UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

"GENERACIÓN AUTOMÁTICA DE DIAGRAMAS UML A PARTIR DE HISTORIAS DE USUARIO EN SCRUM"

TESIS

PARA OBTENER EL GRADO DE
MAESTRA EN INGENIERÍA EN SOFTWARE

PRESENTA

ING. LORENA MARTÍNEZ SIXTO

DIRECTOR DE TESIS
DR. CARLOS ALBERTO FERNÁNDEZ Y FERNÁNDEZ

CODIRECTOR DE TESIS
DR. CHRISTIAN EDUARDO MILLÁN HERNÁNDEZ

HUAJUAPAN DE LEÓN, OAX. A 19 DE JUNIO DE 2026

Tesis presentada el 08/04/2026 ante los siguientes sinodales:

Dr. Eduardo Sánchez Soto

M.C. Everth Haydee Rocha Trejo

Dr. Iván Antonio García Pacheco

Dr. Jorge Rafael Aguilar Cisneros

Director de Tesis:

Dr. Carlos Alberto Fernández y Fernández

Codirector de Tesis:

Dr. Christian Eduardo Millán Hernández

Dedicatoria

A mis padres y hermanas, por su amor incondicional, por su apoyo constante y por ser mi fortaleza a lo largo de este camino. Por sus palabras de aliento, su paciencia y por confiar en mí incluso en los momentos de duda; este logro también les pertenece.

Agradecimientos

Quiero expresar mi más sincero agradecimiento a mi director, el Dr. Carlos Alberto Fernández y Fernández, y a mi codirector, el Dr. Christian Eduardo Millán Hernández por su acompañamiento constante, su orientación académica y las valiosas observaciones que enriquecieron este trabajo. Su disposición, experiencia y compromiso fueron fundamentales durante todo el proceso de investigación y redacción, brindándome guía y claridad en cada etapa.

Asimismo, agradezco al Dr. Iván Antonio García Pacheco coordinador del programa de posgrado, por el apoyo brindado a lo largo de mi formación académica, así como por las facilidades y gestiones que hicieron posible el desarrollo de esta maestría.

De manera muy especial, a mi familia. A mis padres, por estar siempre presentes, por su apoyo incondicional y por darme la fuerza para seguir adelante incluso en los momentos más difíciles. A mis hermanos y primas, por su compañía, su paciencia y por sus palabras de ánimo, que fueron fundamentales a lo largo de este camino.

También a mis amigos de la maestría, con quienes compartí no solo aprendizajes y retos, sino también momentos que hicieron este proceso más llevadero. Su apoyo y compañerismo hicieron que esta experiencia fuera aún más significativa.

Finalmente, a la Secretaría de Ciencias, Humanidades, Tecnología e Innovación (SECIHTI), antes Consejo Nacional de Humanidades, Ciencias y Tecnologías (CONAHCYT), por brindarme el apoyo económico durante el periodo que comprendió la realización de mis estudios de maestría.

Índice

Lista de tablas

Lista de figuras

Resumen

Scrum ha ganado popularidad en el desarrollo de *software* gracias a su enfoque iterativo e incremental. En contraste con las metodologías tradicionales, Scrum a menudo prescinde del modelado detallado con UML (*Unified Modeling Language*). Mientras que UML proporciona una forma estructurada de visualizar y documentar sistemas mediante diagramas, Scrum aboga por la simplicidad y la adaptabilidad. Es decir, en lugar de dedicar tiempo a la creación de modelos detallados antes de comenzar la implementación, los equipos ágiles prefieren la interacción constante con el cliente y la iteración continua, permitiendo así la realización de ajustes ágiles y rápidos a medida que se desarrolla el producto. No obstante, el realizar el modelado UML con la ayuda del Procesamiento del Lenguaje Natural (NLP, por sus siglas en inglés) representa una evolución significativa, puesto que se alcanza un equilibrio entre la rigurosidad del modelado y la agilidad inherente en las metodologías ágiles. De esta manera, se potencia la adaptabilidad al permitir ajustes rápidos y continuos a medida que se recopila y analiza la información. Los equipos ágiles, guiados a menudo por historias de usuario, pueden aprovechar esta capacidad para responder de manera eficiente a los cambios en los requisitos, manteniendo la flexibilidad esencial en entornos dinámicos. Por lo tanto, este trabajo de tesis propone un conjunto de prácticas apoyadas de una herramienta computacional para la generación automática de diagramas UML a partir de historias de usuario utilizando NLP. La metodología empleada se basó en el desarrollo de un sistema que integra Modelos de Lenguaje Largo (LLMs, por sus siglas en inglés) y técnicas de recuperación de información para la interpretación de las historias de usuario y generar los diagramas UML. La propuesta fue evaluada mediante un caso de estudio en el que se utilizaron historias de usuario reales para generar diagramas de clases y de actividades. Los resultados obtenidos se analizaron mediante una evaluación cuantitativa utilizando las métricas de ROUGE, así como una evaluación cualitativa realizada por expertos, empleando una escala Likert para valorar los distintos criterios relacionados con la calidad de los diagramas generados. Los resultados mostraron que la herramienta permite obtener representaciones visuales comprensibles y útiles para apoyar la comprensión del sistema dentro de equipos agiles, contribuyendo a mejorar la documentación y el proceso de modelado UML en entornos Scrum.

1. Introducción

En las últimas décadas, el desarrollo de *software* ha evolucionado significativamente debido a la creciente relevancia de la tecnología en la sociedad y las empresas. A pesar de las considerables inversiones realizadas con el fin de maximizar ganancias y reducir costos, la mayoría de los proyectos han tenido dificultades para ajustarse a las cambiantes necesidades de los usuarios. Esto ha llevado al surgimiento de las metodologías ágiles como un enfoque fundamental en el desarrollo de *software* (Altameem, 2015).

Las metodologías ágiles representan una respuesta innovadora a los problemas históricos del desarrollo de *software*. Su enfoque en la calidad del producto, la motivación de los desarrolladores y la comunicación efectiva han transformado la forma en que se abordan los proyectos de *software* (Jain et al., 2018), las cuales se basan en el Manifiesto Ágil[1]. El Manifiesto Ágil, elaborado por un grupo de expertos en desarrollo de *software* en 2001, estableció cuatro valores clave: individuos e interacciones sobre procesos y herramientas, *software* funcional sobre documentación exhaustiva, colaboración con el cliente durante la negociación de contratos y respuesta al cambio sobre el siguiente plan. Estos valores y los doce principios asociados proporcionan una guía para abordar los proyectos de *software* con flexibilidad y adaptabilidad, lo que ha transformado la forma en que se desarrolla el *software*, permitiendo la creación de productos de alta calidad que satisfacen las cambiantes necesidades de usuarios y empresas en la era tecnológica (Darrin y Devereux, 2017).

En este sentido, Sachdeva (2016) señaló que algunas de las metodologías ágiles más conocidas incluían a Scrum, Kanban, *Extreme Programming* (XP) y Lean. En el caso de Scrum, esta metodología se enfoca en mejorar la eficiencia a través de la comunicación y la planificación, otorgando a los equipos la libertad para explorar en busca de soluciones y ofreciendo un proceso más ágil en caso de requerir la realización de modificaciones fundamentales. Uno de sus objetivos principales es la reducción de costos gracias a la comunicación continua y la mejora de la calidad, asegurándose de que todos los equipos estén informados acerca de los problemas y las modificaciones (Srivastava et al., 2017).

Aunado a lo anterior, Mahmood et al., (2020) destacaron que Scrum se caracteriza por ser un enfoque específico dentro del contexto de las metodologías ágiles. Su rasgo distintivo reside en la incorporación de tres elementos esenciales: roles, artefactos y eventos. Dentro del conjunto de metodologías ágiles, Scrum sobresale por su simplicidad de implementación, lo que justifica su

[1] El Manifiesto Ágil es un documento redactado en 2001 por 17 expertos en programación que supuso un cambio radical en la forma de desarrollar software, puesto que se consideraba que las metodologías formales eran excesivamente "pesadas" y rígidas por su carácter normativo y fuerte dependencia de planificaciones detalladas previas al desarrollo. Dicho manifiesto se puede consultar en: https://agilemanifesto.org/iso/es/manifesto.html

popularidad en muchas organizaciones. De esta manera, Scrum ha desempeñado un papel fundamental en la gestión del ritmo y el alcance del proyecto, además de que ha contribuido a elevar la calidad de los productos entregados.

De acuerdo con Alsaqqa et al. (2020), el ciclo de vida de Scrum consta de tres etapas principales: planificación, desarrollo de *Sprints*, y revisión y retrospectiva. Estas etapas se repiten de manera iterativa para gestionar proyectos de forma efectiva y adaptativa, y se describen de la siguiente manera:

- **Planeación:** Se establecen los objetivos generales del sistema a desarrollar, las herramientas a utilizar, equipo de trabajo y recursos necesarios para el desarrollo del proyecto (Abrahamsson et al., 2017).
- **Desarrollo:** Se llevan a cabo iteraciones llamadas *Sprints*. Cada *Sprint* tiene una duración de dos a cuatro semanas y se enfoca en la implementación de un conjunto específico de funcionalidades (Schmidt, 2016).
- **Revisión y retrospectiva:** Al final de cada *Sprint*, se realiza una revisión para evaluar el trabajo completado y recopilar retroalimentación. También se lleva a cabo una retrospectiva para analizar el proceso y buscar oportunidades de mejora. Estas evaluaciones ayudan a ajustar la planificación para los futuros *Sprints* (Rodríguez et al., 2019).

En este contexto, un *Sprint* es el marco temporal durante el cual se desarrolla un conjunto específico de funcionalidades del producto. La ejecución comprende varias etapas, iniciando con la planificación, en la que el equipo selecciona los requisitos a implementar. Posteriormente, durante el *Sprint* que generalmente tiene una duración de dos a cuatro semanas, el equipo trabaja en los requisitos seleccionados. A lo largo del *Sprint* se llevan a cabo reuniones diarias para mantenerse al tanto del progreso. Al final del *Sprint*, se realiza una revisión en la que se muestra el trabajo completado, seguida de una reunión de retrospectiva orientada a mejorar el proceso. Finalmente, se procede con la planificación del próximo *Sprint*, considerando las lecciones aprendidas durante la iteración. Este ciclo se repite de manera continua a lo largo del proyecto, lo que permite al equipo adaptarse a los cambios, mejorar su proceso de trabajo y entregar valor de forma incremental, promoviendo la colaboración y la comunicación constante durante el desarrollo de *software* (Srivastava et al., 2017).

En el desarrollo ágil de *software*, las historias de usuario es uno de los métodos más utilizados para representar los requisitos (Lucassen et al., 2016). De hecho, un enfoque ampliamente adoptado en la Ingeniería de Requisitos dentro de entornos ágiles se fundamenta en el uso de este tipo de artefactos. Sommerville (2020) enfatiza que las historias de usuario son una forma efectiva de capturar los requisitos desde la perspectiva del usuario final, y señala que estas deben ser breves y enfocarse en un solo aspecto o característica del sistema. Por su parte, Dimitrijević et al. (2015) describen una historia de usuario como una breve descripción en lenguaje común o de negocio utilizada por los usuarios finales de un sistema de *software*, que captura la esencia de la actividad que un usuario realiza o necesita realizar con el sistema. En este sentido, diversas técnicas empleadas a lo largo del proceso de desarrollo ágil se apoyan en las historias de usuario, tales como la planificación de lanzamientos, la planificación de iteraciones y el seguimiento del avance del proyecto. En consecuencia, las historias de usuario no solamente representan un formato sencillo para documentar los requisitos, sino que también funcionan como un medio de interacción entre los desarrolladores, los usuarios finales y los clientes. Además, constituyen la base para la definición de las funcionalidades del sistema y su posterior implementación (Lucassen et al., 2016).

Las historias de usuario no abarcan la totalidad de los requisitos definitivos del *software*, ya que funcionan como el punto de partida de las conversaciones entre profesionales y partes interesadas. De esta manera, los requisitos en sí se derivan a partir de las historias proporcionadas por las partes interesadas. De acuerdo con la investigación de Gilson y Irwin (2018), las historias de usuario tienen la capacidad de desglosar de manera detallada las necesidades del *software*, lo que puede resultar en listas extensas de requisitos. Por lo tanto, es de suma importancia dedicar un esfuerzo significativo a la adecuada documentación de las historias de usuario, dado que historias mal redactadas pueden dar lugar a diversos problemas, incluyendo la omisión de requisitos no funcionales y la complejidad en cuanto a la priorización de los requisitos.

Por otro lado, el Lenguaje Unificado de Modelado (UML, por sus siglas en inglés) desempeña un papel esencial en la creación de sistemas de *software*, ya que se presenta como un lenguaje visual empleado para, entre otras tareas, la representación y documentación de los requisitos de *software*. UML facilita a los desarrolladores la comprensión de cómo opera el *software* y, además, puede ser de utilidad para organizar las necesidades de los usuarios. En la Ingeniería de *Software* se hace un amplio uso de diversos tipos de modelos de UML que incluyen a los diagramas de clases, casos de uso, diagramas de actividades, diagramas de secuencia, diagramas de objetos, diagrama de modelo de dominio, diagramas de máquinas de estados, diagramas de comunicación y paquetes (Koç et al., 2021). Cada uno de estos diagramas posee características y propósitos específicos que podrían contribuir en gran medida a mejorar los resultados en el desarrollo ágil de *software*. En palabras de Mornie et al. (2023), estos diagramas se pueden construir a partir de las historias de usuario, ya que estas proporcionan los requisitos generales y esenciales de las partes interesadas. Sin embargo, transformar las historias de usuario en diagramas UML no es una tarea sencilla. Este proceso requiere un esfuerzo importante, dedicación de tiempo y experiencia, ya que las historias de usuario suelen presentar inconsistencias, lagunas y tareas incorrectas.

La investigación de Alsaqqa et al., (2020) señala que en el desarrollo ágil de *software* se prioriza la funcionalidad sobre la documentación extensa. Si bien la documentación desempeña un papel importante en el proceso de desarrollo, resulta fundamental gestionar adecuadamente los recursos y el tiempo dedicados a su elaboración para evitar que se convierta en un obstáculo para el progreso del proyecto. La excesiva dependencia de la documentación ha sido un desafío frecuente, ya que mantener los documentos actualizados en paralelo con el código puede ser una tarea compleja, especialmente cuando los requisitos cambian con frecuencia. En este contexto, Scrum reduce la carga documental para los equipos de desarrollo en comparación con enfoques que requieren una documentación extensa, como la elaboración de múltiples artefactos UML, los cuales a menudo se consideran tareas pesadas. En las metodologías ágiles, el uso de documentación y artefactos UML no son un requisito obligatorio. De hecho, uno de los principios del Manifiesto Ágil establece la importancia de construir proyectos en torno a personas motivadas, lo que se puede aplicar a la decisión de incorporar artefactos UML. Es decir, cuando el equipo de desarrollo comprende claramente cómo el uso de UML puede beneficiar la creación de soluciones modulares y escalables, así como mejorar la comunicación entre los miembros del equipo y las partes interesadas, es probable que sugieran proactivamente la inclusión de un diagrama UML en sus tareas ágiles (Gallud y Fardoun, 2018).

En este sentido, la comunicación adquiere una importancia sustancial en contextos complejos como los proyectos de Ingeniería de *Software*, donde la participación de múltiples individuos, la presencia de incertidumbre y desacuerdos son comunes. La colaboración entre personas que desempeñan roles diversos y provienen de distintos trasfondos se convierte en un elemento fundamental en el proceso de desarrollo de *software*, lo que subraya la necesidad de garantizar una transmisión de información precisa entre las partes interesadas (de Pinho, 2020).

Las historias de usuario se han vuelto esenciales en el desarrollo ágil de *software*, siendo consideradas eficaces para describir los objetivos del sistema. Sin embargo, la gestión continua de la acumulación de productos puede resultar laboriosa y propensa a errores, especialmente al evaluar la calidad o alcance de las historias y al supervisar el panorama general del sistema. La generación de historias a diagramas UML posibilita la validación de conceptos y pasos funcionales, la detección de historias con formato incorrecto o redundante, y abre la puerta para un análisis sistemático automatizado. Además de que el modelado UML ha sido reconocido como una herramienta eficaz para fines de comunicación y análisis (Gilson e Irwin, 2018). Dado que las historias de usuario son parte de la documentación de facto dentro del enfoque ágil y ésta se percibe como un reflejo de la implementación, pueden considerarse como una estrategia para mitigar riesgos y mejorar el proyecto. La documentación ágil se dedica a registrar de manera concisa los conceptos estables, poniendo énfasis en las necesidades reales del cliente. Por lo tanto, una documentación precisa facilita la comprensión del proyecto, actuando como una guía directa que se conecta con los puntos relevantes. En metodologías ágiles, se enfatiza una documentación precisa y centrada en el desarrollo, evitando paquetes extensos y priorizando la implementación (Shafiq y Waheed, 2018).

Para Shaikh et al., (2021) la detección de errores resulta más simple y económica en las etapas iniciales en comparación con las fases posteriores, ya que el ciclo de desarrollo precede al incremento de costos y esfuerzos. El modelo UML de clases es esencial en las metodologías de *software* actuales y se construye en las fases iniciales del desarrollo, por lo que la identificación y corrección de errores en este modelo puede representar un ahorro significativo en tiempo y costos asociados al desarrollo de *software*. En este mismo orden de ideas, Kochbati et al., (2021) consideran que la tarea de derivar manualmente modelos arquitectónicos a partir de requisitos expresados en lenguaje natural es laboriosa y consume un tiempo considerable, especialmente si se trata del desarrollo de *software* de mayor complejidad. Asimismo, el crecimiento en el tamaño y la complejidad del *software* resalta la necesidad de descomponerlo en subsistemas durante las fases iniciales del desarrollo, ya que esta descomposición contribuirá a obtener un diseño más eficaz de la arquitectura del *software*. Así, la automatización de esta descomposición es altamente deseable en las etapas iniciales del desarrollo, especialmente en el caso de sistemas a gran escala. Este enfoque automatizado no solo facilita el proceso de diseño, sino que también conlleva ahorros significativos en costos y tiempo.

En este sentido, diversas investigaciones han explorado mecanismos para automatizar la generación de diagramas UML a partir de requisitos expresado en lenguaje natural. La investigación de Elallaoui et al., (2015), por ejemplo, presentó una propuesta que buscó simplificar el proceso de convertir historias de usuario en diagramas de secuencia de UML. Es decir, se trató del diseño de un algoritmo para automatizar la conversión de historias de usuario en diagramas de secuencia en el contexto del proceso establecido por Scrum, con el objetivo de facilitar la creación de casos de prueba. Para llevar a cabo esta automatización, se implementó un algoritmo que analizaba un archivo de texto que contenía un conjunto de historias de usuario, para generar posteriormente un archivo XML para cada una de ellas. El archivo XML resultante se transformaba luego en un diagrama de secuencia mediante el uso del complemento SDK de la herramienta UML2 para Eclipse. El objetivo principal de esta investigación fue simplificar el trabajo de los diseñadores y desarrolladores al procesar historias de usuario y, al mismo tiempo, facilitar la generación automática de casos de prueba para los *testers*. Sin embargo, la investigación no presenta resultados empíricos que demuestren la eficiencia del algoritmo mencionado.

Por otro lado, la investigación de Elallaoui et al., (2018) destacó el creciente respaldo de la comunidad de desarrolladores hacia los métodos ágiles, en especial Scrum, resaltando la utilidad de las historias de usuario en la redacción de requisitos funcionales. Aprovechando el trabajo realizado

en la transformación de modelos bajo el enfoque de la Arquitectura Dirigida por Modelos (MDA, por sus siglas en inglés), su propuesta se centró en la automatización del proceso de convertir historias de usuario en diagramas de casos de uso UML, empleando técnicas del Procesamiento de Lenguaje Natural (NLP, por sus siglas en inglés), específicamente con el analizador TreeTagger[2]. Esta investigación se validó mediante la comparación entre los modelos de historias de usuario generados mediante la aplicación del NLP y los modelados generados manualmente, obteniendo precisiones que oscilaron entre el 87% y el 98%.

En este sentido, se argumenta que las metodologías ágiles se han vuelto esenciales en el ámbito empresarial debido a su enfoque en la obtención de beneficios, la activa participación de las partes interesadas y la capacidad para adaptarse rápidamente a requisitos cambiantes. Aunque Scrum, como una metodología ágil ampliamente adoptada, proporciona un modelo ágil para el desarrollo de *software*, aún existen deficiencias en cuanto a la arquitectura de la solución y al modelado de requisitos. En este contexto, Kussunga y Ribeiro (2019) buscaron abordar estas limitaciones presentando una propuesta de un entorno visual diseñado para respaldar el *Product Backlog* y el *Sprint backlog* dentro de Scrum. Este entorno tuvo como objetivo facilitar la priorización, el seguimiento y la toma de decisiones en relación con requisitos valiosos y fácilmente implementables. El estudio no presenta resultados de una evaluación empírica que permitan validar la propuesta presentada.

De manera similar, la investigación de Chen et al., (2019) presentó un caso de estudio en el que se evaluaron diferentes proyectos de *software* a gran escala que enfrentaron requisitos poco claros por parte de los clientes. Adoptando Scrum como método ágil de desarrollo y utilizando diagramas UML, se exploró la creación de un sistema automatizado de programación de cursos para escuelas primarias y secundarias. A través de entrevistas y la aplicación de diagramas UML se logró integrar los requisitos de los clientes, facilitando la documentación del proceso de desarrollo de *software* y el diseño de funciones que respondieran las necesidades de los clientes. Así, se implementó un método que permitía verificar su viabilidad y abordar los desafíos surgidos durante el desarrollo. Los resultados obtenidos se presentaron como una valiosa referencia para empresas de *software* que enfrentan la complejidad de requisitos poco claros al llevar a cabo proyectos a gran escala.

La investigación de Kochbati et al., (2021) propuso una estrategia fundamentada en el aprendizaje automático para segmentar de manera automática el sistema en subsistemas y generar modelos arquitectónicos preliminares a partir de las historias de usuario en lenguaje natural dentro del marco del proceso de Scrum. La estrategia se sustentó en tres componentes fundamentales. En primer lugar, se calculó la similitud de requisitos a nivel de palabra utilizando word2vec[3] como modelo predictivo. En segundo lugar, se calculó la similitud a nivel de requisitos mediante una fórmula de puntuación. En tercer lugar, se empleó el algoritmo de agrupación aglomerativa jerárquica para agrupar requisitos con similitud semántica, proporcionando así una descomposición temprana del sistema. Finalmente, se implementó un conjunto de heurísticas específicas del NLP para extraer elementos relevantes necesarios para construir modelos a partir de los grupos identificados. Este enfoque fue sometido a evaluación a través de tres casos de estudio, respaldados por un conjunto de indicadores clave de rendimiento (KPI, por sus siglas en inglés). En relación con la solución de agrupación, se logró prescindir de la intervención manual al proponer una cantidad óptima de

[2] TreeTagger es una herramienta de etiquetado de texto, análisis de las oraciones y extracción del lema. Fue desarrollada por Helmut Schmid en el proyecto TC del Instituto de Lingüística Computacional de la Universidad de Stuttgart con el objetivo de ser utilizado para el etiquetado y lematización de voz.

[3] Word2vec es una técnica de Inteligencia Artificial que permite el análisis algorítmico de textos mediante la conversión de palabras en vectores numéricos (Ayyadevara, 2018).

agrupaciones. Adicionalmente, los resultados de la evaluación de los conjuntos generados en comparación con los grupos reales, determinados por expertos, revelaron *F-measures*[4] razonables, alcanzando aproximadamente el 80%. En lo que respecta a la precisión y recuperación de los modelos de casos de uso de UML generados, se alcanzaron tasas del 100% para la detección de actores, hasta un 98% para la identificación de casos de uso y hasta un 97% para la identificación de relaciones.

Finalmente, el estudio realizado por Cámara et al., (2023) se enfocó en analizar las habilidades actuales de ChatGPT en el ámbito del modelado, con el propósito de identificar tanto sus aspectos positivos como las limitaciones que presenta. Los resultados indican que la eficacia de ChatGPT en el modelado de *software* se ve restringida por problemas de rendimiento que se manifiestan en deficiencias sintácticas y semánticas, falta de coherencia en las respuestas y desafíos en la escalabilidad. Además, se contemplan consideraciones sobre el potencial de los LLMs en el modelado de *software* a corto plazo. También se ofrecen sugerencias sobre cómo la comunidad de modelado puede contribuir a mejorar las capacidades actuales de ChatGPT.

Como se puede observar, una tendencia actual para fomentar la integración efectiva de los diagramas UML en el ciclo de vida de Scrum es agilizar su generación a partir de las historias de usuario utilizando el NLP. Las técnicas de NLP ofrecen oportunidades para mejorar la calidad de las historias de usuario, ya que permiten analizar, extraer y comprender la información contenida en ellas. De esta manera, el NLP ha demostrado ser una herramienta ampliamente aplicable en diversas áreas de la Ingeniería de *Software*, como la gestión de requisitos de *software*, la identificación de actores y acciones en documentos de requisitos, la extracción de características de *software* y las pruebas de *software* (Raharjana et al., 2021).

Por lo tanto, en el ámbito del desarrollo de *software*, la generación de diagramas UML a partir de historias de usuario en el marco de Scrum se ha convertido en un desafío esencial. Estas representaciones gráficas son cruciales para comprender y comunicar la arquitectura y el diseño de un sistema de *software*. Sin embargo, su creación manual puede resultar costosa en términos de tiempo y recursos, lo que a menudo dificulta una comunicación efectiva entre los equipos de desarrollo y las partes interesadas.

Considerando todo lo anteriormente expuesto, esta tesis propuso como principal contribución el diseño de un conjunto de prácticas orientadas a la generación de diagramas UML a partir de historias de usuario en entornos de desarrollo ágil basados en Scrum. Dicho conjunto de prácticas fue implementado y validado mediante una herramienta de apoyo que integra técnicas de NLP, modelos LLMs y sistemas RAG. En este contexto, la investigación se enfocó en el uso del NLP como un mecanismo para interpretar historias de usuario escritas en lenguaje natural y transformarlas en representaciones visuales estructuradas mediante diagramas UML. El propósito fundamental de la propuesta consistió en mejorar la comprensión de las historias de usuario, facilitando una comunicación más clara y efectiva entre los miembros del equipo de desarrollo y las partes interesadas del proyecto. Al proporcionar representaciones visuales derivadas directamente de requisitos descritos en lenguaje natural, se buscó reducir posibles ambigüedades o interpretaciones incorrectas durante las etapas tempranas del desarrollo. Asimismo, la investigación incluyó una evaluación comparativa de modelos de lenguaje y una validación cuantitativa y cualitativa de los diagramas generados, con el fin de analizar la viabilidad de la propuesta y su aplicación en entornos ágiles de desarrollo de *software*.

[4] *F-measures*, también conocida como *F-score*, es una métrica estadística que combina las métricas de precisión y exhaustividad en un solo valor, proporcionando así una medida equilibrada del rendimiento de un modelo de clasificación. La definición se puede consultar en: https://deepai.org/machine-learning-glossary-and-terms/f-score

De esta manera, la propuesta buscó fortalecer las actividades de análisis y diseño en proyectos ágiles de desarrollo de *software*, favoreciendo una mejor alineación entre los requisitos del sistema y su implementación. En consecuencia, se esperó que la integración del conjunto de prácticas, con el apoyo de la herramienta desarrollada, contribuyera a mejorar la eficiencia y la calidad de los procesos de desarrollo en entornos ágiles, promoviendo una comprensión más profunda de los requisitos y facilitando la toma de decisiones a lo largo del proceso de desarrollo de *software*.

1.1. Limitaciones y delimitaciones de la tesis

La investigación consideró las siguientes limitaciones:

1. La calidad de las historias de usuario puede influir significativamente en la efectividad de la herramienta, especialmente si éstas carecen de detalles claros o presentan inconsistencias.
2. La eficacia de la herramienta depende en gran medida del modelo de NLP que se incorpore. La herramienta podría enfrentar desafíos si el modelo NLP no logra comprender y procesar adecuadamente las historias de usuario.
3. La evolución del modelo de NLP puede ser un factor crítico, ya que posibles cambios en el modelo a lo largo del tiempo podrían afectar la consistencia y estabilidad de la herramienta. La falta de compatibilidad con versiones anteriores podría generar complicaciones.

De manera similar, se consideraron las siguientes delimitaciones:

1. La herramienta de generación de diagramas UML se enfoca exclusivamente en la creación de diagramas de clase iniciales y diagramas de actividades.
2. La herramienta está diseñada exclusivamente para procesar historias de usuario redactadas en inglés.
3. La implementación de la herramienta está especialmente orientada a integrarse con la metodología Scrum.

1.2. Hipótesis

La hipótesis que sustentó esta investigación fue la siguiente:

"El uso de un conjunto de prácticas apoyadas por una herramienta para la generación de diagramas UML a partir del lenguaje natural facilitará la comprensión de las historias de usuario en el contexto de las metodologías ágiles".

1.3. Objetivos del trabajo

Tomando en cuenta lo anteriormente expuesto, los objetivos abordados con el desarrollo de este proyecto de tesis fueron los siguientes:

1.3.1. Objetivo general

"Establecer un conjunto de prácticas basadas en la creación automática de diagramas UML para mejorar la comprensión de las historias de usuario en el contexto de metodologías ágiles".

1.3.2. Objetivos específicos

Es importante enfatizar en que la consecución de este objetivo general dependió de los siguientes objetivos específicos:

- Investigar las buenas prácticas relacionadas a mejorar la comprensión de historias de usuario en metodologías ágiles.
- Diseñar un conjunto de prácticas para comprender las historias de usuario en metodologías ágiles mediante diagramas UML.
- Desarrollar una herramienta para la generación automática de diagramas de UML a partir de un grupo de historias de usuario como soporte para las prácticas.
- Realizar una evaluación empírica sobre el conjunto de prácticas propuestas mediante un juicio de expertos.

1.4. Aproximación a la solución

La comprensión efectiva de las historias de usuario en el desarrollo de *software* en entornos ágiles, como Scrum, se revela como un factor crucial para el éxito de los proyectos. Con el fin de afrontar este desafío, se planteó la implementación de un conjunto de prácticas diseñadas específicamente para potenciar la comprensión de las historias de usuario. Este enfoque se fortalece mediante el desarrollo de una herramienta especializada que posibilite la generación automática de diagramas UML. Con esta estrategia integrada, se buscó no solamente mejorar la interpretación de las historias de usuario, sino también elevar la calidad del diseño y la documentación en el marco de los proyectos ágiles, asegurando así un desarrollo más robusto y alineado con las expectativas del cliente.

En este sentido, la Figura 1.1 muestra las fases que se desarrollaron en esta tesis para el diseño, desarrollo y evaluación de una solución alternativa al problema identificado. Las fases se describieron de la siguiente manera:

1. En la fase inicial se empleó la metodología de revisión integrativa para identificar y analizar la literatura en busca de las mejores prácticas orientadas a mejorar la comprensión de las historias de usuario dentro de las metodologías ágiles. Al elegir esta metodología se buscó obtener una visión completa y enriquecedora del panorama actual (Gómez-Luna et al., 2014). Esta revisión desempeñó un papel fundamental en la construcción de una base conceptual sólida que facilitó la identificación y formulación de un conjunto específico de prácticas que se adaptaron a esta tesis.

Figura 1 Fases propuestas para desarrollar una solución alternativa.

2. En la segunda fase se procedió al diseño meticuloso de un conjunto innovador de prácticas diseñadas para su implementación en entornos ágiles de desarrollo de *software*, con el objetivo concreto de potenciar la comprensión de historias de usuario. Este conjunto de nuevas prácticas se basó en principios fundamentales identificados durante la revisión literaria, estableciendo criterios clave para su aplicación. Esto garantizó su pertinencia y utilidad práctica en el contexto ágil, asegurando así que contribuyeran de manera efectiva a la mejora del proceso de desarrollo.
3. En la tercera fase se procedió con el desarrollo de una herramienta computacional para la generación automática de diagramas de UML a partir de historias de usuario haciendo uso de un modelo de NLP (Moreira et al., 2021).

   Para el desarrollo de la herramienta, se llevaron a cabo los siguientes pasos:

   - **Selección del conjunto de datos:** Se realizó una revisión minuciosa de conjuntos de datos públicos que contuvieran historias de usuario. La elección se basó en criterios de calidad, relevancia y disponibilidad, asegurando una adecuada representación para el entrenamiento del modelo. Se empleó el conjunto de datos público denominado "*Requirements data sets (user stories)*" (Dalpiaz, 2018).
   - **Limpieza de datos:** Después de la selección del conjunto de datos, se inició con la etapa de preprocesamiento, una fase esencial en el análisis de datos. Este procedimiento implicó la identificación, corrección o eliminación de errores y discrepancias presentes en el conjunto de datos, los cuales pudieran afectar en gran medida la capacidad y desempeño de los LLMs. El objetivo primordial de este proceso fue mejorar la calidad de los datos, asegurando que fueran más precisos, coherentes y aptos para un análisis detallado. En este sentido, la eficacia de un modelo de aprendizaje automático se ve restringida por la calidad de los datos. Por lo tanto, es esencial comprender exhaustivamente el conjunto de datos antes de utilizarlo en cualquier aplicación, ya que la falta de comprensión puede dar lugar a análisis imprecisos y decisiones poco fiables (Jain et al., 2020).
   - **Generación de *prompt*:** En el contexto de los LLMs, el término "*prompt*" se refiere a una instrucción o entrada expresada en lenguaje natural, proporcionada al modelo con el objetivo de generar una respuesta (Webson y Pavlick, 2021). La generación del *prompt* tiene como propósito el potenciar la capacidad del modelo para comprender y brindar respuestas precisas. La elaboración de preguntas específicas y la inclusión de detalles contextualmente relevantes fueron elementos cruciales para optimizar la interacción del modelo con las consultas. Este proceso buscó garantizar que las entradas del modelo fueran claras y efectivas, maximizando así su rendimiento (Kirchenbauer et al., 2023).
   - **Implantación del LLM:** Los LLMs han logrado avances significativos en el procesamiento de lenguaje natural, estos modelos se entrenan con grandes cantidades de datos de texto y tienen la capacidad de generar texto similar a un humano, responder preguntas y completar tareas con alta precisión (Kasneci et al., 2023). Zhao et al., (2023) argumentaron que para la aplicación de los LLMs existen cinco tipos de tareas clásicas de NLP: a nivel de palabra, nivel de oración, etiquetas de secuencia, extracción de relaciones y generación de texto. Por lo tanto, la implementación del modelo de lenguaje largo constituyó la piedra angular de la solución presentada en esta tesis, puesto que se planteó utilizar una arquitectura LLM avanzada capaz de aprender patrones complejos y entender la semántica de las consultas. El entrenamiento se realizó con el conjunto de datos limpio y preparado, buscando así maximizar la capacidad del modelo para generar

respuestas coherentes y contextualmente adecuadas. Esta aproximación integral buscó no solo abordar el problema de manera efectiva, sino también establecer un marco metodológico sólido que respalde la validez y la confiabilidad de los resultados generados por el LLM. Por otro lado, para llevar a cabo el proceso de desarrollo de la herramienta, se optó por el uso de la metodología ágil Scrum. Esta elección se fundamentó en la capacidad de Scrum para proporcionar flexibilidad y adaptabilidad, permitiendo con esto ajustar la estrategia propuesta ante posibles cambios a medida que los modelos de NLP evolucionen (Alsaqqa et al., 2020).

4. De acuerdo con Hron y Obwegeser (2022), el marco ágil Scrum, ampliamente adoptado en el desarrollo de *software*, ofrece una estructura iterativa y flexible para proyectos. En este contexto, en la cuarta fase se planteó la integración de un conjunto de prácticas específicas con el objetivo de mejorar la comprensión de las historias de usuario. Esta integración se llevó a cabo mediante el uso de una herramienta especializada para la generación de diagramas UML. La implementación del conjunto de prácticas y la herramienta correspondiente se enfocaron en mejorar la comprensión de los requisitos del usuario, contribuyendo así a la eficacia del desarrollo. La generación de diagramas UML es una herramienta clave para expresar de manera clara y visual la arquitectura y estructura del sistema, facilitando la comunicación y colaboración entre los miembros del equipo (Lopes et al., 2019).

   Es crucial destacar que esta integración tuvo como objetivo mejorar los procesos de diseño y documentación sin comprometer la entrega de valor en ciclos cortos y continuos. La agilidad inherente a Scrum se mantiene intacta, y el enfoque en la generación eficiente de diagramas UML se alineó con la necesidad de adaptabilidad y rapidez en la entrega de soluciones de *software*. Por lo tanto, la propuesta de integrar el conjunto de prácticas específicas y la herramienta para la generación de diagramas UML en el marco de Scrum potencia la comprensión de historias de usuario y por consecuencia amplia la documentación existente, todo ello sin comprometer la esencia ágil que caracteriza a Scrum y sin afectar la entrega de valor en ciclos rápidos y continuos (Wang et al., 2017).

   De esta manera, se propuso la integración de un conjunto de prácticas durante los eventos de Planeación del *Sprint* y Ejecución del *Sprint* (véase Figura 1.2), centrándose en la mejora de la comprensión de las historias de usuario incorporadas en el *Sprint backlog*. Para facilitar este proceso, se utilizó una herramienta de generación automática de diagramas UML. Así, esta iniciativa tuvo como objetivo principal mejorar la calidad y claridad en la planificación y ejecución de las actividades del *Sprint*.

   La herramienta de generación de diagramas UML es una aliada fundamental al proporcionar una visualización efectiva de las interrelaciones y estructuras relacionadas con las historias de usuario. Además de los beneficios visuales, es importante destacar cómo esta implementación impacta a la colaboración dentro de los involucrados en los eventos de Planeación del *Sprint* y Ejecución del *Sprint*. La herramienta de generación de diagramas UML facilita la comunicación al proporcionar un lenguaje visual común para discutir y comprender las historias de usuario. La visualización de las interrelaciones pretende promover un análisis más efectivo durante la Planeación del *Sprint*, permitiendo una asignación de tareas más informada.

   Para respaldar la efectividad del conjunto de prácticas, se planificó una evaluación continua. La retroalimentación del equipo y de los *stakeholders* fue fundamental para ajustar y mejorar el enfoque a medida que evolucionaba el proyecto. La integración de estas prácticas con otros elementos clave de Scrum, como la Revisión del *Sprint* y la Retrospectiva del *Sprint*, se consideró para garantizar una sinergia efectiva en todas las fases del ciclo de desarrollo.

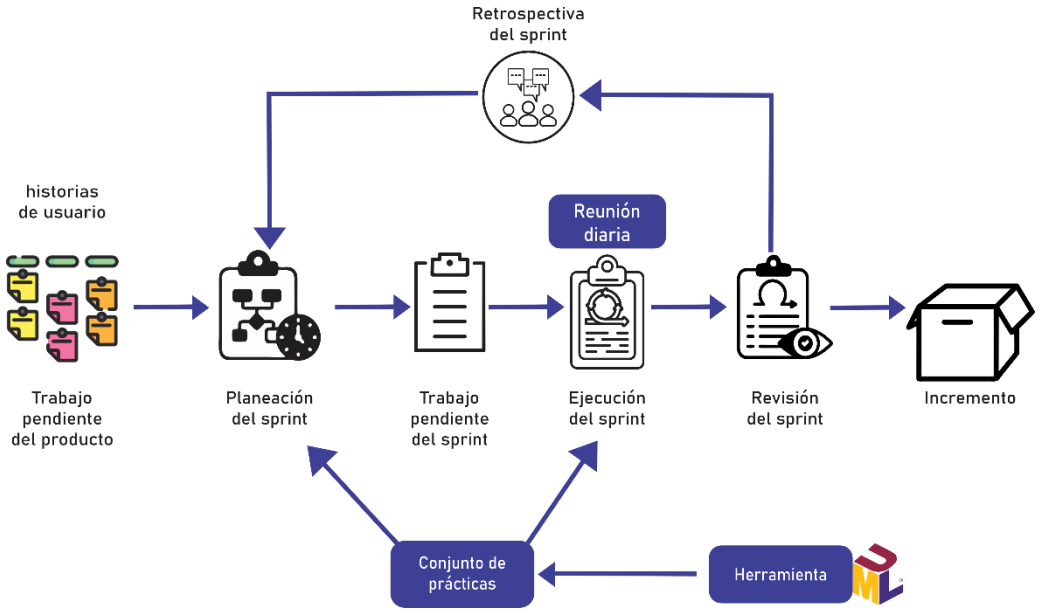


Figura 2 Integración del conjunto de prácticas en el ciclo de vida de Scrum.

5. Después de la implementación del conjunto de prácticas y el uso de la herramienta para la generación de diagramas UML, se realizó una evaluación empírica mediante un juicio de expertos, iniciando con la identificación y selección de expertos en entornos ágiles como Scrum, para continuar con el diseño de un cuestionario y entrevistas estructuradas con preguntas específicas para la recopilación de sus opiniones. Esta evaluación se fundamentó en la experiencia y conocimientos especializados de expertos clave en desarrollo de *software*, arquitectura de sistemas y metodologías ágiles. De esta manera, la implementación del conjunto de prácticas y la herramienta de generación de diagramas UML se sometió a un análisis detallado, considerando aspectos como la eficacia en la comprensión de las historias de usuario, la mejora en la calidad del diseño y la agilidad del proceso. Este enfoque empírico permitió la obtención de una visión informada y cuantificable de cómo el conjunto de prácticas, con ayuda de la herramienta de generación de diagramas UML, impactan en la eficiencia y la entrega de valor en ciclos cortos, contribuyendo así a una toma de decisiones respaldada por datos concretos en el contexto de un proyecto de desarrollo de *software* (Felderer y Travassos, 2020).

Finalmente, una vez descrito el enfoque general de la investigación y las fases propuestas para el diseño, desarrollo y evaluación de la solución, esta tesis se estructura en seis capítulos. El Capítulo 1 presentó el contexto de la investigación, el planteamiento del problema, la justificación, los objetivos, la hipótesis y la propuesta general de solución. El Capítulo 2 expone los fundamentos teóricos y los trabajos relacionados que sustentan la investigación. El Capítulo 3 describe la metodología empleada para el desarrollo del estudio. El Capítulo 4 presenta el diseño del conjunto de prácticas y el desarrollo de la herramienta para la generación automática de diagramas UML. El Capítulo 5 presenta el caso de estudio desarrollado para evaluar la propuesta, así como los resultados obtenidos mediante la evaluación cualitativa realizada por expertos. Finalmente, el Capítulo 6 presenta las conclusiones, las contribuciones alcanzadas, las limitaciones identificadas y las líneas de trabajo futuro derivadas de la investigación.

2. Marco teórico

En este capítulo se presentan los fundamentos teóricos necesarios para contextualizar la generación automática de diagramas UML a partir de historias de usuario dentro del marco de desarrollo ágil Scrum. El objetivo principal de este enfoque es mejorar la comprensión y la comunicación de los requisitos del *software* en un entorno ágil. Además, se analizan diversas propuestas previas aplicadas en diferentes contextos para identificar similitudes y diferencias con respecto a esta investigación.

2.1. Metodologías ágiles

La Ingeniería de *Software* es una disciplina fundamental en el desarrollo de sistemas informáticos. Surgió en 1968, cuando un grupo de estudio de la OTAN se enfocó en los desafíos del *software* y promovió su aplicación en diversas áreas de nuestras vidas. Este campo se considera parte de la familia de la Ingeniería debido a su enfoque en procesos estructurados y metodológicos para el análisis, diseño y desarrollo de productos de *software* (Alsaqqa et al., 2020). Además, es un campo que ha experimentado notables mejoras, buscando mantenerse actualizado con los avances tecnológicos y las exigencias comerciales actuales. El desarrollo de enfoques efectivos para la producción de productos de *software* final es fundamental en este proceso, siendo el desarrollo ágil de *software* uno de los logros destacados en este sentido (Behutiye et al., 2020).

Las metodologías ágiles surgen como una alternativa a los enfoques tradicionales de desarrollo, especialmente adecuadas para productos cuya definición detallada es difícil de obtener desde el principio o cuyo valor sería menor si se definiera previamente, en comparación con la retroalimentación continua durante el proceso de construcción. Esto es especialmente relevante en sistemas donde los usuarios no pueden identificar completamente sus necesidades hasta que ven un componente funcional (Altameem, 2015). Las características tanto de estos productos como de los usuarios, combinadas con un entorno de cambio acelerado, hacen que las metodologías ágiles sean una opción viable para comprender mejor los requisitos. Manteniendo aspectos esenciales de las metodologías tradicionales, las metodologías ágiles se enfocan en otros aspectos del proyecto, como la reducción de la documentación al mínimo necesario, favoreciendo la comunicación directa y cara a cara entre todos los miembros del equipo. También promueven la colaboración activa de los usuarios en todas las etapas del proceso de desarrollo, así como el desarrollo incremental del *software* con iteraciones cortas que ofrecen soluciones a medida. Además, buscan reducir drásticamente el tiempo de desarrollo sin comprometer la calidad del producto (Calo et al., 2010).

De acuerdo con la investigación realizada por Gutiérrez et al. (2020), el propósito de las metodologías ágiles es resolver los problemas que surgen tras la implementación y uso de un *software* o producto, considerando que las expectativas y demandas de los usuarios son ahora más urgentes y

cambiantes. Las metodologías ágiles, son reconocidas por su flexibilidad y enfoque colaborativo, subrayan la importancia de la comunicación efectiva en el entorno de trabajo. La habilidad de comunicarse de manera clara y abierta no solo fortalece la unidad del equipo, sino que también desempeña un papel crucial en el logro de resultados efectivos. La comunicación eficaz permite a los miembros del equipo establecer relaciones sólidas que son fundamentales para el éxito del proyecto. Del mismo modo, la implementación de prácticas ágiles en el desarrollo de *software* promueve una comunicación fluida entre todas las partes interesadas y los equipos involucrados. Al entender la importancia de una comunicación efectiva y compartir información de manera adecuada, los miembros del equipo pueden lograr el éxito del proyecto de manera más efectiva (Altameem, 2015).

2.1.1. Manifiesto Ágil

De acuerdo con Abrahamsson et al. (2002), el movimiento ágil en la industria del *software* tuvo su origen con la publicación del Manifiesto Ágil para el desarrollo de *software* en 2001, elaborado por un grupo de profesionales y consultores del ámbito. El manifiesto destaca cuatro valores fundamentales que respaldan el proceso de desarrollo de *software*.

a. **Individuos e interacciones sobre procesos y herramientas:** Las metodologías ágiles, en busca de una mayor productividad, valoran al equipo humano como factor crucial para el éxito. Reconocen que un equipo bien capacitado, con habilidades técnicas, adaptabilidad y habilidades de colaboración, supera en eficacia a las herramientas y procesos estrictos. Por consiguiente, priorizan la formación de equipos sólidos sobre la implementación de estructuras rígidas, permitiendo que el equipo defina el entorno más adecuado según las circunstancias (Gutiérrez et al., 2020).
b. ***Software* funcionando sobre documentación extensiva:** Los profesionales del desarrollo de *software*, aunque no consideren la producción de documentos como su actividad principal, reconocen su importancia y son conscientes del tiempo y los costos asociados con mantener una documentación completa y actualizada. Las metodologías ágiles valoran la documentación como parte integral del proceso y del producto final en un proyecto de desarrollo de *software*. Sin embargo, enfatizan la necesidad de crear únicamente los documentos esenciales, los cuales deben ser breves y centrarse en lo fundamental, dando prioridad al contenido sobre la presentación. Dentro del enfoque ágil, se promueven mecanismos más dinámicos y económicos para la documentación, como la comunicación directa, el trabajo en equipo, la auto documentación y el cumplimiento de estándares (Alsaqqa et al., 2020).
c. **Colaboración con el cliente sobre negociación contractual:** Las metodologías ágiles promueven la integración directa del cliente en el equipo de desarrollo de *software*, reconociendo su participación como esencial para el éxito del proyecto. Esto contrasta con la tradición de posturas distantes entre cliente y equipo, favoreciendo una colaboración mutuamente beneficiosa. La participación activa del cliente a lo largo del proyecto, proporcionando correcciones y recomendaciones, se considera fundamental debido a su conocimiento de sus propias necesidades y deseos (Gutiérrez et al., 2020).
d. **Respuesta ante el cambio sobre seguir un plan:** Dado el constante cambio en la tecnología y la dinámica de la sociedad moderna, los proyectos de desarrollo de *software* enfrentan frecuentemente modificaciones durante su ejecución. Estos cambios pueden variar desde simples ajustes en la personalización del *software* hasta alteraciones en las leyes, la introducción de nuevos productos en el mercado, cambios en el comportamiento de la competencia y las tendencias tecnológicas emergentes, entre otros. En contraste con las

> metodologías tradicionales que tienden a buscar una definición exhaustiva desde el principio, las metodologías ágiles reconocen la necesidad de la flexibilidad en la planificación. Por lo tanto, en lugar de una planificación rígida, se adopta una estrategia más adaptable. Esto implica establecer planificaciones detalladas para cortos períodos y mantener planificaciones más abiertas para los meses siguientes, permitiendo así una mejor respuesta a los cambios que puedan surgir (Alsaqqa et al., 2020).

Las metodologías ágiles introducen nuevos enfoques de trabajo que no están abrumados por procedimientos administrativos, como la planificación, el control y la documentación, ni abogan por la total ausencia de procesos. Reconociendo que los cambios son inevitables, el objetivo es minimizar el costo de volver a hacer partes del producto debido a las modificaciones introducidas. Se busca orientar el desarrollo hacia un objetivo que puede evolucionar a medida que aumenta el conocimiento sobre el *software* que se está construyendo (Gutiérrez et al., 2020). A continuación, se enlistan las metodologías ágiles más utilizadas.

- Programación Extrema (XP).
- Scrum.
- Lean.
- Kanban.

2.1.2. Buenas prácticas en el desarrollo ágil de *software*

El desarrollo ágil se caracteriza por el uso de buenas prácticas que impactan de manera significativa tanto en el proceso como en los resultados del proyecto. Estas prácticas son esenciales para asegurar que los equipos de desarrollo se adapten rápidamente a los cambios y mantengan una alta calidad en sus entregables. Dentro de Scrum, prácticas como el uso de artefactos, roles y eventos juegan un papel fundamental en la correcta interpretación e implementación de las historias de usuario. Este enfoque iterativo permite a los equipos desarrollar soluciones alineadas con las necesidades del usuario y cumplir con los objetivos del proyecto (Sampietro, 2016; Sidky et al., 2007).

Pressman y Maxim (2020) destacan que la adopción de buenas prácticas es fundamental en cualquier enfoque de desarrollo de *software,* ya que garantiza tanto la calidad como la sostenibilidad a largo plazo del producto final. Aunque muchos de los principios que proponen tienen sus raíces en modelos más tradicionales, como el desarrollo en cascada y el modelo V, conceptos como la iteración, la mejora continua y la revisión frecuente del progreso han sido adaptados eficazmente dentro de los marcos ágiles. En estos contextos, dichos principios se transforman para maximizar la flexibilidad, fomentar la colaboración entre equipos y permitir una rápida respuesta a los cambios en los requisitos, aspectos que son pilares clave de las metodologías ágiles (Suaza et al., 2015).

La correcta adopción y aplicación de las prácticas ágiles es clave para alcanzar un resultado óptimo en los proyectos de *software*, independientemente del tipo de *software* que se desarrolle en dicho marco. La forma de trabajo ágil, definida por principios de flexibilidad, colaboración y entrega incremental, sólo puede maximizar su efecto mediante una adecuada implementación de estas prácticas. Tal como señalan Wang et al., (2012), integrar estas prácticas en la cultura organizacional requiere un esfuerzo consciente y planificado que, al llevarse a cabo de manera adecuada, incrementa notablemente su efecto positivo. De esta forma, una correcta implementación de las prácticas ágiles optimiza los beneficios que estas ofrecen, sin importar el tipo de *software* que se esté desarrollando.

Además, una buena comunicación dentro de los equipos es determinante para el éxito de los proyectos ágiles. Las prácticas ágiles, como las reuniones diarias y las retrospectivas, están diseñadas específicamente para mejorar tanto la comunicación interna del equipo como la colaboración con los

stakeholders externos. Esto influye directamente en la comprensión y correcta implementación de las historias de usuario. Como mencionan Pikkarainen et al., (2008), la capacidad de los equipos ágiles para interactuar de manera efectiva es uno de los factores más importantes que contribuyen al éxito de la metodología ágil.

Por otro lado, la adopción de buenas prácticas ágiles facilita la claridad y precisión en la definición de las historias de usuario, promoviendo una mejor comunicación y colaboración dentro del equipo. Esto resulta en un desarrollo de *software* más alineado con las expectativas de los *stakeholders* (Sampietro, 2016). Así, estas prácticas no solo facilitan el desarrollo técnico, sino que también fortalecen la capacidad de los equipos para responder a los cambios y cumplir con las demandas de un entorno de trabajo dinámico y en constante evolución (Diebold y Dahlem, 2014).

En un entorno global, la integración de prácticas ágiles presenta desafíos adicionales, que deben gestionarse para asegurar la eficacia del proceso de desarrollo. Jalali y Wohlin (2012) identifican que, a medida que los equipos de desarrollo se dispersan geográficamente, la implementación de prácticas ágiles como reuniones diarias, revisiones retrospectivas y entregas incrementales se vuelve más compleja. Sin embargo, estos desafíos no son insuperables y, cuando se gestionan adecuadamente, las metodologías ágiles pueden ser altamente eficaces en contextos globales (Vallon et al., 2018).

Finalmente, las historias de usuario en la Ingeniería de Requisitos Ágil, son una herramienta de comunicación que combina las ventajas de los medios escritos y verbales. Permiten registrar información de manera clara y estructurada, como en la comunicación escrita, pero también fomentan la interacción dinámica y la retroalimentación inmediata, características de la comunicación verbal (Menzinsky et al., 2018). De esta manera, combinan lo mejor de ambos enfoques para mejorar la comprensión y manejo de los requisitos. La adopción adecuada de estas prácticas ágiles es esencial para asegurar que los equipos comprendan y definan correctamente las historias de usuario, lo que mejora la comunicación y la colaboración dentro del equipo. Diebold y Dahlem (2014) señalan que la adopción adecuada de estas prácticas no solo fortalece la capacidad de los equipos para adaptarse al cambio, sino que también fomenta una mejora continua dentro del proceso de desarrollo.

2.1.3. Metodología Scrum

Scrum, creado por el ingeniero Ken Schwaber y el médico Jeff Sutherland en 1995, es un enfoque ágil para el desarrollo de *software* (véase Figura 2.1). De acuerdo con sus creadores, el ciclo de vida definido por Scrum es altamente adaptable, marcado por incrementos y repeticiones. Abrahamsson et al., (2002) considera que Scrum es un marco de trabajo que se apoya en principios ágiles, con el fin de mantener un control constante sobre el estado actual del *software*. En este enfoque, el cliente establece las prioridades, mientras que el equipo de Scrum se autoorganiza para determinar la manera más efectiva de entregar los resultados.

Para Alberto et al., (2020), Scrum introduce un enfoque que promueve la adopción de una metodología de colaboración entre los equipos implicados en la creación de nuevos productos. Su objetivo es fomentar una comunicación más efectiva, una integración más estrecha y un mayor conocimiento entre todos los roles involucrados, con el fin de agilizar el trabajo, desarrollar productos eficientes y garantizar la satisfacción del cliente. Scrum se concentra en comprender a fondo al equipo de trabajo, identificando tanto sus puntos fuertes como sus áreas de mejora. La esencia de Scrum radica en la comunicación, la colaboración y el continuo proceso de mejora, adaptación y construcción de mejores desarrollos de *software*. Su fundamento principal es el trabajo en equipo, donde cada miembro mantiene en mente los aspectos clave de las tareas a realizar en todo momento.

Scrum es un proceso empírico fundamentado en la observación, la experiencia y la experimentación, respaldado por tres pilares: transparencia, inspección y adaptación. Esto subraya la

idea de trabajar de forma iterativa (Srivastava et al., 2017). Los principios de Scrum se alinean con el Manifiesto Ágil y se emplean para orientar las actividades de desarrollo, que abarcan requisitos, análisis, diseño, implementación y entrega. En cada una de estas etapas, las tareas se realizan en períodos de tiempo definidos, conocidos como *Sprints*. El trabajo realizado en cada *Sprint* se adapta al problema específico en cuestión, y el equipo Scrum lo define y ajusta según sea necesario en tiempo real. La cantidad de *Sprints* requeridos puede variar según el tamaño y la complejidad del producto (Pressman y Maxim, 2020).

La investigación de Srivastava et al. (2017) argumentó que una ventaja de Scrum es su enfoque en incrementar la productividad mediante una comunicación efectiva y una planificación que otorgan autonomía a los equipos para explorar nuevas vías en el diseño de soluciones. Además, ofrece un proceso más ágil para adaptarse eficazmente ante cambios fundamentales. Por su parte, Estrada-Velasco et al. (2021) señalan que al implementar la metodología Scrum en proyectos de desarrollo, se favorece la adopción de un conjunto de prácticas y métodos que inciden de manera significativa en el rendimiento del equipo, lo cual se refleja notoriamente en el avance y la calidad del producto de *software*.

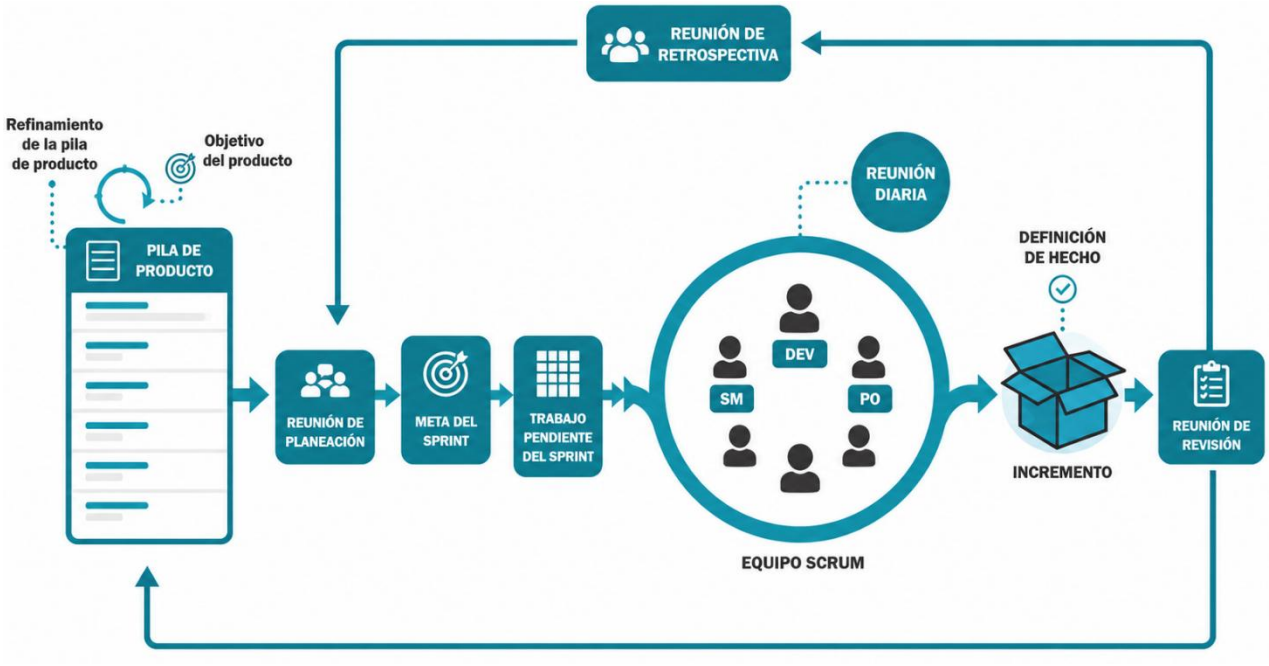


Figura 3 Ciclo de vida de Scrum[5]

De acuerdo con la literatura existente, los principales componentes de la metodología Scrum son los siguientes:

a. Roles Scrum

De acuerdo con Hema et al. (2020), los roles principales en Scrum son los siguientes:

- **Dueño del producto:** Responsable de maximizar el valor del producto y gestionar la pila del producto.

[5] Para más información sobre el ciclo de vida de Scrum, por favor referirse a: https://www.scrum.org/learning-series/what-is-scrum/

- ***Scrum Master*:** Facilitador del equipo Scrum, responsable de garantizar que el equipo comprenda y siga los principios y prácticas de Scrum.
- **Equipo de desarrollo:** Equipo encargado de llevar a cabo el trabajo necesario para entregar un incremento de producto al final de cada *Sprint*. Este equipo es autoorganizado y multidisciplinario.

b. Eventos Scrum

Por otro lado, Sommerville (2020) afirma que los eventos principales en Scrum son:

- ***Sprint*:** Un período de tiempo fijo, generalmente de una a cuatro semanas, durante el cual se desarrolla un incremento potencialmente entregable del producto.
- **Reunión de planeación:** Una reunión al inicio de cada *Sprint* en la que el equipo colabora para seleccionar y planificar el trabajo que se realizará durante el *Sprint*. Durante esta reunión, se establece el objetivo del *Sprint* y se define el trabajo pendiente del *Sprint*, esta reunión suele durar entre dos y cuatro horas para un *Sprint* de un mes.
- **Reunión diaria:** Una reunión diaria de quince minutos en la que el equipo sincroniza su trabajo, identifica cualquier impedimento y planifica el trabajo para las próximas 24 horas.
- **Reunión de revisión:** Una reunión al final de cada *Sprint* en la que el equipo presenta el incremento completado al Dueño del Producto y a los interesados. Durante esta reunión, se revisa el trabajo realizado y se recibe retroalimentación para ajustar el producto o la pila del producto, con una duración de hasta cuatro horas para un *Sprint* de cuatro semanas.
- **Reunión de retrospectiva:** Una reunión al final de cada *Sprint* en la que el equipo reflexiona sobre su desempeño durante el *Sprint*. Se identifican los puntos fuertes, las oportunidades de mejora y se establecen acciones para mejorar el proceso en el próximo *Sprint*, con una duración de hasta tres horas para un *Sprint* de cuatro semanas.

c. Artefactos Scrum

India et al., 2020) argumentan que Scrum cuenta con tres artefactos:

- **Pila del producto:** Una lista prioritaria y dinámica de todas las funcionalidades, requisitos, mejoras y correcciones pendientes de realizar en el producto. Es responsabilidad del Dueño del Producto gestionarlo y priorizarlo.
- **Trabajo pendiente del *Sprint*:** Una lista de elementos seleccionados de la pila del producto que el equipo se compromete a completar durante un *Sprint*. Es responsabilidad del equipo de desarrollo gestionarlo y actualizarlo durante el *Sprint*.
- **Incremento:** Es una unidad de funcionalidad de un elemento de trabajo que está lista para ser publicada en el entorno de producción.

d. Pilares de Scrum

El estudio de Alberto et al., 2020) establece que Scrum tiene como base tres pilares fundamentales:

- **Transparencia:** Todos los elementos del proceso están claramente expuestos para lograr el objetivo deseado.
- **Inspección:** Es fundamental examinar regularmente cada etapa del seguimiento para una planificación adecuada.
- **Revisión:** Se debe garantizar el cumplimiento de todos los criterios de aceptación.

2.1.4. Buenas prácticas en Scrum

Scrum es uno de los marcos ágiles más utilizados para la gestión de proyectos de desarrollo de *software* (Gutiérrez et al., 2020). Su estructura basada en roles, artefactos y eventos proporciona una base sólida para la implementación de buenas prácticas que optimizan el trabajo en equipo y la entrega de valor de manera continua e incremental. Las buenas prácticas en Scrum son esenciales para asegurar que los equipos funcionen de manera eficiente y que se logren los objetivos del proyecto (Ghimire y Charters, 2022).

- **Roles claramente definidos:** Como señalan Hema et al. (2020), uno de los elementos clave en Scrum es la definición clara de los roles: Dueño del Producto, el *Scrum Master* y el equipo de desarrollo. Cada rol tiene responsabilidades bien delimitadas que aseguran una adecuada distribución de las tareas y una comunicación fluida. El Dueño del Producto se encarga de maximizar el valor del producto mediante la gestión de la pila del producto, el scrum master facilita el proceso y remueve impedimentos, mientras que el equipo de desarrollo colabora de manera autoorganizada para cumplir con los objetivos del *Sprint*. Esta separación de roles fomenta una especialización funcional que mejora la eficiencia del equipo (Cruz et al., 2023).
- **Gestión eficiente del *Product Backlog*:** El *Product Backlog* es el artefacto central en Scrum y su gestión es fundamental para el éxito del proyecto. Una buena práctica es mantener el *Product Backlog* refinado y priorizado, de manera que el equipo de desarrollo siempre tenga una visión clara de las tareas más importantes. Este refinamiento continuo permite al equipo adaptarse rápidamente a los cambios en los requisitos y a las prioridades del cliente, optimizando el uso del tiempo y los recursos del equipo (Pressman y Maxim, 2020). Además, la colaboración constante entre el Dueño del Producto y el equipo de desarrollo asegura que las historias de usuario estén bien definidas y comprendidas antes de ser implementadas (Cohn, 2004; Sassa et al., 2023).
- **Establecimiento de objetivos claros en la Planeación del *Sprint*:** Durante la Planeación del *Sprint*, el equipo Scrum establece metas claras y alcanzables que guían su trabajo. El equipo se enfoca en un objetivo bien definido y alineado con las prioridades del cliente, lo que le permite ser más eficaz en la entrega de valor. Esta práctica asegura que todos los miembros del Equipo Scrum comprendan el propósito del *Sprint* y trabajen de manera colaborativa para alcanzar los objetivos establecidos. La definición clara de los objetivos mantiene la alineación entre las expectativas del cliente y los entregables del equipo, optimizando el enfoque y el uso de los recursos (Estrada-Velasco et al., 2021; Sassa et al., 2023; Singh, 2023).
- **Revisiones y retrospectivas continuas:** Scrum enfatiza la importancia de la mejora continua a través de las revisiones y retrospectivas. Durante la revisión del *Sprint*, el equipo presenta el incremento de producto completado y recibe retroalimentación del Dueño del Producto y otros *stakeholders*. Esta práctica asegura que el producto en desarrollo cumple con las expectativas y permite realizar ajustes en el siguiente *Sprint*. La retrospectiva del *Sprint*, por otro lado, se enfoca en la mejora de los procesos del Equipo Scrum. Durante esta reunión, se analizan tanto los logros como los aspectos a mejorar del *Sprint* recién finalizado. Esta reflexión permite al equipo ajustar su forma de trabajo y ser más eficiente en futuros *Sprints*. La implementación continua de estos ciclos de retroalimentación y mejora asegura que el equipo esté constantemente optimizando su rendimiento y la calidad de su trabajo (Diebold y Dahlem, 2014; Estrada-Velasco et al., 2021; Sassa et al., 2023; Singh, 2023).
- **Transparencia y comunicación continua:** Scrum fomenta la transparencia a través de reuniones diarias, conocidas como *Daily* Scrum, en las que los miembros del Equipo Scrum sincronizan su trabajo y comparten avances, desafíos y próximos pasos. Esta práctica permite

la detección temprana de obstáculos y promueve una colaboración efectiva para resolver problemas de manera ágil. La comunicación constante es fundamental para mantener la alineación entre el Equipo de Desarrollo, el Dueño del Producto y los *stakeholders*, asegurando que todos trabajen hacia las mismas prioridades (Sandsto y Reme-Ness, 2021; Sassa et al., 2023).

- **Entregas incrementales y adaptación continua:** Una de las prácticas más destacadas en Scrum es la entrega incremental de valor. En lugar de desarrollar todo el producto de una sola vez, el equipo se enfoca en proporcionar versiones funcionales al final de cada *Sprint*. Este enfoque no solo permite la entrega continua de valor al cliente, sino que también facilita la obtención de retroalimentación regular, lo que permite ajustar el producto según las necesidades y expectativas cambiantes. La adaptabilidad es un pilar fundamental de Scrum, permitiendo a los equipos responder de manera ágil a un entorno dinámico. A través de la entrega incremental, el Equipo Scrum puede aprender y realizar mejoras en cada *Sprint*, ajustando su estrategia para maximizar la efectividad del desarrollo. Este ciclo de entrega y retroalimentación mejora la calidad del producto y asegura que se alinee con las necesidades del mercado y de los usuarios finales (Diebold y Dahlem, 2014; Estrada-Velasco et al., 2021; Sandsto y Reme-Ness, 2021; Sassa et al., 2023; Singh, 2023).

2.2. Historias de usuario en Scrum

Para Sommerville (2020), las historias de usuario son descripciones más elaboradas que proporcionan una visión más completa y detallada de lo que un usuario requiere de un sistema de *software*. Estas narrativas están diseñadas para capturar de manera estructurada las necesidades y expectativas de los usuarios, describiendo con precisión un único elemento que se espera del sistema. Además de exponer las funcionalidades deseadas, las historias de usuario también pueden incluir detalles sobre el contexto de uso, los criterios de aceptación y cualquier otro aspecto relevante para comprender plenamente la solicitud del usuario.

Como mencionan Pokharel y Vaidya (2020), la esencia de la metodología ágil radica en priorizar la satisfacción del cliente mediante la entrega constante y oportuna de *software* que aporte valor. En este enfoque, los requisitos se describen como historias de usuario, ideadas desde la perspectiva del usuario final. La efectividad de las historias de usuario radica en la capacidad de los profesionales del *software* para comprender su calidad y su potencial para generar valor tangible para el cliente. Una historia de usuario se define como la descripción de una funcionalidad que será beneficiosa tanto para el usuario como para el comprador de la aplicación. Dado que la metodología ágil se enfoca en proporcionar valor a través de la satisfacción del cliente, redactar historias de usuario de manera adecuada adquiere una importancia relevante en el desarrollo de *software* ágil.

En el desarrollo de *software* ágil, las historias de usuario reemplazan al tradicional documento de especificaciones utilizado en los enfoques tradicionales de desarrollo. En las metodologías ágiles, las historias de usuario se convierten en el principal medio para capturar los requisitos, proporcionando una descripción breve de una funcionalidad específica desde la perspectiva del usuario. Por lo general, se utiliza una plantilla estándar para su redacción, centrándose principalmente en los requisitos funcionales del sistema. Aunque las historias de usuario son ampliamente aceptadas como una forma accesible de documentar y comunicar las características del *software*, su implementación práctica puede llevar a la ambigüedad, lo que conlleva riesgos como la imprecisión, inconsistencia y redundancia en los requisitos (Gupta et al., 2022).

Dentro de la metodología Scrum, los equipos emplean diversas técnicas para gestionar las actividades relacionadas con la Ingeniería de Requisitos en un contexto ágil. Entre estas técnicas se encuentra el Desarrollo Impulsado por el Comportamiento (BDD, por sus siglas en inglés) y el Desarrollo Impulsado por Pruebas (TDD, por sus siglas en inglés). El enfoque de BDD se basa en el uso de historias de usuario y criterios de aceptación para definir los requisitos del sistema de manera detallada y comprensible (Smart y Molak, 2023). Una historia de usuario correctamente escrita sigue el siguiente formato (Amna y Poels, 2022, Sommerville, 2020; Thamrongchote y Vatanawood, 2016):

Como <tipo de usuario>, **quiero** <objetivo deseado> **para que** <valor alcanzado>

1. **Tipo de usuario:** Indica el tipo o rol del usuario.
2. **Objetivo deseado:** Describe el resultado específico que el usuario espera lograr al utilizar la funcionalidad que se está solicitando.
3. **Valor alcanzado:** Es el beneficio o la razón por la que el usuario desea alcanzar el objetivo deseado.

2.3. UML

UML presentado por el Grupo de Gestión de Objetos (OMG, por sus siglas en inglés) a finales de la década de 1990, es reconocido como un lenguaje versátil para modelar *software*. Ofrece una amplia variedad de diagramas visuales que permiten representar los requisitos, la arquitectura y el diseño del sistema de *software* en diferentes niveles de abstracción (Ozkaya, 2019).

UML es un lenguaje utilizado para documentar y visualizar los requisitos del *software* (Mornie et al., 2023). Su principal función es permitir a profesionales, como desarrolladores, comprender el funcionamiento del *software* y organizar las necesidades de los usuarios de manera efectiva. Por otro lado, la metodología ágil hace un amplio uso de diferentes tipos de modelos UML.

Este estándar de la industria del *software* se ha consolidado como una herramienta integral para la documentación, construcción, visualización y especificación de componentes de *software*. El cual surgió con el objetivo de unificar diversos lenguajes de modelado, aspirando a ser una herramienta universal para el modelado de *software*. El uso de UML facilita la comunicación entre desarrolladores y clientes durante el desarrollo del *software*, lo cual es crucial para el éxito del proyecto. Esta eficaz comunicación ha contribuido a mantener la relevancia de UML en la industria del desarrollo de *software* hasta la fecha (Choma Neto et al., 2021; Sarduy y Álvarez, 2021).

UML se ha establecido como el estándar predominante en la industria, en gran parte gracias al respaldo de los creadores de los tres métodos de orientación a objetos más utilizados: Grady Booch, Ivar Jacobson y Jim Rumbaugh (Rumbaugh et al., 1999). Además, en el desarrollo de UML han intervenido otras empresas prominentes en la industria, como Microsoft, Hewlett-Packard, Oracle e IBM, así como diversos grupos de analistas y desarrolladores (Grau y Segura, 2008). En esencia, UML es un lenguaje destinado a describir modelos, que son representaciones simplificadas de la realidad creadas para comprender mejor el sistema que se está desarrollando. Proporciona planos del sistema, abarcando tanto una visión general como detalles específicos de sus partes. Aunque UML no está ligado a ninguna metodología de análisis y diseño ni a ningún lenguaje de programación específico, se fundamenta en el paradigma de la orientación a objetos, por lo que resulta particularmente idóneo para el desarrollo de sistemas desde esta perspectiva. La especificación, visualización, construcción y documentación de cualquier sistema *software* requiere que se examine desde diversas perspectivas. Los usuarios finales tienen necesidades distintas de las de los analistas o programadores. UML incluye una variedad de diagramas y notaciones gráficas y textuales diseñadas

para mostrar el sistema desde diferentes ángulos, los cuales pueden emplearse en distintas etapas del ciclo de desarrollo del *software* (Koç et al., 2021; Ohst et al., 2003; Reggio et al., 2013).

2.3.1. Diagramas UML

Los diagramas UML proporcionan una representación visual que facilita la comprensión de la estructura, comportamiento e interacciones de los elementos del sistema, como clases, objetos, relaciones, actividades, estados, entre otros. En conjunto, constituyen una herramienta eficaz para el análisis, diseño y desarrollo de *software*, permitiendo a los equipos de desarrollo comunicarse de manera efectiva y capturar los requisitos y la estructura del sistema de manera clara y concisa).

Estos diagramas se dividen en dos categorías principales: estáticos y dinámicos. Los diagramas estáticos se centran en el modelado de la estructura del sistema, describiendo la disposición estática de los elementos del sistema y sus relaciones. Los tipos de diagramas estáticos empleados son (Bhatt y Nandu, 2021):

- **Diagrama de casos de uso:** Describe las funcionalidades del sistema desde el punto de vista del usuario.
- **Diagrama de clases:** Muestra las clases del sistema, sus atributos y métodos, así como las relaciones entre ellas.
- **Diagrama de objetos:** Representa instancias específicas de clases y sus relaciones en un momento determinado.
- **Diagrama de componentes:** Muestra los componentes del sistema y las dependencias entre ellos.
- **Diagrama de implementación:** Describe la arquitectura física del sistema y cómo los componentes se despliegan en el *hardware*.

Por otro lado, los diagramas dinámicos se centran en el modelado del comportamiento del sistema, mostrando cómo interactúan los objetos y cómo el sistema responde a eventos y cambios. Los tipos de diagramas dinámicos utilizados son (Bhatt y Nandu, 2021):

- **Diagrama de secuencia:** Muestra la secuencia de interacciones entre objetos en un escenario específico.
- **Diagrama de colaboración:** Similar al diagrama de secuencia, pero enfocado en las relaciones entre objetos.
- **Diagrama de actividades:** Describe el flujo de actividades dentro del sistema.
- **Diagrama de máquina de estado:** Muestra los estados posibles de un objeto y cómo cambia de un estado a otro en respuesta a eventos.

Estos diagramas proporcionan una visión completa y detallada del sistema, desde su estructura estática hasta su comportamiento dinámico, lo que ayuda a los desarrolladores a comprender, diseñar y comunicar de manera efectiva la arquitectura y funcionalidades del *software*. La calidad del diseño de los diagramas UML juega un papel crucial en su comprensión (Ozkaya, 2019).

2.3.2. Diagrama de clases

Uno de los diagramas más comunes es el diagrama de clases, que pertenece a los diagramas que representan la estructura del *software*. Estos diagramas muestran los componentes fundamentales de un sistema y las relaciones entre ellos. Los componentes se dibujan como rectángulos, y las relaciones se representan con líneas entre los rectángulos (véase Figura 2.2), a veces con flechas que indican la dirección de la relación (Badreddin et al., 2021; Bergström et al., 2022; Reggio et al., 2013, 2014).

Los diagramas de clases UML son esenciales para entender los sistemas de *software* y para facilitar la comunicación entre los equipos de desarrollo. La calidad del diseño de estos diagramas tiene un gran impacto en la comprensión de los sistemas por parte de las personas. Esta calidad puede depender tanto de la apariencia estética del diagrama como de la facilidad con la que los usuarios pueden interpretarlo. Existen dos métodos principales para crear diagramas de clases: como una guía preliminar antes de la programación o como una documentación posterior al desarrollo del sistema. Los diagramas preliminares se consideran de ingeniería avanzada y sirven como referencia durante el desarrollo. Estos diagramas son usualmente creados por personas, quienes suelen tener un sentido intuitivo del diseño. En cambio, los diagramas creados después del desarrollo se consideran de ingeniería inversa y actúan como documentación del sistema (Bergström et al., 2022; Fernández-Sáez et al., 2015; Scanniello et al., 2018).

En el contexto de un proyecto de desarrollo de *software* ágil, los diagramas UML pueden respaldar el proceso de desarrollo de varias maneras. Gallud y Fardoun (2019) sostienen que los modelos UML, como el diagrama de clases, pueden ser muy útiles para el equipo ágil. Por ejemplo, el diagrama de clases describe los objetos y sus relaciones dentro del sistema, lo que puede ayudar al equipo ágil a decidir cuál es el mejor diseño para implementar las funciones requeridas por las partes interesadas.

Los diagramas de clases son cruciales para ayudar a los desarrolladores a construir el código ejecutable del *software*. Su objetivo principal es describir las responsabilidades del sistema, analizar y diseñar la estructura de la aplicación, y proporcionar una base para los diagramas de componentes y de implementación. Además, desempeñan un papel crucial en la ingeniería directa e inversa. Estos diagramas se utilizan principalmente para mostrar cómo se modelan diferentes tipos de elementos dentro del sistema, como clases, interfaces, tipos de datos y componentes. Son útiles tanto para los desarrolladores, quienes pueden entender fácilmente la estructura del sistema, como para otros miembros del equipo, como los analistas de negocios. Los diagramas de clases están estrechamente vinculados con la codificación estructural, por lo que es esencial identificar con precisión cada elemento y sus relaciones de antemano.

En un diagrama de clases, los elementos esenciales incluyen el nombre de la clase, los atributos y las operaciones. Las clases se representan mediante rectángulos, con el nombre de la clase en la parte superior del rectángulo, y divididos en dos secciones: la mitad superior contiene los atributos y la inferior, las operaciones. Estos diagramas ofrecen diferentes tipos de relaciones o asociaciones entre las clases, como visibilidad, uso, agregación, dependencia y generalización, que son fundamentales para la elaboración de varias declaraciones en la filosofía orientada a objetos. Representan los elementos del sistema y proporcionan una visión de los componentes principales y cómo interactúan para alcanzar un objetivo común (Bhatt y Nandu, 2021; F. Chen et al., 2022; Nikiforova et al., 2011).

Figura 4 Estructura de un diagrama de clases (Bhatt y Nandu, 2021).

2.3.3. Diagrama de actividades

Los diagramas de actividades son una herramienta visual poderosa en el desarrollo de *software*, ampliamente utilizada para modelar el flujo de trabajo de un sistema. Estos diagramas representan el comportamiento del sistema a través de actividades y acciones, mostrando cómo se llevan a cabo las tareas y procesos. En estos diagramas, las actividades se representan mediante nodos, y las transiciones entre ellas se muestran con flechas que indican la secuencia de ejecución (véase Figura 2.3). Estos diagramas son particularmente útiles para modelar procesos empresariales, algoritmos, flujos de trabajo y cualquier secuencia de pasos que describa cómo se realiza una tarea específica. Facilitan una comprensión clara y concisa de la lógica detrás de un proceso, lo cual es fundamental para la colaboración efectiva entre equipos y *stakeholders*. Una de las ventajas clave de los diagramas de actividades es su capacidad para identificar posibles mejoras o problemas en el flujo de trabajo existente. Esto se debe a que proporcionan una representación visual detallada de los procesos, lo que permite analizar y optimizar el rendimiento del sistema. Además, pueden integrarse eficazmente con otros diagramas UML, como los de casos de uso y de clases, para ofrecer una visión más completa del sistema y su estructura. (Abbas et al., 2021; Kulkarni y Srinivasa, 2021; Lima et al., 2019; Raffel et al., 2020).

De acuerdo con Afiansyah et al. (2023), Elmansouri et al. (2021), Lima et al. (2019) y Reggio et al. (2013), el diagrama de actividades es uno de los tipos de diagramas UML diseñados para capturar el comportamiento dinámico de un sistema. Se utiliza para modelar el flujo de actividades de un sistema, teniendo en cuenta la secuencia y las condiciones del flujo. Este tipo de diagrama describe un proceso de negocio como una serie de acciones que pueden ser realizadas por personas, componentes de *software* o computadoras. La finalización de una acción, la disponibilidad de datos y la ocurrencia de eventos externos pueden iniciar otras acciones. El diagrama de actividad UML representa diferentes tipos de flujos, como flujos paralelos, ramificados, concurrentes y simples.

2.4. UML en Scrum

El uso del modelado UML como técnica de modelos conceptuales y su integración en metodologías ágiles no es algo novedoso. Recker y Green (2019) señalan que, a pesar del crecimiento de la popularidad de las metodologías ágiles como Scrum, los modelos conceptuales aún tienen un lugar importante en el desarrollo de *software*. Algunos investigadores incluso sugieren la inclusión de más documentación, incluyendo modelos visuales, en los procesos ágiles. Vithana, (2015) argumentó que la falta de documentación formal puede resultar en una verificación deficiente de los requisitos. Para abordar los problemas de comunicación en proyectos ágiles, Sundararajan et al., (2014) sugieren la creación de documentos de diseño para la arquitectura general de los sistemas, las bases de datos y las interfaces.

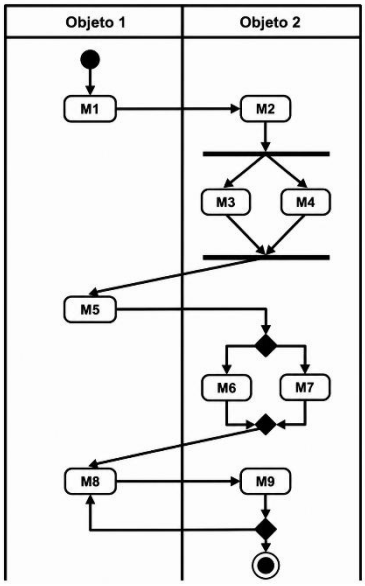


Figura 5 Ejemplo de un diagrama de actividades (Kulkarni y Srinivasa, 2021).

En relación con las historias de usuario, Kannan et al. (2019) proponen la utilización de diagramas de casos de uso que se derivan de las historias de usuario. Estos diagramas proporcionan una representación visual de cómo los usuarios interactúan con el sistema y son útiles para capturar y comunicar los requisitos de manera efectiva.

Por otro lado, Schön et al. (2017) señalaron que algunas organizaciones ágiles emplean diagramas de casos de uso, tarjetas de historias y mapas mentales para fomentar una comprensión compartida y la participación de los usuarios potenciales. Los diagramas de casos de uso ayudan a establecer un lenguaje común entre los equipos técnicos y no técnicos, mientras que las tarjetas de historias y los mapas mentales facilitan la colaboración al permitir una visualización clara y concisa de los requisitos y procesos del proyecto. Esta combinación de herramientas y técnicas no solo promueve una mejor comprensión de los requisitos por parte de los equipos de desarrollo, sino que también facilita la participación activa de los usuarios en la definición y validación de las funcionalidades del sistema.

Helmy et al. (2012) sugirieron que la creación de modelos de dominio de alto nivel puede ofrecer una orientación valiosa tanto para el diseño del modelo de datos físicos como para el diseño de clases en proyectos de desarrollo de *software*. Además, Trkman et al., (2016) argumentaron que la incorporación de modelos de procesos de negocio mejora significativamente la comprensión de las interdependencias entre las historias de usuario en el contexto de metodologías ágiles como Scrum. En una revisión exhaustiva de la literatura, Amna y Poels (2022a) observan que muchos estudios proponen el uso de modelos como una estrategia para abordar la ambigüedad inherente a las historias de usuario en el desarrollo de *software* ágil. Sin embargo, señalan que la validación de estas propuestas aún está en proceso, con solo algunas soluciones confirmadas como efectivas hasta el momento. Esta situación plantea un desafío continuo en la práctica de cómo desarrollar y aplicar eficazmente modelos en proyectos ágiles.

El uso de diagramas UML dentro del marco de trabajo ágil Scrum ha sido objeto de estudio en diversas investigaciones, las cuales han explorado tanto la viabilidad como los beneficios de esta integración. Estas investigaciones han revelado cómo los diagramas UML, con su capacidad intuitiva para visualizar y modelar sistemas de *software*, pueden complementar eficazmente las prácticas ágiles de Scrum. Además, se ha analizado cómo la inclusión de diagramas UML puede facilitar la

comunicación y colaboración entre los miembros del equipo, al proporcionar un lenguaje común y una representación gráfica clara de los requisitos y diseños del sistema. Asimismo, se ha investigado cómo la utilización de diagramas UML puede contribuir a una mejor comprensión y gestión de la complejidad en el desarrollo de *software* ágil, al permitir una visualización estructurada y detallada de los diferentes aspectos del sistema en desarrollo. Estos estudios han demostrado que la combinación de UML con Scrum puede resultar en un enfoque poderoso y efectivo para el desarrollo de *software*, que aprovecha lo mejor de ambos enfoques para alcanzar resultados óptimos. Sin embargo, también se ha señalado la necesidad de adaptar y personalizar esta integración según las necesidades y características específicas de cada proyecto y equipo de desarrollo (Chen et al., 2019; Gallud y Fardoun, 2018; Kussunga et al., 2019).

La investigación realizada por Gallud y Fardoun (2018) propuso una metodología para definir la arquitectura de *software* en entornos ágiles denominada Xcrum. Xcrum es una metodología ágil que comparte similitudes con Scrum en términos de roles, iteraciones, artefactos y reuniones, pero también presenta diferencias significativas. En Xcrum, los roles de negocio y desarrollo tienen responsabilidades específicas, comparables al Dueño del Producto y al equipo de desarrollo en Scrum, respectivamente. Las iteraciones en Xcrum, se asemejan a los *Sprints* en Scrum y tienen una duración de una a cuatro semanas. Los artefactos en Xcrum incluyen historias de usuario, listas de historias del sistema e iteración, y un gráfico de progreso, todos similares a los artefactos en Scrum. Las reuniones de Xcrum siguen la estructura de Scrum, pero incorporan actividades de XP para mejorar la productividad del equipo. Además, Xcrum presta especial atención a las pruebas, esperando que el equipo escriba casos de prueba para garantizar un producto de alta calidad. En resumen, Xcrum ofrece una alternativa a Scrum con una estructura familiar, pero con enfoques distintivos que pueden adaptarse a las necesidades específicas del equipo y del proyecto.

El enfoque propuesto por Kussunga et al., (2019) para apoyar las ceremonias de Scrum se centró en dos bloques innovadores: la transformación de historias de usuario a casos de uso y la posterior transformación de casos de uso en el modelo arquitectónico, utilizando técnicas de modelado UML. El ciclo de vida del entorno visual se divide en dos fases principales: Planeación del *Sprint* y Ejecución del *Sprint*. Durante la Planeación del *Sprint*, se emplean artefactos como la pila del producto y la arquitectura global, junto con los modelos de casos de uso resultantes del proceso de transformación de historias de usuario. En la Ejecución del *Sprint*, se llevan a cabo las actividades planificadas del *Sprint*, así como la revisión y retrospectiva del mismo, con el objetivo de entregar un incremento de producto utilizable en cada iteración. Este enfoque, influenciado por Scrum, UML y las prácticas de desarrollo basadas en casos de uso, se enfoca en la visualización y análisis eficiente de los requisitos. Destaca la importancia de recopilar y analizar efectivamente los requisitos durante todo el proceso de desarrollo.

La investigación de Chen et al. (2019) se centró en abordar una metodología para analizar proyectos de *software* a gran escala, que enfrentan requisitos poco definidos por parte de los clientes. Se opta por implementar Scrum como enfoque ágil de desarrollo de *software*, utilizando diagramas UML para mejorar la documentación del diseño y agilizar el proceso de desarrollo. El estudio examina cómo una empresa específica desarrolla un sistema automático de programación de cursos para escuelas primarias y secundarias utilizando la metodología Scrum. A través de entrevistas, la empresa logra capturar los requisitos de los clientes utilizando los diagramas UML correspondientes, facilitando la documentación del proceso de desarrollo y la conceptualización de funciones para satisfacer las necesidades del cliente. Posteriormente, se implementa el método propuesto en el proyecto de la empresa, evaluando su viabilidad. Durante el proceso de desarrollo, surgen desafíos que se discuten junto con posibles soluciones. Estos resultados pueden servir como referencia para

empresas de *software* que enfrenten proyectos de gran envergadura con requisitos poco claros por parte de los clientes.

Falah et al. (2020) propusieron una metodología de modelado independiente que integra el uso de diagramas UML con técnicas de generación de casos de prueba basadas en modelos, las cuales fueron aplicadas en el marco de Scrum. La investigación presenta un enfoque innovador para abordar los desafíos comunes asociados con el uso de UML en el desarrollo de *software*. En primer lugar, propone la selección de un conjunto reducido de diagramas UML, denominado UML esencial, basado en una investigación exhaustiva que incluye la revisión de libros, cursos universitarios, encuestas de opinión y consultas a profesionales de las Tecnologías de la Información (TI). Este conjunto comprende los diagramas más utilizados y relevantes en la práctica de desarrollo de *software*. Además, introduce una técnica novedosa para la generación de casos de prueba a partir de los diagramas UML seleccionados. Esta técnica utiliza los diagramas UML como base para generar conjuntos de pruebas, que pueden utilizarse para verificar la conformidad del *software* con los requisitos establecidos. Esta aproximación integra el modelado UML en la fase de prueba del ciclo de vida del desarrollo de *software*, mejorando la coherencia y eficiencia del proceso. Por último, propone la integración de este enfoque con Scrum, un marco de trabajo ágil ampliamente utilizado en la industria del *software*. Esta combinación, denominada Scrum-UML, aprovecha los principios ágiles de Scrum, como la entrega iterativa y la adaptación continua, para mejorar el proceso de desarrollo de *software* basado en UML. Al centrarse en entregables rápidos y frecuentes, este enfoque promueve una colaboración efectiva entre los miembros del equipo y una mayor capacidad de respuesta a los cambios en los requisitos del cliente.

El estudio realizado por Páez et al., (2021) tuvo como objetivo desarrollar una aplicación móvil para gestionar la entrada y salida de bicicletas en la sede Bogotá de la Universidad Cooperativa de Colombia, dirigida a estudiantes, docentes y personal administrativo. En este estudio, se utilizaron los diagramas UML, especialmente los casos de uso, como herramienta para abstraer y comprender el contexto real del proyecto. Estos diagramas posibilitaron la representación visual de los requisitos y funcionalidades del sistema, facilitando así la comunicación entre los miembros del equipo y los *stakeholders*. Además, se adoptó la metodología ágil Scrum como marco de trabajo para la gestión del proyecto, ofreciendo un enfoque flexible y adaptable que permitió al equipo realizar entregas iterativas y responder de manera ágil a los cambios y necesidades del cliente. Dentro de Scrum, se definieron tres roles conocidos, asegurando una distribución eficiente de responsabilidades y un flujo de trabajo organizado. La integración del modelado UML y la metodología Scrum permitió un enfoque integral y colaborativo en el desarrollo del proyecto, garantizando una comprensión clara de los requisitos y una gestión efectiva de las actividades de desarrollo.

2.5. Automatización de UML a partir de historias de usuario

En la actualidad, el desarrollo ágil de *software* está experimentando un aumento significativo en su adopción. Sin embargo, a diferencia del extenso estudio de la automatización de la Ingeniería de Requisitos en el desarrollo de *software* convencional, la automatización en el ámbito ágil aún no ha recibido la atención necesaria, especialmente en lo que respecta al modelado de requisitos (Raharjana et al., 2021). Este modelado es esencial en el ciclo de vida de la Ingeniería de *Software*, ya que proporciona una base sólida para el producto final y contribuye en gran medida al éxito de los proyectos (Nasiri et al., 2021).

La elaboración de diagramas UML para proyectos de desarrollo de *software* presenta desafíos considerables (Mornie et al., 2023). La transformación de historias de usuario en estos diagramas es especialmente compleja debido a la naturaleza a menudo inconsistente, incompleta y errónea de las

historias de usuario (Mornie et al., 2023). Por lo tanto, se vuelve crucial incorporar herramientas o técnicas que faciliten la generación automatizada o semiautomatizada de diagramas UML a partir de historias de usuario, lo cual puede lograrse aprovechando el Procesamiento del Lenguaje Natural (NLP, por sus siglas en inglés) (Kochbati et al., 2021).

El análisis de requisitos es fundamental para el éxito del desarrollo de *software*, ya que los desarrolladores deben comprender completamente los requisitos antes de iniciar la implementación. Las historias de usuario, cada vez más utilizadas en Scrum, son una forma semiestructurada de expresar estos requisitos (Kassab, 2015; Lucassen et al., 2016). A medida que los sistemas crecen en tamaño debido al aumento de socios en el proceso, el número de historias de usuario también aumenta. Para abordar este crecimiento, se sugiere dividir el sistema en subsistemas que cubren conjuntos de historias de usuario similares, lo que facilita la comprensión del *software* y la construcción de una cartera de productos. A pesar de la eficacia de los modelos UML en la especificación de características y la reducción de ambigüedades, la creación manual de estos modelos sigue siendo una tarea laboriosa, especialmente para sistemas grandes (Kochbati et al., 2021). La integración de enfoques basados en modelos en Scrum también ha enfrentado desafíos debido a la falta de herramientas de automatización y al enfoque centrado en la implementación en lugar del análisis o la documentación.

2.6. Inteligencia Artificial

La Inteligencia Artificial (IA, por siglas en inglés) ha emergido como una disciplina multidisciplinaria que busca desarrollar sistemas capaces de realizar tareas que requieran inteligencia humana. Desde su concepción, la IA ha evolucionado de manera significativa, abarcando áreas como el aprendizaje automático, la visión por computadora, la robótica y el NLP. Dentro de este amplio espectro, el NLP se destaca como un campo crucial que se enfoca en la interacción entre las computadoras y el lenguaje humano (Chowdhary, 2020; Nemeskey, 2021).

En la última década, la IA ha emergido como el campo líder en tareas de procesamiento y generación de información, gracias al desarrollo del aprendizaje automático basado en redes neuronales. Este avance ha ampliado su aplicabilidad a diversos campos, incluyendo la robótica, el procesamiento de voz, la visión artificial y el NLP, con el objetivo de dotar a los sistemas computacionales de la capacidad de aprender (Beltrán y Mojica, 2021).

El Aprendizaje Automático (ML, por sus siglas en inglés), como parte de la IA, se ha convertido rápidamente en una herramienta poderosa en muchos campos, incluido el reconocimiento de imágenes y voz, el NLP e incluso la medicina. El ML implica la capacidad de aprender automáticamente relaciones y patrones significativos a partir de ejemplos y observaciones. Esta tecnología ha dado lugar a sistemas inteligentes cuya capacidad cognitiva se asemeja a la humana y se ha extendido tanto en el ámbito empresarial como en el personal. El campo del ML ha experimentado notables avances en algoritmos sofisticados y técnicas de preprocesamiento eficientes, con el objetivo principal de automatizar la creación de modelos analíticos para realizar tareas cognitivas, como la detección de objetos o la traducción de lenguaje natural. Estos algoritmos aprenden de manera iterativa a partir de datos de entrenamiento específicos, permitiendo a las computadoras descubrir conocimientos ocultos y patrones complejos sin necesidad de ser programadas explícitamente. Han sido aplicados con éxito en una amplia gama de áreas, como la detección de fraudes, la evaluación crediticia, el análisis para ofrecer la mejor oferta, el reconocimiento de voz e imágenes, y el NLP (Janiesch et al., 2021; Sharifani y Amini, 2023).

El ML, como parte de la IA, mejora la capacidad de las máquinas para aprender a través de algoritmos y el análisis de diversos tipos de datos. Dentro de este campo, el aprendizaje profundo y las redes neuronales tienen un papel fundamental. Los algoritmos de aprendizaje profundo analizan

repetidamente conjuntos de datos, lo que permite a las máquinas mejorar su conocimiento en función de los resultados obtenidos. Esta mejora en la inteligencia de las máquinas es esencial para el NLP, un campo integral de la informática que utiliza tanto el aprendizaje automático como la lingüística computacional. El NLP busca facilitar una interacción eficiente entre humanos y computadoras, permitiendo que las máquinas comprendan tanto la sintaxis como el significado del lenguaje humano (Badillo et al., 2020; Chowdhury, 2020). El aprendizaje automático se divide en tres enfoques principales:

- **Aprendizaje supervisado:** El modelo se entrena utilizando un conjunto de datos que incluye tanto las entradas como las salidas deseadas, también conocidas como etiquetas. El objetivo es desarrollar una función que mapee las entradas a las salidas esperadas. Dentro del aprendizaje supervisado, existen dos tipos principales: la clasificación, utilizada cuando las salidas corresponden a categorías o etiquetas discretas, y la regresión, empleada cuando las salidas son valores numéricos continuos (Ayodele, 2010; Badillo et al., 2020; Oladipupo, 2010).
- **Aprendizaje no supervisado:** El modelo se entrena utilizando únicamente un conjunto de datos que consta de las entradas, sin incluir las salidas esperadas o las etiquetas. El modelo debe encontrar patrones o estructuras en los datos por sí mismo. En este enfoque, el objetivo es explorar y descubrir la estructura subyacente de los datos sin la guía de las etiquetas (Ayodele, 2010; Badillo et al., 2020; Oladipupo, 2010).
- **Aprendizaje por refuerzo:** El modelo aprende a través de la interacción con un entorno. El modelo recibe retroalimentación en forma de recompensas o castigos por las acciones que realiza. El objetivo es que el modelo aprenda a tomar decisiones secuenciales que maximicen una recompensa acumulada a lo largo del tiempo. Este enfoque es útil en situaciones donde el entorno es dinámico y no se dispone de un conjunto de datos previamente etiquetado (Ayodele, 2010; Badillo et al., 2020; Oladipupo, 2010).

2.7. Procesamiento de Lenguaje Natural

De acuerdo con Chowdhury (2020) y Sonbol et al. (2022), el NLP es un campo de investigación y aplicación que se centra en cómo las computadoras pueden entender y procesar texto o voz en lenguaje natural para llevar a cabo diversas tareas. Sus aplicaciones abarcan una amplia gama de áreas, tales como la traducción automática, el resumen de textos, las interfaces de usuario, el multilingüismo, la recuperación de información en varios idiomas, el reconocimiento de voz, la IA y los sistemas expertos, entre otros.

Para Jain et al. (2018), el NLP implica la capacidad de las computadoras para entender, interpretar y generar lenguaje humano de manera efectiva. Esto engloba una serie de tareas complejas, como comprender el significado de las palabras, la sintaxis, la semántica, el contexto y el tono en el que se expresan. Estas habilidades son cruciales para una amplia gama de aplicaciones, desde la traducción automática hasta los *chatbots* y los sistemas de análisis de sentimientos en redes sociales (Javaid et al., 2022). El NLP es una de las principales áreas de la IA, con el objetivo de permitir a las computadoras comprender y procesar textos escritos en lenguaje natural para alcanzar objetivos específicos (Chowdhury, 2020; Sonbol et al., 2022). Se divide en tres niveles: léxico y morfológico, sintáctico y semántico.

En el nivel morfológico se analizan las palabras en función de su estructura interna, considerando elementos como prefijos, sufijos y raíces. En este nivel se realizan tareas como la tokenización, que consiste en dividir un texto en una lista de unidades denominadas *tokens*, las cuales

pueden ser palabras, números o signos de puntuación. Asimismo, se lleva a cabo la lematización, que implica identificar la forma base o lema de cada palabra (Sonbol et al., 2022).

El nivel sintáctico se enfoca en analizar la estructura gramatical de las oraciones, lo que incluye el etiquetado de partes de la oración (*POS-Tagging*), la fragmentación, el análisis de dependencia y el reconocimiento de entidades nombradas. El *POS-Tagging* consiste en etiquetar cada token en una oración con su categoría gramatical (sustantivo, verbo, adjetivo) basado en su contexto sintáctico, mientras que la fragmentación identifica los constituyentes sintácticos como frases nominales y verbales en una oración. El análisis de dependencia analiza la estructura sintáctica de una oración para descubrir las relaciones gramaticales entre las palabras y el tipo de relación entre ellas, mientras que el reconocimiento de entidades nombradas busca localizar y clasificar las entidades mencionadas en la oración en categorías predefinidas como personas, organizaciones y ubicaciones (Chowdhury, 2020).

En el nivel semántico, se busca comprender profundamente el significado del texto y convertirlo en una representación formal. Esto se logra mediante diversas técnicas, como la representación basada en ontología, donde se utilizan ontologías como WordNet para representar un conjunto de conceptos y sus relaciones dentro de un dominio específico, facilitando la asignación de palabras a conceptos predefinidos y la medición de la similitud semántica entre ellos. Otro enfoque es el Modelo Espacial Vectorial (VSM, por sus siglas en inglés), que representa el texto como una matriz término por documento, utilizando las frecuencias de las palabras como pesos y aplicando factores de ponderación como IDF y TF-IDF en el caso de la Bolsa de Palabras (BoW, por sus siglas en inglés). También se emplea la representación basada en modelado de tópicos, una aproximación estadística que busca descubrir temas latentes en un conjunto de textos. Por último, se utilizan técnicas avanzadas de integración, que aprenden representaciones vectoriales de palabras de alta calidad a partir de grandes cantidades de texto no estructurado (Sonbol et al., 2022).

2.8. Modelos de Lenguaje Largo

El lenguaje desempeña un papel fundamental en la comunicación humana y en la interacción con las computadoras. En respuesta a la creciente demanda de sistemas capaces de manejar tareas lingüísticas complejas, han surgido los LLMs. Estos avances se han visto impulsados por mejoras tecnológicas como los *transformers*, el aumento de la capacidad computacional y la disponibilidad de grandes conjuntos de datos de entrenamiento (Naveed et al., 2024).

Los LLMs han desencadenado una transformación revolucionaria al aproximarse al desempeño humano en una variedad de tareas lingüísticas. Trabajos iniciales, como T5 y mT5, han demostrado la capacidad de estos modelos para adaptarse a nuevas tareas sin necesidad de ajustes adicionales. Además de responder con precisión a las solicitudes de tareas específicas, modelos como GPT-3 y sus versiones posteriores han exhibido habilidades emergentes como razonamiento, planificación y toma de decisiones, incluso sin un entrenamiento específico para estas capacidades. Esta versatilidad ha propiciado la adopción generalizada de los LLMs en diversos campos, desde aplicaciones multimodales hasta el desarrollo de agentes autónomos. Los LLMs están redefiniendo los límites de lo que es posible en la comprensión y generación de lenguaje natural, estableciéndose como una herramienta poderosa y adaptable en la investigación y la práctica aplicada (Brown et al., 2020; Naveed et al., 2024; Xue et al., 2021).

De acuerdo con Minaee et al. (2024), los LLMs son modelos estadísticos basados en redes neuronales que se entrenan a gran escala. Estos modelos, principalmente basados en *transformers*, contienen una gran cantidad de parámetros y se pre-entrenan en grandes conjuntos de datos, como PaLM (Chowdhery et al., 2023), LLaMA (Touvron et al., 2023), GPT-4 (OpenAI et al., 2024) entre

otros. Los LLMs exhiben capacidades como el aprendizaje contextual, el seguimiento de instrucciones y el razonamiento de múltiples pasos. Además, tienen la capacidad de expandir su conocimiento y mejorar continuamente a través de la retroalimentación de usuarios y el aprendizaje reforzado. Estos modelos tienen una amplia gama de aplicaciones, que incluyen *chatbots*, traducción de idiomas, resumen de texto y respuesta a preguntas. Su versatilidad y capacidad para adaptarse a diferentes tareas y dominios han impulsado su adopción en diversos campos de la IA y de NLP. Los LLMs están redefiniendo las posibilidades en la generación y comprensión de lenguaje natural, posicionándose como herramientas fundamentales en la investigación y la práctica aplicada.

Aunque los LLMs han tenido una aplicación predominante en tareas lingüísticas, su potencial en la Ingeniería de *Software* ha emergido como un área de investigación prometedora. Recientes investigaciones han explorado cómo los LLMs pueden automatizar varios procesos cruciales en el desarrollo de *software*. Por ejemplo, se ha investigado cómo los LLMs pueden automatizar la generación de código (Thakur et al., 2023; Xu et al., 2022), permitiendo a los desarrolladores generar automáticamente fragmentos de código complejos a partir de descripciones en lenguaje natural. Además, se ha demostrado su utilidad en la documentación de *software* (Lin et al., 2024; Wang et al., 2023), facilitando la creación automática de documentación clara y detallada sobre el funcionamiento de sistemas y aplicaciones. Además, los LLMs han mostrado capacidad para la comprensión de requisitos (Marques et al., 2024; Peer et al., 2024), interpretando y sintetizando información compleja de los usuarios y transformándola en requisitos detallados y comprensibles para los equipos de desarrollo. Esta capacidad de los LLMs para automatizar procesos críticos en la Ingeniería de *Software* está redefiniendo las prácticas tradicionales y promete mejorar la eficiencia y la precisión en el desarrollo de *software* moderno.

En el contexto actual, donde los LLMs han demostrado su eficacia en una amplia gama de tareas, optimizar su uso para resolver problemas complejos es una cuestión fundamental. La ingeniería rápida, es decir, el arte de formular indicaciones precisas y claras, se ha destacado como el método más directo y efectivo para interactuar con estos modelos. Al proporcionar instrucciones detalladas, los usuarios pueden mejorar significativamente la precisión y relevancia de las respuestas generadas por los LLMs, adaptándolas al contexto y a la audiencia específica. Esta práctica no solo facilita la gestión del tono y estilo del resultado, sino que también minimiza las ambigüedades, haciendo la interacción más eficiente y productiva. Por lo tanto, la ingeniería rápida es esencial para aprovechar al máximo el potencial de los LLMs, asegurando respuestas altamente beneficiosas y contextualmente apropiadas (Li et al., 2023; Lin, 2024).

Aunque interactuar con un LLM puede parecer sencillo simplemente formular una pregunta para obtener una respuesta, lograr un compromiso efectivo con estos modelos es más desafiante y complejo de lo que parece. La utilidad de los LLMs está estrechamente vinculada a la calidad de las indicaciones proporcionadas, un aspecto a menudo subestimado en las discusiones sobre sus capacidades. Una indicación bien estructurada puede llevar a una respuesta precisa y relevante, mientras que una mal formulada puede resultar en una salida vaga o incorrecta. Esto se debe en gran medida a la naturaleza inherente de los LLMs. A pesar de su sofisticación y del gran volumen de datos con los que han sido entrenados, que incluye desde páginas web y artículos académicos hasta publicaciones en redes sociales y libros, estos modelos son esencialmente algoritmos matemáticos que predicen la probabilidad de secuencias de texto. No están diseñados para producir verdades absolutas, sino para generar texto basado en patrones estadísticos. Cada predicción influye en las siguientes, por lo que un error inicial puede desencadenar una cadena de imprecisiones. Así, una indicación bien elaborada no solo mejora la precisión de cada token generado, sino que también reduce el riesgo de errores acumulativos. Además, la capacidad de los LLMs para el aprendizaje en contexto permite que se

adapten temporalmente a las indicaciones recibidas, subrayando la importancia de formular instrucciones claras y detalladas (Arefeen et al., 2024; Ge et al., 2024; Yao et al., 2024).

2.9. Modelos de Lenguaje Largo: GPT, LLaMA, PaLM

Los LLMs constituyen un avance significativo en el campo de NLP, destacándose por su capacidad para abordar una amplia gama de tareas lingüísticas y aplicaciones (Kalyan, 2024). Tres de los modelos más prominentes en esta categoría son GPT (*Generative Pre-trained Transformers)*, LLaMA (*Large Language Model Meta-Learning Approach*) y PaLM, cada uno diseñado con enfoques y objetivos específicos que optimizan su desempeño en diferentes contextos (Minaee et al., 2024).

GPT es una serie de modelos de lenguaje basados en decodificadores *transformers*, desarrollados por OpenAI. Desde el lanzamiento inicial de GPT-1, estos modelos han experimentado una evolución notable hasta llegar a GPT-4o. Cada iteración ha mejorado considerablemente en cuanto a sus capacidades en generación de texto, comprensión del lenguaje y respuestas contextualmente relevantes (Sonoda et al., 2024; Wilbanks et al., 2024). GPT-3, por ejemplo, se destaca por su escalabilidad y potencia. Este modelo está pre-entrenado en una vasta cantidad de datos, lo que le permite manejar una amplia variedad de tareas complejas, desde la generación de texto hasta la traducción automática y más. Esta versatilidad lo ha convertido en una herramienta fundamental en diversas aplicaciones, incluyendo InstrucGPT para la generación de instrucciones detalladas y ChatGPT para mantener conversaciones fluidas y naturalmente interactivas (Minaee et al., 2024; OpenAI et al., 2024). Desde su introducción, los modelos GPT han sido adoptados ampliamente en la industria y la investigación, demostrando su capacidad para mejorar significativamente la eficiencia y la calidad en una variedad de aplicaciones de NLP (Sayin y Gierl, 2024).

LLaMA es una iniciativa de Meta, compuesta por una colección de modelos de lenguaje de código abierto. Estos modelos están diseñados para el aprendizaje meta, lo que les permite adaptarse rápidamente a nuevos dominios y tareas con un ajuste mínimo. Basados en la arquitectura *transformers*, similar a GPT-3, pero con modificaciones adaptadas a sus propios fines, los modelos de LLaMA incluyen una serie de variantes como Code LLaMA, Gorilla, Giraffe, y otros que se especializan en aplicaciones específicas y en la mejora del rendimiento general en tareas complejas de NLP (Minaee et al., 2024; Touvron et al., 2023).

PaLM, desarrollado por Google, presenta una familia de modelos de lenguaje que se distinguen por su escala masiva y su capacidad para manejar una amplia gama de tareas de NLP. PaLM ha sido entrenado en un corpus enorme de datos de alta calidad, con el modelo inicial contando con 540 mil millones de parámetros y pre-entrenado en 780 mil millones de tokens. Esta familia de modelos incluye variantes como U-PaLM, PaLM-2, y otros modelos especializados como Med-PaLM para aplicaciones médicas, que demuestran resultados de vanguardia en tareas de generación y comprensión de lenguaje (Chowdhery et al., 2023; Minaee et al., 2024).

De acuerdo con la investigación realizada por Carlà et al. (2024), el 8 de febrero de 2024, Google presentó una actualización de su plataforma de IA, Gemini, que incorpora funciones mejoradas y un análisis multimodal optimizado. Gemini combina las capacidades de asistentes digitales y *chatbots*, sobresaliendo en el manejo de entradas de voz y texto para realizar una amplia variedad de tareas. Diseñada para satisfacer diversas necesidades, esta plataforma representa la culminación del desarrollo de Google hacia la IA multimodal. La IA multimodal, como señalan Ahmed y Islam (2024), representa un avance significativo al integrar distintos tipos de datos, como texto, imágenes, audio y video, mejorando así la comprensión, el análisis y la toma de decisiones. Gemini es un modelo innovador capaz de procesar múltiples tipos de datos, como texto, código, audio, imágenes y video, gracias a su diseño multimodal y a su capacitación previa en diversas modalidades, lo que le permite

realizar ajustes precisos (Gemini Team et al., 2024; Kahng et al., 2024; Kevian et al., 2024; Lee et al., 2023).

En conjunto, estos modelos representan una revolución en la capacidad de los sistemas de IA para manejar y comprender el lenguaje natural en diversas aplicaciones. Su desarrollo continuo y aplicación en escenarios del mundo real continúa expandiendo los límites de lo posible en el NLP, mejorando constantemente su capacidad para adaptarse y aprender de nuevos datos y contextos (Minaee et al., 2024; Yao et al., 2024).

2.10. Ingeniería de *prompts*

En la investigación realizada por Vatsal y Dubey (2024), se destaca el avance significativo de la IA con la introducción de los LLMs, modelos entrenados utilizando grandes cantidades de texto que abarcan millones, e incluso miles de millones de tokens. Se ha demostrado que aumentar la cantidad de parámetros en un modelo mejora su rendimiento, un principio que también se ha manifestado en los LLMs, los cuales han logrado un rendimiento sin precedentes en una amplia variedad de tareas de NLP. Para Chang et al. (2023), estos avances han generado un gran interés tanto en la academia como en sectores industriales como la medicina, el derecho y las finanzas.

Actualmente, la investigación sobre LLMs se centra en evaluar su capacidad de razonamiento a través de *prompts*, instrucciones en lenguaje natural, en lugar de centrarse únicamente en la predicción del siguiente token. Este enfoque ha dado lugar al campo emergente de Ingeniería de *prompts*, que consiste en diseñar instrucciones que permiten extraer de manera organizada el conocimiento incrustado en los LLMs, sin necesidad de ajustes extensivos en sus parámetros (Chang et al., 2024).

Para Gu et al. (2023), la Ingeniería de *prompts* adapta modelos preentrenados a nuevas tareas mediante la inclusión de pistas específicas en la entrada del modelo. Estas pistas, conocidas como *prompts*, pueden ser instrucciones en lenguaje natural, ya sea generadas manualmente o de forma automática, o representaciones vectoriales generadas automáticamente. Las instrucciones en lenguaje natural se denominan *prompts* discretos o *hard prompts*, mientras que las representaciones vectoriales son *prompts* continuos o *soft prompts*. Este enfoque permite que los modelos preentrenados realicen tareas sin necesidad de reentrenamiento.

2.10.1. *Hard prompts*

Para Gu et al. (2023), *los hard prompts* son instrucciones diseñadas manualmente que consisten en texto interpretativo, creado específicamente para guiar el comportamiento de los modelos hacia tareas concretas. Este enfoque resulta especialmente valioso porque permite aprovechar los modelos preentrenados sin necesidad de modificar sus parámetros, lo que los hace más versátiles y fácilmente adaptables a nuevas tareas.

Existen varios métodos para implementar *hard prompts*, cada uno enfocado en resolver problemas específicos y optimizar la eficiencia del modelo. Entre estos métodos destacan:

- ***Task Instruction Prompting:*** Descrito por Radford et al. (2023) y Sahoo et al. (2024), se basa en diseñar instrucciones claras y explícitas para orientar al modelo en la ejecución de una tarea particular. Estas instrucciones, cuidadosamente formuladas, proporcionan el contexto necesario y aseguran que el modelo comprenda de manera precisa la actividad a realizar.
- ***In-Context Learning:*** Es un enfoque ampliamente estudiado por Dong et al. (2024) e Izacard et al. (2022). Este método utiliza ejemplos relacionados dentro del *prompt* para ofrecer un contexto que facilite el aprendizaje. Al presentar al modelo secuencias de

ejemplos relevantes, se mejora su capacidad para generalizar y responder de manera coherente, promoviendo un entendimiento más profundo de las instrucciones y los datos proporcionados.

- ***Retrieval-Based Prompting:*** Descrito por Li et al. (2023), Rubin et al. (2022), Yang et al. (2022) y Ye et al. (2023), emplea recursos externos como bases de conocimiento o conjuntos de datos para seleccionar contextos relevantes que guíen al modelo en la generación de respuestas. Este método incrementa significativamente el desempeño al permitir que el modelo se base en información precisa y previamente almacenada, siendo particularmente útil en tareas complejas o altamente especializadas.
- ***Chain-of-Thought Prompting:*** Estudiado por Qiao et al. (2023), Wei et al., (2022) y Zhang et al. (2022), guía al modelo mediante una secuencia de instrucciones o preguntas diseñadas de forma progresiva. En este enfoque, cada paso añade información o delimita el enfoque, ayudando al modelo a mantener una lógica coherente y generar respuestas estructuradas. Este método es especialmente valioso en escenarios que demandan razonamiento detallado o un enfoque iterativo.

Los *hard prompts* destacan por la capacidad que tiene de aprovechar el conocimiento integrado en los modelos, facilitando su adaptación a una amplia gama de tareas sin necesidad de reentrenamiento. Además, debido a que son instrucciones claras y diseñadas manualmente, ofrecen un alto grado de personalización, lo que los convierte en herramientas esenciales para aplicaciones donde la precisión y la coherencia son cruciales (Gu et al., 2023).

2.10.2. Alucinaciones en LLMs (*Hallucination*)

Las alucinaciones en los LLMs ocurren cuando el modelo genera respuestas que no están relacionadas con el contexto o que son incorrectas. Esto puede suceder cuando el modelo produce información ficticia o errónea, afectando la precisión de las respuestas generadas. De acuerdo con Martino et al., 2023 y Rocco et al. 2024), las causas de las alucinaciones incluyen a las siguientes:

- **Falta de comprensión del contexto:** Aunque los LLMs son entrenados con grandes volúmenes de texto, pueden no interpretar correctamente el contexto de una conversación, lo que resulta en respuestas incoherentes.
- **Desconexión del tema:** Las alucinaciones también surgen cuando el modelo genera respuestas que no se ajustan al tema en discusión, especialmente en conversaciones largas.
- **Complejidad del lenguaje natural:** El lenguaje humano tiene ambigüedades que los modelos a veces no logran entender completamente, lo que puede llevar a respuestas incorrectas.
- **Sesgos en los datos de entrenamiento:** Si los datos de entrenamiento contienen información sesgada o errónea, el modelo puede generar respuestas equivocadas.

2.10.3. Técnicas para reducir las alucinaciones

Para Sahoo et al. (2024), las alucinaciones son un reto importante, existen diversas técnicas que se están desarrollando para mitigarlas y mejorar la precisión de los LLMs. Entre las más efectivas se encuentran:

- **Generación Aumentada por Recuperación (RAG, por sus siglas en inglés):** Esta técnica permite al modelo consultar bases de datos externas para mejorar la precisión de las respuestas, reduciendo la dependencia del conocimiento preexistente (Ram et al., 2023).

- ***ReAct Prompting*:** Esta técnica integra de forma simultánea el razonamiento y la ejecución de acciones en los modelos de lenguaje, lo que mejora su capacidad para planificar, ajustar estrategias y manejar excepciones de manera eficiente durante la resolución de tareas (Yao et al., 2023).
- ***Chain-of-Verification* (CoVe, por sus siglas en inglés):** Esta técnica verifica las respuestas generadas en varias etapas. El modelo primero genera una respuesta, luego plantea preguntas de verificación y ajusta la respuesta inicial según las correcciones necesarias (Dhuliawala et al., 2023).
- ***Chain-of-Note* (CoN, por sus siglas en inglés):** Es una técnica diseñada para reducir las alucinaciones en los modelos de lenguaje aumentados por recuperación. Filtra documentos irrelevantes y maneja escenarios desconocidos, lo que mejora la precisión y relevancia de las respuestas, evitando respuestas incorrectas o imprecisas (Yu et al., 2024).
- ***Chain-of-Knowledge* (CoK, por sus siglas en inglés):** Divide tareas complejas en pasos secuenciales, reuniendo evidencia de diversas fuentes para mejorar la precisión en tareas que requieren razonamiento complejo (Li et al., 2024).

Las alucinaciones son un desafío importante en los LLMs, pero con enfoques como RAG, ReAct, CoVe y CoK, se puede mejorar la precisión y coherencia de las respuestas generadas, reduciendo las alucinaciones y aumentando la fiabilidad del modelo.

2.10.4. Limitaciones de los LLMs

A pesar de los avances significativos logrados con los LLMs, su implementación enfrenta diversas limitaciones que afectan tanto su desempeño como su confiabilidad. Una de las principales problemáticas radica en la capacidad de estos modelos para interpretar de manera adecuada el contexto de las consultas o peticiones realizadas por los usuarios. Aunque son entrenados con grandes volúmenes de datos textuales, no siempre logran comprender completamente el entorno en el que se desarrollan las conversaciones, lo que puede dar lugar a respuestas imprecisas o irrelevantes (Martino et al., 2023). Además, en ocasiones, los modelos generan información incorrecta o fabricada, incluso cuando no existe respaldo en los datos para sustentar dichas afirmaciones, lo que representa un desafío crítico en aplicaciones que exigen altos estándares de precisión (Rocco et al., 2024).

Otra limitación significativa se observa en conversaciones extensas, donde los LLMs tienden a perder coherencia temática, desviándose del tópico principal y produciendo respuestas desconectadas del contenido previamente discutido. Esta desconexión dificulta su uso en tareas donde la continuidad es esencial (Gu et al., 2023). Por otro lado, la ambigüedad y la complejidad inherentes al lenguaje humano plantean desafíos adicionales. Los matices lingüísticos, las interpretaciones contextuales y las múltiples formas de expresar ideas pueden superar las capacidades de los modelos, lo que resulta en respuestas que no cumplen con las expectativas de exactitud y relevancia (Dong et al., 2024).

Finalmente, los sesgos presentes en los datos utilizados para entrenar estos modelos representan una preocupación importante. Si los conjuntos de datos contienen errores, información parcializada o prejuicios culturales, estos sesgos se reflejan en las respuestas generadas, comprometiendo la objetividad y utilidad de los LLMs en aplicaciones sensibles (Martino et al., 2023). Estas limitaciones resaltan la necesidad de continuar investigando y desarrollando estrategias que permitan optimizar el desempeño y la confiabilidad de los LLMs, especialmente en contextos que demandan precisión, coherencia y responsabilidad (Rocco et al., 2024).

Una de las principales limitaciones de los LLMs es su capacidad para generar respuestas precisas y contextualizadas. Estos modelos, aunque son potentes, pueden producir respuestas incorrectas debido a la falta de comprensión profunda del contexto o a la limitación de datos en su entrenamiento. Para abordar estos desafíos, se ha propuesto el uso de sistemas RAG, que combinan la capacidad de recuperación de información externa con la generación de texto, mejorando la precisión y relevancia de las respuestas generadas.

2.11. Generación aumentada por recuperación

La RAG es una técnica avanzada que busca enriquecer los LLMs al incorporar información externa y actualizada. De acuerdo con Ram et al. (2023), este enfoque permite a los LLMs acceder a fragmentos específicos de información obtenidos mediante Sistemas de Recuperación de Información (IR, por sus siglas en inglés), lo que mejora la calidad de la generación de texto al añadir datos relevantes y en tiempo real. Este proceso resulta especialmente útil en contextos donde la información cambia rápidamente y no puede ser almacenada estáticamente en los modelos preentrenados.

La investigación de Shi et al., (2023) destaca que los sistemas RAG permiten superar las limitaciones de los LLMs tradicionales, ya que amplía el contexto del modelo con datos externos, lo que mejora la precisión y relevancia de las respuestas generadas, especialmente en situaciones que requieren datos específicos y actualizados. Esto es especialmente valioso en aplicaciones como *chatbots* y asistentes virtuales especializados, donde es esencial ofrecer respuestas precisas y pertinentes (Van Veen et al., 2023; Zeng et al., 2024).

A pesar de su destacado rendimiento en tareas como la generación de texto y la respuesta a preguntas complejas, los LLMs tradicionales tienen dificultades para manejar grandes volúmenes de información o contextos extensos. Esta limitación hace que dependan en gran medida de su conocimiento preentrenado, el cual, debido a su naturaleza estática, puede resultar insuficiente cuando se necesita información nueva o específica (Brown et al., 2020; Zhao et al., 2024). Los sistemas RAG mejoran este aspecto al permitir que los LLMs accedan a datos externos y actualizados, lo que permite una mayor precisión en las respuestas. Además, la integración de datos externos permite a los LLMs capturar conocimientos más especializados y específicos de ciertos dominios, lo que aumenta la calidad de las respuestas. Las consultas realizadas a estos sistemas pueden clasificarse según la complejidad y el nivel de detalle requerido, lo que refleja la capacidad del modelo para manejar diferentes tipos de información (Gaoa et al., 2023; Shi et al., 2023; Wang et al., 2024; Zhao et al., 2024).

De acuerdo con la investigación de Cuconasu et al. (2024), RAG se presenta como una solución innovadora para mejorar la precisión de los LLMs al incorporar información externa relevante. Al acceder a fuentes externas a través de un sistema de IR, los modelos pueden enriquecer el contexto de las consultas en tiempo real, reduciendo así los errores y aumentando la confiabilidad de las respuestas, lo cual es crucial en aplicaciones que requieren datos detallados y actualizados (Nguyen et al., 2024).

Los sistemas RAG se componen de dos elementos clave: el recuperador y el generador. El recuperador es responsable de activar un sistema de IR, que puede ser denso o disperso, para obtener fragmentos relevantes de una base de datos de documentos, que en muchos casos puede incluir toda la web. Una vez que el recuperador obtiene los fragmentos más relevantes, los envía al generador para que los procese e incluya en la respuesta final (Li et al., 2023; Chen et al., 2024; Meng et al., 2024).

Por otro lado, la investigación de Chen et al., (2024) menciona que los LLMs interactúan con el usuario mediante la formulación de consultas que el recuperador utiliza para extraer la información

necesaria. Esta interacción permite que el LLM acceda a datos que no se encuentran almacenados dentro de su conocimiento preentrenado, lo que enriquece las respuestas generadas con información más actualizada y específica. Para mejorar la precisión y la relevancia de los datos recuperados, el recuperador generalmente emplea un modelo de incrustación y, en algunos casos, un reclasificador. Estas técnicas ayudan a refinar la selección de los documentos relevantes, asegurando que los fragmentos de información más pertinentes sean utilizados en la generación de la respuesta final (Li et al., 2023; Muennighoff et al., 2023; Meng et al., 2024).

De este modo, los sistemas RAG amplían las capacidades de los LLMs al proporcionarles acceso a información adicional que no forma parte de su entrenamiento inicial (véase Figura 2.4). Esto permite aumentar la precisión y la confiabilidad de las respuestas generadas, especialmente en escenarios que requieren conocimiento detallado, actualizado y específico. Al integrar información externa relevante, RAG supera las limitaciones de los modelos de lenguaje tradicionales, mejorando sustancialmente la calidad y la relevancia de las respuestas en aplicaciones que demandan un alto grado de precisión y contexto (Lin et al., 2024; Zeng et al., 2024; S. Zhao et al., 2024).

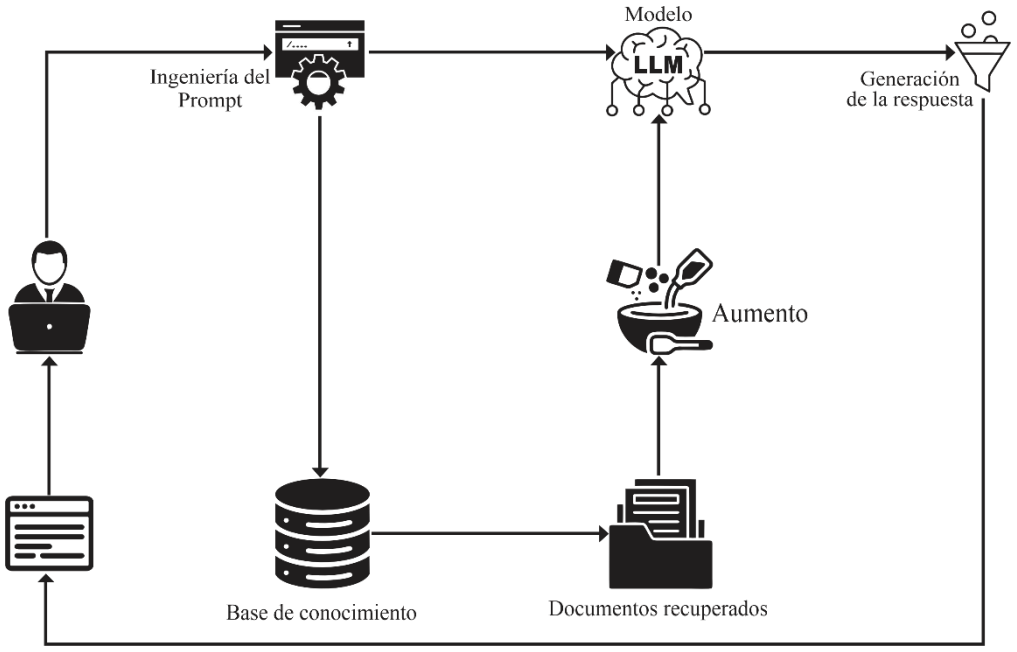


Figura 6 Diagrama de RAG, adaptado de la investigación realizada por Rocco et al., (2024).

2.12. Estado del arte

En esta sección se presenta una revisión exhaustiva de la literatura relevante, centrándose en los estudios más significativos y recientes que han abordado la generación automática de diagramas UML en contextos ágiles. Se realiza un análisis crítico de las diferentes aproximaciones metodológicas y herramientas utilizadas, evaluando sus fortalezas, limitaciones y aplicaciones prácticas. Esta revisión, no solo establece una base sólida de conocimiento teórico, sino que también identifica las áreas de oportunidad y las lagunas actuales en la investigación, las cuales motivan el desarrollo de la presente investigación.

A lo largo de la historia de la Ingeniería de *Software*, se han realizado múltiples esfuerzos para automatizar los procesos de recolección y gestión de requisitos; sin embargo, la automatización de los requisitos ágiles, en particular las historias de usuario, ha sido una tarea pendiente. Dado que los métodos ágiles son ampliamente adoptados en el desarrollo de *software*, surge la necesidad de

desarrollar herramientas que faciliten la transformación de estas historias en modelos comprensibles. Las historias de usuario son vitales en el proceso de comunicación entre los usuarios y los equipos de desarrollo, lo que subraya su importancia en la eficacia de la Ingeniería de *Software*.

2.12.1. Propuesta de un entorno visual para soportar Scrum

El enfoque propuesto por Kussunga y Ribeiro (2019) tuvo como objetivo proporcionar un entorno visual para respaldar las ceremonias de la metodología ágil Scrum, facilitando la transformación y gestión eficiente de requisitos de *software*. Este enfoque se diferenciaba de un ciclo de vida convencional de Scrum en varios aspectos clave (véase Figura 2.5). Algunas diferencias entre el enfoque propuesto y un ciclo convencional de vida de Scrum son las siguientes:

- Durante la Planeación del *Sprint*, se utilizó el modelado UML para crear un plan basado en la pila del producto, derivado de la transformación de historias de usuario en casos de uso mediante el método 4SRS[6]. Este método aseguró una alineación precisa de los requisitos con los casos de uso y facilitó una mejor comprensión de las funcionalidades que el equipo de desarrollo implementaría.
- En la ejecución del *Sprint*, se realizaron iteraciones que condujeron a la entrega de un incremento potencialmente utilizable del producto. Se ejecutaron las tareas planificadas, incluyendo la implementación de los casos de uso y el refinamiento continuo de los requisitos mediante múltiples iteraciones del método 4SRS. Esto garantizó que los requisitos fueran evaluados, validados y ajustados de manera iterativa y colaborativa.
- El método 4SRS transformó los requisitos del usuario en un modelo arquitectónico a través de pasos estructurados y recursivos. Este enfoque promovió una mayor transparencia, comunicación y alineación entre el equipo de desarrollo y las partes interesadas, asegurando una arquitectura global coherente y bien estructurada.

El enfoque propuesto combinó prácticas ágiles con técnicas de modelado UML y el método 4SRS con el objetivo de mejorar la eficiencia, la calidad y la adaptabilidad en el desarrollo de *software*. Proporcionó un marco estructurado y visual que facilitó la gestión y transformación de requisitos en proyectos ágiles basados en Scrum. Al integrar técnicas de modelado y transformación arquitectónica, este enfoque promovió una mayor transparencia, comunicación y alineación entre el equipo de desarrollo y las partes interesadas, asegurando la entrega continua de valor al cliente.

De acuerdo con Kussunga y Ribeiro (2019), su enfoque propuesto no solo permitió una planificación más eficiente de los *Sprints* y una transformación efectiva de las historias de usuario en casos de uso, sino que también facilitó la adaptación continua a los cambios en los requisitos del sistema y mejoró la comunicación y colaboración dentro del equipo y con las partes interesadas. La combinación de prácticas ágiles y técnicas de modelado UML aseguró una mayor visibilidad y control sobre el proceso de desarrollo de *software*, garantizando que los productos entregados cumplieran con los estándares de calidad y satisficieran las necesidades del cliente de manera efectiva.

[6] De acuerdo con Pereira et al., (2023) 4SRS es un método empleado en el muestreo estadístico y el aprendizaje supervisado. Este método se enfoca en la selección secuencial y aleatoria de muestras, con el propósito de optimizar la eficiencia y la precisión en la estimación de parámetros o en la predicción de resultados dentro de conjuntos de datos extensos y complejos.

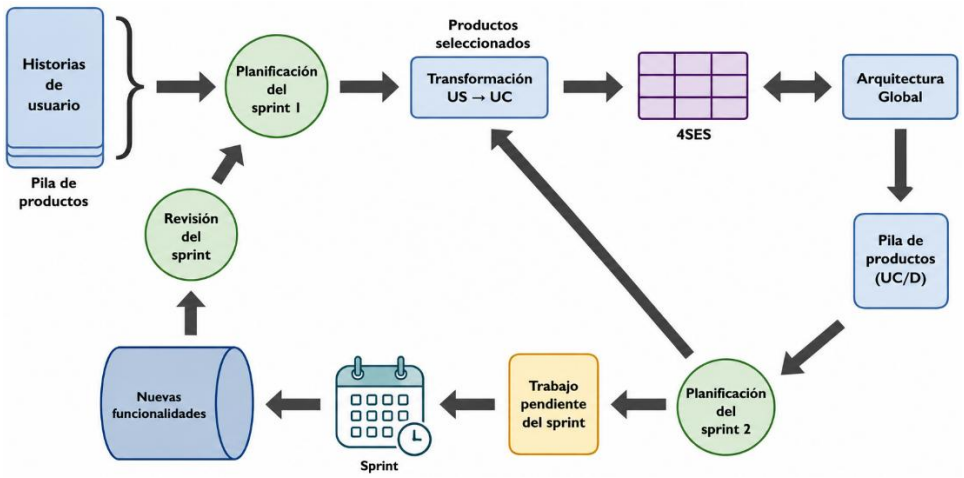


Figura 7 Proceso de aproximación al entorno visual (Kussunga y Ribeiro 2019).

2.12.2. Generación de diagramas de clases a partir de historias de usuario en métodos ágiles

La MDA es un marco conceptual para los procesos de desarrollo de *software* que facilita la transformación automática de un modelo de proceso de negocio a un modelo de código. En el contexto de MDA, se identifican dos tipos principales de transformaciones:

- **Transformación del Modelo Independiente de Computación (CIM, por sus siglas en inglés) al Modelo Independiente de la Plataforma (PIM, por sus siglas en inglés):** Esta transformación se enfoca en convertir un modelo que describe los requisitos y la funcionalidad del sistema sin considerar detalles técnicos de implementación, en un modelo que, aunque aún abstracto, está más cerca de cómo se implementará el sistema en términos de arquitectura y diseño (Azanzi et al., 2023; Kharmoum et al., 2020; Melouk et al., 2020).

- **Transformación del PIM al Modelo Específico de Plataforma (PSM, por sus siglas en inglés):** Esta transformación consiste en adaptar el modelo independiente de la plataforma a un modelo específico para una plataforma tecnológica particular. En esta etapa, se añaden detalles específicos de la plataforma seleccionada, tales como el lenguaje de programación, las bases de datos y otros componentes tecnológicos (Azanzi et al., 2023; Kharmoum et al., 2020; Melouk et al., 2020).

En este sentido, la investigación realizada por Nasiri et al., (2020) presentó un enfoque para la extracción de elementos de diseño implementado en Python utilizando la librería Stanford CoreNLP[7]. Este enfoque automatizó la transformación del nivel CIM, que representa un conjunto de historias de usuario, al nivel PIM. Mediante esta transformación automática, se generó un archivo XMI en el cual describe un diagrama de clases (véase Figura 2.6).

[7] Para más información sobre la librería Stanford CoreNLP, por favor referirse a: https://stanfordnlp.github.io/CoreNLP/

Figura 8 Arquitectura del enfoque propuesto por Nasiri et al., (2020).

Para construir un diagrama de clases a partir de varias historias de usuario, fue necesario extraer componentes clave como clases, atributos, relaciones y operaciones de clases. Este proceso empleó técnicas de tokenización, análisis de partes del discurso y resolución de correferencias para el procesamiento de las historias de usuario. Posteriormente, se aplicaron las dependencias entre palabras para cada oración en una historia de usuario. El proceso de derivación, que reducía una palabra a su raíz, se utilizó para evitar la duplicación de clases, plural y singular. Además, se empleó WordNet[8] para eliminar redundancias en las clases y convertir formas verbales a su forma básica, así como para encontrar sinónimos de verbos que indicaban relaciones de composición. La extracción de elementos de diseño se basó en reglas detalladas en la Tabla 1, Tabla 2 y Tabla 3, las cuales analizaron las dependencias gramaticales de cada palabra en las historias de usuario. En primer lugar, se extrajeron los actores y las clases; después, se identificaron las relaciones. La extracción de atributos se fundamentó en las relaciones de composición entre clases. Si había una composición entre dos clases y una de ellas no dependía de otras, esta se transformaba en un atributo. Posteriormente, se asignaron atributos y operaciones a las clases relacionadas. Finalmente, la herramienta utilizó la API PyEcore para generar un archivo XMI y la API PlantUML para visualizar el diagrama de clases en una imagen PNG.

La evaluación del desempeño de la herramienta de extracción de elementos de diseño reveló resultados que destacaron las ventajas del enfoque propuesto, mostrando un mejor rendimiento en comparación con enfoques anteriores, con una precisión del 98% al comparar los resultados obtenidos automáticamente con los obtenidos manualmente. Esto subraya la eficiencia y la precisión de la herramienta en la extracción automática de elementos de diseño a partir de historias de usuario, facilitando así la integración de los requisitos del usuario con el diseño del sistema en el proceso de desarrollo de *software*.

Tabla 1. Reglas de extracción de actores (Nasiri et al., 2020).

Reglas	Descripción
AC1	En la historia de usuario, el primer sustantivo después de "*As*" es un actor, si no es un sustantivo compuesto.
AC2	Si el primer sustantivo después de "As" es un nombre propio, seguido de otro nombre, entonces el segundo sustantivo es un actor.
AC3	El sustantivo compuesto después de "*As*" es actor. Si el primer sustantivo de un sustantivo compuesto es un adjetivo y el segundo sustantivo es un actor, entonces existen relaciones de herencia entre este último y el sustantivo compuesto que es un actor.

[8] WordNet es una base de datos léxica para el idioma inglés, desarrollada en 1986 en la Universidad de Princeton (McCrae et al., 2020).

Tabla 2. Reglas de extracción de clases (Nasiri et al., 2020).

Reglas	Descripción
C1	El sujeto de una oración es una clase, siempre que sea un sustantivo y no un pronombre.
C2	El objeto directo de la oración es una clase.
C3	Se aplica la regla C2 pero el objeto directo forma parte de un sustantivo compuesto.
C4	El sustantivo que precede al apóstrofe posesivo es una clase.
C5	El sustantivo después de estas preposiciones (*of, for, to*) es una clase.
C6	Conjunción entre clases.

Tabla 3. Reglas para extraer relaciones y su clasificación (Nasiri et al., 2020).

Reglas	Descripción
R1	El verbo es una asociación entre el objeto directo de la oración y el sujeto: verbo (objeto directo, sujeto).
R2	Verbo con preposición.
R3	Preposiciones como "*of*", "*to*", "for" y "*about*" detectan una asociación.
R4	Caso posesivo: "*s´*", "*my*", "*his*", determinan una asociación; Correlación de pronombres con sustantivos para extraer las asociaciones utilizando la resolución de correferencia de la herramienta Stanford CoreNLP.
H1	Los siguientes verbos indican una relación de composición: "*Contain, part of, consist, include, have, comprise*".
H2	Sustantivo compuesto: si ambos sustantivos son clases, entonces existe una relación de composición entre ellos.
H3	El verbo al que pertenece lista de verbos que indican una relación de agregación.

2.12.3. Generación de diagramas de actividades y casos de uso utilizando técnicas de NLP y reglas heurísticas

En la investigación realizada por Maatuk y Abdelnabi (2021), se desarrolló un enfoque automatizado para la generación de diagramas UML a partir de requisitos expresados en lenguaje natural. Este enfoque buscó abordar y superar las dificultades inherentes al análisis manual de dichos requisitos, tales como la ambigüedad, la inconsistencia y la incompletitud, mediante el uso de técnicas de NLP y reglas heurísticas. Se centró específicamente en la generación de diagramas de actividades y de casos de uso, con el fin de facilitar el proceso de análisis de requisitos y mejorar la eficiencia y precisión en la producción de modelos UML.

El enfoque propuesto por Maatuk y Abdelnabi (2021) abarcó cuatro fases principales (véase Figura 2.7), diseñadas para transformar requisitos en lenguaje natural en diagramas UML de manera automatizada:

1. **Reconstrucción y normalización del texto en lenguaje natural:** En esta fase se propusieron un conjunto de reglas sintácticas para dividir los requisitos complejos en declaraciones más simples, facilitando así la extracción de elementos UML y mejorando la precisión de los resultados. En total, se definieron dieciséis reglas de reconstrucción

sintáctica que fueron aplicadas durante la redacción y normalización de los documentos de requisitos.

2. **Procesamiento automático del texto de los requisitos mediante NLP:** En esta fase se empleó el NLP para analizar y procesar automáticamente el texto de los requisitos, evitando así el procesamiento manual. Este proceso incluyó varios subprocesos: tokenización de texto, etiquetado de partes del discurso, derivación y lematización, tipado de dependencias y extracción de información abierta. Para realizar estas tareas, se utilizó Stanford CoreNLP.
3. **Aplicación de reglas heurísticas a partir de los resultados del NLP:** En esta fase, se implementaron reglas de mapeo con el objetivo de extraer conceptos de UML a partir de los resultados obtenidos mediante el NLP. Para ello, se empleó un conjunto de reglas heurísticas diseñadas para convertir los conceptos expresados en lenguaje natural en elementos UML, como actores, casos de uso y nodos de actividad. Estas reglas desempeñaron un papel fundamental al facilitar una transformación precisa y estructurada de los requisitos en lenguaje natural hacia elementos UML, garantizando así la coherencia y la precisión en la representación de los mismos.
4. **Generación de diagramas UML:** En esta fase se generaron los diagramas UML utilizando los elementos UML obtenidos en la fase anterior. Estos diagramas incluyeron tanto los diagramas de casos de uso como los diagramas de actividades. Fueron creados manualmente o mediante el uso de herramientas especializadas de dibujo UML, lo que permitió una documentación estructurada y detallada de los requisitos del sistema.

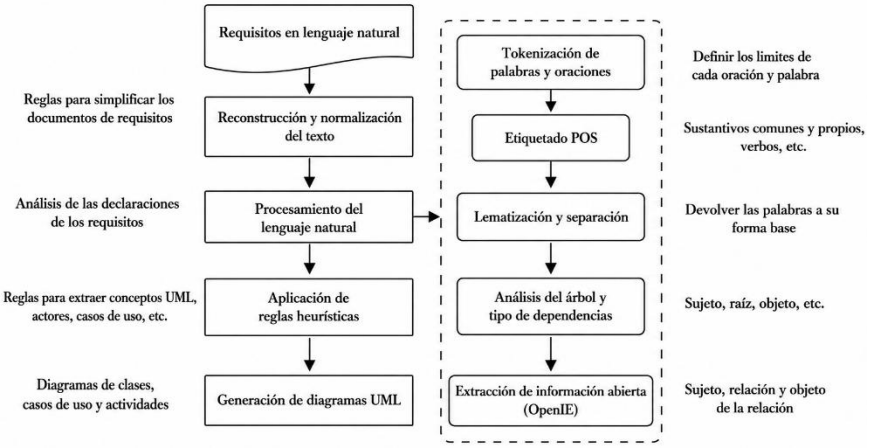


Figura 9 Arquitectura del enfoque propuesto por M. Maatuk y A. Abdelnabi (2021).

En resumen, el enfoque propuesto se centró en mejorar la generación de diagramas de casos de uso y actividades a partir de requisitos informales expresados en lenguaje natural. Inicialmente, se revisaron diversos trabajos relacionados para identificar las limitaciones de las herramientas existentes y las técnicas más eficientes utilizadas en esta área. Se aplicó el enfoque propuesto al Sistema de Verificación de Calificación (QVS, por sus siglas en inglés) como caso de estudio, demostrando sus ventajas. Para validar su efectividad, se comparó con otros métodos documentados en la literatura que también emplean técnicas de NLP para extraer elementos UML de los requisitos de *software* en lenguaje natural. Esta comparación reveló que ninguno de los métodos evaluados era capaz de extraer diagramas de clases, casos de uso y actividades de manera tan efectiva como el enfoque propuesto en este trabajo. Los resultados mostraron claramente que los diagramas producidos por este enfoque fueron significativamente superiores.

2.12.4. Generación automática de modelos de procesos de negocio a partir de historias de usuario

En el ámbito del desarrollo de *software,* ha surgido un creciente interés en la automatización de la generación de modelos de procesos de negocio a partir de requisitos textuales. En este contexto, Nasiri et al., (2023) propusieron un enfoque que se centró en la extracción automatizada y generación de modelos de procesos de negocio utilizando técnicas de NLP y el marco MDA. Su objetivo fue reducir el esfuerzo manual necesario para derivar diagramas de actividad a partir de historias de usuario y criterios de aceptación, con el fin de mejorar la eficiencia y precisión en el desarrollo de *software* ágil.

La implementación del enfoque se realizó en Python, para el preprocesamiento de historias de usuario y criterios de aceptación, se utilizó Stanford CoreNLP, que facilitó la correferencia, tokenización, análisis de partes del discurso y análisis de dependencias. Las reglas de extracción de artefactos se definieron en Prolog, un lenguaje de programación lógica, para detectar y clasificar actividades, condiciones y acciones en el diagrama de actividades. Estas reglas se integraron con Python utilizando la API PySwip, lo que permitió la comunicación bidireccional con SWI-Prolog.

El proceso de extracción de artefactos siguió un flujo sistemático. En primer lugar, se aplicó la correferencia para reemplazar pronombres en los criterios de aceptación y asegurar la coherencia en el texto. Posteriormente, los criterios de aceptación se dividieron en sus partes "dado", "cuándo" y "entonces" para identificar condiciones, acciones y actividades, respectivamente. El análisis de partes del discurso y el análisis de dependencias proporcionaron relaciones gramaticales entre las palabras de una oración, lo que facilitó la definición de reglas de extracción de artefactos. Estas reglas fueron aplicadas de manera consistente a las tres partes del formato "dado/cuándo/entonces" para extraer las actividades y acciones necesarias. Una vez que los componentes del diagrama de actividades fueron extraídos, se agruparon automáticamente en un archivo de texto interpretado por la API PlantUML. Esta API, implementada en Python, generaba una imagen visual del diagrama de actividades UML que representaba el modelo de proceso de negocio derivado de las historias de usuario y criterios de aceptación (véase Figura 2.8).

De esta manera, este enfoque contribuyó significativamente al desarrollo ágil de *software* al minimizar el esfuerzo manual requerido para derivar modelos de procesos de negocio a partir de requisitos textuales. Mejoró la precisión y la eficiencia del proceso de desarrollo al integrar de manera efectiva los requisitos del usuario con el diseño del sistema, facilitando así la comunicación y comprensión entre los desarrolladores y las partes interesadas. Para evaluar el desempeño del enfoque propuesto, se utilizaron dos casos de estudio con diferentes criterios de aceptación. Para aumentar la cantidad de pruebas, se modificaron estos criterios alterando las partes de aceptación para evaluar diversas estructuras de oraciones. Los resultados se evaluaron mediante la construcción manual de un diagrama de actividad para cada archivo que contenía la historia de usuario con varios criterios de aceptación, comparando luego estos diagramas manuales con los generados automáticamente.

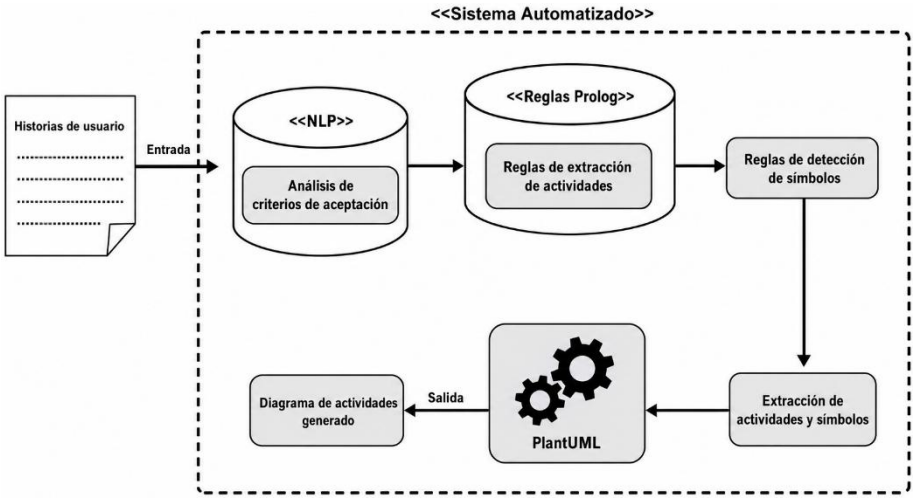


Figura 10 Arquitectura del enfoque propuesto por Nasiri et al., (2023).

Los resultados de la evaluación del enfoque propuesto por Nasiri et al. (2023) para la extracción automatizada de actividades mostraron que se identificaron un total de 50 actividades en el conjunto de datos de prueba. En comparación, el enfoque manual identificó 48 actividades, lo que sugiere que el enfoque propuesto pudo identificar dos actividades adicionales. En términos de rendimiento, el enfoque automatizado demostró un alto nivel. Alcanzó una tasa de aciertos positivos del 97%, indicando que identificó correctamente el 97% de las actividades necesarias. La precisión fue del 94%, lo que significa que el 94% de las actividades identificadas fueron correctas. Además, la exactitud fue del 94%, lo que indica que el 94% de todas las actividades identificadas fueron correctas según el enfoque propuesto. Estos resultados muestran que el enfoque automatizado no solo mejoró la eficiencia al identificar más actividades en comparación con el enfoque manual, sino que también mantuvo un alto nivel de precisión y exactitud en la identificación de las actividades requeridas.

2.12.5. Generador automatizado de diagramas de casos de uso utilizando NLP y ML

El enfoque propuesto por Dias et al., (2023) se centró en desarrollar un método automatizado para generar diagramas de casos de uso a partir de historias de usuario utilizando técnicas avanzadas de NLP y ML. Este enfoque buscó optimizar y acelerar el proceso de diseño del ciclo de vida del desarrollo de *software*, específicamente en la fase de análisis y diseño, reduciendo el tiempo y esfuerzo requerido para la creación manual de diagramas de casos de uso. La Figura 2.9 muestra el proceso dividido en varias fases, las cuales se definen a continuación.

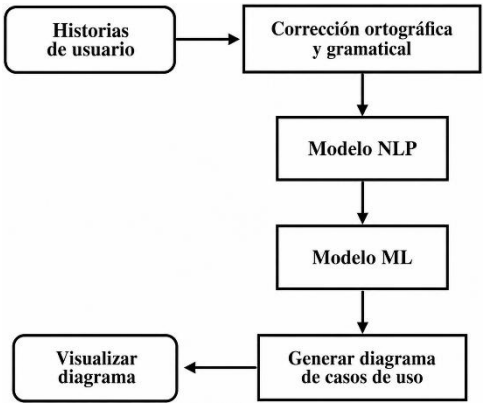


Figura 11 Modelo del enfoque propuesto por Dias et al., (2023).

1. **Corrección ortográfica y gramatical:** En esta fase se aplicó un modelo de Corrección de Errores Gramaticales (GEC, por sus siglas en inglés) para corregir tanto la ortografía como la gramática de las historias de usuario. Este modelo garantizó que la entrada estuviera libre de errores gramaticales y coherente desde el punto de vista lingüístico. La corrección fue crucial, ya que cualquier error gramatical podría haber llevado a predicciones inexactas en las etapas subsiguientes del proceso de generación de diagramas de casos de uso.
2. **Modelo NLP:** Tras la realización de la corrección de errores gramaticales, se aplicó un modelo de NLP para identificar actores y casos de uso a partir de las historias de usuario. El proceso de NLP comenzó con la segmentación de oraciones, donde las historias de usuario se dividieron en oraciones individuales. Posteriormente, se procedió con la tokenización y el Etiquetado de Partes del Discurso (POS, por sus siglas en inglés) para cada palabra en las oraciones. El etiquetado POS permitió identificar si una palabra era un sustantivo, verbo, adjetivo, etc. Posteriormente, se llevó a cabo el análisis de dependencias para comprender las relaciones gramaticales entre las palabras en las oraciones. Este análisis generó una estructura de datos que mostraba cómo las palabras de una oración se relacionaban entre sí, lo cual resultaba fundamental para identificar los roles de los actores y las acciones representadas en las historias de usuario.
3. **Modelo ML:** Después de identificar las relaciones gramaticales, se empleó un modelo de ML, específicamente Bayes Ingenuo, para filtrar y eliminar los casos de uso redundantes o incorrectamente identificados por el modelo de NLP. Bayes Ingenuo es un algoritmo de clasificación supervisado simple pero efectivo que se utilizó en este contexto para mejorar la precisión del sistema final. El modelo de ML fue entrenado con más de 700 casos de uso extraídos del modelo de NLP, junto con su clasificación como correctos o incorrectos.
4. **Generar diagrama de casos de uso:** Finalmente, utilizando la herramienta PlantUML, se utilizaron los actores y casos de uso identificados para generar automáticamente el diagrama de casos de uso, visualizando los componentes de acuerdo con la notación UML estándar. Este proceso aseguró que los diagramas generados reflejaran de manera precisa y completa los requisitos y las interacciones descritas en las historias de usuario, facilitando así una mejor comprensión y comunicación entre los *stakeholders* y los desarrolladores de *software*. Este enfoque automatizado no solo mejoró la eficiencia del proceso de generación de diagramas de casos de uso, sino que también redujo significativamente el riesgo de errores humanos y garantizó la coherencia en la interpretación de los requisitos del sistema, contribuyendo así a un desarrollo de *software* más efectivo y preciso.

El enfoque propuesto para la generación automatizada de diagramas de casos de uso a partir de historias de usuario utilizando técnicas de NLP y ML demostró resultados prometedores en la evaluación realizada. En primer lugar, el modelo de GEC utilizado logró corregir la mayoría de los errores gramaticales y ortográficos en las historias de usuario, aunque se identificaron algunas limitaciones en la corrección de errores ortográficos específicos. Esto permitió que el modelo de NLP procesara las historias de usuario gramaticalmente correctas de manera efectiva.

En la fase de NLP, el modelo logró identificar correctamente el 86.67% de los actores y el 71.43% de los casos de uso, utilizando un conjunto de 8 historias de usuario como prueba. Estos resultados indicaron una capacidad significativa para extraer y comprender los elementos esenciales de las historias de usuario, como actores, verbos y objetos, mediante técnicas de tokenización, etiquetado POS y análisis de dependencias. Además, el modelo de ML, implementado con el algoritmo Bayes Ingenuo, alcanzó una precisión del 75.52% en la clasificación y filtrado de casos de

uso identificados por el modelo de NLP. Este enfoque contribuyó a mejorar la precisión general del proceso y a reducir la inclusión de casos de uso innecesarios en el diagrama final.

Finalmente, el diagrama de casos de uso generado visualmente utilizando PlantUML reflejó de manera precisa y eficiente las relaciones identificadas entre actores y casos de uso en las historias de usuario. Aunque se identificaron algunos errores menores que requirieron edición manual posterior, el enfoque automatizado proporcionó una base sólida para optimizar el proceso de diseño del ciclo de vida del desarrollo de *software*. Esto no solo mejoró la eficiencia del proceso, sino que también redujo los errores humanos asociados con la creación manual de diagramas de casos de uso, permitiendo a los equipos de desarrollo enfocarse en tareas más estratégicas y creativas.

2.12.6. Evaluación de la IA generativa en tareas de modelado: Un informe de experiencia con ChatGPT y UML

La evaluación llevada a cabo por Cámara et al., (2023) tuvo como objetivo explorar y evaluar el uso de LLM, específicamente ChatGPT, en el contexto del modelado de *software*. ChatGPT, desarrollado por OpenAI, es un modelo de lenguaje basado en *transformers* entrenado con una vasta cantidad de datos para generar respuestas humanas en conversaciones. Aunque inicialmente fue diseñado para tareas de NLP, su aplicación se ha expandido hacia el modelado y otras áreas de la Ingeniería de *Software*. La versión específica de ChatGPT utilizada no fue mencionada.

El estudio evaluó la capacidad de ChatGPT para generar diagramas de clases UML a partir de descripciones en lenguaje natural, explorando tanto las posibilidades como las limitaciones de esta tecnología emergente. Los autores estructuraron la evaluación en dos fases principales: una exploratoria inicial y una segunda fase enfocada y sistemática. A continuación, se dará una breve explicación de cada una:

1. **Exploración detallada del comportamiento de ChatGPT:** En la primera fase de los experimentos, los autores realizaron una exploración detallada del comportamiento de ChatGPT en el modelado de *software*. Iniciaron con interacciones individuales con ChatGPT, usando distintos *prompts* para solicitar la creación de modelos de *software* variados en tamaño y complejidad. Todas estas interacciones y descubrimientos fueron documentados en un diario compartido, permitiendo así un registro minucioso de las capacidades y limitaciones observadas. Durante esta fase, se identificaron varios hallazgos clave:
    - ChatGPT funcionaba de manera más efectiva cuando los nombres de las entidades a modelar eran significativos y representativos del dominio del problema.
    - Los modelos generados por ChatGPT mostraron ser más precisos y completos cuando el modelo tenía un conocimiento semántico profundo del dominio, destacando una relación directa entre la familiaridad de ChatGPT con el tema y la calidad del modelo generado.
    - Se encontró que el dominio del problema influía en la estructura y contenido de los modelos generados, con variaciones significativas en su nivel de abstracción dependiendo del contexto específico.
    - ChatGPT enfrentaba dificultades al generar modelos de clase con más de ocho a diez clases desde cero, pero mostró mejoras cuando se le proporcionó un modelo inicial pequeño para refinar progresivamente.

Un ejemplo ilustrativo de esta fase incluyó la solicitud a ChatGPT de generar un diagrama de clases UML para un sistema de videoclub. Utilizando un *prompt* específico (véase Figura 2.10), la

calidad de las respuestas mejoró notablemente cuando se proporcionaron descripciones detalladas y significativas de cada entidad. Sin embargo, ChatGPT mostró dificultades al manejar más de diez clases simultáneamente, requiriendo ajustes iterativos para alcanzar un modelo preciso.

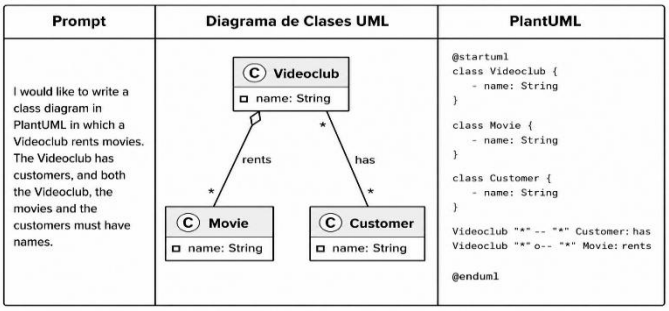


Figura 12 *Prompt* utilizado en ChatGPT que generar un diagrama de clases (Cámara et al., 2023).

2. **Experimentos más enfocados y rigurosos**

   En la segunda fase, los autores utilizaron los conocimientos adquiridos para llevar a cabo experimentos más enfocados y rigurosos. Desarrollaron un conjunto de cuarenta modelos de intención UML, diseñados para explorar conceptos y mecanismos específicos de modelado en una variedad de dominios y estructuras. Entre los principales hallazgos de esta segunda fase se encontró que:

   - ChatGPT era capaz de capturar adecuadamente asociaciones y herencias, aunque con variabilidad en su precisión y completitud dependiendo del dominio específico.
   - Los resultados de ChatGPT eran bastante aleatorios y no deterministas, lo que presentaba un desafío significativo para la reproducibilidad de los experimentos.
   - Se observaron errores semánticos comunes en los modelos generados por ChatGPT, a pesar de su precisión sintáctica general.
   - Desarrollar modelos precisos con ChatGPT requería un proceso iterativo y un diálogo continuo.

En la evaluación de ChatGPT para el modelado de diagramas UML, se observaron resultados mixtos en términos de precisión y completitud. Aunque demostró una capacidad razonable para identificar clases, atributos y métodos, hubo casos en los que omitió atributos importantes o no identificó correctamente las relaciones entre clases. La especificación de relaciones, como las de asociación, agregación o composición, fue inconsistente. Se identificaron varios desafíos y limitaciones significativas:

- Dificultades para manejar descripciones complejas y detalladas.
- La coherencia del modelo fue problemática, con modelos que a veces incluían redundancias o inconsistencias.
- Necesidad de mejoras en la capacidad de ChatGPT para interpretar y generar modelos que cumplan con las convenciones estándar de UML.

A pesar de las limitaciones observadas, el estudio sugiere que ChatGPT podría ser útil en escenarios donde se requiere una generación rápida de modelos preliminares. En particular, su capacidad para interpretar descripciones en lenguaje natural y generar diagramas de clases UML de forma automática ofrece una herramienta valiosa para ingenieros de *software* que buscan prototipar y visualizar rápidamente los conceptos iniciales de sus proyectos. Esta utilidad se extiende a actividades

como la enseñanza del modelado de *software*, la creación de documentación preliminar y la facilitación de discusiones iniciales en equipos de desarrollo.

Sin embargo, para que ChatGPT se convierta en una herramienta práctica y confiable en el modelado de *software*, es necesario realizar más investigación y desarrollo en varias áreas clave. Primero, es esencial mejorar la precisión de ChatGPT en la generación de modelos UML, especialmente en la identificación y representación de relaciones complejas como asociaciones, agregaciones y composiciones. Esto podría implicar el desarrollo de algoritmos más avanzados de NLP y técnicas de aprendizaje profundo que manejen de manera más efectiva las descripciones detalladas y complejas. Además, es necesario abordar la inconsistencia y variabilidad en los resultados generados por ChatGPT. La naturaleza no determinista del modelo plantea desafíos significativos para la reproducibilidad y coherencia de los diagramas UML. Soluciones potenciales incluyen la implementación de mecanismos para mantener el contexto de la conversación de manera más consistente y la integración de técnicas de verificación y validación para asegurar que los modelos generados cumplan con las convenciones estándar de UML. También se debe considerar el desarrollo de interfaces y herramientas que faciliten la interacción iterativa y el refinamiento de los modelos generados por ChatGPT. Esto permitiría a los usuarios ajustar y corregir los diagramas de manera más eficiente, garantizando que los modelos finales sean precisos y útiles. La integración de ChatGPT con entornos de desarrollo integrados y otras herramientas de Ingeniería de *Software* (SE, por sus siglas en inglés) podría proporcionar un flujo de trabajo más fluido y productivo para los desarrolladores.

2.12.7. Transformación de historias de usuario y requisitos funcionales en diagramas UML de casos de uso mediante NLP y BERT

El estudio realizado por Molla et al., (2024) tuvo como objetivo comparar la transformación automática de historias de usuario y requisitos funcionales en diagramas UML de casos de uso utilizando técnicas de NLP y aprendizaje automático. La propuesta se enfocó en automatizar la identificación de actores, casos de uso y relaciones a partir de requisitos escritos en lenguaje natural, buscando reducir el esfuerzo manual requerido durante la fase de diseño de software. Para ello, los autores utilizaron un enfoque basado en el modelo BERT, el cual fue entrenado para extraer los elementos principales del diagrama UML.

El estudio planteó una metodología basada en varias etapas de procesamiento textual y aprendizaje automático. Inicialmente, las historias de usuario y requisitos funcionales fueron preprocesados mediante técnicas de NLP como tokenización, *stemming* y etiquetado gramatical. Posteriormente, el modelo BERT fue utilizado para generar representaciones contextualizadas del texto y realizar tareas de reconocimiento de entidades, permitiendo identificar actores, casos de uso y relaciones entre ellos (véase Figura 2.11). Finalmente, los elementos extraídos fueron transformados automáticamente en diagramas UML de casos de uso utilizando PlantUML.

Para evaluar la propuesta, los autores utilizaron un conjunto de once historias de usuario y once requisitos funcionales obtenidos de proyectos académicos. La evaluación se realizó mediante métricas de exactitud y precisión, comparando los elementos identificados automáticamente contra los elementos reales presentes en los diagramas de referencia. Los resultados mostraron valores de hasta 88% tanto en exactitud como en precisión, observándose un mejor desempeño cuando se utilizaron requisitos funcionales en comparación con historias de usuario.

A pesar de los resultados obtenidos, los autores identificaron diversas limitaciones en su propuesta. Entre ellas, destacaron las dificultades para identificar relaciones complejas dentro de los diagramas UML, especialmente relaciones de generalización, inclusión y extensión. Asimismo, el estudio se enfocó únicamente en diagramas de casos de uso y no contempló otros tipos de diagramas

UML, como diagramas de clases o actividades. Además, aunque el enfoque utilizó técnicas de aprendizaje automático basadas en BERT, no incorporó modelos generativos de lenguaje de gran escala (LLMs) ni técnicas RAG que permitieran mejorar la comprensión semántica y contextual de las historias de usuario.

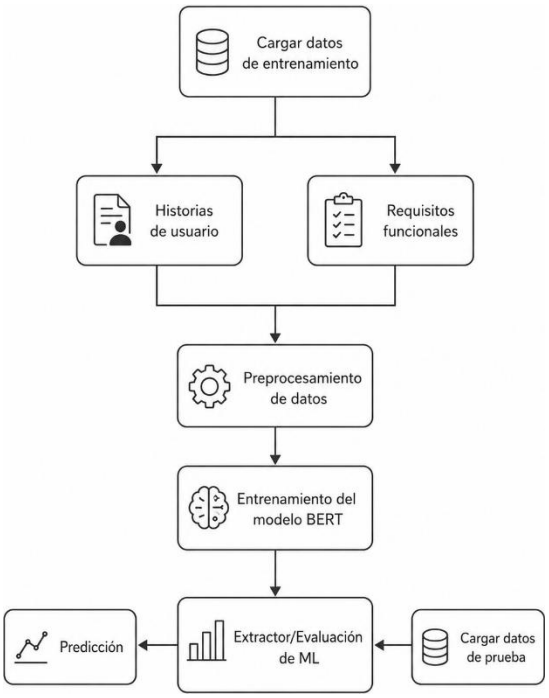


Figura 13 Flujo del enfoque basado en BERT propuesto por Molla et al., (2024).

2.12.8. Generación de diagramas de clases UML a partir de historias de usuario mediante LLMs

Ojha et al., (2025) exploraron el uso de LLMs para automatizar la generación de diagramas de clases UML a partir de historias de usuario. Los autores señalaron que la elaboración manual de este tipo de diagramas constituye una actividad costosa y propensa a errores debido a la ambigüedad y la incompletitud presentes en los requisitos escritos en lenguaje natural. En este contexto, propusieron un enfoque orientado a aprovechar las capacidades de comprensión contextual de los LLMs para identificar entidades, atributos y relaciones descritos en las historias de usuario y transformarlos en diagramas de clases UML. En particular, emplearon ChatGPT, mediante la OpenAI API, para interpretar los requisitos expresados en lenguaje natural y generar automáticamente la sintaxis PlantUML correspondiente.

La propuesta fue evaluada utilizando el conjunto de datos Dalpiaz2020, conformado por historias de usuario y sus correspondientes diagramas de clases UML validados en distintos dominios de aplicación. Como parte del proceso, las historias de usuario fueron sometidas a un preprocesamiento básico para reducir ruido lingüístico y posteriormente fueron proporcionadas a ChatGPT para generar la sintaxis PlantUML correspondiente a los diagramas de clases UML. Posteriormente, dicha sintaxis fue utilizada para producir automáticamente los diagramas, los cuales fueron comparados con los diagramas de referencia disponibles en el conjunto de datos (véase Figura 2.12).

Los autores destacaron que el uso de LLMs ofrece ventajas significativas respecto a los enfoques tradicionales basados en reglas, debido a su capacidad para interpretar requisitos ambiguos, inferir información implícita y adaptarse a diferentes dominios sin necesidad de configuraciones específicas. Asimismo, señalaron que los LLMs facilitan la generación automática de modelos más flexibles y

contextualmente coherentes, favoreciendo la automatización de actividades relacionadas con el modelado de *software*.

No obstante, los autores reconocen que la investigación constituye un primer acercamiento hacia la automatización de diagramas de clases mediante LLMs. Entre las principales limitaciones identificadas se encuentran la ausencia de una evaluación exhaustiva de la precisión de los diagramas generados, la dependencia de un único modelo de lenguaje y la falta de mecanismos adicionales para enriquecer el contexto de entrada. Además, el estudio se restringió exclusivamente a diagramas de clases UML, sin considerar otros tipos de diagramas utilizados durante el análisis y diseño de *software*.

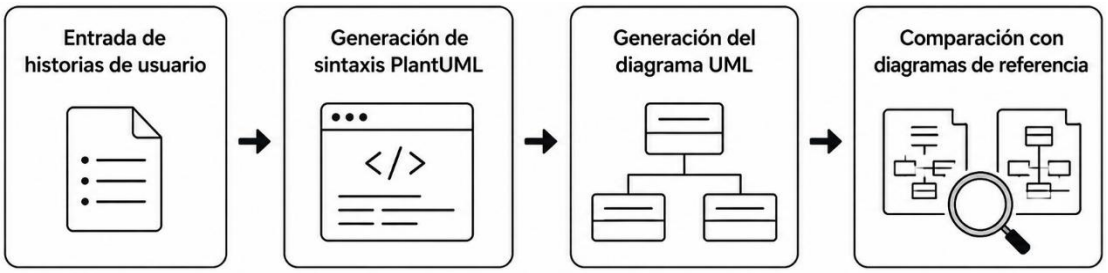


Figura 14 Flujo de generación de diagramas de clases UML mediante ChatGPT utilizado por Ojha et al., (2025).

2.12.9. Visualización de historias de usuario hacia diagramas UML de casos de uso mediante NLP y reglas lógicas

La investigación desarrollada por Mornie et al., (2026) tuvo como propósito proponer un enfoque semiautomatizado para visualizar diagramas UML de casos de uso a partir de historias de usuario estructuradas. Los autores señalaron que la construcción manual de diagramas UML continúa siendo un proceso complejo, costoso y propenso a errores debido a la ambigüedad inherente del lenguaje natural. Además, identificaron que varios estudios previos no lograban representar adecuadamente las relaciones dentro de los diagramas de casos de uso.

La propuesta se desarrolló en cuatro etapas principales: recopilación de requisitos, procesamiento de lenguaje natural, aplicación de reglas lógicas y generación del diagrama UML. Para el procesamiento textual, los autores utilizaron Stanford CoreNLP junto con diversas técnicas de NLP, incluyendo tokenización, *stemming*, lematización, etiquetado gramatical y análisis de dependencias (véase Figura 2.13). Posteriormente, se aplicaron reglas lógicas específicas para identificar actores, casos de uso y relaciones dentro de las historias de usuario. Finalmente, los elementos obtenidos fueron utilizados para generar automáticamente diagramas UML mediante PlantUML.

Como parte del estudio, los autores realizaron una revisión sistemática de investigaciones relacionadas con la generación automática de modelos UML, identificando problemas frecuentes asociados al modelado manual, tales como el alto consumo de tiempo, la complejidad del lenguaje natural, la ambigüedad semántica y la posibilidad de omitir información importante durante el análisis de historias de usuario. Asimismo, destacaron que los procesos manuales suelen ser propensos a errores debido al volumen de requisitos que deben analizarse durante el desarrollo de *software*.

Aunque la propuesta permitió automatizar parcialmente la generación de diagramas UML de casos de uso, los autores reconocieron varias limitaciones. El estudio se restringió únicamente a diagramas de casos de uso y no abordó otros diagramas UML, como diagramas de clases o actividades. Además, el enfoque dependía principalmente de reglas heurísticas y técnicas tradicionales de NLP, sin considerar LLMs o mecanismos de recuperación contextual como RAG, los cuales podrían mejorar significativamente la interpretación semántica de las historias de usuario y la calidad de los diagramas generados.

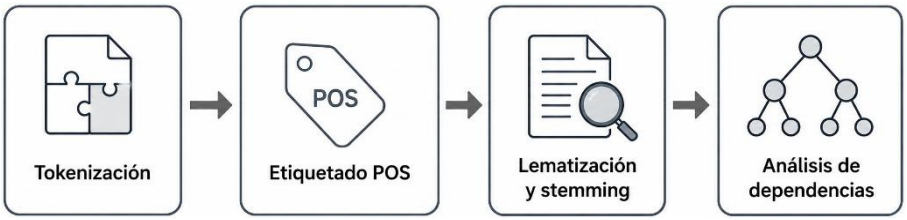


Figura 15 Etapas del NLP utilizado por Mornie et al., (2026).

2.13. Conclusión del estado del arte

El análisis del estado del arte en el contexto del modelado de *software* y las metodologías ágiles, como Scrum, revela una evolución significativa en la automatización y precisión del diseño de *software*. Los estudios revisados demuestran que la integración de técnicas avanzadas como el NLP, ML y los entornos visuales ha permitido abordar diversas facetas del modelado de *software* de manera más eficiente y efectiva.

Diversas investigaciones han explorado la generación automática de diagramas UML a partir de descripciones en lenguaje natural, destacando tanto las oportunidades como los desafíos inherentes a este enfoque. Por ejemplo, se ha demostrado que es posible transformar historias de usuario en diagramas de secuencia y diagramas de clases, lo cual agiliza el proceso de diseño y asegura una documentación más precisa de los requisitos. Sin embargo, persisten desafíos en la generación de modelos complejos y la consistencia de los resultados, lo que subraya la necesidad de un desarrollo continuo en estas áreas.

La implementación de entornos visuales y metodologías integradas ha facilitado la adopción de Scrum y otras prácticas ágiles, mejorando la colaboración y la claridad en la gestión de proyectos. Herramientas que combinan UML con generación automática de casos de prueba han mostrado ser eficaces para garantizar una cobertura de pruebas exhaustiva y alineada con las historias de usuario. Estos avances destacan la importancia de un análisis lingüístico preciso y el uso de reglas heurísticas para mejorar la calidad y precisión de los modelos generados.

Además, la evaluación de tecnologías emergentes como ChatGPT en la generación de modelos UML ha revelado su potencial para la creación rápida de prototipos y la facilitación de discusiones iniciales en equipos de desarrollo. No obstante, se identificaron limitaciones significativas en la precisión y consistencia de los modelos generados, así como en la capacidad para manejar descripciones complejas y detalladas. En conjunto, estos estudios indican una tendencia clara hacia la automatización y mejora del modelado de *software* utilizando técnicas avanzadas de IA. La capacidad de generar modelos precisos y completos de manera automatizada tiene el potencial de transformar la Ingeniería de *Software*, aunque se requiere un esfuerzo continuo para superar las limitaciones actuales y asegurar la reproducibilidad y coherencia de los resultados.

3. Diseño de la propuesta de solución

En este capítulo se presentan los elementos principales utilizados en el desarrollo de la propuesta planteada en este trabajo de tesis, tales como el proceso de selección del conjunto de prácticas que serán integradas en la Planeación del *Sprint* y la Ejecución del *Sprint*, así como la planificación de la implementación de la herramienta para la generación automática de diagramas UML.

Para el diseño del conjunto de prácticas y el desarrollo de la herramienta para la generación automática de diagramas UML a partir de historias de usuario, se utilizó el Marco de Investigación de Sistemas de Información (ISRF, por sus siglas en inglés) propuesto por Hevner et al. (2004), el cual fue empleado como guía metodológica para estructurar y validar la investigación. Este marco se compone de tres elementos principales: el entorno, el diseño/investigación y la base de conocimiento, los cuales se relacionan a través de tres ciclos: relevancia, diseño y rigor (véase Figura 3.1). A continuación, se describen los elementos que conforman el ISRF en el contexto de este trabajo de tesis.

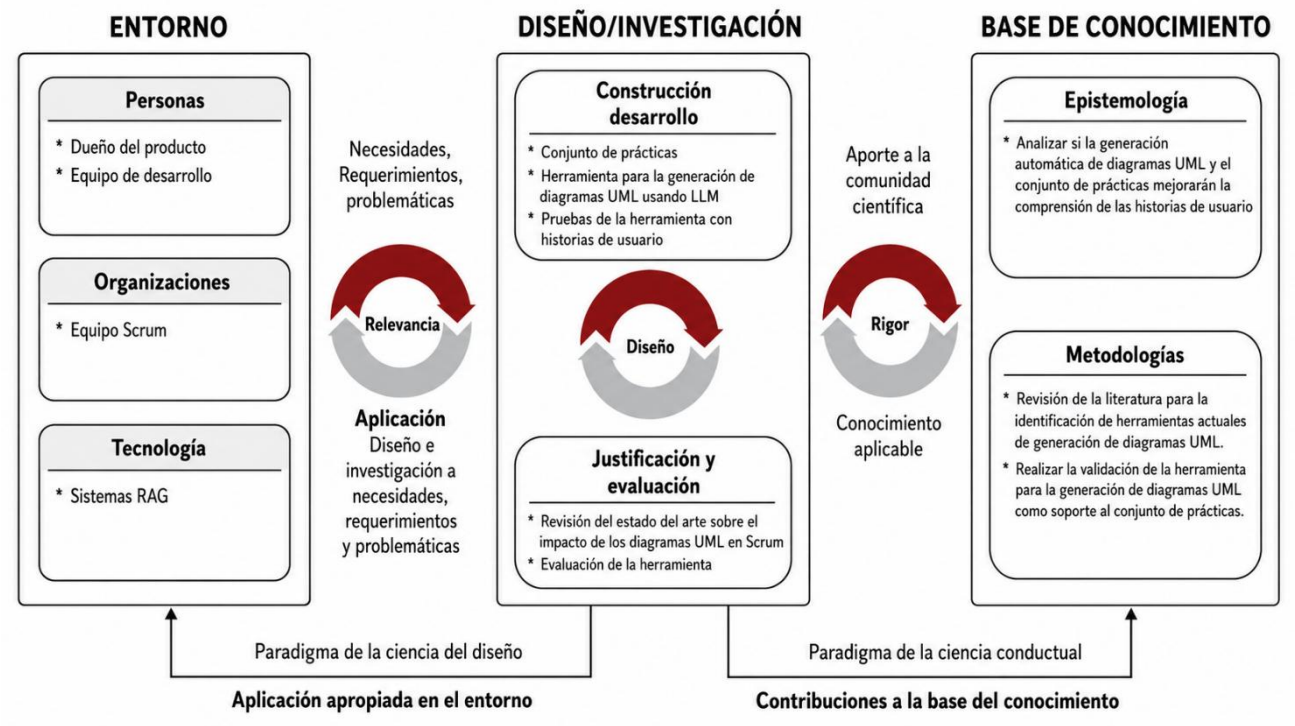


Figura 16 Ajuste del ISRF propuesto por Hevner et al., (2004).

3.1. Entorno

De acuerdo con Raharjana et al., (2021) las historias de usuario son un artefacto utilizado en el desarrollo de *software* ágil para expresar un requisito de *software*; sin embargo, al ser escritas en lenguaje natural pueden presentar inconvenientes como ambigüedad, inconsistencia e incompletitud. Como consecuencia, el equipo de desarrollo no logra comprender correctamente las necesidades de los *stakeholders*.

En este trabajo de tesis se propone la integración de un conjunto de prácticas en los eventos de Planeación del *Sprint* y Ejecución del *Sprint*, así como la implementación de una herramienta para la generación automática de diagramas de clases inicial y diagramas de actividades inicial con base en el conjunto de prácticas definidas, con el fin de mejorar la comprensión de las historias de usuario por parte del dueño del producto y del equipo de desarrollo. En este sentido, se definieron tres elementos fundamentales para alcanzar este objetivo:

- **Personas:** Los roles clave corresponden a los miembros del equipo Scrum, específicamente el dueño del producto y el equipo de desarrollo, quienes aplican el conjunto de prácticas ágiles e interactúan con la herramienta propuesta para la generación de diagramas UML.
- **Organizaciones:** El marco organizacional incluye al equipo Scrum, el cual adopta el conjunto de prácticas ágiles propuestas con el propósito de mejorar la comunicación y la comprensión de las historias de usuario mediante los diagramas UML generados.
- **Tecnología:** El sistema RAG se utilizó para la generación automática de los diagramas UML. Asimismo, en este apartado se incluyen las historias de usuario como el elemento fundamental para la generación de dichos diagramas.

Como parte del entorno tecnológico del proyecto, se diseñó una comparativa entre LLMs representativos, tanto propietarios como de código abierto, con el fin de analizar su desempeño en el contexto de un sistema RAG orientado a la generación de diagramas de clases y de actividades UML. Esta evaluación permitió observar cómo diferentes arquitecturas y niveles de acceso influyen en la calidad de los artefactos generados cuando se incorpora contexto relevante al proceso de generación de los diagramas UML.

3.2. Diseño/Investigación

En esta sección se hace mención de la importancia del uso de las prácticas ágiles, para mejorar la comunicación del equipo Scrum y la comprensión de las historias de usuario. Por lo tanto, se detalla el proceso del diseño del conjunto de prácticas para su integración en la metodología Scrum, así como la planificación de la implementación de la herramienta para la generación automática de diagramas UML.

3.2.1. Construcción

Pressman y Maxim (2020) destacaron que, en cualquier enfoque de desarrollo de *software*, la adopción de buenas prácticas es esencial para garantizar tanto la calidad como la sostenibilidad a largo plazo del producto final. Aunque sus propuestas se originan en metodologías más tradicionales como cascada y V, muchos de los principios que mencionan, tales como la iteración, la mejora continua y la revisión frecuente del progreso, son aplicables y han sido adaptados eficazmente dentro de los marcos ágiles. En el contexto del desarrollo ágil de *software*, estos principios se transforman para maximizar la flexibilidad, la colaboración entre los equipos y la capacidad de adaptarse de manera

rápida y efectiva a los cambios en las historias de usuario, que representan los pilares fundamentales de las metodologías ágiles (Suaza et al., 2015).

La adecuada adopción y aplicación de las prácticas ágiles es clave para alcanzar un buen resultado en los proyectos de *software*, independientemente del tipo de *software* que se esté desarrollando. Tal como señalaron Wang et al. (2012), la correcta adopción de las prácticas ágiles no ocurre de forma automática, requiere un enfoque disciplinado y una adaptación constante a las necesidades del equipo y del entorno, lo que permite integrarlas a la cultura organizacional y así maximizar su impacto positivo.

De acuerdo con Pikkarainen et al. (2008), la buena comunicación dentro de los equipos de desarrollo de *software* es fundamental para el éxito de los proyectos. En este sentido, las prácticas ágiles están específicamente diseñadas para mejorar la comunicación dentro del equipo, como la colaboración con los *stakeholders*, lo que influye de manera directa en la comprensión y correcta implementación de las historias de usuario.

Cabe destacar que en un entorno global la integración de prácticas ágiles presenta desafíos adicionales, los cuales deben gestionarse para asegurar la eficacia del proceso de desarrollo. Jalali y Wohlin (2012) argumentaron que, a medida que los equipos de desarrollo se distribuyen geográficamente, la implementación de prácticas ágiles como reuniones diarias, revisiones retrospectivas y entregas incrementales se vuelve más compleja. Sin embargo, estos desafíos no son insuperables, solo requieren de una gestión adecuada, por lo que las metodologías ágiles resultan ser efectivas en entornos globales (Vallon et al., 2018).

La adopción adecuada de las prácticas ágiles contribuye significativamente a mejorar la claridad y precisión en la definición de las historias de usuario. Al hacerlo, se fomenta una mejor comunicación y colaboración dentro del equipo, lo que resulta en un desarrollo de *software* alineado con las expectativas de los *stakeholders* (Sampietro, 2016). Así, estas prácticas no solo facilitan el desarrollo técnico, sino que también fortalecen la capacidad de los equipos para responder a los cambios y cumplir con las demandas de un entorno de trabajo dinámico y en constante evolución (Diebold y Dahlem, 2014).

Las historias de usuario en un entorno ágil, son una herramienta de comunicación que combina lo mejor de los medios escritos y verbales. En este sentido, estas permiten registrar información de manera clara y estructurada, a la vez que fomentan la interacción dinámica y la retroalimentación inmediata (Menzinsky et al., 2018). Por ello, una adopción adecuada de estas prácticas ágiles es esencial para garantizar que los equipos comprendan y definan correctamente las historias de usuario, lo que mejora la comunicación y la colaboración interna. Tal como señalan Diebold y Dahlem (2014), las prácticas ágiles son directrices fundamentales que fortalecen la capacidad de los equipos para adaptarse al cambio y promover la mejora continua en el proceso de desarrollo.

3.2.1.1. Proceso de diseño del conjunto de prácticas para su integración en Scrum

A partir de lo expuesto, se describe el proceso desarrollado para definir las prácticas ágiles que integran el conjunto de prácticas, las cuales serán integradas en los eventos de Planeación del *Sprint* y Ejecución del *Sprint*. Este proceso incluyó un análisis detallado de las necesidades del equipo Scrum y las características de las metodologías ágiles, con el objetivo de seleccionar y proponer las prácticas que mejor contribuyan a mejorar la comprensión de las historias de usuario y a fortalecer la colaboración entre los miembros del equipo Scrum.

El propósito de implementar estas prácticas es promover una cultura de transparencia y confianza en el equipo Scrum, facilitando su adaptación a los cambios y desafíos propios del desarrollo ágil de *software*. Alineado con los objetivos de este trabajo de investigación, se espera que, mediante el uso de la herramienta de generación automática de diagramas UML, estas prácticas

contribuyan a una mejor comprensión de las historias de usuario y a una comunicación más fluida entre los miembros del equipo Scrum. Además, se espera que la incorporación de estas prácticas impulse un desempeño más eficiente del equipo, lo que, a su vez, contribuya a mejorar la calidad del producto final de *software*. Sin embargo, dichos resultados deberán ser evaluados y validados al término del desarrollo de la herramienta.

Al realizar la integración del conjunto de prácticas y la herramienta de generación de diagramas UML en el proceso de desarrollo ágil, además de mejorar la comprensión de las historias de usuario, también se busca promover un entorno de trabajo donde la comunicación sea clara y los malentendidos sean mínimos. De esta manera, el equipo Scrum puede enfocarse en la entrega de valor al cliente, garantizando que las historias de usuario sean entendidas e implementadas con una mayor precisión y eficiencia.

El desarrollo de *software* en entornos ágiles, particularmente dentro del marco de Scrum, requiere un enfoque meticuloso y estratégico para garantizar que los equipos comprendan y comuniquen eficazmente las historias de usuario. Estas historias no son solo un conjunto de requisitos, son la base que guía el trabajo del equipo y, por lo tanto, su comprensión clara y compartida es vital para el éxito de un proyecto.

Con este objetivo en mente, se llevó a cabo el proceso para seleccionar el conjunto de prácticas, las cuales se encuentren alineadas con los principios ágiles, y promuevan una colaboración efectiva y una comunicación fluida entre los miembros del equipo Scrum. A continuación, se describen las etapas fundamentales que integran este proceso:

1. Revisión bibliográfica sobre las prácticas ágiles

Las metodologías ágiles han surgido como una respuesta a la necesidad de flexibilidad, colaboración y adaptación en proyectos de desarrollo de *software*. Dentro de estas metodologías, las prácticas ágiles juegan un papel fundamental, ya que representan las acciones y enfoques específicos que los equipos adoptan para alcanzar los principios ágiles establecidos en el Manifiesto Ágil. La presente revisión bibliográfica tiene como objetivo identificar y analizar las prácticas documentadas en la literatura que contribuyen a mejorar la comprensión de las historias de usuario, la comunicación y la colaboración dentro de los equipos, con el fin de integrarlas de manera efectiva en los eventos de Planeación del *Sprint* y Ejecución del *Sprint* en Scrum.

El primer paso de la revisión bibliográfica consistió en la definición clara y precisa de los objetivos de la revisión que se pretende realizar. El propósito principal fue identificar las prácticas ágiles que han sido estudiadas en la literatura y cómo se integran en las metodologías ágiles, específicamente en el contexto de Scrum. Este enfoque permitió establecer una base para la selección de estudios relevantes y guiar el análisis posterior.

Esta revisión no solo incluyó artículos académicos de revistas especializadas, sino también investigaciones recientes y casos de estudio que documentaron la experiencia de diversos equipos en la implementación de Scrum en diferentes contextos (Kussunga y Ribeiro, 2019; Maatuk y Abdelnabi, 2021; Nasiri et al., 2020; Nasiri et al., 2023). A través de esta revisión, se identificaron enfoques exitosos que varios equipos han utilizado para mejorar el proceso de desarrollo de *software* (Dada y Sanusi, 2022; Diebold y Dahlem, 2014; Jalali y Wohlin, 2012; Pikkarainen et al., 2008; Sampietro, 2016; Wang et al., 2012).

Con una sólida base teórica establecida, el siguiente paso consistió en explorar el estado del arte en relación con el uso de diagramas UML en el contexto del desarrollo ágil bajo el marco de Scrum. Esta fase implicó investigar cómo diferentes equipos han integrado estas

herramientas visuales en sus prácticas diarias, así como los beneficios y desafíos que han encontrado en el camino. Al comprender el papel de los diagramas UML, se hizo evidente que estas representaciones gráficas no solo ayudan a clarificar conceptos complejos y abstractos, sino que también fomentan una mejor alineación y entendimiento entre los diferentes roles de Scrum (Alberto et al., 2020; Hema et al., 2020; Pressman y Maxim, 2020).

2. Criterios para la identificación de las prácticas relevantes

La identificación de las prácticas relevantes en las metodologías ágiles es fundamental para comprender cómo estas contribuyen a la mejora de la comunicación, la colaboración y la comprensión de las historias de usuario dentro del equipo de trabajo Scrum. Los criterios utilizados para la identificación de las prácticas relevantes incluyeron:

- **Impacto en la comunicación:** Se identificaron las prácticas que facilitan la transmisión efectiva de información entre los miembros del equipo Scrum.
- **Mejora en la colaboración:** Se seleccionaron aquellas prácticas que favorecen la colaboración entre los miembros del equipo Scrum.
- **Aplicación en los eventos de Scrum:** Se buscaron prácticas que se implementan directamente en los eventos del marco de trabajo de Scrum, como en la Planeación del *Sprint* y Ejecución del *Sprint*.
- **Relevancia en la mejora continua:** Se incluyeron prácticas que contribuyen a la mejora continua dentro del proceso de desarrollo ágil, como la retroalimentación y cambios constantes durante el desarrollo del producto de *software*.

3. Prácticas identificadas en la literatura por fase en Scrum

El proceso de desarrollo ágil se encuentra organizado en varias fases y en cada una de ellas incluye prácticas específicas que contribuyen a la eficiencia y efectividad del desarrollo del producto de *software*. A continuación, se presentan las principales prácticas identificadas en la literatura, organizadas por fase, específicamente en el marco de trabajo Scrum (Alberto et al., 2020; Hema et al., 2020; Sommerville, 2020).

Planeación del producto (*Product Backlog*)

Esta fase es crucial, ya que en ella se realiza la creación y priorización del *Product Backlog*, una lista organizada de todos los requisitos, mejoras y correcciones que se desean implementar en el producto de *software*. Las prácticas identificadas en esta fase son:

- **Definición clara de las historias de usuario:** Es esencial que las historias de usuario sean claras, comprensibles y bien definidas. Esto asegura que el equipo de desarrollo tenga una comprensión más clara de lo que se espera lograr en cada funcionalidad del producto.
- **Priorización basada en valor:** La priorización de las historias de usuario debe basarse en el valor que aportan al cliente o al usuario final. Esta práctica permite al equipo enfocarse en las funcionalidades más importantes y maximizar el valor entregado al cliente.

Planeación del *Sprint* (*Sprint Planning*)

En esta fase, al inicio de cada *Sprint*, se realiza una reunión de planificación en la que el equipo de desarrollo selecciona los elementos del *Product Backlog*, los cuales serán trabajados durante el *Sprint* e integrados al *Sprint backlog*, y establece los objetivos del *Sprint*:

- **Definición de objetivos claros:** Los objetivos del *Sprint* deben ser específicos y alcanzables. Esta práctica ayuda a que el equipo tenga una visión clara de lo que se espera lograr durante el *Sprint*.
- **Estimación en equipo:** La estimación de tareas en equipo permite una distribución más equitativa del trabajo y asegura que todos los miembros del equipo comprendan el esfuerzo necesario para completar el trabajo comprometido en el *Sprint*.
- **División de tareas en pequeñas unidades:** Las tareas deben dividirse en unidades pequeñas y manejables, lo que facilita la ejecución diaria y la gestión del progreso durante el *Sprint*.

Ejecución del *Sprint*

En esta fase, el equipo de desarrollo trabaja en las historias de usuario seleccionadas en la fase de Planeación del *Sprint*. El trabajo se realiza de forma colaborativa, con reuniones diarias de seguimiento, donde el equipo discute el progreso, los impedimentos y planifica las actividades del día. Las prácticas relevantes durante la Ejecución del *Sprint* incluyen:

- **Integración continua:** La integración continua permite que las modificaciones realizadas por los desarrolladores sean integradas en el sistema de manera frecuente y fluida, lo que reduce los conflictos y mejora la calidad del código.
- **Revisión de código:** La revisión de código es una práctica importante para asegurar que el código cumpla con los estándares de calidad, además de identificar posibles errores o áreas de mejora antes de que se integren al producto final.
- **Reuniones diarias (*Daily* Scrum):** Estas reuniones permiten al equipo revisar el progreso y detectar posibles obstáculos de manera oportuna, asegurando que se mantenga un ritmo constante durante el *Sprint*.

Revisión del *Sprint*

Esta fase se lleva a cabo al final de cada *Sprint*, donde el equipo de desarrollo presenta el Incremento del producto, es decir, las funcionalidades completadas durante la Ejecución del *Sprint*. También se obtiene la retroalimentación de las partes interesadas para realizar los ajustes al *Product Backlog* en caso de ser necesario. Las prácticas clave de esta fase incluyen:

- **Demostración del incremento funcional del producto:** La presentación de un incremento funcional es una práctica crucial para mostrar el progreso y verificar que los entregables cumplen con los requisitos del cliente.
- **Retroalimentación con las partes interesadas para mejorar el producto:** La retroalimentación obtenida durante la revisión permite ajustar las funcionalidades y mejorar el producto según las expectativas del cliente.

Retrospectiva del *Sprint* (*Sprint Retrospective*)

Después de la fase de revisión del *Sprint*, el equipo Scrum realiza una retrospectiva del *Sprint*, donde se reflexiona sobre lo que salió bien, lo que no y cómo se puede mejorar en el siguiente *Sprint*. Las prácticas esenciales en esta fase incluyen:

- **Crear un ambiente seguro y colaborativo:** Para fomentar una comunicación abierta y constructiva, es necesario un ambiente de confianza donde los miembros del equipo puedan expresar sus opiniones y sugerencias sin temor a represalias.

- **Enfoque en la mejora continua:** La retrospectiva debe centrarse en identificar áreas de mejora y establecer acciones concretas para optimizar el proceso de trabajo en el próximo *Sprint*.
- **Realizar compromisos en equipo:** El equipo debe comprometerse a implementar los cambios necesarios y mejorar continuamente en el proceso de desarrollo del *software*.
- **Documentar discusiones y acuerdos:** Es crucial documentar las discusiones y acuerdos alcanzados durante la retrospectiva para garantizar el seguimiento de las acciones en el próximo *Sprint*.

4. Prácticas identificadas en la literatura por rol en Scrum

En el marco de las metodologías ágiles, Scrum define roles específicos que son esenciales para el éxito del proyecto. Cada uno de estos roles tienen responsabilidades y prácticas clave que, cuando se aplican correctamente, aseguran la eficiencia del equipo y la calidad del producto final. A continuación, se presentan las buenas prácticas identificadas en la literatura para cada uno de los roles en Scrum:

Dueño del Producto

El Dueño del Producto tiene la responsabilidad de maximizar el valor del producto y realizar la gestión del *Product Backlog*. Las prácticas esenciales de este rol incluyen:

- **Mantenimiento del *Product Backlog* refinado:** Es importante que el Dueño del Producto mantenga el *Product Backlog* actualizado y refinado, de modo que esté siempre listo para ser trabajado por el equipo.
- **Colaboración constante con los *stakeholders*:** El Dueño del Producto debe mantener una comunicación fluida con los *stakeholders* para comprender sus necesidades, recibir retroalimentación y ajustar el *Product Backlog* en consecuencia. Este enfoque asegura que el producto final sea lo que el cliente realmente necesita.

Scrum Master

El Scrum Master tiene el objetivo de que el equipo siga el proceso Scrum correctamente y de eliminar cualquier impedimento que pueda bloquear el progreso del equipo. También es responsable de facilitar la comunicación y colaborar en la mejora continua del equipo. Las prácticas esenciales de este rol incluyen:

- **Facilitación efectiva de los eventos Scrum:** El Scrum Master debe garantizar que todas las reuniones Scrum (*Daily Scrum*, *Sprint Planning*, *Sprint* Review y *Sprint* Retrospective) se realicen de manera eficiente, con un propósito claro y respetando los tiempos establecidos.
- **Fomentar la transparencia:** El Scrum Master debe fomentar la transparencia en todo el proceso. Esto incluye mantener tableros visibles, gráficos de progreso y otros recursos que ayuden a todos los miembros del equipo y a los *stakeholders* a mantenerse informados sobre el estado del proyecto.
- **Proactividad en la eliminación de bloqueos:** El *Scrum Master* debe prestar atención a cualquier impedimento que pueda afectar al equipo, ya sea técnico, organizacional o de comunicación. La eliminación de estos obstáculos de manera proactiva asegura que el equipo pueda avanzar sin interrupciones.

Equipo de Desarrollo

El Equipo de Desarrollo es el encargado de convertir las historias de usuario en incrementos del producto funcionales. Para desempeñar su trabajo de manera eficiente, deben adoptar prácticas que promuevan la colaboración, la calidad del código y la mejora continua. Las prácticas esenciales de este rol incluyen:

- **Trabajo colaborativo:** El equipo de desarrollo debe fomentar un entorno de colaboración, donde todos los miembros compartan conocimientos y habilidades.
- **Revisiones de código:** Las revisiones de código son fundamentales para mantener la calidad del *software* y compartir el conocimiento técnico dentro del equipo.
- **Definición clara de "Hecho":** El equipo debe tener una definición clara de qué significa que una historia de usuario esté "hecha", que debe incluir pruebas, documentación y cualquier otro criterio de calidad relevante.

5. Identificación de prácticas potenciales

Una vez finalizada la identificación de prácticas existentes en la literatura por fase y por rol en Scrum, se dio inicio a la creación de un listado de prácticas potenciales a ser seleccionadas como parte del conjunto de prácticas a integrar en la Planeación del *Sprint* y Ejecución del *Sprint*. En esta fase, se buscó identificar prácticas que, aunque no se mencionan explícitamente en la literatura revisada, pueden deducirse a partir de las experiencias y enfoques exitosos descritos por otros equipos en contextos similares encontrados en el estado del arte. Estas prácticas tienen el potencial de enriquecer el marco de trabajo propuesto en esta tesis.

El listado se elaboró considerando las ideas y conceptos extraídos de la literatura revisada, priorizando aquellas prácticas que se alinean con los objetivos de este trabajo de investigación, especialmente en lo que respecta a mejorar la comprensión de las historias de usuario y optimizar la comunicación dentro del equipo Scrum. Se consideraron diferentes metodologías y enfoques que han demostrado ser efectivos en el ámbito del desarrollo ágil, buscando prácticas que complementen y refuercen el trabajo del equipo.

Un aspecto clave en la generación de este listado de prácticas fue la consideración de una herramienta que permite la generación automática de diagramas UML de clases iniciales y de actividades. Esta herramienta no solo busca facilitar la creación de representaciones gráficas, sino que también respalda y potencia las prácticas seleccionadas, garantizando que estén alineadas con el objetivo de mejorar la comprensión de las historias de usuario y optimizar la comunicación dentro del equipo Scrum.

6. Identificación de la fase y el rol involucrado en las prácticas identificadas

Después de haber elaborado el listado de prácticas potenciales y haberlas priorizado, el siguiente paso fue identificar los roles del equipo Scrum que participarán en la implementación de cada una de ellas. En esta fase, se realizó un análisis detallado de las responsabilidades y habilidades de los distintos miembros del equipo, teniendo en cuenta su participación en las diferentes fases del ciclo de trabajo de Scrum, como la Planeación del Producto, la Planeación del *Sprint* y la Ejecución del *Sprint*.

Este proceso fue esencial para garantizar que las prácticas se distribuyeran de manera adecuada y que se ajustaran tanto al rol del PO como al DT, quienes desempeñan funciones clave dentro del marco Scrum. Al realizar este análisis, se aseguró que las prácticas no solo fueran viables en términos operativos, sino que también se alinearan con la dinámica

colaborativa del equipo, mejorando el flujo de trabajo y minimizando posibles dificultades en su adopción.

Además, esta etapa fue crucial para anticipar cualquier inconveniente que pudiera surgir durante la implementación de las prácticas, garantizando así que cada práctica priorizada fuera apropiada para los roles involucrados y las fases correspondientes. De este modo, se estableció una distribución clara y coherente de las responsabilidades, lo que facilita la integración efectiva de las prácticas dentro del equipo Scrum y contribuye al cumplimiento de los objetivos de la herramienta de generación automática de diagramas UML.

7. Priorización de las prácticas potenciales

Tras la elaboración del listado de prácticas potenciales y la identificación de la fase y el rol involucrado, se llevó a cabo un proceso de priorización que clasificó las prácticas en tres categorías: alta, media y baja. Esta clasificación resultó crucial para enfocar los esfuerzos en aquellas prácticas que son más relevantes y están alineadas con los objetivos de la herramienta destinada a la generación automática de diagramas UML. Las prácticas de alta prioridad se identificaron como esenciales, ya que su integración asegura que el equipo pueda generar diagramas de manera efectiva en etapas tempranas, como la planificación del Sprint, cumpliendo así con el propósito de integrar la herramienta propuesta.

En contraste, las prácticas de prioridad media incluyen funcionalidades que complementan las de alta prioridad, brindando un soporte adicional. Finalmente, las prácticas de baja prioridad fueron catalogadas como no esenciales para las etapas iniciales de integración del conjunto de prácticas, quedando reservadas para futuras consideraciones.

Esta estrategia de priorización optimiza el uso de recursos, incluida la herramienta, y dirige la atención hacia las prácticas que aportan el mayor valor en el proceso de desarrollo. Al centrar los esfuerzos en estas áreas, se maximiza el impacto positivo en el flujo de trabajo del equipo y se garantiza que las decisiones tomadas contribuyan efectivamente al avance del proyecto.

3.2.1.2. Planificación para la implementación de la herramienta

Asimismo, se detalla la planificación para la implementación de la herramienta para la generación automática de diagramas UML. El objetivo principal de esta herramienta es servir como soporte para el conjunto de prácticas diseñadas para su integración en Scrum.

Para la implementación de la herramienta se optó por utilizar la metodología Scrum, tomando en cuenta las numerosas ventajas que esta metodología ofrece en términos de flexibilidad, colaboración y adaptabilidad a los cambios. Scrum facilita la entrega incremental y la mejora continua, lo cual es esencial para el desarrollo ágil de la herramienta, permitiendo iterar y ajustar el producto conforme se obtienen retroalimentaciones a lo largo del proceso de los *Sprints* (Alberto et al., 2020; Estrada-Velasco et al., 2021; Hema et al., 2020).

De acuerdo con lo mencionado, se adopta Scrum, adaptado a las necesidades específicas de este proyecto. Aunque Scrum es un marco de trabajo bien estructurado que define roles, eventos, artefactos y *Sprints*, en este caso se ha optado por simplificar y ajustar estos elementos con el fin de alinearlos con los recursos disponibles y el alcance de este trabajo de investigación. De este modo, se preserva el enfoque ágil, iterativo y colaborativo, con la flexibilidad necesaria para asegurar que los objetivos del proyecto se alcancen de manera eficiente.

La tarea principal de la herramienta es generar diagramas de clases iniciales y diagramas de actividades iniciales a partir de historias de usuario previamente definidas. Durante su desarrollo, se

empleará un enfoque iterativo que permitirá la incorporación gradual de funcionalidades, así como la realización de ajustes conforme se avanza en cada *Sprint*. Esta planificación integra las prácticas de Scrum con adaptaciones específicas, lo que garantiza que las funcionalidades se desarrollen en función con los objetivos planteados para el proyecto.

Para adaptar el marco de trabajo Scrum a las necesidades de este proyecto de investigación y optimizar los recursos disponibles, se han modificado ciertos aspectos de su estructura tradicional. En lugar de seguir estrictamente la organización clásica de roles, artefactos y eventos, se ha optado por una simplificación que prioriza la eficiencia del proceso y la adaptación continua a los resultados obtenidos en cada *Sprint*. Esta adaptación implica que algunos roles y eventos serán omitidos. Además, los roles se asignan de manera estratégica, considerando los recursos humanos disponibles y las tareas específicas de cada fase del ciclo de desarrollo de *software* de Scrum. El propósito de esta adaptación es maximizar la efectividad del proceso de desarrollo, asegurando que la herramienta cumpla con los requisitos establecidos y satisfaga las necesidades del usuario final.

1. **Definición de roles:** En el contexto del desarrollo de la herramienta para la generación automática de diagramas UML, se han definido dos únicos roles, adaptados específicamente a las necesidades del proyecto y a la optimización de los recursos humanos disponibles. Dichos roles son los siguientes:
   - **Dueño del Producto:** El director de Tesis asume este rol, siendo el responsable de proporcionar las historias de usuario y garantizar que los entregables de cada *Sprint* estén alineados con los objetivos del proyecto. Además de supervisar el progreso de la herramienta y tomar decisiones sobre las funcionalidades que deben desarrollarse en cada *Sprint*, asegurándose de que el desarrollo esté alineado con las necesidades del proyecto.
   - **Equipo de Desarrollo:** En este trabajo de investigación, el equipo de desarrollo está compuesto únicamente por la tesista, quien se encarga de implementar las historias de usuario proporcionadas por el dueño del producto en la planeación de cada *Sprint*.

2. **Artefactos:** Para la implementación de la herramienta para la generación automática de diagramas UML, se hará uso de los tres artefactos Scrum: el *Product Backlog*, el *Sprint backlog* y el Incremento del producto.
   - ***Product Backlog*:** Contiene las historias de usuario priorizadas, que se actualizan conforme se ajustan los requisitos durante el desarrollo de la herramienta.
   - ***Sprint backlog*:** Incluye las historias de usuario seleccionadas del *Product Backlog* para el *Sprint* actual, gestionadas por el equipo de desarrollo para cumplir con los objetivos del *Sprint*.
   - **Incremento:** Al final de cada *Sprint*, se genera un incremento funcional de la herramienta, que es evaluado para asegurar su calidad y cumplimiento de los requisitos.

3. **Eventos:** En el desarrollo de la herramienta para la generación automática de diagramas UML, los eventos del marco de trabajo Scrum se adaptan para garantizar una planificación eficiente, un seguimiento continuo del progreso y una mejora constante del proceso. Estos eventos son esenciales para promover la colaboración, identificar problemas de manera temprana y asegurar que el proyecto se mantenga alineado con los objetivos. Se emplean los eventos de Planeación del *Sprint*, Revisión del *Sprint* y la Retrospectiva del *Sprint*, adaptándose a las necesidades específicas del proyecto y al ciclo iterativo de desarrollo.

4. **Identificación de los *Stakeholders:*** La identificación de *stakeholders* es un paso crucial para garantizar que el desarrollo de la herramienta de generación automática de diagramas UML satisfaga las necesidades y expectativas de todas las partes interesadas. Al reconocer a las personas y roles clave involucrados en el proyecto, se facilita la planificación, ejecución y evaluación de los avances, asegurando una alineación constante con los objetivos definidos.

5. **Clasificación de los *Stakeholders*:** Los *stakeholders* involucrados en este proyecto se dividen en dos categorías principales: internos, que forman parte directa del desarrollo del proyecto, y externos, que interactúan con la herramienta o influyen en su implementación.

   ***Stakeholders* Internos**

   - **Dueño del Producto:** Responsable de definir los requisitos, priorizar el *Product Backlog* y establecer la visión general del proyecto.
   - **Equipo de Desarrollo:** Encargado de implementar las funcionalidades de la herramienta y garantizar su correcto funcionamiento.

   ***Stakeholders* Externos**

   - **Usuarios finales:** Los usuarios finales incluyen al dueño del producto, quien generará los diagramas, y al equipo de desarrollo, encargado de refinarlos. Su retroalimentación es crucial para garantizar que la herramienta sea funcional, fácil de usar y cumpla con los requisitos esperados.
   - **Director de Tesis:** Supervisa el progreso del proyecto, asegura el cumplimiento de los objetivos académicos y proporciona orientación técnica y metodológica para garantizar la calidad del producto final.
   - **Co-director de Tesis:** Apoya al director en la supervisión y evaluación del proyecto, brindando retroalimentación adicional y asegurando el cumplimiento de los estándares académicos y la calidad del desarrollo.

La colaboración activa de los *stakeholders*, tanto internos como externos, es esencial para el éxito en el desarrollo de la herramienta. Su participación continua a lo largo de todas las fases del proyecto, desde la planificación y el diseño hasta la implementación y evaluación, garantiza que el producto final esté alineado con las necesidades y expectativas de los usuarios finales. Mediante la retroalimentación constante y la revisión de los avances, los *stakeholders* contribuyen a identificar áreas de mejora y ajustar los requisitos conforme a los desafíos y avances del proyecto, lo que favorece el desarrollo de una herramienta funcional, accesible y acorde con los objetivos establecidos. Esta colaboración es crucial para asegurar que la herramienta cumpla con su propósito y ofrezca una solución efectiva a los usuarios.

6. **Planificación inicial de los *Sprints:*** La planificación inicial de los *Sprints* estableció una estructura general que guía el desarrollo iterativo e incremental de la herramienta destinada a la generación automática de diagramas UML. Este proceso se fundamentó en los principios del marco de trabajo Scrum, adaptados a las necesidades específicas de este trabajo de investigación. Su propósito principal es garantizar la entrega gradual de funcionalidades clave, permitiendo un desarrollo progresivo que facilite la evaluación constante de los avances y la incorporación de mejoras necesarias. Este enfoque asegura que en cada iteración se generen entregables que aporten valor tangible al proyecto, maximizando el

aprovechamiento de los recursos disponibles y minimizando los riesgos asociados al desarrollo. Asimismo, busca mantener una alineación constante con los objetivos planteados, promoviendo un proceso organizado y eficiente que permita alcanzar las metas del proyecto de manera estructurada y satisfactoria.

Entregables principales

Los *Sprints* están diseñados para abordar las funcionalidades esenciales del proyecto, teniendo como entregables principales:

- **Diagramas de clases iniciales:** Generados automáticamente a partir de las historias de usuario previamente definidas.
- **Diagramas de actividades iniciales:** Representaciones visuales que describen el flujo de trabajo asociado a las historias de usuario.

7. **Estructura general de los *Sprints* para el desarrollo de la herramienta:** La planificación inicial para el desarrollo de la herramienta destinada a la generación automática de diagramas UML se organizó en *Sprints* iterativos, adoptando un enfoque ágil que promovió la flexibilidad y la adaptabilidad en la gestión del proyecto. Estos ciclos de trabajo, delimitados temporalmente, estuvieron diseñados para facilitar un avance progresivo en la implementación de las funcionalidades clave, garantizando la entrega incremental de resultados tangibles y útiles. Este modelo iterativo asegura un progreso constante y estructurado, al tiempo que permite responder de manera oportuna a las necesidades y desafíos que puedan surgir durante el desarrollo. Cada *Sprint* se fundamenta en los roles, eventos y artefactos definidos en el marco de trabajo Scrum, adaptados cuidadosamente a las particularidades y objetivos del proyecto. En esta planificación inicial, se establecieron metas generales para los *Sprints*. Entre las funcionalidades prioritarias se encuentran la generación de diagramas de clases iniciales y diagramas de actividades, elementos esenciales dentro del alcance del proyecto. El Dueño del Producto desempeña un papel fundamental al priorizar las historias de usuario en el *Product Backlog*, garantizando que las tareas seleccionadas para cada *Sprint* estén orientadas a los objetivos generales de este proyecto de tesis. Por su parte, el Equipo de Desarrollo, representado exclusivamente por la tesista, se encargó de implementar estas tareas, gestionando el trabajo de manera autónoma y aplicando principios de comunicación y retroalimentación interna para evaluar y ajustar el progreso según sea necesario. Este enfoque aseguró una alineación constante con los objetivos planteados y fomenta la coherencia en cada etapa del desarrollo. En el contexto del desarrollo ágil, uno de los retos más comunes es la falta de una comprensión clara de las historias de usuario, lo cual puede afectar la comunicación y coordinación del equipo. En particular, en el marco de Scrum, se carece de herramientas automatizadas que apoyen visualmente la Planeación del *Sprint* y Ejecución del *Sprint*. Este proyecto propuso el diseño e implementación de una herramienta capaz de generar diagramas UML automáticamente a partir de historias de usuario, con el objetivo de mejorar la comprensión de las historias de usuario. La implementación de esta herramienta responde a la necesidad de proporcionar un soporte visual automático que ayude al equipo Scrum a interpretar y utilizar las historias de usuario de manera más eficiente. Al generar diagramas de clase iniciales y diagramas de actividades, se busca fortalecer la comunicación entre los roles involucrados, optimizar la planificación y proporcionar una documentación útil durante el desarrollo del proyecto. Esta solución no solo aborda un problema práctico en la industria del *software*, sino que también

representa una contribución académica significativa al incorporar tecnologías de lenguaje natural en el ámbito del desarrollo ágil.

Objetivos

Objetivo general: Diseñar e implementar una herramienta para la generación automática de diagramas UML (clase y actividades) basada en historias de usuario redactadas en lenguaje natural, para apoyar los eventos y artefactos del marco Scrum.

Objetivos específicos:

- Analizar los requisitos funcionales y no funcionales de la herramienta.
- Diseñar un prototipo que integre un LLM para la interpretación de historias de usuario.
- Desarrollar el prototipo de la herramienta utilizando tecnologías adecuadas para la generación de diagramas UML.
- Validar la efectividad de la herramienta mediante un juicio de expertos.

8. **Artefacto: *Product Backlog:*** El *Product Backlog* es una lista estructurada de historias de usuario que define los requisitos para el desarrollo de la herramienta para la generación de diagramas UML. Su elaboración se llevó a cabo mediante reuniones con los *stakeholders* internos, quienes fueron identificados previamente. Durante estas reuniones, se recopilaron y definieron las historias de usuario que reflejan las necesidades y expectativas de los *stakeholders*. Estas historias de usuario describen, de manera clara y concisa, las funcionalidades más relevantes de la herramienta. Cada historia de usuario se estructuró siguiendo la siguiente plantilla estándar: **Como** [tipo de usuario], **quiero** [objetivo o funcionalidad], **para** [beneficio o valor esperado].

   El *Product Backlog* resultante constituye la base del desarrollo, ya que permite priorizar e iterar sobre las funcionalidades más relevantes en función del valor que aportan al equipo Scrum. A medida que la implementación avance conforme a los *Sprints* y se reciba retroalimentación de los *stakeholders*, el *Product Backlog* será refinado y ajustado según las necesidades emergentes. Este proceso iterativo permitirá que la herramienta se adapte a los cambios y mejore su funcionalidad de manera iterativa, garantizando que el producto final cumpla con los objetivos planteados. La elaboración del *Product Backlog* permite establecer una visión preliminar de los requisitos del proyecto en general, proporcionando una línea base para la planificación y desarrollo de la herramienta. La definición precisa de las historias de usuario y su priorización facilitarán la implementación de un producto de *software* alineado con las necesidades del equipo Scrum, asegurando su integración efectiva en el proceso de desarrollo.

3.3. Comparativa de LLMs en un sistema RAG para la generación de diagramas UML

En el marco del desarrollo de la herramienta orientada a la generación automática de diagramas de clases y de actividades UML a partir de historias de usuario, es crucial seleccionar los LLMs más adecuados para realizar esta tarea. Si bien los LLMs presentan capacidades destacadas en generación de texto y razonamiento estructurado, su rendimiento puede variar significativamente en función de la tarea específica, la arquitectura del sistema en que se integran y el tipo de entrada procesada. Por esta razón, se llevó a cabo una evaluación comparativa de distintos LLMs, todos incorporados dentro

de un sistema RAG, con el fin de identificar los tres mejores modelos con mejor desempeño para ser integrados a la herramienta antes mencionada.

Con el objetivo de asegurar una evaluación rigurosa y representativa, se definió un proceso metodológico compuesto por tres etapas principales:

1. Selección y preparación de las historias de usuario.
2. Implementación de un sistema RAG.
3. Evaluación del desempeño de los LLMs seleccionados.

3.3.1. Selección y preparación de las historias de usuario

El conjunto de historias de usuario empleado en este estudio comparativo fue construido a partir de una colección de diagramas de clases y de actividades disponibles en el repositorio *Real World PlantUML*. Este repositorio fue seleccionado debido a la calidad, claridad y diversidad de sus diagramas, así como a su alineación con escenarios del mundo real, lo cual lo convierte en una fuente adecuada para evaluar la capacidad de generalización de los LLMs.

Con el objetivo de garantizar la validez del proceso de evaluación, las historias de usuario fueron redactadas de manera manual en inglés, siguiendo una estructura estandarizada del tipo: *As a [*rol*], I want to [*acción*], so that [*objetivo*]*.

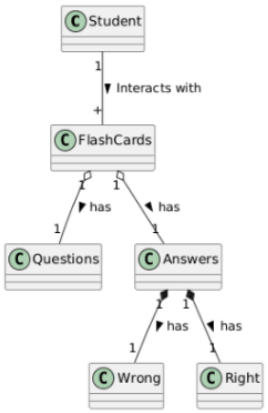
Este formato, ampliamente utilizado en metodologías ágiles, permite una formulación clara de los requisitos funcionales desde la perspectiva del usuario final. La redacción se realizó asegurando una correspondencia semántica directa entre los elementos descritos en los diagramas originales y la información expresada en las historias de usuario, de modo que cada historia de usuario refleja con precisión las entidades, relaciones, acciones y flujos clave del diagrama UML de referencia, ya sea de clases o de actividades.

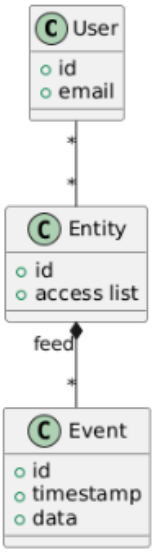
Para este estudio comparativo, las historias de usuario se clasificaron en dos conjuntos, de acuerdo con el tipo de diagrama UML que se buscaba generar: diagramas de clases y diagramas de actividades. Esta clasificación permitió analizar de forma diferenciada el desempeño de los LLMs en la generación de cada tipo de diagrama.

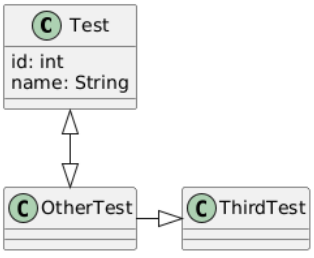
El primer conjunto, compuesto por 68 historias de usuario, se utilizó para la generación de diagramas de clases UML. Estas historias de usuario están orientadas a representar entidades, atributos, relaciones y responsabilidades dentro del sistema. En la Tabla 4 se presentan 10 historias de usuario junto con su diagrama correspondiente en código PlantUML, seleccionadas del conjunto total. Debido a la extensión del material utilizado durante la evaluación, el conjunto completo de historias de usuario empleadas para la generación de diagramas de clases UML se encuentra disponible para su consulta en el repositorio digital del proyecto:

https://github.com/LoreMartinez/Repositorio_Tesis_DiagramasUML_Archivos/blob/main/Archivos_evaluacion_LLMs/HU_generacion_diagramas_clases.pdf

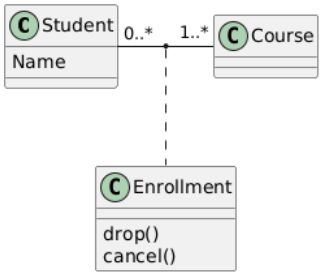
Tabla 4. Historias de usuario para la generación de diagramas de clases.

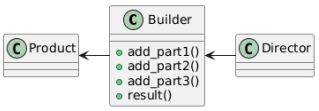
Núm.	Historia de usuario	Código en PlantUML	Diagrama de clases UML
1	***As** a student, **I want** to interact with flashcards containing questions and answers (right and wrong) **so that** I can practice and strengthen my knowledge effectively.*	@startuml class student class flashcard class question class answer student"1" -- "+"flashcard : interact with > flashcard"1" o-- "1"question : has > flashcard"1" o-- "1"answer : has > answer"1" *-- "1"wrong : has > answer"1" *-- "1"right : has > @enduml	

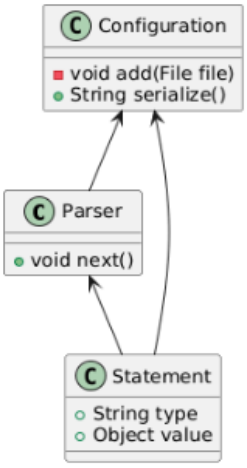
2	***As** a system user, **I want** to access the entities I have permissions for and view their event feed in chronological order, **so that** I can check the relevant information of each entity and stay updated on their activities in real time.*	@startuml
		class User {
		+ id
		+ email
		}
		class Entity {
		+ id
		+ access list
		}
		class Event {
		+ id
		+ timestamp
		+ data
		}
		User "*" -- "*" Entity
		Entity "feed" *-- "*" Event
		@enduml
			

3	***As** a system user, **I want** to manage tests (Test) and their relationships with other tests (OtherTest and ThirdTest), **so that** I can organize and link different types of tests, maintain clear identifiers and names, and access related information in a structured way.*	@startuml class Test Test : id: int Test : name: String class OtherTest Test <|--|> OtherTest OtherTest -|> ThirdTest @enduml	

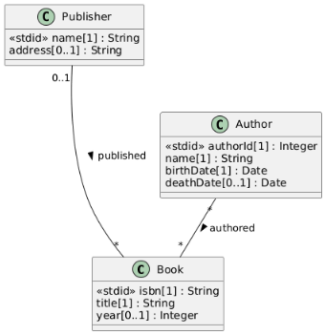
Núm.	Historia de usuario	Código en PlantUML	Diagrama de clases UML

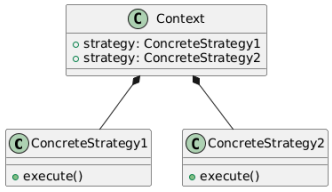
4	***As** a student, **I want** to enroll in courses and manage my enrollments through the application, **so that** I can participate in courses of interest and have the option to cancel or drop my enrollments when needed.*	@startuml
		class Student {
		Name
		}
		Student "0..*" - "1..*" Course
		(Student, Course) . Enrollment
		class Enrollment {
		drop()
		cancel()
		}
		@enduml
			

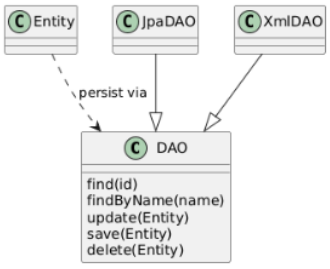
5	***As** a system user, **I want** to build a product step by step using defined methods, **so that** I can generate complete products flexibly and consistently, ensuring that all necessary parts are included before obtaining the final result.*	@startuml
		Builder <--R Director
		Product <--R Builder
		class Builder {
		+ add_part1()
		+ add_part2()
		+ add_part3()
		+ result()
		}
		@enduml
			

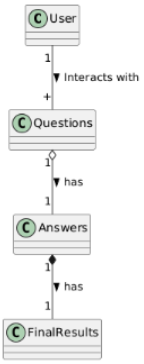
6	***As** a configuration system user, **I want** to load configuration files, process their statements, and export the resulting configuration, **so that** I can manage and apply configurations efficiently and consistently.*	@startuml
		class Parser {
		+ void next()
		}
		class Statement {
		+ String type
		+ Object value
		}
		class Configuration {
		- void add(File file)
		+ String serialize()
		}
		Parser <-- Statement
		Configuration <-- Parser
		Configuration <-- Statement
		@enduml
			

Núm.	Historia de usuario	Código en PlantUML	Diagrama de clases UML

7	***As** a user of the library system, **I want** to register books, authors, and publishers, as well as the relationships between them, **so that** I can maintain a complete catalog, know who wrote each book, and which publisher released it.*	@startuml
		class Book {
		«stdid» isbn[1] : String
		title[1] : String
		year[0..1] : Integer
		}
		class Publisher {
		«stdid» name[1] : String
		address[0..1] : String
		}
		class Author {
		«stdid» authorId[1] : Integer
		name[1] : String
		birthDate[1] : Date
		deathDate[0..1] : Date
		}
		Publisher "0..1" --- "*" Book : published >
		Author "*" -- "*" Book : authored >
		@enduml
			

8	***As** a software developer, **I want** a context to use interchangeable execution strategies, **so that** I can change the system's behavior without modifying its internal structure and keep the code flexible and reusable.*	@startuml
		class ConcreteStrategy1 {
		+execute()
		}
		class ConcreteStrategy2 {
		+execute()
		}
		class Context {
		+strategy: ConcreteStrategy1
		+strategy: ConcreteStrategy2
		}
		Context *-- ConcreteStrategy1
		Context *-- ConcreteStrategy2
		@enduml
			

9	***As** a system developer, **I want** entities (Entity) to be persisted through different data access implementations (DAO, JpaDAO, XmlDAO), **so that** I can store, update, query, and delete information flexibly and consistently.*	@startuml
		class DAO{
		find(id)
		findByName(name)
		update(Entity)
		save(Entity)
		delete(Entity)
		}
		@enduml
			

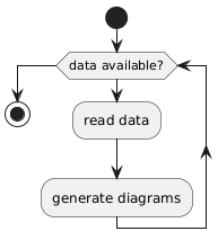
Núm.	Historia de usuario	Código en PlantUML	Diagrama de clases UML
		Entity ..> DAO : "persist via" JpaDAO --|> DAO XmlDAO --|> DAO @enduml	
10	***As** a user of the assessment system, **I want** to interact with questions, answer them, and receive a final result, **so that** I can measure my performance and get feedback on my answers.*	@startuml class User class Questions class Answers class FinalResults User"1" -- "+"Questions : Interacts with > Questions"1" o-- "1"Answers : has > Answers"1" *-- "1"FinalResults : has > @enduml	

El segundo conjunto estuvo compuesto por cincuenta historias de usuario, pero en este caso se utilizó para la generación de diagramas de actividades UML, los cuales capturan el flujo de acciones o procesos asociados a un determinado comportamiento funcional. En la Tabla 5 se presentan 10 historias de usuario seleccionadas del conjunto total de 50 utilizadas para la generación de diagramas de actividades UML. Debido a la extensión del material utilizado durante la investigación, el conjunto completo de historias de usuario puede consultarse en el repositorio digital del proyecto:

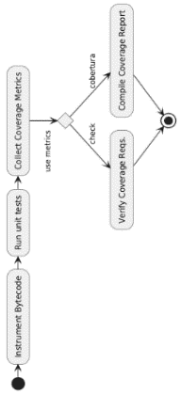
https://github.com/LoreMartinez/Repositorio_Tesis_DiagramasUML_Archivos/blob/main/Archivos_evaluacion_LLMs/HU_generacion_diagramas_actividades.pdf

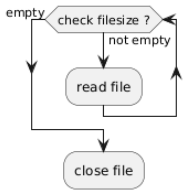
Tabla 5. Historias de usuario para la generación de diagramas de actividades.

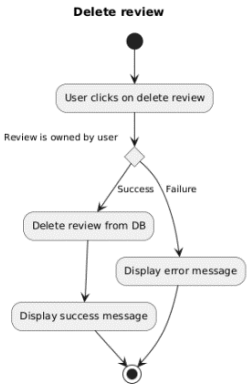
Núm.	Historia de usuario	Código en PlantUML	Diagrama de actividades UML

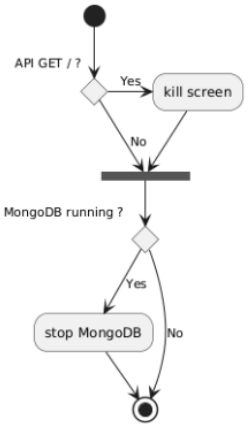
1	***As** a system user **I want** available data to be read and automatically used to generate diagrams **so that** I can clearly and effectively visualize the information as long as there is data to process.*	@startuml
		start
		while (data available?)
		:read data;
		:generate diagrams;
		endwhile
		stop
		@enduml
			

Núm.	Historia de usuario	Código en PlantUML	Diagrama de actividades UML

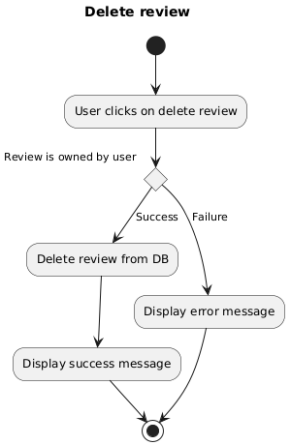
2	***As** a software developer, **I want** to automatically analyze the code coverage whenever the unit tests are run to validate that quality thresholds are met and get a detailed report that helps me identify areas of the code that need more testing.*	@startuml
		(*) -right-> "Instrument Bytecode"
		-right-> "Run unit tests"
		-right-> "Collect Coverage Metrics"
		if "use metrics" then
		--> [check] "Verify Coverage Reqs."
		--> (*)
		else
		--> [cobertura] "Compile Coverage Report"
		--> (*)
		endif
		@enduml
			

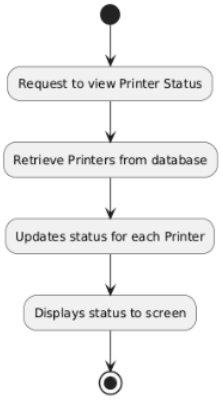
3	***As** a system user **I want** files to be automatically read until no data remains and then properly closed **so that** data is fully processed and system resources are released correctly.*	@startuml while (check filesize ?) is (not empty) :read file; endwhile (empty) :close file; @enduml	

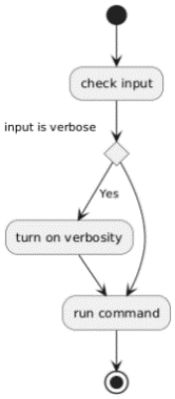
4	***As** a system user **I want** to be able to delete my reviews **so that** I can manage my content and remove information I no longer wish to display.*	@startuml
		title Delete review
		(*) --> "User clicks on delete review"
		--> if "Review is owned by user"
		--> [Success] "Delete review from DB"
		--> "Display success message"
		--> (*)
		else
		--> [Failure] "Display error message"
		--> (*)
		endif
		@enduml
			

5	***As** a system administrator, **I want** the system to check the status of the API and MongoDB, **so that** the corresponding processes are stopped if necessary and services are properly managed.*	@startuml
		(*) --> if "API GET / ?" then
		-right-> [Yes] "kill screen"
		--> ===A===
		else
		--> [No] ===A===
		endif
		if "MongoDB running ?" then
		--> [Yes] "stop MongoDB"
		--> (*)
		else
		--> [No] (*)
		endif
		@enduml
			

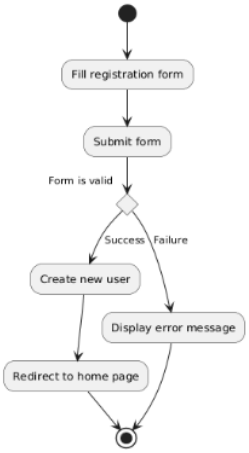
Núm.	Historia de usuario	Código en PlantUML	Diagrama de actividades UML

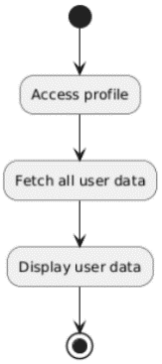
6	***As** a system user **I want** to be able to delete my reviews **so that** I can manage my content and remove information I no longer wish to display.*	@startuml
		title Delete review
		(*) --> "User clicks on delete review"
		--> if "Review is owned by user"
		--> [Success] "Delete review from DB"
		--> "Display success message"
		--> (*)
		else
		--> [Failure] "Display error message"
		--> (*)
		endif
		@enduml
			

7	***As** a system user, **I want** to request the status of printers, **so that** the system retrieves their data, updates their status, and displays it on the screen.*	@startuml (*) --> "Request to view Printer Status" --> Retrieve Printers from database --> Updates status for each Printer -->Displays status to screen -->(*) @enduml	

8	***As** a system user, **I want** the system to detect if the input requires verbosity, **so that** it enables verbose mode before executing the command and provides detailed information if needed.*	@startuml
		(*) --> "check input"
		If "input is verbose" then
		--> [Yes] "turn on verbosity"
		--> "run command"
		else
		--> "run command"
		Endif
		-->(*)
		@enduml
			

Núm.	Historia de usuario	Código en PlantUML	Diagrama de actividades UML

9	***As** a new user, **I want** to fill out and submit the registration form, **so that** I can create an account and be redirected to the home page, or receive an error message if the form is invalid.*	@startuml
		(*) --> "Fill registration form"
		--> "Submit form"
		--> if "Form is valid"
		--> [Success] "Create new user"
		--> "Redirect to home page"
		--> (*)
		else
		--> [Failure] "Display error message"
		--> (*)
		endif
		@enduml
			

10	***As** a system user, **I want** to access my profile, **so that** the system displays all my data clearly and completely.*	@startuml (*) --> "Access profile" --> "Fetch all user data" --> "Display user data" --> (*) @enduml	

Esta distinción facilitó la evaluación de los LLMs en tareas de modelado UML orientadas tanto a la estructura como al comportamiento del sistema. Los diagramas de clases muestran la estructura estática del sistema mediante entidades, atributos y relaciones, mientras que los diagramas de actividades describen el flujo de acciones o procesos. Al emplear historias de usuario con una estructura similar, pero con propósitos distintos, fue posible observar con mayor claridad cómo responden los modelos en cada tipo de diagrama, evaluando la similitud de los diagramas generados con los diagramas de referencia.

3.3.2. Implementación del sistema RAG para la generación de diagramas UML

Con el propósito de evaluar la capacidad de los distintos LLMs para generar diagramas de clases y de actividades UML a partir de historias de usuario, se desarrolló e implementó un sistema RAG. En el contexto de este estudio, el sistema RAG tuvo como propósito proporcionar información adicional mediante un glosario de sustantivos elaborado a partir de las historias de usuario. Para ello, se elaboraron dos glosarios independientes: uno correspondiente a las historias de usuario utilizadas para generar los diagramas de clases y otro para las historias de usuario asociadas a los diagramas de actividades. Esta integración tuvo como finalidad mejorar la precisión, coherencia y alineación semántica de los diagramas generados con respecto a los requisitos funcionales descritos en las historias de usuario. Esta integración busca evaluar si el acceso al contexto recuperado puede influir positivamente en el desempeño de los modelos en tareas de generación de contenido estructurado basado en especificaciones técnicas.

3.3.2.1. Revisión de LLMs actuales disponibles

Durante el periodo en que se llevó a cabo esta investigación, se identificaron diversos LLMs que destacaban por su rendimiento, disponibilidad y grado de adopción en el ámbito académico e industrial. Para sustentar la selección de los modelos evaluados, se realizó una revisión de estudios recientes en el campo del NLP, tales como los trabajos de Kalyan (2024), Yao et al. (2024) y Zhao et al. (2023), que comparan de manera sistemática distintos LLMs en tareas de generación de texto, comprensión semántica y generación de código.

A partir del análisis de trabajos recientes y revisiones comparativas, se identificaron seis modelos de lenguaje particularmente relevantes por su impacto, accesibilidad y diversidad de características técnicas: GPT (OpenAI), Gemini (Google), Claude (Anthropic), LLaMA (Meta), DeepSeek (DeepSeek AI) y Mistral (Mistral AI). Esta revisión abarcó tanto modelos de código abierto como propietarios, lo que permitió considerar un panorama más amplio y representativo del estado actual de los LLMs disponibles durante el periodo en que se realizó el estudio. A continuación, se presentan brevemente las principales características de cada uno:

- **GPT:** Perteneciente a la reconocida familia de modelos basados en la arquitectura *transformers*, GPT se ha posicionado como una de las herramientas más utilizadas en tareas de NLP. A pesar de no ser un modelo de código abierto, su gran capacidad dada por el alto número de parámetros le permite generar respuestas detalladas y coherentes. No obstante, su uso puede verse limitado por sus altos requerimientos computacionales y por su naturaleza propietaria. Aun así, su destacado desempeño y presencia constante en estudios del área respaldan su consideración en este análisis (Neha y Bhati, 2025; Sonoda et al., 2024).
- **Gemini:** Este modelo se distingue por su capacidad para manejar tareas multimodales, ya que es capaz de procesar no solo texto, sino también imágenes y otros formatos de entrada. Gracias a su arquitectura avanzada, Gemini puede gestionar grandes volúmenes de información de forma eficiente. Aunque no se encuentra disponible como código abierto, su versatilidad y alto rendimiento lo posicionan como un referente dentro del panorama actual de modelos de lenguaje (Islam y Ahmed, 2024; Lewandowska y Liebeskind, 2024).
- **Claude:** Diseñado con un enfoque en seguridad y alineación con valores humanos, Claude se caracteriza por ofrecer respuestas claras, coherentes y contextualmente adecuadas. Destaca por su habilidad para mantener la coherencia en interacciones extensas y su capacidad de interpretar instrucciones complejas. Estos atributos lo posicionan como un modelo confiable para tareas que requieren precisión semántica y consistencia discursiva (Minaee et al., 2024; Zhao et al., 2023).
- **LLaMA:** Este modelo de código abierto ha sido ampliamente valorado en entornos académicos y de desarrollo por su eficiencia, adaptabilidad y capacidad de personalización. LLaMA permite ajustes finos para tareas específicas, lo que resulta particularmente útil en proyectos de investigación. A pesar de contar con menos parámetros que algunos modelos propietarios, ha demostrado un rendimiento competitivo, siendo considerado una opción accesible y potente (Liu et al., 2024; Wilbanks et al., 2024; Zhao et al., 2023).
- **DeepSeek:** Introducido en 2023, DeepSeek fue diseñado con la meta de avanzar hacia una Inteligencia Artificial General (AGI). Es un modelo de código abierto optimizado para funcionar en contextos con recursos computacionales limitados. Su diseño modular y escalable lo hace apropiado para múltiples entornos de aplicación, desde sistemas ligeros hasta soluciones en la nube, manteniendo un buen rendimiento en tareas exigentes de NLP (Anthropic, 2024; Zhao et al., 2023).

- **Mistral:** Modelo de reciente desarrollo, Mistral ha destacado rápidamente por su habilidad para generar textos estructurados, comprender contextos complejos y adaptarse a diversas tareas lingüísticas. Al ser de código abierto, ofrece una gran transparencia y flexibilidad, cualidades que lo han vuelto especialmente atractivo en escenarios académicos donde se valora el control sobre la arquitectura y el comportamiento del modelo (Tsai et al., 2024).

Esta revisión permitió identificar un conjunto representativo de LLMs, combinando por un lado LLMs propietarios de alto rendimiento, y por otro, alternativas de código abierto valoradas por su flexibilidad y capacidad de adaptación. La selección se basó no solo en su presencia en la literatura y su disponibilidad al momento de realizar el estudio, sino también en su potencial para abordar tareas de generación de contenido estructurado, como la creación de diagramas UML.

3.3.2.2. Criterios de selección

La selección de los modelos evaluados en este estudio no solo respondió a su popularidad o disponibilidad en el momento de la investigación, sino que se fundamentó en una serie de criterios técnicos y operativos orientados a asegurar la pertinencia y viabilidad del experimento. A continuación, se describen los principales criterios considerados:

1. **Número de parámetros:** El tamaño del modelo, medido en número de parámetros, se consideró como un indicador de su capacidad de generalización y profundidad semántica. Si bien los modelos más grandes tienden a ofrecer mejores resultados en tareas complejas, también exigen una mayor infraestructura computacional. Por tanto, se procuró un equilibrio entre la complejidad del modelo y su eficiencia operativa.
2. **Disponibilidad del código (código abierto vs propietario)**: Se incluyeron tanto modelos de código abierto como LLaMA, DeepSeek y Mistral, que ofrecen ventajas en términos de personalización, gratuidad y transparencia, modelos propietarios como GPT-4, Claude y Gemini, cuya calidad ha sido ampliamente validada, a pesar de sus restricciones de acceso y modificación.
3. **Capacidad de ajuste (*Fine-Tuning*)**: Se valoró la posibilidad de adaptar los modelos a tareas específicas, especialmente en aquellos de código abierto. Esta capacidad permite optimizar su rendimiento en dominios particulares, como el modelado UML, sin requerir un entrenamiento completo desde cero.
4. **Requerimientos computacionales:** Las limitaciones técnicas del entorno de desarrollo también fueron determinantes. Dado que muchos modelos de gran tamaño requieren infraestructura avanzada como múltiples GPUs y amplia memoria, se priorizó el uso de modelos que pudieran ejecutarse de manera eficiente en las condiciones disponibles, ya sea mediante implementación local o acceso vía API.

Estos criterios permitieron seleccionar un conjunto equilibrado de modelos, representativo del estado del arte al momento del estudio, y adaptado tanto a las condiciones prácticas del proyecto como a las necesidades específicas de la tarea de generación automática de diagramas UML a partir de historias de usuario.

La Tabla 6 resume las características técnicas de los modelos revisados. Los modelos propietarios como GPT, Claude y Gemini solo están disponibles mediante API y no permiten ajustes, mientras que LLaMA, DeepSeek y Mistral, al ser de código abierto, permiten personalización y ejecución local con menores requerimientos computacionales. Esta combinación proporciona un panorama equilibrado entre alto rendimiento y flexibilidad de uso.

Tabla 6. Características técnicas de los LLMs seleccionados

Modelo/versión	Número de parámetros	Disponibilidad del código	Capacidad de ajuste	Requerimientos computacionales
GPT-4	~1.7T	✕	✕	Acceso vía API
Claude Sonnet 4	-	✕	✕	Acceso vía API
Claude Opus 4.1	-	✕	✕	Acceso vía API
Gemini 2.5 Pro	-	✕	✕	Acceso vía API
LlaMA3.2	3B	✓	✓	2.0 GB en HD/SSD
DeepSeek-R1	8B	✓	✓	5.2 GB en HD/SSD
Mistral	7B	✓	✓	4.1 GB en HD/SSD

3.3.2.3. Implementación del sistema RAG

Con el propósito de evaluar la capacidad de distintos LLMs para generar diagramas UML a partir de historias de usuario, se diseñó e implementó un sistema automatizado basado en la arquitectura RAG. Esta arquitectura se ha consolidado como una estrategia efectiva en tareas de generación asistida por contexto, ya que permite complementar las capacidades de los modelos generativos con información adicional extraída de documentos relevantes. En esencia, RAG combina técnicas de recuperación semántica de información con la generación de texto, lo que permite a los modelos producir respuestas más precisas, coherentes y contextualizadas, especialmente en dominios técnicos o especializados.

Es importante precisar que el sistema RAG desarrollado en esta investigación opera sobre un corpus cerrado y específico del dominio del problema, conformado principalmente por documentos que contienen un glosario de sustantivos y conceptos derivados de las historias de usuario. A diferencia de enfoques RAG aplicados a grandes bases de conocimiento o repositorios documentales extensos, el mecanismo de recuperación implementado no consulta fuentes externas ni colecciones masivas de documentos. En su lugar, se limita a enriquecer semánticamente la entrada del modelo mediante la recuperación de conceptos previamente definidos y alineados con el modelado UML. Esta delimitación permitió mantener un control preciso sobre el contexto incorporado al proceso generativo, asegurando coherencia temática y reduciendo la variabilidad asociada a corpus abiertos de gran escala.

En el contexto de este trabajo, el sistema RAG fue concebido como una solución modular capaz de recibir una historia de usuario como entrada, recuperar información contextual de modelado UML, y generar un diagrama de clases UML en formato PlantUML que refleje adecuadamente los elementos descritos en la historia. Esta solución no solo permite evaluar comparativamente el desempeño de diferentes LLMs, sino que también proporciona una base funcional para futuras aplicaciones prácticas en la SE asistida por la IA.

A continuación, se detallan los componentes clave del sistema RAG desarrollado, el flujo de ejecución empleado, la construcción del *prompt* utilizado por los modelos, así como las reglas y lineamientos establecidos para asegurar la validez y consistencia de los diagramas generados en sintaxis PlantUML.

- **Arquitectura general del sistema:** La arquitectura del sistema RAG se diseñó siguiendo un enfoque modular y se implementó en el lenguaje de programación Python. Esta estructura modular facilitó la integración de componentes especializados y permitió una ejecución ordenada y eficiente del proceso completo. La arquitectura se dividió en cinco etapas principales, que se desarrollaron de manera secuencial y coordinada para cumplir con el objetivo de generar diagramas UML a partir de historias de usuario. Las etapas incluyeron:

  1. **Carga de documentos:** En esta fase se incorporaron las historias de usuario junto con la documentación contextual relacionada, principalmente en formato PDF. Los documentos fueron procesados para extraer fragmentos semánticamente coherentes que sirvieron como unidad de consulta posterior.
  2. **Indexación semántica:** Los fragmentos extraídos fueron convertidos en vectores mediante técnicas de *embeddings*, y almacenados en un índice vectorial utilizando FAISS, optimizando así la recuperación eficiente de información relevante durante la consulta.
  3. **Recuperación contextual:** Mediante consultas al índice semántico, el sistema recuperó fragmentos contextuales clave que enriquecieron la información disponible para la generación del diagrama. Esta etapa permitió aportar un contexto más preciso y específico para cada historia de usuario.
  4. **Generación del diagrama UML:** Empleando LLMs, se integró la información recuperada con las historias de usuario en un *prompt* cuidadosamente diseñado. Para facilitar esta integración, se utilizó la clase RetrievalQA de la biblioteca LangChain, la cual coordinó la recuperación y generación de texto, asegurando que la entrada al modelo fuera completa y contextualizada.
  5. **Evaluación y renderización:** Los diagramas generados fueron almacenados en archivos con sintaxis PlantUML (.puml y .txt), organizados según modelo, tipo de diagrama e historia de usuario. Posteriormente, se utilizó PlantUML con Java para renderizar los archivos en formato de imagen (.png), lo que permitió tanto una inspección visual como una evaluación cuantitativa mediante la métrica ROUGE, utilizada para comparar los diagramas generados con los diagramas de referencia.

  Para la implementación del sistema RAG, se integraron diversas tecnologías especializadas: LangChain para la coordinación entre recuperación y generación, FAISS para la indexación y búsqueda semántica eficiente, PlantUML para la creación visual de los diagramas, y ROUGE para la evaluación automática de la calidad y similitud de los diagramas generados. Estas herramientas permitieron asegurar un flujo de trabajo coherente y reproducible, maximizando la precisión y validez de los resultados obtenidos.

- **Preparación y segmentación de documentos:** Como parte del proceso de implementación, se diseñó una etapa dedicada a la preparación y segmentación de documentos, con el objetivo de organizar la información contextual relevante para cada historia de usuario. Esta información se obtuvo a partir de documentos en formato PDF, los cuales contenían detalles del dominio del sistema, descripciones funcionales y demás especificaciones del proyecto relacionadas con la historia de usuario en cuestión. Una vez cargado el documento correspondiente, se aplicó un proceso de segmentación semántica mediante un algoritmo recursivo, con el fin de dividir el contenido en fragmentos textuales coherentes. Esta segmentación buscó preservar la estructura lógica del texto, evitando divisiones que interrumpieran el sentido de las ideas, y facilitando así una posterior recuperación contextual

precisa. Cada uno de los fragmentos resultantes fue transformado en una representación vectorial mediante el modelo *all-MiniLM-L6-v2*, lo que permitió capturar el significado semántico del texto en un formato numérico adecuado para su búsqueda. Estas representaciones vectoriales se almacenaron en un índice de similitud semántica construido con la herramienta FAISS, especializada en búsquedas eficientes de vectores de alta dimensión. El índice resultante sirvió como base para la recuperación de contexto, permitiendo acceder de forma rápida y precisa a los fragmentos de texto más relevantes en función de la historia de usuario procesada. De este modo, se buscó garantizar que la información aportada al modelo generativo estuviera alineada con el dominio y los requerimientos específicos de cada caso analizado.

- **Construcción del *prompt* para el LLM:** Con el propósito de estandarizar la generación de diagramas de clases y de actividades en UML, y de garantizar su validez sintáctica bajo la notación PlantUML, se diseñó un *prompt* especializado redactado en inglés. Este *prompt* actuó como interfaz de comunicación entre el sistema y los LLMs, proporcionando directrices claras sobre el formato de salida esperado, el idioma requerido y las restricciones específicas del modelado UML. A continuación, se presentan los *prompts* utilizados para la generación de diagramas de clases (véase Figura 3.2) y diagramas de actividades (véase Figura 3.3), respectivamente:

You are a software analyst.

*Based on the previously provided context and the user story, generate a class diagram at the **analysis level**.*

STRICT INSTRUCTIONS:
- Your output MUST consist ONLY of valid PlantUML code.
- DO NOT include any text, explanation, comment, or tag such as <think>, <note>, <explanation>, etc.
- Output must start with "@startuml" and end with "@enduml", with NOTHING outside this block.

Diagram requirements:
- Focus on user responsibilities, not technical implementation.
- Exclude infrastructure-related classes (controllers, databases, etc.).
- DO NOT include a class named "System" or "Sistema".
*- Use class names, attributes, and methods in **English**, consistent with the user story.*
- Group responsibilities into meaningful and relevant classes.

Output format:
Only output PlantUML code between "@startuml" and "@enduml". Nothing else.

Figura 17 *Prompt* para la generación de diagramas de clases UML.

You are a software analyst.

*Based on the previously provided context and the user story, generate an activity diagram at the **analysis level**.*

STRICT INSTRUCTIONS:

- Your output MUST consist ONLY of valid PlantUML code.

- DO NOT include any text, explanation, comment, or tag such as <think>, <note>, <explanation>, etc.

- Output must start with "@startuml" and end with "@enduml", with NOTHING outside this block.

Diagram requirements:

- Focus on user responsibilities and workflow, not technical implementation.

- Represent each main activity or decision point clearly.

- Use correct PlantUML syntax for activity diagrams: initial node, final node, activities, decisions (if/else), merges, and swimlanes if needed.

- Avoid including elements related to infrastructure, backend, or system internals.

*- All labels and activity names must be in **English**, consistent with the user story.*

Output format:

Only output PlantUML code between "@startuml" and "@enduml". Nothing else.

Figura 18 *Prompt* para la generación de diagramas de actividades UML.

La construcción del *prompt* constituyó una etapa esencial en el proceso de generación automática de diagramas UML. Su diseño se enfocó en orientar la respuesta del modelo para mantener la coherencia, la pertinencia semántica y la calidad sintáctica de los diagramas de clases y de actividades generados en PlantUML. Mediante una estructura cuidadosamente definida, el *prompt* buscó representar con precisión los requisitos descritos en las historias de usuario, reduciendo la inclusión de elementos innecesarios o interpretaciones erróneas que pudieran afectar la calidad de los resultados.

- **Recuperación y generación por LLMs:** El componente encargado de la generación dentro del sistema se basó en la utilización de la clase RetrievalQA proporcionada por la biblioteca LangChain. Esta clase facilitó la integración eficiente entre los procesos de recuperación de información y generación de texto, permitiendo un flujo armonioso entre ambas etapas. En concreto, RetrievalQA posibilitó la consulta directa sobre el vectorstore previamente construido, diseñado para almacenar y gestionar fragmentos textuales que contenían información contextual relevante para cada historia de usuario. Los fragmentos recuperados a partir de este *vectorstore* se emplearon como contexto adicional que acompañó a la historia de usuario durante la consulta al modelo generativo. De esta forma, se enriqueció la entrada proporcionada al modelo de lenguaje, lo que contribuyó a incrementar la precisión y coherencia de las respuestas generadas. La metodología RAG implementada integró efectivamente tres elementos fundamentales: el contexto recuperado, el *prompt* diseñado para la tarea y la propia historia de usuario. Esta combinación conformó la entrada compuesta enviada al LLM seleccionado para la generación del código PlantUML correspondiente. Este enfoque permitió aprovechar el conocimiento tanto almacenado en la base de datos semántica como la capacidad generativa de los LLMs, logrando así una generación asistida que responde

con mayor fidelidad a las especificaciones planteadas en las historias de usuario, buscando mejorar la calidad de los diagramas UML generados.

- **Renderización automática y almacenamiento estructurado:** Los resultados generados por los modelos se almacenaron en archivos con extensiones .puml y .txt, siguiendo una organización jerárquica que consideró criterios como el modelo de lenguaje utilizado, el tipo de diagrama generado (clases o actividades) y el identificador único de la historia de usuario correspondiente. Esta estructura permitió mantener un orden lógico y fácilmente navegable dentro del repositorio de resultados. A partir de estos archivos .puml, se realizó la renderización automática mediante la ejecución de PlantUML a través de una invocación programática con Java, generando imágenes en formato .png que representan visualmente los diagramas UML creados. Este proceso de renderización automatizada facilitó la inspección rápida y efectiva de la calidad tanto estructural como semántica de los diagramas, brindando soporte para la evaluación visual y la detección de posibles inconsistencias o errores en la representación. La integración de este sistema de almacenamiento estructurado y renderización automática posibilitó un manejo sistemático y ordenado de grandes volúmenes de diagramas, favoreciendo procesos de comparación entre modelos, análisis cuantitativo con métricas automáticas y evaluaciones cualitativas manuales. De este modo, se estableció una base sólida para la mejora continua del sistema, facilitando la retroalimentación y refinamiento de los modelos y *prompts* utilizados en la generación automática de diagramas UML.

3.3.3. Evaluación realizada mediante la métrica de ROUGE

Para evaluar la calidad de los diagramas generados automáticamente en este estudio, se utilizaron los diagramas generados por los LLMs seleccionados. Con el propósito de favorecer la transparencia y reproducibilidad de la investigación, dichos diagramas se encuentran disponibles para su consulta en: https://github.com/LoreMartinez/Repositorio_Tesis_DiagramasUML_Archivos/tree/main/Archivos_evaluacion_LLMs/Diagramas_generados_LLMs. La evaluación se realizó mediante la métrica ROUGE (*Recall-Oriented Understudy for Gisting Evaluation*), reconocida en el ámbito del procesamiento de lenguaje natural para medir la similitud entre textos generados y textos de referencia elaborados manualmente. Aunque ROUGE fue inicialmente diseñada para evaluar resúmenes, investigaciones recientes han extendido su aplicación a la evaluación de textos estructurados en formatos específicos, tales como representaciones textuales de modelos y diagramas UML (Di Rocco et al., 2025; Lin, 2004).

En este trabajo de tesis, la evaluación abarcó dos tipos principales de diagramas UML: diagramas de clases y diagramas de actividades. Ambos se generan en formato textual en sintaxis PlantUML, permitiendo aplicar la métrica ROUGE para cuantificar la correspondencia entre las versiones automáticas y sus diagramas de referencia.

Los diagramas de clases evaluados tienen un enfoque orientado al análisis, buscando representar de forma clara y comprensible las entidades, atributos y relaciones fundamentales del dominio, sin entrar en detalles técnicos excesivos. Los diagramas de actividades, por su parte, se enfocan en la representación del flujo de trabajo y la secuencia de acciones o eventos, destacando el orden y la lógica de las operaciones que modelan el comportamiento del sistema. Para evaluar la similitud en ambos tipos de diagramas, se consideraron las siguientes variantes de ROUGE:

- **ROUGE-1:** Mide la superposición de palabras individuales (unigramas), lo que facilita la evaluación básica de coincidencia léxica.

- **ROUGE-2:** Evalúa la coincidencia de secuencias de dos palabras consecutivas (bigramas), aportando una medida más estricta que captura relaciones y estructuras locales relevantes para la semántica del diagrama.
- **ROUGE-L:** Basada en LCS (*Longest Common Subsequence*), esta métrica considera la subsecuencia común más larga entre texto generado y referencia, lo que ayuda a evaluar la estructura global y la coherencia en el orden de los elementos, especialmente importante en diagramas de actividades donde el flujo y la secuencia son críticos.

Dado el objetivo de evaluar la coherencia estructural de los diagramas generados, se optó por emplear la variante ROUGE-L como métrica principal en este estudio. A diferencia de otras variantes que se centran en coincidencias léxicas simples o secuencias cortas, ROUGE-L permite capturar similitudes en el orden y la disposición de los elementos, lo cual resulta particularmente relevante en la representación de diagramas estructurados, como los de clases y actividades. Esta elección se justifica por la necesidad de valorar no solo la presencia de conceptos clave, sino también su disposición dentro de una secuencia coherente, aspecto esencial para garantizar la validez semántica y funcional de los diagramas UML.

La variante ROUGE-L, es una métrica ampliamente reconocida en el campo del NLP, utilizada para comparar textos generados automáticamente con textos de referencia. Esta métrica se fundamenta en tres medidas clave:

- **Precisión:** Mide la proporción de coincidencias correctas con respecto a los elementos generados por el modelo, es decir, cuántos de los elementos producidos están efectivamente presentes en el diagrama de referencia.
- ***Recall*:** Evalúa la proporción de coincidencias correctas en relación con los elementos esperados en el texto de referencia, reflejando la capacidad del modelo para recuperar la mayor cantidad posible de información relevante.
- **Medida F1:** Representa la media entre la precisión y el *recall*, proporcionando una métrica equilibrada que resume el rendimiento global del LLM.

Con el propósito de evaluar el desempeño de los LLMs en la tarea de generación automática de diagramas de clases UML, se adoptó un enfoque metodológico sustentado en un sistema RAG. Este enfoque integra técnicas de recuperación de contexto y generación con LLMs, lo que permite enriquecer la entrada con información contextual relevante antes de que el modelo produzca la salida deseada. Dada la naturaleza estructurada de los diagramas generados, representados mediante la sintaxis de PlantUML, fue posible establecer una base uniforme para la comparación automatizada entre los diagramas generados por los modelos y los diagramas de referencia.

La Tabla 7 presenta el tipo de acceso disponible para cada uno de los modelos de lenguaje evaluados en este estudio. Se distingue entre aquellos que pueden ejecutarse localmente, gracias a su carácter de código abierto, y aquellos cuyo uso se encuentra restringido al acceso mediante API proporcionada por el proveedor. Esta distinción es relevante tanto en términos de flexibilidad para el ajuste del modelo como en lo referente a los requerimientos técnicos y costos de implementación.

Tabla 7. Tipo de acceso de los LLMs evaluados.

Modelo/versión	Tipo de acceso
GPT-4	Acceso remoto (API Key - OpenAI)
Claude Sonnet 4	Acceso remoto (API Key - Anthropic)

Modelo/versión	Tipo de acceso
Claude Opus 4.1	Acceso remoto (API Key - Anthropic)
Gemini 2.5 Pro	Acceso remoto (API Key - Google)
LlaMA 3.2	Ejecución local (Ollama 0.7.0)
Deepseek-R1	Ejecución local (Ollama 0.7.0)
Mistral	Ejecución local (Ollama 0.7.0)

3.3.4. Resultados de la evaluación cuantitativa

En esta sección se presentan los resultados cuantitativos obtenidos en la evaluación del desempeño de los siete modelos de lenguaje, tanto propietarios como de código abierto como GPT-4, Claude Sonnet 4, Claude Opus 4.1, Gemini 2.5 Pro, LLaMA3.2, DeepSeek-R1 y Mistral en la generación automática de diagramas UML de clases y de actividades a partir de historias de usuario. Tal como se describió en la sección 3.3.3, la evaluación se realizó utilizando la métrica ROUGE en sus variantes ROUGE-1, ROUGE-2 y ROUGE-L, considerando para cada una las medidas de precisión, r*ecall* y F1. Estas medidas permitieron comparar los diagramas generados por los LLMs ates mencionados con los diagramas de referencia obtenidos del repositorio *Real World PlantUML*, evaluando el grado de coincidencia tanto en términos léxicos como estructurales. En particular, los valores obtenidos hacen posible identificar en qué medida los modelos fueron capaces de recuperar información relevante presente en las historias de usuario, conservar términos y relaciones propias del dominio y reproducir de manera consistente la estructura básica del diagrama UML en sintaxis PlantUML. Este análisis proporciona, por tanto, una visión detallada del rendimiento de cada modelo, así como de los patrones de acierto, omisión y variabilidad en la representación textual de los diagramas generados.

Los resultados se analizan de forma separada para cada tipo de diagrama, dado que los diagramas de clases y de actividades presentan distintas exigencias semánticas y estructurales. Mientras que los diagramas de clases evalúan principalmente la correcta identificación de entidades, atributos y relaciones, los diagramas de actividades se centran en la representación de flujos de control, secuencias de acciones y decisiones.

3.3.4.1. Resultados para diagramas de clases

En esta sección se presentan los resultados cuantitativos obtenidos en la generación automática de diagramas de clases, evaluados mediante las métricas ROUGE-1, ROUGE-2 y ROUGE-L, considerando las medidas de precisión, *recall* y F1. Los valores reportados corresponden a los promedios obtenidos a partir de un conjunto de 68 historias de usuario obtenidas del repositorio *Real World PlantUML*. Los resultados completos de la evaluación cuantitativa de los diagramas de clases se encuentran disponibles para su consulta en:

https://github.com/LoreMartinez/Repositorio_Tesis_DiagramasUML_Archivos/blob/main/Archivos_evaluacion_LLMs/Resultados_evaluacion_cuantitativa_diagramas_clases.pdf

3.3.4.2. Resultados de ROUGE-1 para diagramas de clases

El resumen de los resultados de la métrica ROUGE-1 se presentan en la Tabla 8. En esta métrica, **Gemini 2.5 Pro** destaca como el modelo con mejor desempeño global, al obtener el valor más alto en la medida F1 con **0.334** y la mayor precisión con **0.443**. Este comportamiento indica que Gemini 2.5

Pro no solo identifica correctamente una mayor proporción de elementos relevantes, sino que lo hace con un mayor grado de exactitud. En segundo lugar, se ubica LLaMA3.2 con 0.286, seguido por GPT-4 con 0.274. Ambos modelos muestran un rendimiento intermedio estable, lo que sugiere una capacidad adecuada para recuperar los elementos básicos de los diagramas, aunque con menor exactitud que Gemini 2.5 Pro. Los modelos Claude Sonnet 4, Claude Opus 4.1 y Mistral presentan valores de F1 ligeramente inferiores, pero relativamente cercanos entre sí, lo que indica un comportamiento homogéneo dentro de este grupo. Por último, DeepSeek-R1 registra los valores más bajos en las tres medidas, lo que refleja una capacidad limitada para capturar correctamente la información léxica fundamental de los diagramas de clases.

Tabla 8. Promedios por modelo ordenado por la medida F1 de ROUGE-1.

Posición	Modelo	ROUGE-1		
		Precisión	***Recall***	**F1**
1	**Gemini 2.5 Pro**	**0.443**	0.319	**0.334**
2	LLaMA3.2	0.357	0.303	0.286
3	GPT-4	0.341	0.289	0.274
4	Claude Opus 4.1	0.241	**0.392**	0.273
5	Claude Sonnet 4	0.258	0.355	0.269
6	Mistral	0.337	0.284	0.267
7	DeepSeek-R1	0.263	0.289	0.229

En conjunto, estos resultados permiten establecer una primera jerarquía clara de desempeño entre los modelos evaluados. Gemini 2.5 Pro se posiciona como el referente principal en la recuperación de información léxica básica, mientras que LLaMA3.2 y GPT-4 conforman un segundo nivel con resultados competitivos, pero menos precisos. Este comportamiento justifica la necesidad de analizar métricas más estrictas como ROUGE-2 y ROUGE-L en los apartados posteriores.

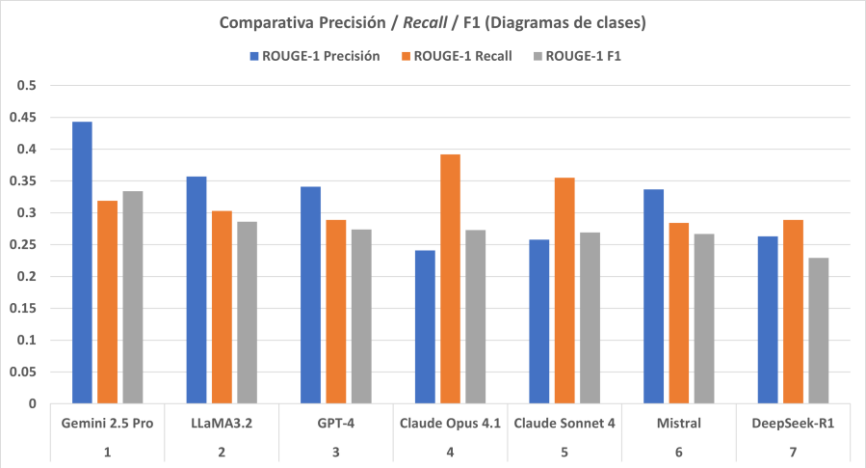


Figura 19 Comparación del desempeño en precisión, *recall* y F1 para ROUGE-1.

La Figura 3.4 muestra las diferencias de desempeño entre los modelos bajo la métrica ROUGE-1. Se observa que Gemini 2.5 Pro mantiene una ventaja clara tanto en precisión como en F1, lo que confirma su mejor aproximación léxica a los diagramas de referencia. LLaMA3.2 y GPT-4 conforman

un grupo intermedio con resultados equilibrados, mientras que Claude Sonnet 4 y Claude Opus 4.1 destacan por valores de recall más altos, aunque con menor precisión. DeepSeek-R1 se mantiene en el nivel inferior en las tres medidas.

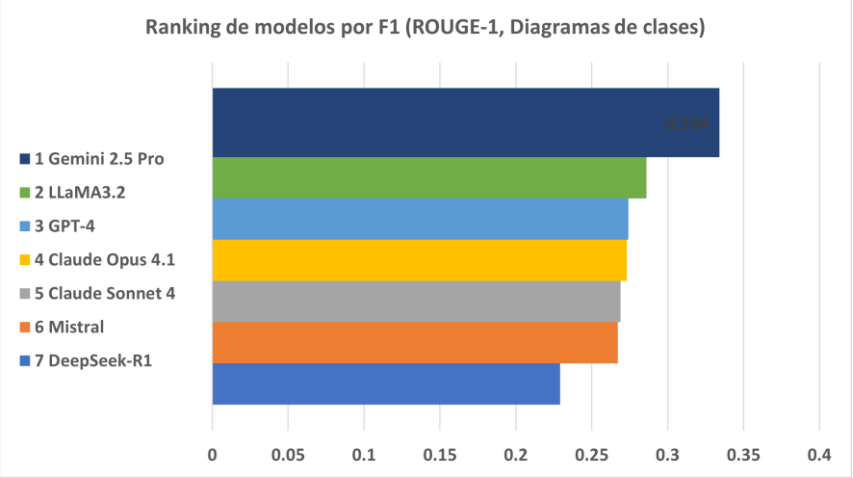


Figura 20 *Ranking* de los modelos con base en F1 para ROUGE-1.

La Figura 3.5 sintetiza el desempeño global de los modelos mediante la medida F1. Se confirma a Gemini 2.5 Pro en la primera posición con una diferencia marcada frente al resto. LLaMA3.2 y GPT-4 ocupan los lugares siguientes con valores muy similares, lo que refleja un comportamiento competitivo. Claude Sonnet 4, Mistral y Claude Opus 4.1 se sitúan en posiciones intermedias, mientras que DeepSeek-R1 permanece en la última posición de la clasificación.

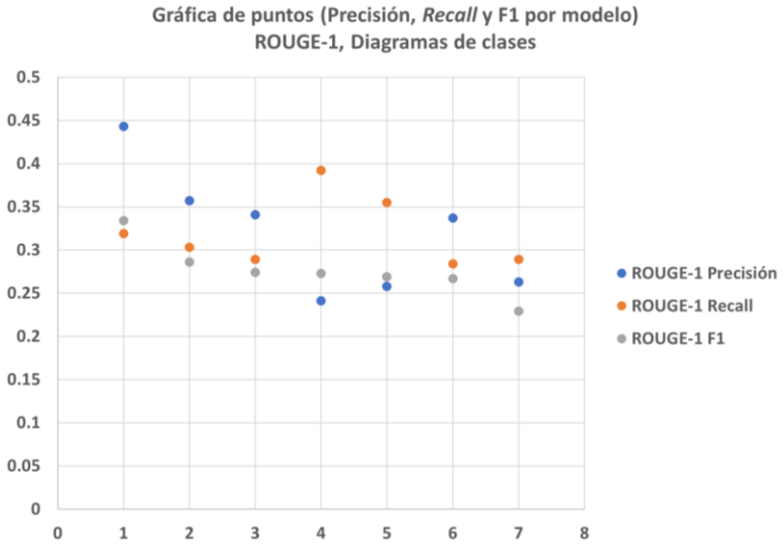


Figura 21 Precisión, *recall* y F1 por cada modelo para ROUGE-1.

La Figura 3.6 permite analizar la variación individual de las métricas sin superposición visual. Se observa nuevamente que Gemini 2.5 Pro alcanza los valores más altos en precisión y F1. Claude Sonnet 4 y Claude Opus 4.1 registran valores elevados de *recall*, acompañados de una disminución en precisión, lo que limita su F1. LLaMA3.2, GPT-4 y Mistral muestran un comportamiento más equilibrado entre las tres medidas, mientras que DeepSeek-R1 se mantiene de forma constante en los valores más bajos.

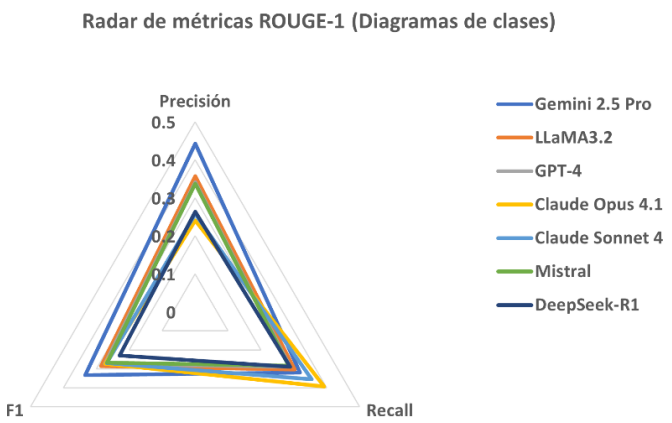


Figura 22 Radar comparativo de precisión, *recall* y F1 para ROUGE-1.

La Figura 3.7 resume de manera conjunta el comportamiento de los modelos en las tres métricas. El polígono de Gemini 2.5 Pro presenta la mayor extensión, reflejando su ventaja global en precisión y F1, así como un nivel competitivo de *recall*. LLaMA3.2 y GPT-4 conforman un segundo grupo con áreas amplias y equilibradas. Claude Sonnet 4 y Claude Opus 4.1 muestran mayor cobertura, con valores altos de *recall*, pero menor precisión. Mistral presenta un comportamiento intermedio, mientras que el polígono de DeepSeek-R1 es claramente el más reducido del conjunto.

3.3.4.3. Resultados de ROUGE-2 para diagramas de clases

Los valores correspondientes a la métrica ROUGE-2 se presentan en la Tabla 9. Al tratarse de una métrica más estricta que ROUGE-1, todos los modelos exhiben una disminución en sus puntuaciones, lo que refleja la mayor dificultad para reproducir secuencias consecutivas de palabras. En esta métrica, **Gemini 2.5 Pro** mantiene el mejor desempeño con un valor F1 de **0.120**, seguido por GPT-4 con 0.094 y LLaMA3.2 con 0.093. Estos resultados indican que Gemini 2.5 Pro conserva una mayor capacidad para preservar relaciones léxicas locales entre términos consecutivos, aspecto clave para representar de manera adecuada asociaciones entre clases, atributos y relaciones. Claude Sonnet 4 y Claude Opus 4.1 presentan valores moderados, caracterizados por un *recall* superior a la precisión, lo que sugiere una mayor recuperación de bigramas relevantes, aunque con menor exactitud. Mistral se ubica en un nivel similar, mientras que DeepSeek-R1 ocupa nuevamente la última posición, confirmando las limitaciones ya observadas cuando el nivel de exigencia aumenta.

Tabla 9. Promedios por modelo ordenado por la medida F1 de la métrica ROUGE-2.

Posición	Modelo	ROUGE-2		
		Precisión	***Recall***	**F1**
1	**Gemini 2.5 Pro**	**0.148**	0.121	**0.120**
2	GPT-4	0.107	0.088	0.094
3	LLaMA3.2	0.116	0.084	0.093
4	Claude Sonnet 4	0.077	0.121	0.085
5	Mistral	0.071	0.091	0.081
6	Claude Opus 4.1	0.067	**0.125**	0.080
7	DeepSeek-R1	0.064	0.095	0.061

En conjunto, los resultados de ROUGE-2 refuerzan el patrón identificado en ROUGE-1, pero con diferencias más acentuadas entre los modelos. La mayor exigencia en la coincidencia de bigramas pone de manifiesto la ventaja de Gemini 2.5 Pro para mantener coherencia local, así como las dificultades del resto de los modelos para sostener una correspondencia precisa entre los diagramas generados y los de referencia.

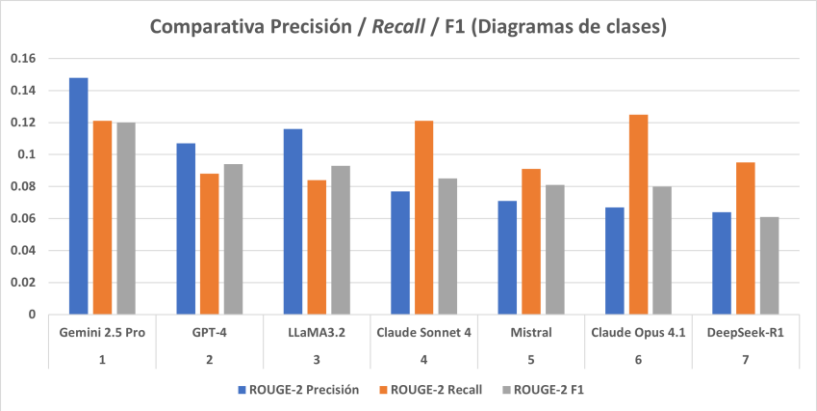


Figura 23 Comparación del desempeño en precisión, *recall* y F1 para ROUGE-2.

La Figura 3.8 muestra que Gemini 2.5 Pro conserva los valores más altos en las tres medidas. Claude Sonnet 4, Claude Opus 4.1 y Mistral registran valores más elevados de recall, acompañados de menores niveles de precisión, lo que limita su F1. GPT-4 y LLaMA3.2 presentan un comportamiento intermedio con un equilibrio más estable entre las métricas, mientras que DeepSeek-R1 se mantiene en el nivel inferior.

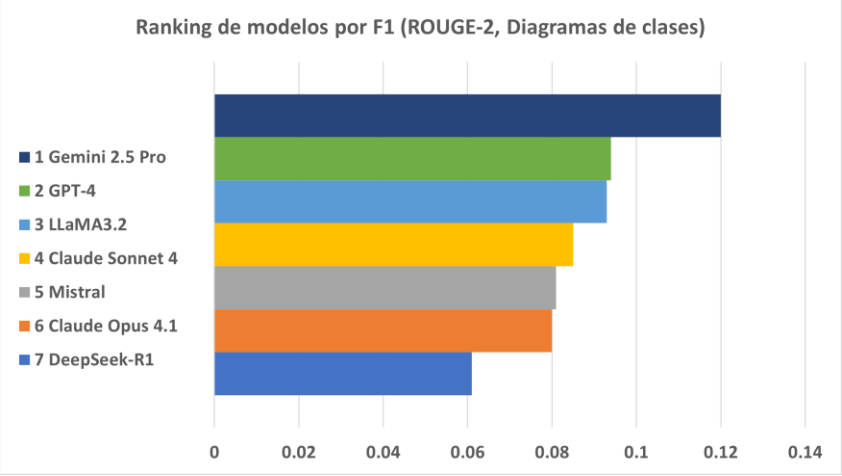


Figura 24 *Ranking* de modelos con base en F1 para ROUGE-2.

La Figura 3.9 sintetiza el desempeño global mediante la medida F1. Gemini 2.5 Pro ocupa la primera posición, seguido por GPT-4 y LLaMA3.2 con valores muy similares entre sí. Claude Sonnet 4, Mistral y Claude Opus 4.1 se agrupan en posiciones intermedias, mientras que DeepSeek-R1 se conserva en el último lugar de la clasificación.

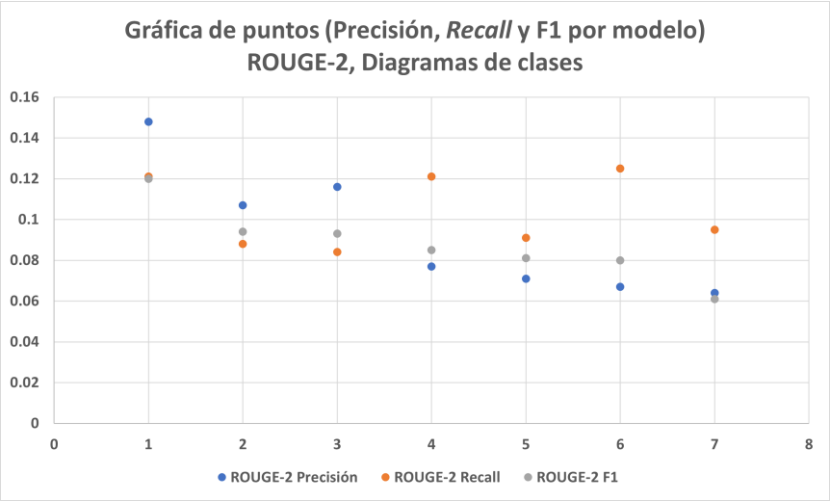


Figura 25 Precisión, *recall* y F1 por cada modelo para ROUGE-2.

La Figura 3.10 permite observar la distribución individual de las métricas sin superposición visual. Se confirma que Gemini 2.5 Pro presenta los valores más altos en precisión y F1. Claude Sonnet 4, Claude Opus 4.1 y Mistral destacan por valores elevados de *recall*, acompañados de una reducción en precisión. GPT-4 y LLaMA3.2 mantienen una distribución más equilibrada entre las tres medidas, mientras que DeepSeek-R1 se concentra nuevamente en los valores más bajos.

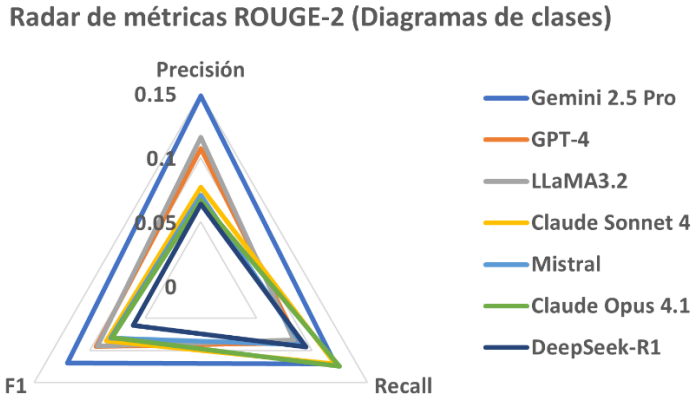


Figura 26 Radar comparativo de precisión, *recall* y F1 para ROUGE-2.

La Figura 3.11 resume de manera conjunta el comportamiento de los modelos en ROUGE-2. El polígono de Gemini 2.5 Pro presenta la mayor extensión, reflejando su ventaja global en las tres medidas. GPT-4 y LLaMA3.2 conforman un segundo grupo con áreas relativamente amplias y equilibradas. Claude Sonnet 4, Claude Opus 4.1 y Mistral muestran mayor cobertura en *recall*, aunque con menor precisión. El polígono de DeepSeek-R1 es el de menor tamaño, confirmando su desempeño limitado bajo esta métrica.

3.3.4.4. Resultados de ROUGE-L para diagramas de clases

Los resultados correspondientes a la métrica ROUGE-L se presentan en la Tabla 10, donde se muestran los valores promedio de precisión, *recall* y F1 para cada modelo. En esta métrica, **Gemini 2.5 Pro** vuelve a posicionarse como el modelo con mejor desempeño al alcanzar un valor F1 de **0.279**, seguido por LLaMA3.2 con 0.242 y GPT-4 con 0.236. Estos tres modelos conforman el grupo con mayor capacidad para aproximarse a la estructura global de los diagramas de referencia. Los modelos Claude Sonnet 4, Mistral y Claude Opus 4.1 presentan valores intermedios y muy cercanos entre sí,

mientras que DeepSeek-R1 registra nuevamente el valor más bajo. Destaca que Claude Sonnet 4 y Claude Opus 4.1 obtienen valores relativamente altos de *recall*, lo que indica una mayor recuperación de fragmentos estructurales del diagrama, aunque con menor precisión en la secuencia exacta.

Tabla 10. Promedios por modelo ordenado por la medida F1 de la métrica ROUGE-L.

Posición	Modelo	ROUGE-L		
		Precisión	***Recall***	**F1**
1	**Gemini 2.5 Pro**	**0.365**	0.268	**0.279**
2	LLaMA3.2	0.298	0.212	0.242
3	GPT-4	0.299	0.222	0.236
4	Claude Sonnet 4	0.208	0.315	0.224
5	Mistral	0.292	0.216	0.223
6	Claude Opus 4.1	0.191	**0.330**	0.221
7	DeepSeek-R1	0.203	0.257	0.191

Estos resultados muestran que, al evaluar la similitud a partir de las subsecuencias comunes más largas, las diferencias de desempeño entre los modelos se hacen aún más evidentes y favorecen de manera consistente a Gemini 2.5 Pro. Este comportamiento confirma su mayor capacidad para preservar la estructura global de los diagramas en términos de secuencias textuales más extensas, aspecto fundamental para representar de forma coherente las relaciones entre clases, atributos y asociaciones en los diagramas de clases generados. En contraste, el resto de los modelos presenta una disminución más pronunciada en sus puntuaciones, particularmente en la medida F1, lo que refleja mayores dificultades para mantener una correspondencia estructural completa entre los diagramas generados y los de referencia. De este modo, ROUGE-L no solo confirma el liderazgo de Gemini 2.5 Pro observado en las variantes ROUGE-1 y ROUGE-2, sino que también pone de relieve las limitaciones del resto de los modelos cuando se requiere una mayor fidelidad en la conservación de la estructura global del diagrama. A continuación, se presentan las figuras correspondientes a ROUGE-L, que comparan el desempeño de los modelos en precisión, *recall* y F1 para los diagramas de clases.

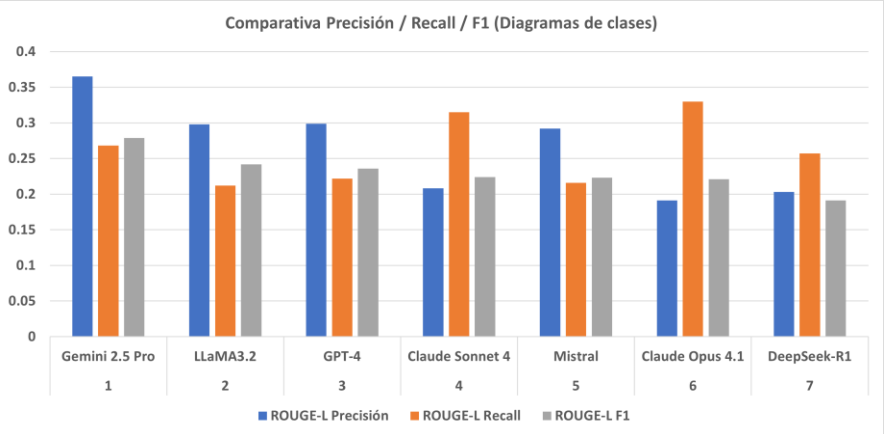


Figura 27 Comparación del desempeño en precisión, *recall* y F1 para ROUGE-L.

La Figura 3.12 muestra que Gemini 2.5 Pro mantiene los valores más altos en precisión y F1, con una diferencia clara frente al resto de los modelos. LLaMA3.2 y GPT-4 presentan valores muy

próximos entre sí, ubicándose en un segundo nivel de desempeño. Claude Sonnet 4, Mistral y Claude Opus 4.1 se sitúan en un nivel intermedio, con valores de *recall* relativamente altos, pero con menor precisión, lo que reduce su F1. DeepSeek-R1 permanece en el nivel inferior en las tres medidas.

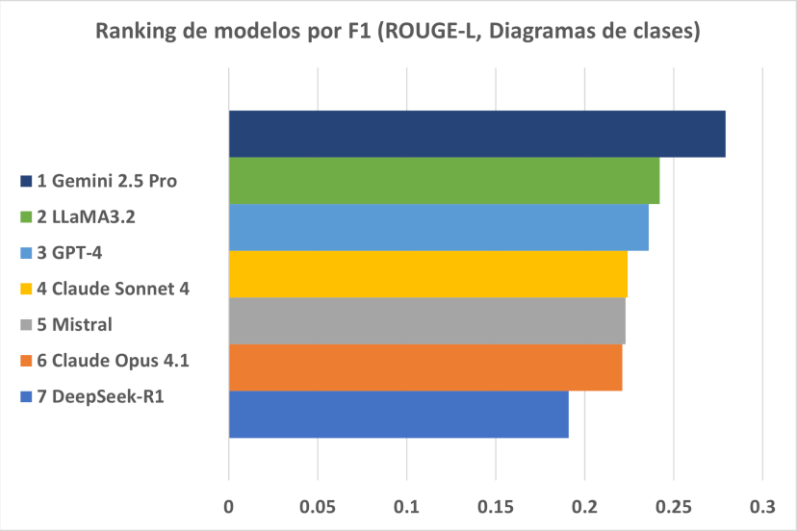


Figura 28 *Ranking* de los modelos con base en F1 para ROUGE-L.

La Figura 3.13 confirma el patrón observado en las métricas anteriores. Gemini 2.5 Pro encabeza la clasificación con una diferencia apreciable respecto al resto. LLaMA3.2 y GPT-4 ocupan las siguientes posiciones con valores similares, mientras que Claude Sonnet 4, Mistral y Claude Opus 4.1 se agrupan en posiciones intermedias. DeepSeek-R1 se mantiene en la última posición de la clasificación.

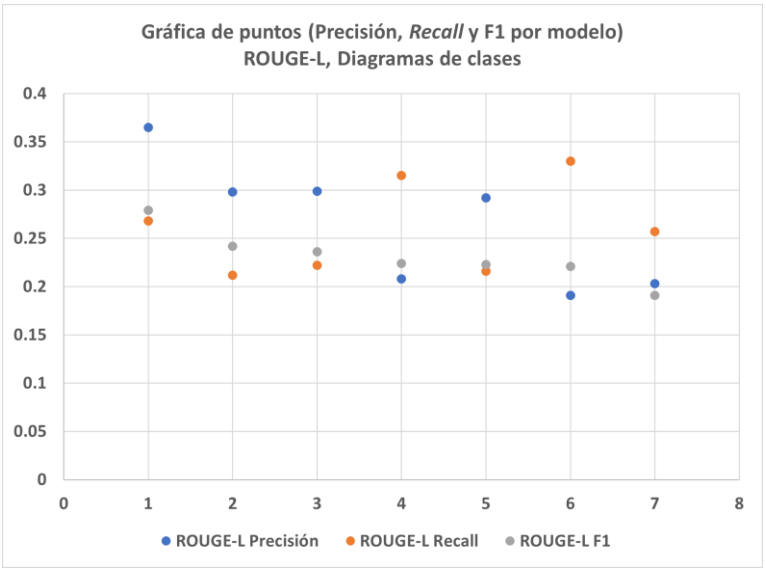


Figura 29 Precisión, *recall* y F1 por cada modelo para ROUGE-L.

La Figura 3.14 permite analizar la variación individual de las métricas sin superposición visual. Gemini 2.5 Pro vuelve a destacar con los valores más altos en precisión y F1. LLaMA3.2 y GPT-4 muestran valores muy cercanos entre sí, con un desempeño equilibrado dentro del grupo superior. Claude Sonnet 4 y Claude Opus 4.1 alcanzan los mayores valores de *recall*, acompañados de una

disminución en precisión, lo que limita su F1. Mistral presenta resultados moderados y consistentes, mientras que DeepSeek-R1 se mantiene claramente por debajo del resto.

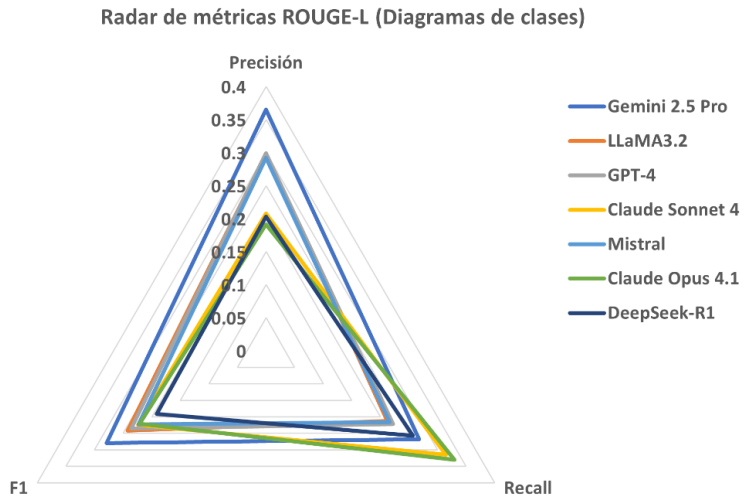


Figura 30 Radar comparativo de precisión, *recall* y F1 para ROUGE-L.

La Figura 3.15 resume de manera conjunta el comportamiento de los modelos en ROUGE-L. El polígono de Gemini 2.5 Pro es el de mayor extensión, reflejando su ventaja en precisión y F1, junto con un recall competitivo. LLaMA3.2 y GPT-4 conforman un segundo grupo con áreas equilibradas. Claude Sonnet 4 y Claude Opus 4.1 muestran mayor cobertura, con valores altos de *recall*, pero menor precisión. Mistral presenta un área intermedia bien distribuida, mientras que el polígono de DeepSeek-R1 es el más reducido del conjunto.

3.3.4.5. Análisis global del desempeño de los LLMs en los diagramas de clases

De manera general, la Tabla 11 muestra que **Gemini 2.5 Pro** domina con claridad al obtener el mejor desempeño en 32 historias de usuario, lo que representa el **47.1%** del total. En segundo lugar, aparecen Mistral, Claude Sonnet 4 y GPT-4, con resultados considerablemente menores que oscilan entre siete y nueve historias destacadas. Esta distribución confirma que Gemini 2.5 Pro no solo alcanza el mejor promedio global, sino que también mantiene un rendimiento superior de forma consistente en una cantidad significativa de casos individuales.

Tabla 11. Frecuencia de historias de usuario donde cada modelo obtuvo el mejor desempeño.

Modelo	Cantidad de historias de usuario con mejor desempeño	Proporción del total de historias (%)
Gemini 2.5 Pro	32	47.1%
Mistral	9	13.2%
Claude Sonnet 4	7	10.3%
GPT-4	7	10.3%
Claude Opus 4.1	5	7.4%
LLaMA3.2	5	7.4%
DeepSeek-R1	3	4.4%

La Tabla 12 presenta el análisis de estabilidad considerando la media, la desviación estándar y los valores mínimo y máximo del F1 obtenidos por cada modelo a lo largo de las 68 historias de usuario. En este análisis, Gemini 2.5 Pro destaca nuevamente como el modelo con mejor desempeño general, logrando una media de F1 de **0.244**, la más alta entre todos los modelos evaluados. Además,

alcanza un valor máximo de **0.798**, lo que demuestra su capacidad para producir resultados excepcionalmente precisos cuando la historia de usuario proporciona información clara y estructurada. Aunque su rango de variación es amplio, con una diferencia de **0.754** entre su valor mínimo y máximo, esta dispersión puede interpretarse como una alta capacidad de adaptación ante historias de distinta complejidad. En los casos donde la entrada contiene suficiente detalle semántico, Gemini 2.5 Pro es capaz de aprovecharlo para generar resultados muy superiores a los del resto de los modelos. De este modo, la variabilidad observada no representa inconsistencia, sino una respuesta flexible y sensible a la calidad del insumo, un aspecto favorable en contextos donde se puede enriquecer o mejorar la especificación de las historias de usuario. Por su parte, LLaMA3.2 muestra un comportamiento más estable, con un rango de **0.479**, pero acompañado de una media de F1 inferior, ubicada en **0.207**, lo que indica un desempeño consistentemente menor. Los modelos GPT-4, Claude Sonnet 4 y Claude Opus 4.1 presentan valores intermedios tanto en promedio como en dispersión, mientras que DeepSeek-R1 obtiene la media más baja, situada en **0.160**, con un rango de variación moderado de **0.509**.

Tabla 12. Estabilidad por modelo de la medida F1.

Posición	Modelo	F1				
		Media	Desv. Est.	Min.	Máx.	Rango
1	Gemini 2.5 Pro	0.244	0.153	0.044	0.798	0.754
2	LLaMA3.2	0.207	0.115	0.051	0.53	0.479
3	GPT-4	0.197	0.121	0.041	0.635	0.594
4	Claude Sonnet 4	0.192	0.117	0.026	0.659	0.633
5	Claude Opus 4.1	0.191	0.107	0.025	0.628	0.604
6	Mistral	0.188	0.114	0.027	0.594	0.567
7	DeepSeek-R1	0.16	0.094	0.045	0.555	0.509

En conjunto, los resultados evidencian que Gemini 2.5 Pro combina el mejor desempeño promedio con la capacidad de alcanzar los valores más altos entre todos los modelos analizados, lo que lo posiciona como la opción más sólida cuando se busca maximizar la calidad de los diagramas generados. Su comportamiento sugiere que, al utilizar historias mejor definidas o mediante estrategias de enriquecimiento del contexto, puede superar de manera notable a los demás modelos evaluados. Además, su capacidad para adaptarse a distintos niveles de complejidad en las historias de usuario refuerza su utilidad en entornos reales de desarrollo de *software*.

3.3.4.6. Resultados para diagramas de actividades

En esta sección se presentan los resultados cuantitativos obtenidos en la generación automática de diagramas de actividades, evaluados mediante las métricas ROUGE-1, ROUGE-2 y ROUGE-L, considerando las medidas de precisión, *recall* y F1. Los valores reportados corresponden a los promedios obtenidos a partir de un conjunto de 50 historias de usuario obtenidas del repositorio *Real World PlantUML*. Los resultados completos de la evaluación cuantitativa de los diagramas de actividades se encuentran disponibles para su consulta en:

https://github.com/LoreMartinez/Repositorio_Tesis_DiagramasUML_Archivos/blob/main/Archivos_evaluacion_LLMs/Resultados_evaluacion_cuantitativa_diagramas_actividades.pdf

3.3.4.7. Resultados de ROUGE-1 para diagramas de actividades

Los resultados de la métrica ROUGE-1 se presenta en la Tabla 13. **Claude Sonnet 4** obtuvo el mejor desempeño con un valor de F1 de **0.428**, seguido de Claude Opus 4.1 con 0.415 y Gemini 2.5 Pro con 0.408. Estos valores reflejan la capacidad de recuperar los elementos básicos presentes en los

diagramas de referencia. Gemini 2.5 Pro destaca por alcanzar el mayor valor de *recall*, con un promedio de 0.555, lo que indica una elevada cobertura de los elementos relevantes del diagrama, aunque con una precisión ligeramente inferior a la de los modelos de Claude Sonnet 4 y Claude Opus 4.1. GPT-4 presenta un desempeño intermedio con un F1 de 0.346, mientras que LLaMA3.2, Mistral y DeepSeek-R1 registran los valores más bajos, con F1 de 0.286, 0.266 y 0.238, respectivamente. Estos resultados indican mayores dificultades para capturar de manera consistente la información léxica fundamental de los procesos representados.

Tabla 13. Promedios por modelo ordenado por la medida F1 de la métrica ROUGE-1.

Posición	Modelo	ROUGE-1		
		Precisión	***Recall***	**F1**
1	Claude Sonnet 4	**0.391**	**0.532**	**0.428**
2	Gemini 2.5 Pro	0.372	0.552	0.415
3	Claude Opus 4.1	0.353	0.555	0.408
4	GPT-4	0.329	0.474	0.346
5	LLaMA3.2	0.276	0.390	0.286
6	Mistral	0.259	0.404	0.266
7	DeepSeek-R1	0.256	0.437	0.238

De manera general, los resultados de ROUGE-1 evidencian que los modelos Claude Sonnet 4, Claude Opus 4.1 y Gemini 2.5 Pro concentran los niveles más altos de recuperación léxica en la generación de diagramas de actividades, lo que refleja una mayor coincidencia en el uso de términos respecto a los diagramas de referencia. En contraste, el resto de los modelos presenta un desempeño notablemente inferior en este nivel básico de coincidencia textual, poniendo de manifiesto mayores dificultades para reproducir con precisión el texto esperado, tal como se muestra en las figuras que se presentan a continuación.

Las Figuras 3.16 a la 3.19 presentan las gráficas comparativas del desempeño de los modelos bajo ROUGE-1 mediante gráficas de barras, clasificación por F1, gráfica de puntos y radar. En conjunto, estas visualizaciones confirman la ventaja de Claude Sonnet 4 y Claude Opus 4.1 en términos de F1, así como la elevada cobertura lograda por Gemini 2.5 Pro a través de sus altos valores de *recall*. Los modelos intermedios muestran un comportamiento más equilibrado, mientras que DeepSeek-R1 se mantiene de forma consistente en los niveles más bajos.

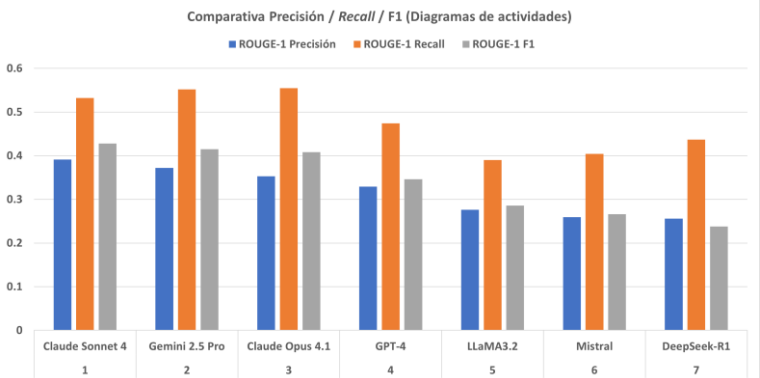


Figura 31 Comparación del desempeño en precisión, *recall* y F1 para ROUGE-1.

La Figura 3.16 muestra que Claude Sonnet 4, Gemini 2.5 Pro y Claude Opus 4.1 alcanzan los valores más altos de precisión, *recall* y F1 bajo ROUGE-1, evidenciando una mayor coincidencia léxica con los diagramas de referencia. GPT-4 presenta un desempeño intermedio, mientras que LLaMA3.2, Mistral y DeepSeek-R1 registran los valores más bajos en las tres medidas.

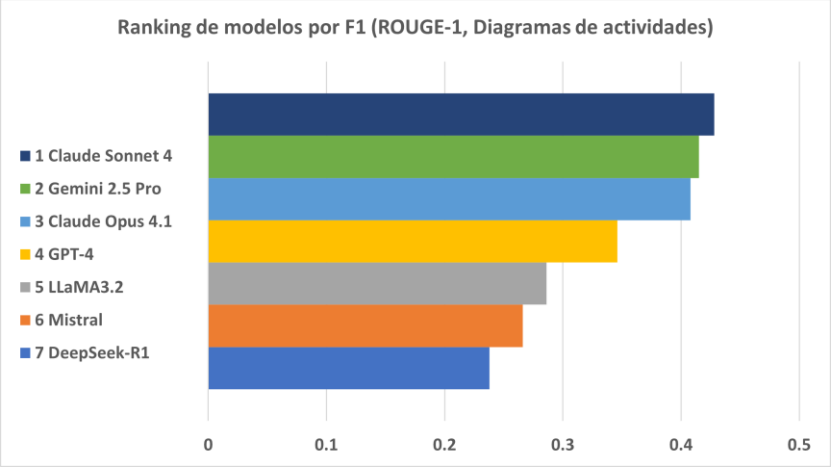


Figura 32 *Ranking* de los modelos con base en F1 para ROUGE-1.

La Figura 3.17 muestra el *ranking* de los modelos en función del valor F1 bajo ROUGE-1 para los diagramas de actividades, donde Claude Sonnet 4 ocupa la primera posición, seguido de Gemini 2.5 Pro y Claude Opus 4.1. GPT-4 se ubica en un nivel intermedio, mientras que LLaMA3.2, Mistral y DeepSeek-R1 presentan los valores más bajos, evidenciando un menor equilibrio entre precisión y recuperación léxica.

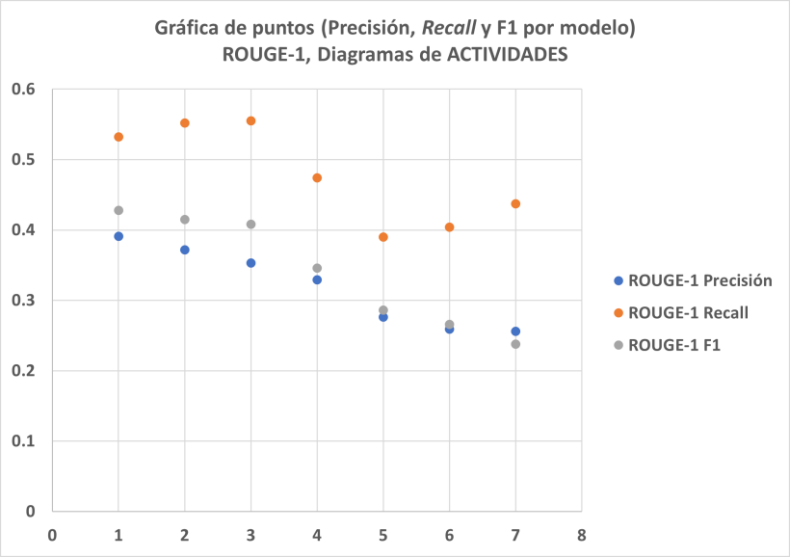


Figura 33 Precisión, *recall* y F1 por cada modelo para ROUGE-1.

La Figura 3.18 evidencia que Claude Sonnet 4, Gemini 2.5 Pro y Claude Opus 4.1 concentran los valores más altos de precisión, *recall* y F1 bajo ROUGE-1 en los diagramas de actividades. GPT-4 presenta un comportamiento intermedio, mientras que LLaMA3.2, Mistral y DeepSeek-R1 muestran los valores más bajos en las tres métricas, confirmando el patrón de desempeño observado en las figuras anteriores.

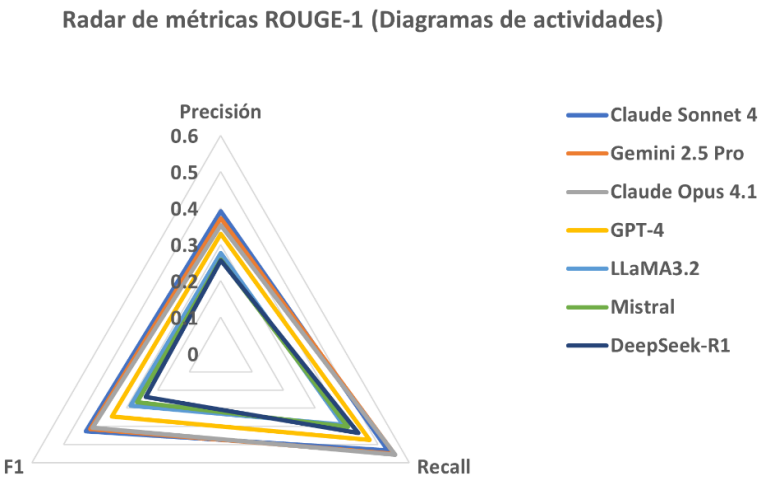


Figura 34 Radar comparativo de precisión, *recall* y F1 para ROUGE-1.

La Figura 3.19 muestra que Claude Sonnet 4, Gemini 2.5 Pro y Claude Opus 4.1 concentran los valores más elevados y equilibrados en precisión, *recall* y F1 bajo ROUGE-1 para los diagramas de actividades. GPT-4 presenta un perfil intermedio, mientras que LLaMA3.2, Mistral y DeepSeek-R1 exhiben áreas notablemente menores, lo que confirma su desempeño inferior en las tres medidas.

3.3.4.8. Resultados de ROUGE-2 para diagramas de actividades

Los valores correspondientes a ROUGE-2 se muestran en la Tabla 14. Al tratarse de una métrica más exigente, orientada a la evaluación de la coincidencia de pares consecutivos de palabras, se observa una disminución generalizada de las puntuaciones en todos los modelos con respecto a ROUGE-1. **Gemini 2.5 Pro** ocupa la primera posición con un valor F1 de **0.144**, seguido por Claude Sonnet 4 con 0.132 y Claude Opus 4.1 con 0.118. Gemini 2.5 Pro destaca además por registrar simultáneamente los valores más altos de precisión con 0.128 y recall con 0.194, lo que evidencia un equilibrio favorable entre exactitud y cobertura en la recuperación de secuencias de acciones. GPT-4 con 0.117 y LLaMA3.2 con 0.104 presentan un desempeño intermedio, mientras que DeepSeek-R1 con 0.103 y Mistral con 0.087 registran los valores más bajos, lo que pone de manifiesto las mayores dificultades para reproducir relaciones secuenciales coherentes cuando se incrementa el nivel de exigencia estructural.

Tabla 14. Promedios por modelo ordenado por la medida F1 de la métrica ROUGE-2.

Posición	Modelo	ROUGE-2		
		Precisión	***Recall***	**F1**
1	**Gemini 2.5 Pro**	**0.128**	**0.194**	**0.144**
2	Claude Sonnet 4	0.116	0.170	0.132
3	Claude Opus 4.1	0.100	0.165	0.118
4	GPT-4	0.101	0.156	0.117
5	LLaMA3.2	0.090	0.136	0.104
6	DeepSeek-R1	0.089	0.152	0.103
7	Mistral	0.074	0.124	0.087

En términos generales, los resultados de ROUGE-2 confirman que mantener la coherencia entre actividades consecutivas en los diagramas de actividades es una tarea más demandante que reproducir términos de manera aislada. Esta métrica acentúa con mayor claridad las diferencias entre los modelos que logran conservar la estructura y el orden del flujo, y aquellos que presentan mayores dificultades para preservar la coherencia secuencial, lo cual se refleja directamente en sus valores de precisión, *recall* y F1. Las Figuras 3.20 a la 3.23 muestran las gráficas comparativas bajo ROUGE-2, donde se aprecia una reducción generalizada de los valores respecto a ROUGE-1, así como un mayor contraste entre los modelos líderes y el resto.

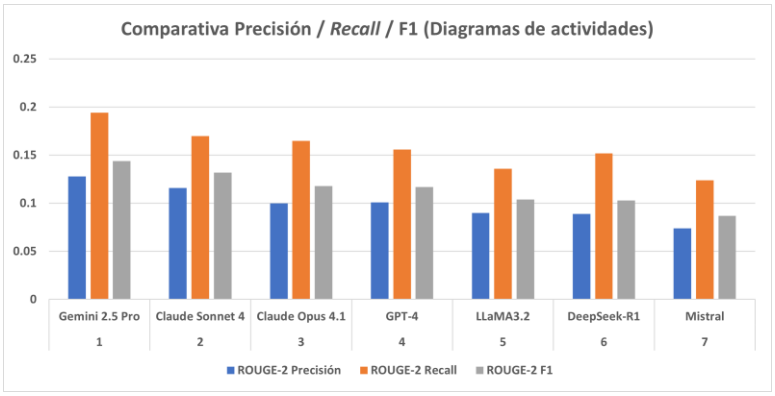


Figura 35 Comparación del desempeño en precisión, *recall* y F1 para ROUGE-2.

La Figura 3.20 muestra la comparativa de precisión, *recall* y F1 bajo la métrica ROUGE-2 en los diagramas de actividades. Se observa que Gemini 2.5 Pro alcanza los valores más altos en las tres métricas, destacando especialmente en *recall*, lo que indica una mayor capacidad para reproducir de manera coherente pares de palabras consecutivas con respecto a las referencias. Claude Sonnet 4 y Claude Opus 4.1 presentan un desempeño cercano, aunque ligeramente inferior al del modelo líder. GPT-4 se ubica en un nivel intermedio, mientras que LLaMA3.2, DeepSeek-R1 y Mistral registran los valores más bajos, reflejando mayores dificultades para conservar la coherencia local del contenido generado.

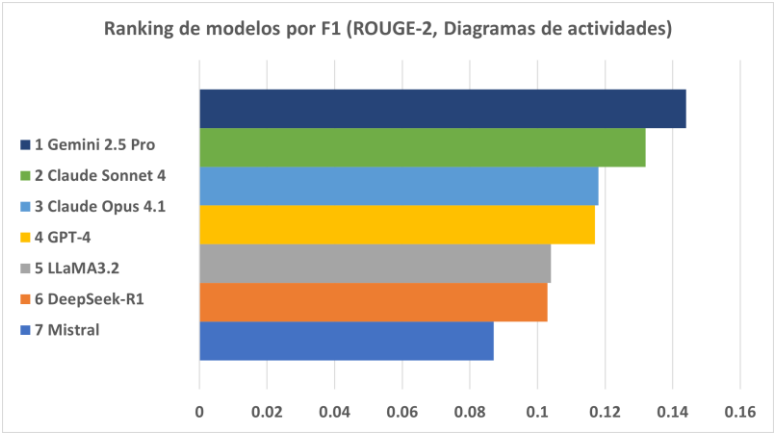


Figura 36 *Ranking* de los modelos con base en F1 para ROUGE-2.

La Figura 3.21 presenta el *ranking* de los modelos por el valor F1 bajo la métrica ROUGE-2 para los diagramas de actividades. Gemini 2.5 Pro ocupa la primera posición, confirmando su superioridad en la preservación de la coherencia local del contenido generado, seguido de Claude Sonnet 4 y Claude Opus 4.1. GPT-4 muestra un desempeño competitivo, aunque ligeramente inferior al de los modelos líderes. En contraste, LLaMA3.2, DeepSeek-R1 y Mistral se ubican en las últimas

posiciones, lo que refleja una menor capacidad para reproducir correctamente las relaciones consecutivas entre los elementos del diagrama de actividades.

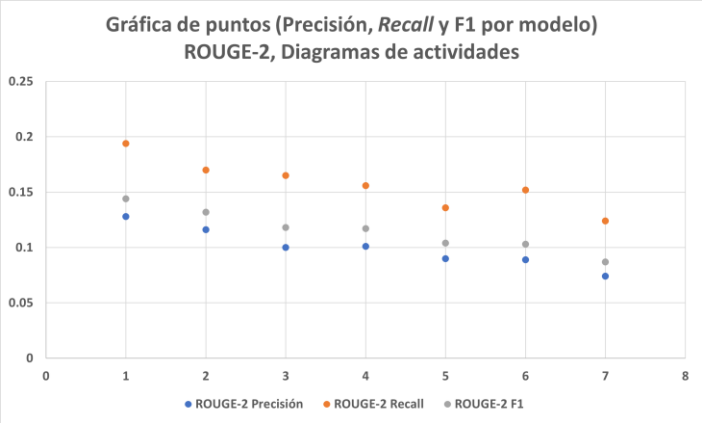


Figura 37 Precisión, *recall* y F1 por cada modelo para ROUGE-2.

La Figura 3.22 muestra la distribución de los valores de precisión, *recall* y F1 bajo ROUGE-2 para cada modelo en los diagramas de actividades. Se observa que Gemini 2.5 Pro concentra los valores más altos en las tres métricas, seguido por Claude Sonnet 4 y Claude Opus 4.1, que mantienen un desempeño cercano. GPT-4 presenta resultados intermedios, mientras que LLaMA3.2, DeepSeek-R1 y Mistral registran los valores más bajos, evidenciando mayores dificultades para conservar la coherencia entre pares consecutivos de elementos del flujo.

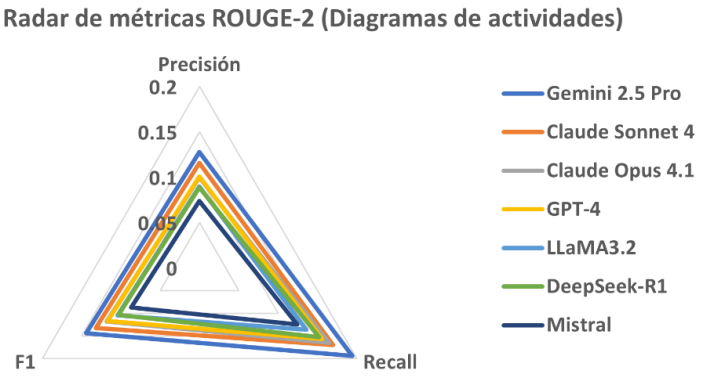


Figura 38 Radar comparativo de precisión, *recall* y F1 para ROUGE-2.

La Figura 3.23 ofrece una visión integrada del desempeño de los modelos en las métricas de precisión, *recall* y F1 bajo ROUGE-2 para los diagramas de actividades. Se aprecia que Gemini 2.5 Pro presenta el comportamiento más sólido y equilibrado, al registrar los valores más altos de forma consistente en las tres métricas evaluadas. Claude Sonnet 4 y Claude Opus 4.1 muestran un desempeño cercano, aunque ligeramente inferior al del modelo líder. GPT-4 se mantiene en un nivel intermedio, mientras que LLaMA3.2, DeepSeek-R1 y Mistral presentan áreas considerablemente menores, lo que evidencia sus mayores limitaciones para preservar la coherencia del flujo entre elementos consecutivos.

3.3.4.9. Resultados de ROUGE-L para diagramas de actividades

Los resultados de ROUGE-L, mostrados en la Tabla 15. En esta métrica, **Claude Sonnet 4** y **Claude Opus 4.1** empatan en la primera posición, ambos con un valor F1 de **0.343**, seguidos muy de cerca por Gemini 2.5 Pro con un F1 de 0.341. Estos valores evidencian que los modelos de Anthropic

presentan una notable capacidad para reproducir la secuencia completa de actividades de forma coherente, mientras que Gemini 2.5 Pro se mantiene como un competidor muy cercano, mostrando además un excelente balance entre precisión y *recall*. GPT-4 ocupa una posición intermedia con un F1 de 0.292, mientras que LLaMA3.2, Mistral y DeepSeek-R1 presentan los resultados más bajos en la concordancia estructural de largo alcance.

Tabla 15. Promedios por modelo ordenado por la medida F1 de la métrica ROUGE-L.

Posición	Modelo	ROUGE-L		
		Precisión	***Recall***	**F1**
1	**Claude Sonnet 4**	**0.319**	0.461	**0.343**
2	Claude Opus 4.1	0.294	**0.472**	**0.343**
3	Gemini 2.5 Pro	0.302	0.471	0.341
4	GPT-4	0.273	0.396	0.292
5	LLaMA3.2	0.240	0.338	0.243
6	Mistral	0.225	0.335	0.229
7	DeepSeek-R1	0.227	0.351	0.221

Los resultados obtenidos con la métrica ROUGE-L confirman la fortaleza de los modelos de Claude Sonnet 4 y Claude Opus 4.1 en la preservación de la estructura secuencial de largo alcance dentro de los diagramas de actividades, lo que refleja una mayor capacidad para mantener relaciones coherentes a lo largo de secuencias extensas del flujo del proceso. En particular, Claude Sonnet 4 y Claude Opus 4.1 destacan por su consistencia en esta métrica. Por su parte, Gemini 2.5 Pro se mantiene como un competidor muy cercano, al exhibir un equilibrio notable entre precisión y *recall*, lo que evidencia su capacidad para combinar una adecuada cobertura de los elementos del diagrama con un bajo nivel de generación de información irrelevante. Las Figuras 3.24 a la 3.27 permiten apreciar de forma clara el comportamiento global de los modelos bajo la métrica ROUGE-L. En ellas se mantiene la jerarquía observada en las métricas anteriores, con los modelos de Claude Sonnet 4 y Claude Opus 4.1 en las primeras posiciones y un segundo nivel de desempeño muy cercano liderado por Gemini 2.5 Pro, lo que refleja una competencia estrecha entre los modelos con mayor capacidad para preservar la coherencia secuencial.

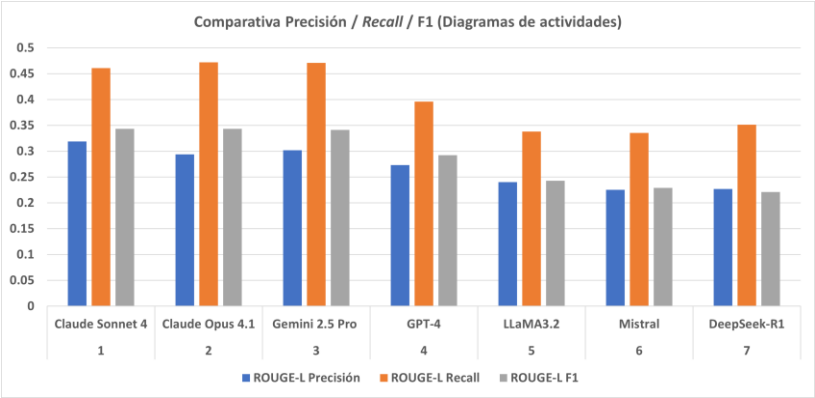


Figura 39 Comparación del desempeño en precisión, *recall* y F1 para ROUGE-L.

La Figura 3.24 muestra que Claude Sonnet 4 y Claude Opus 4.1 alcanzan los valores más altos de F1 en ROUGE-L, seguidos muy de cerca por Gemini 2.5 Pro. Estos resultados indican una mayor capacidad de estos modelos para conservar la estructura global de los diagramas de actividades. GPT-4 presenta un desempeño intermedio, mientras que LLaMA3.2, Mistral y DeepSeek-R1 registran los valores más bajos en las tres métricas.

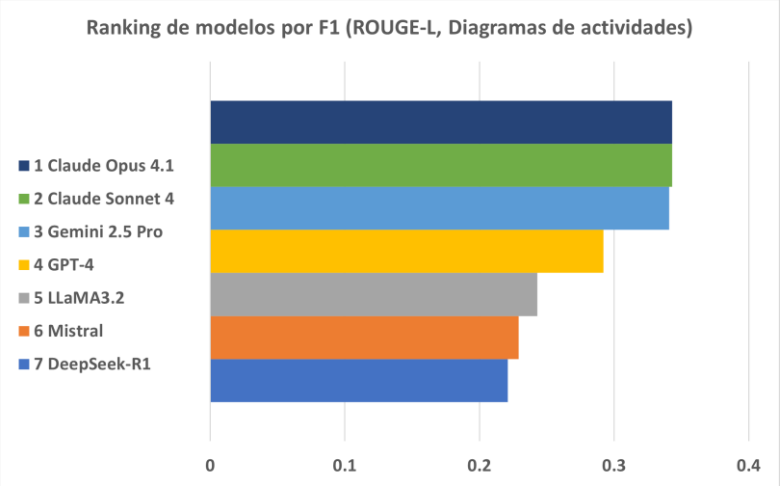


Figura 40 *Ranking* de los modelos con base en F1 para ROUGE-L.

La Figura 3.25 muestra que Claude Opus 4.1 y Claude Sonnet 4 encabezan el *ranking* en términos de la medida F1, seguidos muy de cerca por Gemini 2.5 Pro, lo que confirma el alto desempeño de estos modelos en la conservación de la estructura secuencial global de los diagramas de actividades. GPT-4 se ubica en un nivel intermedio, mientras que LLaMA3.2, Mistral y DeepSeek-R1 presentan los valores más bajos bajo esta métrica.

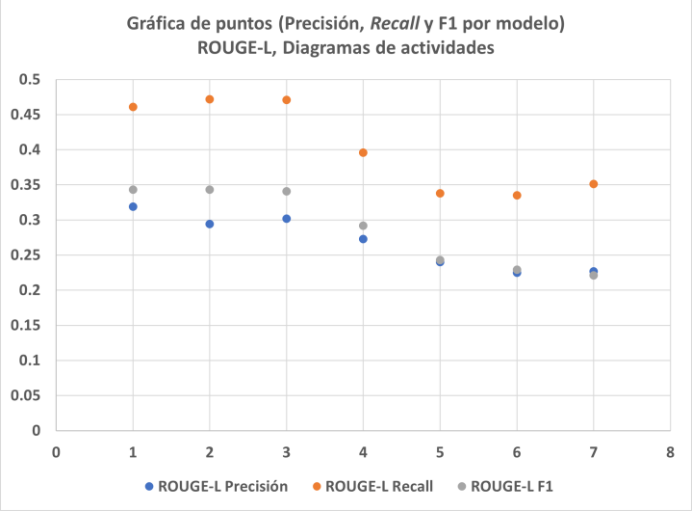


Figura 41 Precisión, *recall* y F1 por cada modelo para ROUGE-L.

La Figura 3.26 permite observar la distribución individual de las tres métricas para cada modelo sin superposición visual. Claude Sonnet 4, Claude Opus 4.1 y Gemini 2.5 Pro concentran los valores más altos de F1, con altos niveles de *recall* y precisión moderada. GPT-4 se ubica en un nivel intermedio, mientras que LLaMA3.2, Mistral y DeepSeek-R1 presentan los valores más bajos, lo que evidencia mayores limitaciones en la conservación de la estructura secuencial global de los diagramas de actividades.

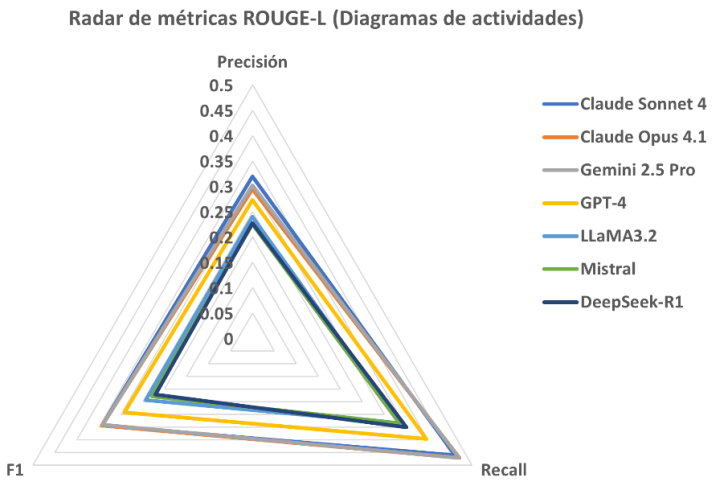


Figura 42 Radar comparativo de precisión, *recall* y F1 para ROUGE-L.

La Figura 3.27 muestra de forma conjunta el comportamiento de los modelos en las tres medidas bajo ROUGE-L. Claude Sonnet 4, Claude Opus 4.1 y Gemini 2.5 Pro presentan las áreas más amplias, lo que refleja su mayor capacidad para preservar la estructura global de los diagramas de actividades. GPT-4 se mantiene en un nivel intermedio, mientras que LLaMA3.2, Mistral y DeepSeek-R1 muestran áreas más reducidas, lo que evidencia un desempeño inferior en esta métrica.

3.3.4.10. Análisis global del desempeño de los LLMs en los diagramas de actividades

La Tabla 16 muestra el número de historias de usuario, tomadas del repositorio *Real World PlantUML*, en las que cada modelo obtuvo el mejor desempeño relativo. Gemini 2.5 Pro domina este criterio con 15 historias, equivalentes al 30% del total, lo que refleja un comportamiento estable y consistente en una amplia variedad de escenarios de modelado de procesos. Le siguen Mistral con 10 historias, equivalente al 20%, y Claude Sonnet 4 con 8 historias, correspondiente al 16%. Claude Opus 4.1 alcanza el mejor rendimiento en 7 historias, es decir el 14% del total, mientras que GPT-4 y DeepSeek-R1 lo hacen en 5 historias cada uno, ambos con el 10%. Resulta particularmente relevante que LLaMA3.2 no haya liderado en ninguna de las 50 historias evaluadas, lo cual evidencia limitaciones importantes en su desempeño relativo para este tipo de diagramas. Este análisis complementa los promedios globales al poner en evidencia el comportamiento de los modelos a nivel individual por historia de usuario.

Tabla 16. *Ranking* por historias de usuario donde cada modelo fue mejor.

Modelo	Cantidad de historias con mejor desempeño	Proporción del total de historias (%)
Gemini 2.5 Pro	15	30%
Mistral	10	20%
Claude Sonnet 4	8	16%
Claude Opus 4.1	7	14%
GPT-4	5	10%
DeepSeek-R1	5	10%
LLaMA3.2	0	0%

La Tabla 17 presenta el análisis de estabilidad del desempeño de los modelos, considerando la media, desviación estándar, valores mínimo y máximo, así como el rango de la medida F1, calculados a partir del conjunto de 50 diagramas generados. Gemini 2.5 Pro obtiene la media más alta con un

valor de 0.314, acompañado de una desviación estándar de 0.138, un valor mínimo de 0.116 y un valor máximo de 0.727, lo que da lugar a un rango de 0.611. Estos resultados indican un alto rendimiento promedio, aunque con una variabilidad moderada asociada a la diversidad de complejidad presente en las historias de usuario del repositorio evaluado. Claude Sonnet 4 y GPT-4 presentan medias ligeramente inferiores, de 0.308 y 0.300 respectivamente, con rangos también superiores a 0.600, lo que refleja un desempeño competitivo, pero con alta sensibilidad a la complejidad del proceso modelado. Claude Opus 4.1 muestra un comportamiento más estable, con una media de 0.289 y uno de los rangos más bajos entre los modelos líderes. Por su parte, LLaMA3.2 presenta la media más baja con 0.232, aunque con el menor rango de todos los modelos, 0.450, lo que refleja un comportamiento más estable pero limitado en términos de calidad global.

Tabla 17. Estabilidad por modelo de la medida F1.

Posición	Modelo	F1				
		Media	Desv. Est.	Min.	Máx.	Rango
1	Gemini 2.5 Pro	0.314	0.138	0.116	0.727	0.611
2	Claude Sonnet 4	0.308	0.129	0.091	0.838	0.747
3	GPT-4	0.300	0.128	0.112	0.728	0.615
4	Claude Opus 4.1	0.289	0.113	0.092	0.648	0.556
5	Mistral	0.271	0.124	0.079	0.795	0.716
6	DeepSeek-R1	0.254	0.110	0.079	0.674	0.594
7	LLaMA3.2	0.232	0.141	0.106	0.556	0.450

De manera general, los resultados obtenidos sobre el conjunto de 50 historias de usuario del repositorio *Real World PlantUML* indican que Gemini 2.5 Pro, Claude Sonnet 4 y Claude Opus 4.1 son los modelos con mejor desempeño para la generación automática de diagramas de actividades. Gemini 2.5 Pro destaca particularmente por liderar en la métrica ROUGE-2, por obtener la mayor cantidad de historias con mejor desempeño individual y por presentar la media más alta en el análisis de estabilidad. Los modelos de Anthropic sobresalen especialmente en la preservación de la estructura secuencial completa, como lo demuestra su liderazgo en ROUGE-L. En contraste, LLaMA3.2, Mistral y DeepSeek-R1 presentan limitaciones importantes tanto en los valores promedio como en su capacidad para dominar escenarios individuales, lo cual sugiere una menor adecuación para tareas de modelado de procesos complejos en contextos reales.

3.3.4.11. *Ranking* cuantitativo de los LLMs evaluados

A partir de los resultados obtenidos en las secciones anteriores, se elaboró un *ranking* cuantitativo de los modelos de lenguaje con el propósito de identificar, en cada tipo de diagrama, al modelo que alcanzó el mejor desempeño global con base en el promedio de la medida F1 de las métricas ROUGE-1, ROUGE-2 y ROUGE-L. Debido a las diferencias estructurales y semánticas entre los diagramas de clases y de actividades, el análisis se realizó de manera independiente para cada caso.

En el caso de los diagramas de clases, los resultados mostrados en la Tabla 18 indican que Gemini 2.5 Pro ocupó la primera posición con un promedio F1 de 0.244, superando al resto de los modelos en el desempeño global. Este posicionamiento se sustentó en sus valores más altos en ROUGE-1 con 0.334 y ROUGE-L con 0.279, así como en su mejor resultado en ROUGE-2 con 0.120. En conjunto, estos valores reflejan una mayor capacidad para recuperar los elementos fundamentales del diagrama y preservar su estructura general en comparación con los demás modelos evaluados.

Tabla 18. Promedio F1 de la métrica ROUGE de los diagramas de actividades.

Posición	Modelo	F1			
		ROUGE-1	ROUGE-2	ROUGE-L	Promedio
1	Gemini 2.5 Pro	0.334	0.120	0.279	0.244
2	LLaMA3.2	0.286	0.093	0.242	0.207
3	GPT-4	0.274	0.094	0.236	0.201
4	Claude Sonnet 4	0.269	0.085	0.224	0.193
5	Claude Opus 4.1	0.273	0.080	0.221	0.191
6	Mistral	0.267	0.081	0.223	0.190
7	DeepSeek-R1	0.229	0.061	0.191	0.160

Por su parte, en los diagramas de actividades, la Tabla 19 muestra que Claude Sonnet 4 alcanzó la primera posición con un promedio F1 de 0.301, resultado que se explicó principalmente por sus altos valores en ROUGE-1 con 0.428 y ROUGE-L con 0.343. Estos indicadores evidenciaron una mayor capacidad para conservar tanto los términos relevantes como la secuencia global de las actividades dentro del diagrama.

Tabla 19. Promedio F1 de la métrica ROUGE de los diagramas de actividades.

Posición	Modelo	F1			
		ROUGE-1	ROUGE-2	ROUGE-L	Promedio
1	Claude Sonnet 4	0.428	0.132	0.343	0.301
2	Gemini 2.5 Pro	0.415	0.144	0.341	0.300
3	Claude Opus 4.1	0.408	0.118	0.343	0.290
4	GPT-4	0.346	0.117	0.292	0.252
5	LLaMA3.2	0.286	0.104	0.243	0.211
6	Mistral	0.266	0.087	0.229	0.194
7	DeepSeek-R1	0.238	0.103	0.221	0.187

De acuerdo con los resultados obtenidos, el *ranking* cuantitativo permitió identificar que Gemini 2.5 Pro fue el modelo con mejor desempeño en la generación de diagramas de clases, mientras que Claude Sonnet 4 lideró en la generación de diagramas de actividades. Esta diferencia confirma que el comportamiento de los modelos no es uniforme para ambos tipos de diagramas y que su desempeño depende del tipo de representación que se busca generar. Con base en estos resultados, ambos modelos se seleccionaron para ser integrados en la herramienta de generación automática de diagramas UML.

3.3.5. Evaluación cualitativa de los diagramas generados

La evaluación cuantitativa basada en la métrica de ROUGE permitió medir el grado de coincidencia léxica y estructural entre los diagramas generados por los LLMs antes mencionados y los diagramas de referencia. No obstante, este tipo de métricas no es suficiente para valorar en su totalidad la calidad conceptual, la comprensibilidad cognitiva ni la utilidad práctica de los diagramas UML generados. Por esta razón, se realizó una evaluación cualitativa complementaria, orientada al análisis de los aspectos semánticos, perceptuales y funcionales de los diagramas, con el objetivo de obtener una valoración integral de la calidad de los diagramas de clases generados por Gemini 2.5 Pro y de los diagramas de actividades generados por Claude Sonnet 4.

3.3.5.1. Criterios de evaluación cualitativa

La evaluación cualitativa se realizó mediante un conjunto de 12 criterios, aplicables tanto a diagramas de clases como a diagramas de actividades. Estos criterios se agruparon en cuatro

dimensiones analíticas: claridad y consistencia visual, comprensión cognitiva, calidad estructural y semántica, e integración y utilidad. Cada dimensión integra aspectos complementarios que permiten evaluar los diagramas desde una perspectiva integral, considerando tanto su correcta construcción conforme a la notación de diagramas UML, como su legibilidad, comprensión por parte del usuario y su adecuación al propósito para el cual fueron generados. Los criterios de evaluación definidos se sustentan en principios teóricos sobre diseño de notaciones visuales, comprensión cognitiva de diagramas y evaluación de modelos, propuestos principalmente por Malinova y Mendling (2021), Moody (2009), Storrle (2011) y Storrle (2018). En la Tabla 20 se presentan las dimensiones consideradas, los criterios de evaluación asociados, sus definiciones operativas y las referencias teóricas que respaldan su selección.

Tabla 20. Criterios para la evaluación cualitativa de diagramas UML.

Dimensión	Criterio de evaluación	Definición	Referencia
1. Claridad y consistencia visual	Claridad semiótica	Evalúa si cada símbolo del diagrama representa un único concepto sin ambigüedades.	(Moody, 2009)
	Discriminabilidad perceptiva	Mide si los elementos del diagrama son fácilmente distinguibles entre sí (formas, líneas, colores, disposición).	(Moody, 2009; Storrle, 2011)
		Evalúa el uso intencional de variables visuales (forma, tamaño, color, orientación, textura) para mejorar la comprensión.	(Moody, 2009)
	Simplicidad	Determina si el diagrama utiliza solo los elementos necesarios, evitando redundancia o sobrecarga visual.	(Moody, 2009)
2. Comprensión cognitiva	Comprensibilidad	Evalúa qué tan fácil es para un lector humano entender el diagrama sin explicaciones adicionales.	(Malinova y Mendling, 2021; Storrle, 2018)
	Carga cognitiva	Mide el esfuerzo mental requerido para interpretar el diagrama; se reduce si la estructura es clara y jerárquica.	(Malinova y Mendling, 2021; Storrle, 2018)
	Memorabilidad	Analiza si el diagrama facilita recordar los conceptos o relaciones representadas.	(Malinova y Mendling, 2021; Storrle, 2018)
3. Calidad estructural y semántica	Corrección sintáctica	Verifica que el diagrama cumpla con las reglas formales de UML (tipos de relaciones, atributos, jerarquías).	(Malinova y Mendling, 2021)
	Consistencia semántica	Evalúa si el contenido del diagrama refleja fielmente el dominio o las historias de usuario.	(Malinova y Mendling, 2021)
	Completitud	Determina si el diagrama incluye todos los elementos esenciales del dominio, sin omisiones importantes.	(Malinova y Mendling, 2021)
4. Integración y utilidad	Integración cognitiva	Evalúa la coherencia entre diferentes diagramas o vistas (por ejemplo, entre clases y actividades).	(Malinova y Mendling, 2021)
	Adecuación al usuario	Mide qué tan apropiado es el diagrama para su propósito (análisis, comunicación o diseño) y para la audiencia que lo usará.	(Malinova y Mendling, 2021)

Para la valoración de cada uno de los criterios definidos en el análisis cualitativo se empleó una escala de *Likert* de cinco niveles, ampliamente utilizada en estudios de evaluación en ingeniería de software por su capacidad para medir percepciones y juicios de manera estructurada y comparable (Clason y Dormody, 1994; Joshi et al., 2015; Lina y de Mello, 2015). Esta escala permitió estandarizar la asignación de puntajes en función del grado de cumplimiento de los criterios en los diagramas generados. La Tabla 21 presenta la interpretación de los valores de la escala, comprendidos entre 0 y 4, desde el nivel Deficiente hasta Excelente, lo que posibilitó expresar los juicios cualitativos mediante valores numéricos comparables.

Tabla 21. Interpretación de los valores de la escala de Likert.

Valor	Nivel de calidad	Interpretación
0	Deficiente	No cumple el criterio evaluado.
1	Bajo	Cumple parcialmente el criterio, requiere interpretación adicional.
2	Aceptable	Cumple los aspectos básicos del criterio, pero presenta errores menores.
3	Bueno	Cumple adecuadamente el criterio; el diagrama es claro y funcional.
4	Excelente	Cumple totalmente el criterio; el diagrama es preciso, coherente y comprensible.

La utilización de esta escala permitió evaluar de manera homogénea todos los diagramas generados por los distintos LLMs, tanto para los diagramas de clases como para los diagramas de actividades. La asignación de un valor numérico a cada criterio facilitó el análisis comparativo del desempeño cualitativo de los modelos, así como la identificación de fortalezas y áreas de oportunidad en los diagramas generados, contribuyendo a una valoración integral de su calidad desde las dimensiones visual, cognitiva, estructural y funcional.

3.3.5.2. Resultados cualitativos para los diagramas de clases

La evaluación cualitativa de los diagramas de clases generados se llevó a cabo con el propósito de analizar su calidad desde una perspectiva integral, considerando no solo su estructura formal, sino también la forma en que estos son percibidos, comprendidos y utilizados. Dicha evaluación fue realizada por un experto en SE con conocimientos en modelado UML, quien empleó un instrumento de evaluación basado en los 12 criterios definidos y en sus correspondientes valores de ponderación, previamente descritos en la sección anterior. A partir de las valoraciones obtenidas en la evaluación cualitativa realizada por el experto sobre los 68 diagramas de clases analizados, se calcularon las medias por dimensión con el propósito de obtener una visión general del desempeño cualitativo. Los resultados se presentan en la Tabla 22, que sintetiza la calidad alcanzada en cada una de las dimensiones evaluadas; el detalle completo de las valoraciones puede consultarse en:

https://github.com/LoreMartinez/Repositorio_Tesis_DiagramasUML_Archivos/blob/main/Archivos_evaluacion_LLMs/Resultados_evaluacion_cualitativa_diagramas_clases.xlsx

Tabla 22. Resultados cualitativos por dimensión en los diagramas de clases.

Dimensión	Media
1. Claridad y consistencia visual	2.92
2. Comprensión cognitiva	2.64
3. Calidad estructural y semántica	2.05
4. Integración y utilidad	2.21

Los resultados mostraron que la dimensión de claridad y consistencia visual fue la que presentó el mejor desempeño, con una media de 2.92. Esto indicó que, en términos generales, los diagramas fueron percibidos como visualmente comprensibles, con una adecuada identificación de los símbolos UML y una organización gráfica coherente, lo cual refleja una correcta interpretación sintáctica por parte de los modelos generadores. No obstante, persistieron oportunidades de mejora en aspectos como la expresividad visual y la simplicidad de algunos diagramas.

En segundo lugar, la dimensión de comprensión cognitiva alcanzó una media de 2.64, lo que reflejó que los diagramas resultaron mayormente comprensibles para el usuario, con una carga cognitiva razonablemente adecuada. Sin embargo, la baja puntuación observada en el criterio de memorabilidad evidenció que, aun cuando los diagramas lograron comunicar la información básica, su retención por parte del usuario resultó limitada, lo que sugiere dificultades en la consolidación cognitiva de los modelos generados automáticamente.

La dimensión de integración y utilidad presentó una media de 2.21, lo que evidenció un nivel intermedio en cuanto a la adecuación de los diagramas para apoyar actividades de análisis, diseño y documentación del software. Este resultado indicó que, si bien los diagramas generados pudieron ser utilizados como apoyo funcional, su nivel de integración dentro de un proceso de desarrollo real aún mostró ciertas restricciones. Finalmente, la dimensión con menor desempeño fue la de calidad estructural y semántica, con una media de 2.05, lo que puso de manifiesto la presencia de deficiencias relacionadas con la consistencia semántica y la completitud de los modelos. Estas limitaciones revelaron que, aunque la estructura sintáctica de los diagramas fue correcta, los modelos presentaron dificultades para representar de manera precisa y completa las relaciones conceptuales propias del dominio descrito en las historias de usuario.

Con el fin de profundizar en el comportamiento de cada indicador, la Tabla 23 muestra el análisis detallado por criterio, incorporando las medidas de media, mediana, valor mínimo y valor máximo. Este desglose permitió identificar con mayor claridad los aspectos mejor logrados, como la claridad semiótica y la corrección sintáctica, que alcanzaron valores promedio de 4.00, lo cual evidenció una adecuada utilización de la notación UML. En contraste, los criterios con mayor debilidad correspondieron a la consistencia semántica, con una media de 0.78, y a la completitud, con una media de 1.57, lo que reflejó dificultades recurrentes para representar de forma íntegra y coherente los conceptos del dominio.

Tabla 23. Resultados cualitativos por criterio en los diagramas de clases.

Dimensión	Criterio	Media	Mediana	Mínimo	Máximo
1. Claridad y consistencia visual	Claridad semiótica	4.00	4.00	4	4
	Discriminabilidad perceptiva	2.68	3.00	2	4
	Expresividad visual	2.04	2.00	2	3
	Simplicidad	2.94	3.00	1	3
2. Comprensión cognitiva	Comprensibilidad	3.00	3.00	3	3
	Carga cognitiva	3.91	4.00	2	4
	Memorabilidad	1.00	1.00	1	1
3. Calidad estructural y semántica	Corrección sintáctica	4.00	4.00	4	4
	Consistencia semántica	0.71	0.00	0	4
	Completitud	1.44	1.00	0	4

Dimensión	Criterio	Media	Mediana	Mínimo	Máximo
4. Integración y utilidad	Integración cognitiva	2.13	2.00	1	3
	Adecuación al usuario	2.29	2.00	2	4

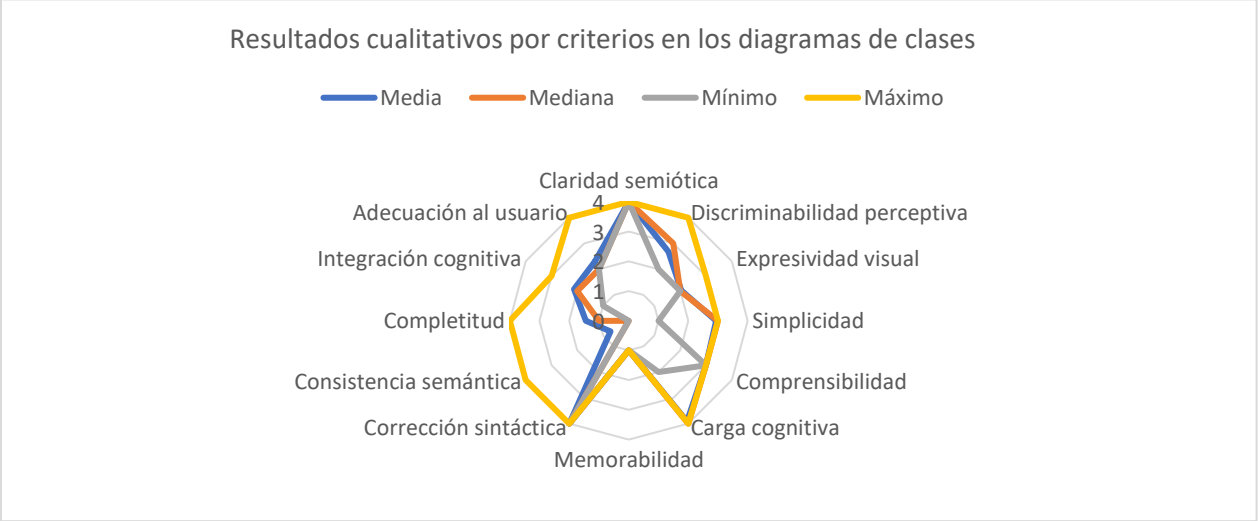


Figura 43 Resultados cualitativos por criterio en los diagramas de clases.

De manera complementaria, la Figura 3.28 presenta una representación gráfica de los valores promedio por criterio, lo que facilitó la comparación visual del desempeño entre los distintos indicadores de calidad evaluados y permitió identificar de forma inmediata los contrastes entre los aspectos mejor consolidados, asociados principalmente a la correcta aplicación de la sintaxis UML, y aquellos que requirieron un mayor fortalecimiento, relacionados con la interpretación semántica y la completitud de los modelos generados.

De manera general, la evaluación realizada por el experto en Ingeniería de Software con experiencia en modelado UML indicó que los diagramas de clases generados presentaron valoraciones favorables en aspectos como la claridad visual, el uso adecuado de la notación UML y la comprensibilidad, lo que facilitó su lectura e interpretación durante las etapas de análisis y diseño. En la mayoría de los casos, la identificación de clases, atributos y relaciones se consideró adecuada y la carga cognitiva se mantuvo en niveles que permitieron una comprensión razonable por parte del usuario.

En paralelo, la evaluación cuantitativa basada en las métricas ROUGE para los diagramas de clases mostró valores moderados en términos de coincidencia léxica y preservación de la estructura global, especialmente en ROUGE-2 y ROUGE-L, lo que sugirió que las salidas de los modelos no siempre reprodujeron con exactitud el texto de referencia. Sin embargo, al contrastar estos resultados con la evaluación cualitativa, fue posible apreciar que, desde una perspectiva perceptual, cognitiva y funcional, muchos de los diagramas de clases ofrecieron un nivel de calidad aceptable para apoyar tareas de documentación y análisis. Por otra parte, los criterios de consistencia semántica, completitud y memorabilidad obtuvieron las valoraciones más bajas, en coherencia con las limitaciones observadas en las métricas automáticas, lo que ayudó a identificar con mayor precisión aquellos aspectos puntuales susceptibles de mejora mediante la revisión y ajuste por parte de un experto humano.

3.3.5.3. Resultados cualitativos para los diagramas de actividades

A partir de las valoraciones obtenidas sobre el conjunto de los 50 diagramas de actividades analizados, se calcularon las medias por dimensión con el propósito de obtener una visión general del desempeño cualitativo. Los resultados obtenidos se presentan en la Tabla 24, que sintetiza el nivel de calidad alcanzado en cada dimensión; el detalle completo de las valoraciones puede consultarse en:

https://github.com/LoreMartinez/Repositorio_Tesis_DiagramasUML_Archivos/blob/main/Archivos_evaluacion_LLMs/Resultados_evaluacion_cualitativa_diagramas_actividades.xlsx

Tabla 24. Resultados cualitativos por dimensión en los diagramas de actividades.

Dimensión	Media
1. Claridad y consistencia visual	2.19
2. Comprensión cognitiva	1.97
3. Calidad estructural y semántica	2.21
4. Integración y utilidad	3.30

Los resultados indicaron que la dimensión de integración y utilidad presentó el mejor desempeño, con una media de 3.30, lo que reflejó que, desde la valoración del experto, los diagramas de actividades resultaron funcionales para apoyar la comprensión general de los procesos representados. Este comportamiento evidenció una adecuada integración de los flujos principales y una correcta adecuación al usuario en términos de interpretación de los procesos. Las dimensiones de calidad estructural y semántica y de claridad y consistencia visual alcanzaron medias de 2.21 y 2.19 respectivamente, lo que reflejó un desempeño intermedio tanto en la representación visual de los elementos como en la coherencia estructural de las historias de usuario modelados. Los diagramas mostraron una identificación adecuada de los símbolos principales, aunque presentaron limitaciones en la discriminabilidad perceptiva. En cuanto a la estructura y semántica, si bien la corrección sintáctica fue adecuada en la mayoría de los casos, persistieron debilidades en la consistencia semántica y en la completitud de algunos flujos de actividades. Por su parte, la dimensión de comprensión cognitiva obtuvo la media más baja, con un valor de 1.97, lo que puso de manifiesto mayores dificultades para facilitar la interpretación y retención de las secuencias de actividades por parte del usuario. Este comportamiento estuvo influido principalmente por las bajas puntuaciones observadas en el criterio de memorabilidad.

Con el fin de profundizar en el comportamiento de cada indicador, la Tabla 25 presenta el análisis detallado por criterio, incorporando las medidas de media, mediana, valor mínimo y valor máximo. Este desglose permitió identificar con mayor precisión los aspectos mejor valorados, así como aquellos que concentraron las principales dificultades en la generación automática de los diagramas de actividades.

Tabla 25. Resultados cualitativos por criterio en los diagramas de actividades.

Dimensión	Criterio	Media	Mediana	Mínimo	Máximo
1. Claridad y consistencia visual	Claridad semiótica	4.00	4.00	4	4
	Discriminabilidad perceptiva	1.00	1.00	1	1
	Expresividad visual	1.90	2.00	1	3
	Simplicidad	2.00	2.00	2	2

Dimensión	Criterio	Media	Mediana	Mínimo	Máximo
2. Comprensión cognitiva	Comprensibilidad	2.00	2.00	2	2
	Carga cognitiva	2.94	3.00	2	3
	Memorabilidad	1.00	1.00	1	1
3. Calidad estructural y semántica	Corrección sintáctica	3.86	4.00	3	4
	Consistencia semántica	0.00	0.00	0	0
	Completitud	2.82	4.00	0	4
4. Integración y utilidad	Integración cognitiva	4.00	4.00	4	4
	Adecuación al usuario	2.70	3.00	2	3

El análisis por criterio mostró que la claridad semiótica y la integración cognitiva alcanzaron las medias más altas, ambas con un valor de 4.00, lo que reflejó que los símbolos UML y la estructura general de los flujos fueron correctamente interpretados por el evaluador. Asimismo, la corrección sintáctica presentó una media elevada de 3.86, lo que indicó un uso adecuado de la notación formal en la mayoría de los diagramas.

En contraste, los valores más bajos se registraron en la consistencia semántica, con una media de 0.00, y en la memorabilidad, con una media de 1.00, lo que evidenció dificultades importantes para mantener la coherencia conceptual de los procesos y para facilitar la retención de la información representada. La completitud, con una media de 2.82, mostró un comportamiento intermedio, lo que sugirió que varios diagramas representaron los procesos de forma parcial, aunque incorporando los elementos principales.

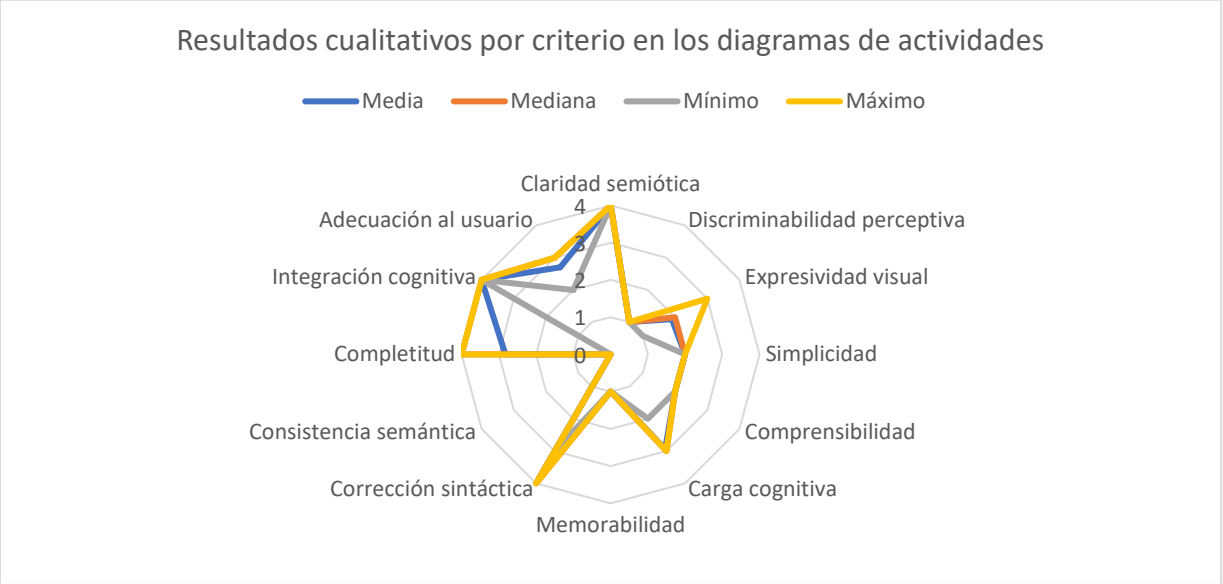


Figura 44 Resultados cualitativos por criterio en los diagramas de actividades.

La Figura 3.29 presenta de manera visual los valores promedio por criterio, lo que facilitó la comparación entre los distintos indicadores de calidad evaluados y permitió identificar con mayor claridad los contrastes entre los aspectos mejor consolidados, asociados principalmente a la claridad

semiótica, la integración cognitiva y la corrección sintáctica, y aquellos que requirieron un mayor fortalecimiento, relacionados con la consistencia semántica y la memorabilidad.

De esta manera los resultados de la evaluación cualitativa permitieron observar que, aunque las métricas cuantitativas no alcanzaron valores elevados en términos absolutos, los diagramas de clases y de actividades generados por Gemini 2.5 Pro presentaron un nivel de calidad aceptable desde una perspectiva visual, cognitiva y funcional. La valoración realizada por el experto destacó de forma positiva la correcta aplicación de la notación UML, la claridad en la identificación de los elementos principales y la organización general de las estructuras y secuencias representadas, aspectos que favorecieron una comprensión global adecuada de los diagramas. Asimismo, se observó que, en muchos casos, los modelos lograron representar de manera coherente los componentes esenciales de las clases y los flujos básicos de las actividades, aun cuando no siempre se alcanzó una correspondencia exacta con los diagramas de referencia.

Estos resultados mostraron que, pese a que la similitud textual medida por las métricas automáticas fue moderada, los diagramas generados resultaron comprensibles y funcionales desde un punto de vista práctico. Al mismo tiempo, las debilidades detectadas en criterios como la consistencia semántica, la completitud y la memorabilidad permitieron identificar con mayor precisión aquellos aspectos que requieren un mayor nivel de refinamiento. En este sentido, la evaluación cualitativa aportó información que no es capturada por las métricas automáticas, lo que reafirmó su papel como un complemento necesario para lograr una valoración más integral y equilibrada de la calidad de los diagramas generados.

3.3.6. Selección del LLM con mejor desempeño para su integración en la herramienta

La selección del modelo de lenguaje a integrar en la herramienta se realizó con base únicamente en los resultados de la evaluación cuantitativa, en la cual se compararon siete modelos mediante métricas automáticas aplicadas a los diagramas de clases y de actividades generados. Esta comparación permitió analizar, de manera objetiva, el nivel de similitud léxica y estructural entre los diagramas generados automáticamente y los diagramas de referencia, a partir de las medidas de precisión, *recall* y F1 de la métrica ROUGE.

Los resultados cuantitativos mostraron diferencias consistentes entre los modelos evaluados, lo que permitió establecer con claridad su desempeño relativo. En este contexto, Gemini 2.5 Pro destacó de forma reiterada al presentar los valores más altos en las métricas globales, así como el mejor desempeño promedio y una mayor estabilidad en los resultados obtenidos a lo largo de las distintas historias de usuario analizadas. De igual manera, este modelo encabezó el número de casos en los que se registró el mejor desempeño relativo, tanto en los diagramas de clases como en los diagramas de actividades, lo que reflejó un comportamiento más consistente ante diferentes escenarios de modelado.

Sin embargo, aun cuando Gemini 2.5 Pro fue el modelo con mejor desempeño en términos comparativos, los valores absolutos alcanzados en la evaluación cuantitativa no resultaron elevados de manera general, considerando que el valor máximo posible de las métricas es 1.0. Esta situación evidenció que las métricas automáticas, si bien son útiles para establecer comparaciones objetivas entre modelos, no resultan suficientes por sí solas para valorar en profundidad la calidad conceptual, perceptual y funcional de los diagramas generados.

Por esta razón, y con base en los resultados cuantitativos obtenidos, Gemini 2.5 Pro fue seleccionado como el modelo a integrar en la herramienta, y posteriormente se aplicó de manera exclusiva una evaluación cualitativa a los diagramas generados por este modelo, con el propósito de reforzar e interpretar con mayor profundidad los hallazgos obtenidos mediante las métricas automáticas. Esta fase permitió analizar aspectos que no pueden ser capturados completamente por la

evaluación cuantitativa, como la claridad visual, la comprensibilidad, la coherencia semántica y la utilidad práctica de los diagramas, desde la perspectiva de un experto en Ingeniería de Software con experiencia en modelado UML.

De este modo, la selección del modelo quedó respaldada por indicadores objetivos de desempeño automático, mientras que la evaluación cualitativa permitió complementar estos resultados y ofrecer una visión más completa sobre la calidad de los diagramas producidos por el modelo finalmente integrado, fortaleciendo así el rigor del proceso de decisión.

4. Implementación de la propuesta de solución

En este capítulo se presenta el desarrollo de la propuesta de solución para la problemática que se aborda en este trabajo de investigación, iniciando con la presentación de cada una de las prácticas seleccionadas para pertenecer al conjunto de prácticas que serán integradas en los eventos de Planeación del *Sprint* y Ejecución del *Sprint*.

Posteriormente, se describe el proceso de desarrollo de la herramienta para la generación automática de diagramas UML, como apoyo a las practicas que se proponen. Este desarrollo se llevó a cabo de forma iterativa a lo largo de cuatro S*prints*. En cada iteración se implementaron nuevas funcionalidades o se mejoraron las existentes, avanzando progresivamente hacia una solución más completa y funcional. Esta metodología de trabajo permitió realizar ajustes continuos en función de los requisitos emergentes, evaluar el progreso de forma constante y mantener la alineación con los objetivos planteados desde el inicio del proyecto.

4.1. Propuesta del conjunto de prácticas

En el desarrollo ágil de *software*, la comprensión de las historias de usuario es un factor clave para garantizar que el equipo Scrum tenga una visión clara de los requisitos del producto de *software*. Sin embargo, en la práctica, los miembros del equipo enfrentan dificultades al tratar de comprender estas historias de usuario, lo que puede derivar en malos entendidos y errores al momento de su implementación (Pikkarainen et al., 2008). Dado que Scrum enfatiza la colaboración y la adaptabilidad, es fundamental contar con prácticas que faciliten una comunicación efectiva entre los miembros del equipo Scrum. En este contexto, la incorporación de los diagramas UML generados de forma automática busca mejorar la comprensión de las historias de usuario al proporcionar una representación visual clara que ayude a aclarar ciertas dudas.

El conjunto de prácticas propuesto busca fortalecer la Planeación del *Sprint* y la Ejecución del *Sprint*, mediante la generación y uso de los diagramas UML. Con esto se pretende que el equipo de Scrum pueda comprender mejor las historias de usuario, facilitando de esta manera la comunicación entre los miembros del equipo durante el proceso de desarrollo del producto de *software*.

La selección de estas prácticas está fundamentada en la revisión bibliográfica sobre las prácticas presentes en las metodologías ágiles, enfocándose en la metodología de Scrum. Recordando que la selección de las prácticas fue en base a los criterios de selección mencionados en el punto siete de la sección 3.2.1.1 del Capítulo 3 de esta tesis. Las prácticas apoyadas en una herramienta, la cual tiene como objetivo la generación de diagramas UML, específicamente diagramas de clases inicial y diagramas de actividades inicial, busca dar soporte a la implementación de las prácticas sin perder el enfoque ágil de Scrum.

Después de realizar una revisión bibliográfica sobre las buenas prácticas existentes en la metodología Scrum y un análisis del estado del arte, se diseñó un conjunto de prácticas. Este proceso de investigación permitió identificar las prácticas más relevantes y efectivas, adaptándolas a las necesidades específicas de los equipos de desarrollo y su interacción con el Dueño del Producto. La integración de estas prácticas en el marco de Scrum busca no solo mejorar la comprensión de las historias de usuario, sino también mejorar la comunicación y colaboración dentro del equipo Scrum, asegurando una mayor eficiencia y alineación durante el ciclo del *Sprint*.

4.1.1. Prácticas priorizadas

Después de realizar un análisis de cada una de las prácticas que se muestran en la Tabla 26, buscando su alineación con el objetivo de este trabajo de investigación, se seleccionaron las primeras cinco prácticas para formar parte del conjunto propuesto, comenzando con una precondición esencial para garantizar el éxito de las siguientes. Estas prácticas se enfocan en aspectos clave como la definición clara de las historias de usuario y la generación automática de diagramas UML, herramientas fundamentales para que todos los miembros del equipo comprendan mejor el producto y el flujo de trabajo durante la Planeación del *Sprint* y la Ejecución del *Sprint*. También se incluyen prácticas que buscan asegurar una documentación adecuada y un uso efectivo de los diagramas generados, para fomentar la colaboración y facilitar la toma de decisiones dentro del equipo Scrum.

Tabla 26. Listado de prácticas priorizadas de acuerdo con el objetivo de esta tesis.

Fase - Rol	Práctica	Prioridad (alta, media y baja)
Planeación del Producto - PO	Definición clara de historias de usuario.	**ALTA**
Planeación del Producto - PO	Generación automática de diagramas de clase inicial.	**ALTA**
Planeación del *Sprint* - PO	Generación automática de diagramas de actividades.	**ALTA**
Planeación del Producto - DT	Uso del diagrama de clase inicial para comprender las historias de usuario.	**ALTA**
Planeación del *Sprint* - DT	Uso de diagrama de actividades para entender el flujo de trabajo.	**ALTA**
Planeación y Ejecución del *Sprint* - PO y DT	Realizar documentación con los diagramas UML generados.	**MEDIA**
Ejecución del *Sprint* - PO	Uso de diagramas generados para clarificar las historias de usuario o el flujo de trabajo.	**MEDIA**
Ejecución del *Sprint* - SM	Facilitar la generación de diagramas y su uso.	**MEDIA**
Ejecución del *Sprint* - SM	Monitoreo del uso efectivo de los diagramas generados.	**MEDIA**
Ejecución del *Sprint* - DT	Revisar y actualizar los diagramas según sea necesario.	**MEDIA**

Fase - Rol	Práctica	Prioridad (alta, media y baja)
Revisión del *Sprint* - PO	Actualización de diagramas según la retroalimentación recibida.	**BAJA**
Retrospectiva del *Sprint* - SM	Facilitar discusiones sobre mejoras en la documentación.	**BAJA**

4.1.2. Descripción del conjunto de prácticas

En el marco de trabajo de Scrum, se ha seleccionado un conjunto de prácticas que buscan fortalecer la comprensión de las historias de usuario y mejorar la comunicación entre los miembros del equipo mediante el uso de diagramas UML generados automáticamente. Estas prácticas se integrarán en dos eventos fundamentales del ciclo Scrum: la Planeación del *Sprint* y la Ejecución del *Sprint*. Los roles clave involucrados en su implementación son el Dueño del Producto y el Equipo de Desarrollo, debido a que estos roles son los que se involucran directamente con la lista de historias de usuario. Ambos roles trabajarán de manera colaborativa para garantizar que estas prácticas contribuyan de manera efectiva y eficiente al avance del proyecto, facilitando la coordinación y alineación de los miembros del equipo para alcanzar los objetivos establecidos del *Sprint*. Es importante señalar que las prácticas seleccionadas no trabajan de manera aislada, sino que están diseñadas para complementarse y reforzarse mutuamente a lo largo del ciclo del *Sprint*.

Precondición: Definición clara de las historias de usuario

Antes de realizar la integración de las prácticas propuestas, es esencial que las historias de usuario sean definidas de manera precisa, clara y comprensible para todos los miembros del equipo. Las historias de usuario mal estructuradas o ambiguas pueden generar confusión y malentendidos, afectando negativamente la calidad de los diagramas UML generados por la herramienta. Si las historias de usuario no se redactan correctamente, los diagramas de clases inicial y los diagramas de actividades iniciales no podrán reflejar de mejor manera las relaciones entre los componentes del sistema y el flujo de trabajo. Una historia de usuario bien definida no solo facilita la creación de diagramas UML claros y precisos, sino que también permite al equipo de desarrollo tener una visión más clara de los requisitos del sistema. Además, contribuye a una mejor organización del trabajo y asegura que las interacciones entre los distintos componentes del sistema sean adecuadamente reflejadas en los diagramas. Por último, contar con historias de usuario claramente definidas facilita la comprensión de las funcionalidades a implementar, mejora la coordinación dentro del equipo y optimiza la ejecución del *Sprint*.

Con el objetivo de guiar de manera estructurada la incorporación de buenas prácticas en las actividades del equipo de desarrollo, se definió un conjunto de prácticas organizadas según su propósito, momento de aplicación y responsables de ejecución. Estas prácticas están alineadas con los eventos del marco Scrum, particularmente en las fases de Planeación y Ejecución del *Sprint*, y buscan mejorar la comprensión de las historias de usuario mediante el uso de artefactos como los diagramas UML. Para la documentación de cada práctica se emplea una tabla individual que detalla aspectos fundamentales como su objetivo, entradas, resultados, criterios de verificación y otros elementos esenciales para su correcta implementación. Esta forma de estructurar la información se basa en una adaptación de la plantilla metodológica propuesta por Hanna Oktaba en su trabajo Kuali-BEH, la cual facilita una descripción ordenada y sistemática de buenas prácticas en el desarrollo de *software* (Oktaba, 2006).

A continuación, en las Tablas 27 a la 31 se presentan las cinco prácticas que conformaron el conjunto de prácticas resultante propuesto en esta investigación.

Tabla 27. Práctica 1: Generación automática de diagramas de clases inicial.

<table>
<tr><td>BP-01</td><td colspan="3">Práctica</td></tr>
<tr><td colspan="4">Nombre</td></tr>
<tr><td colspan="4">Generación automática de diagramas de clases inicial</td></tr>
<tr><td>Fase:</td><td>Planeación del Producto</td><td>Rol del participante:</td><td>Dueño del Producto</td></tr>
<tr><td colspan="4">Objetivo</td></tr>
<tr><td colspan="4">Proporcionar al Dueño del Producto una representación estructurada de los elementos principales del sistema al inicio de la planificación para facilitar la selección de historias de usuario durante la planificación del Sprint.</td></tr>
<tr><td colspan="2">Entrada</td><td colspan="2">Resultado</td></tr>
<tr><td colspan="2">Historia de usuario seleccionada del Sprint Backlog</td><td colspan="2">Diagrama de clases inicial generado automáticamente</td></tr>
<tr><td colspan="4">Criterios de verificación</td></tr>
<tr><td colspan="4">• El diagrama representa correctamente las entidades y relaciones clave descritas en la historia de usuario
• El equipo de desarrollo entiende la información reflejada en el diagrama
• El diagrama fue utilizado como apoyo durante la planificación del Sprint</td></tr>
<tr><td colspan="4">Guía</td></tr>
<tr><td>Actividad</td><td colspan="3">Generación del diagrama de clases inicial</td></tr>
<tr><td colspan="2">Entrada</td><td colspan="2">Salida</td></tr>
<tr><td colspan="2">Historia de usuario seleccionada del Sprint Backlog</td><td colspan="2">Diagrama de clases inicial generado automáticamente</td></tr>
<tr><td>Tareas (opcional)</td><td>Herramientas (opcional)</td><td>Conocimientos y Habilidades</td><td>Métricas (opcional)</td></tr>
<tr><td>1. Seleccionar historia de usuario
2. Ingresar la historia en la herramienta
3. Ejecutar la generación del diagrama</td><td>Herramienta de generación automática de diagramas UML</td><td>• Lectura y análisis de historias de usuario
• Interpretación de diagrama de clases inicial UML</td><td>• Tiempo promedio de generación
• Número de elementos correctamente representados</td></tr>
<tr><td>Actividad</td><td colspan="3">Revisión y aprobación del diagrama de clases inicial generado</td></tr>
<tr><td colspan="2">Entrada</td><td colspan="2">Salida</td></tr>
<tr><td colspan="2">Diagrama de clases inicial generado</td><td colspan="2">Diagrama revisado, aprobado y compartido con el equipo de desarrollo</td></tr>
<tr><td>Tareas (opcional)</td><td>Herramientas (opcional)</td><td>Conocimientos y Habilidades</td><td>Métricas (opcional)</td></tr>
<tr><td>1. Revisar el diagrama generado
2. Validar la coherencia con la historia
3. Compartir con el equipo</td><td>Herramienta de visualización UML (PlantUML)</td><td>• Validación conceptual de diagramas de clases inicial UML
• Comunicación efectiva con el equipo de desarrollo</td><td>• Nivel de comprensión del equipo
• Número de observaciones realizadas</td></tr>
</table>

Tabla 28. Práctica 2: Generación automática de diagramas de actividades.

<table>
<tr><td>BP-02</td><td colspan="3">Práctica</td></tr>
<tr><td colspan="4">Nombre</td></tr>
<tr><td colspan="4">Generación automática de diagramas de actividades</td></tr>
<tr><td>Fase:</td><td>Planeación del Sprint</td><td>Rol del participante:</td><td>Dueño del Producto</td></tr>
<tr><td colspan="4">Objetivo</td></tr>
<tr><td colspan="4">Proporcionar al equipo de desarrollo una representación visual del flujo de trabajo descrito en una historia de usuario, con el fin de reducir malentendidos, mejorar la planificación y clarificar el comportamiento esperado del sistema.</td></tr>
<tr><td colspan="2">Entrada</td><td colspan="2">Resultado</td></tr>
<tr><td colspan="2">Historia de usuario seleccionada del Sprint Backlog</td><td colspan="2">Diagrama de actividades generado automáticamente</td></tr>
<tr><td colspan="4">Criterios de verificación</td></tr>
<tr><td colspan="4">• El diagrama de actividades representa adecuadamente el flujo, decisiones y estados involucrados
• El equipo de desarrollo utiliza el diagrama como apoyo en la planificación del Sprint</td></tr>
<tr><td colspan="4">Guía</td></tr>
<tr><td>Actividad</td><td colspan="3">Generación del diagrama de actividades</td></tr>
<tr><td colspan="2">Entrada</td><td colspan="2">Salida</td></tr>
<tr><td colspan="2">Historia de usuario seleccionada del Sprint Backlog</td><td colspan="2">Diagrama de actividades generado automáticamente</td></tr>
<tr><td>Tareas (opcional)</td><td>Herramientas (opcional)</td><td>Conocimientos y Habilidades</td><td>Métricas (opcional)</td></tr>
<tr><td>1. Seleccionar historia de usuario
2. Ingresar la historia en la herramienta
3. Ejecutar la generación del diagrama</td><td>Herramienta de generación automática de diagramas UML</td><td>• Lectura y análisis de historias de usuario
• Interpretación de diagrama de actividades UML</td><td>• Tiempo promedio de generación
• Número de elementos correctamente representados</td></tr>
<tr><td>Actividad</td><td colspan="3">Revisión y análisis del diagrama en la planificación</td></tr>
<tr><td colspan="2">Entrada</td><td colspan="2">Salida</td></tr>
<tr><td colspan="2">Diagrama de actividades generado</td><td colspan="2">Diagrama revisado, aprobado y compartido con el equipo de desarrollo</td></tr>
<tr><td>Tareas (opcional)</td><td>Herramientas (opcional)</td><td>Conocimientos y Habilidades</td><td>Métricas (opcional)</td></tr>
<tr><td>1. Revisar el diagrama generado
2. Validar la coherencia con la historia
3. Compartir con el equipo</td><td>Herramienta de visualización UML (PlantUML)</td><td>• Validación conceptual de diagramas de actividades UML
• Comunicación efectiva con el equipo de desarrollo</td><td>• Nivel de comprensión del equipo
• Número de observaciones realizadas</td></tr>
</table>

Tabla 29. Práctica 3: Uso del diagrama de clases inicial para comprender las historias de usuario.

<table>
<tr><td>BP-03</td><td colspan="3">Práctica</td></tr>
<tr><td colspan="4">Nombre</td></tr>
<tr><td colspan="4">Uso del diagrama de clases inicial para comprender las historias de usuario</td></tr>
<tr><td>Fase:</td><td>Planeación del Producto</td><td>Rol del participante:</td><td>Equipo de Desarrollo</td></tr>
<tr><td colspan="4">Objetivo</td></tr>
<tr><td colspan="4">Facilitar al Equipo de Desarrollo la comprensión de las historias de usuario mediante el análisis del diagrama de clases inicial, con el fin de reducir ambigüedades, mejorar la alineación con el Dueño del Producto y detectar posibles inconsistencias desde la etapa de Planificación del Sprint.</td></tr>
<tr><td colspan="2">Entrada</td><td colspan="2">Resultado</td></tr>
<tr><td colspan="2">• Historia de usuario presentada con su diagrama de clases inicial previamente generado</td><td colspan="2">• Historia de usuario aclarada y/o ajustada
• Diagrama validado y comprendido por el equipo de desarrollo</td></tr>
<tr><td colspan="4">Criterios de verificación</td></tr>
<tr><td colspan="4">• El Equipo de Desarrollo demuestra comprensión clara de la historia de usuario.
• Se identifican y resuelven ambigüedades o inconsistencias en el diagrama de clases inicial generado.
• El diagrama ajustado se valida en función de los comentarios realizados.</td></tr>
<tr><td colspan="4">Guía</td></tr>
<tr><td>Actividad</td><td colspan="3">Uso del diagrama de clases inicial</td></tr>
<tr><td colspan="2">Entrada</td><td colspan="2">Salida</td></tr>
<tr><td colspan="2">Historia de usuario con su diagrama de clases inicial</td><td colspan="2">Lista de dudas o comentarios del equipo de desarrollo</td></tr>
<tr><td>Tareas (opcional)</td><td>Herramientas (opcional)</td><td>Conocimientos y Habilidades</td><td>Métricas (opcional)</td></tr>
<tr><td>1. Uso del diagrama de clases inicial UML</td><td>• Herramienta de visualización UML (PlantUML)</td><td>• Interpretación de diagrama de clases inicial UML</td><td></td></tr>
<tr><td colspan="4">Consideración (Regla de aplicación)</td></tr>
<tr><td colspan="4">El diagrama de clases/actividades generado debe considerarse un borrador analítico y no una representación definitiva del dominio. La corrección sintáctica del modelo UML no garantiza la validez lógica de la solución de negocio.</td></tr>
</table>

Tabla 30. Práctica 4: Uso de diagramas de actividades para entender el flujo de trabajo.

<table>
<tr><td>BP-04</td><td colspan="3">Práctica</td></tr>
<tr><td colspan="4">Nombre</td></tr>
<tr><td colspan="4">Uso de diagramas de actividades para entender el flujo de trabajo</td></tr>
<tr><td>Fase:</td><td>Planeación del Sprint</td><td>Rol del participante:</td><td>Equipo de Desarrollo</td></tr>
<tr><td colspan="4">Objetivo</td></tr>
<tr><td colspan="4">Facilitar al Equipo de Desarrollo la comprensión del flujo de trabajo descrito en la historia de usuario, mediante el uso de diagramas de actividades.</td></tr>
<tr><td colspan="2">Entrada</td><td colspan="2">Resultado</td></tr>
<tr><td colspan="2">• Historia de usuario con su diagrama de actividades generado</td><td colspan="2">• Historia de usuario comprendida con claridad
• Flujo de trabajo validado y refinado
• Diagrama actualizado en caso de ajustes</td></tr>
<tr><td colspan="4">Criterios de verificación</td></tr>
<tr><td colspan="4">• El equipo identifica correctamente las acciones, decisiones y condiciones del flujo.
• Se resuelven dudas o inconsistencias detectadas.
• El diagrama de actividades actualizado refleja el comportamiento esperado.</td></tr>
<tr><td colspan="4">Guía</td></tr>
<tr><td>Actividad</td><td colspan="3">Uso del diagrama de actividades</td></tr>
<tr><td colspan="2">Entrada</td><td colspan="2">Salida</td></tr>
<tr><td colspan="2">Historia de usuario con su diagrama de actividades</td><td colspan="2">Diagrama validado o ajustado y dudas resueltas</td></tr>
<tr><td>Tareas (opcional)</td><td>Herramientas (opcional)</td><td>Conocimientos y Habilidades</td><td>Métricas (opcional)</td></tr>
<tr><td>1. Uso del diagrama de actividades UML</td><td>• Herramienta de visualización UML (PlantUML)</td><td>• Interpretación de diagrama de actividades UML</td><td></td></tr>
<tr><td colspan="4">Consideración (Regla de aplicación)</td></tr>
<tr><td colspan="4">El diagrama de clases/actividades generado debe considerarse un borrador analítico y no una representación definitiva del dominio. La corrección sintáctica del modelo UML no garantiza la validez lógica de la solución de negocio.</td></tr>
</table>

Tabla 31. Práctica 5: Realizar documentación con los diagramas UML generados.

<table>
<tr><td colspan="2">BP-05</td><td colspan="4">Práctica</td></tr>
<tr><td colspan="6">Nombre</td></tr>
<tr><td colspan="6">Realizar documentación con los diagramas UML generados</td></tr>
<tr><td>Fase:</td><td colspan="2">Planeación del Sprint y Ejecución del Sprint</td><td>Rol del participante:</td><td colspan="2">Dueño del Producto y Equipo de Desarrollo</td></tr>
<tr><td colspan="6">Objetivo</td></tr>
<tr><td colspan="6">Asegurar que toda la documentación técnica esté alineada con los diagramas generados, y que dicha información esté disponible para futuras referencias.</td></tr>
<tr><td colspan="3">Entrada</td><td colspan="3">Resultado</td></tr>
<tr><td colspan="3">• Diagramas UML generados (clases y actividades)</td><td colspan="3">• Documentación técnica actualizada y alineada con los diagramas generados
• Información clara y disponible para el equipo Scrum</td></tr>
<tr><td colspan="6">Criterios de verificación</td></tr>
<tr><td colspan="6">• La documentación refleja los cambios realizados durante el Sprint.
• Los diagramas se integran adecuadamente en los documentos técnicos.
• La documentación es validada por el Dueño del Producto.</td></tr>
<tr><td colspan="6">Guía</td></tr>
<tr><td colspan="2">Actividad</td><td colspan="4">Documentación técnica basada en diagramas UML</td></tr>
<tr><td colspan="3">Entrada</td><td colspan="3">Salida</td></tr>
<tr><td colspan="3">• Diagramas de clases inicial UML
• Diagrama de actividades UML</td><td colspan="3">Documentación técnica actualizada y validada</td></tr>
<tr><td colspan="2">Tareas (opcional)</td><td>Herramientas (opcional)</td><td colspan="2">Conocimientos y Habilidades</td><td>Métricas (opcional)</td></tr>
<tr><td colspan="2">1. Revisar los diagramas generados (clases y actividades).
2. Generar la documentación con los diagramas revisados
3. Validar la documentación final con el Dueño del Producto.</td><td>• Herramienta de visualización UML (PlantUML)</td><td colspan="2">• Lectura e interpretación de diagramas UML</td><td>• Número de diagramas documentados</td></tr>
</table>

4.2. Desarrollo de la herramienta de generación automática de diagramas UML

La implementación de la herramienta de generación automática de diagramas UML se llevó a cabo utilizando la metodología ágil Scrum, seleccionada por su enfoque iterativo e incremental, así como por su capacidad para adaptarse a los cambios a lo largo del ciclo de vida del proyecto. La metodología Scrum resultó ser particularmente adecuada para un proyecto de investigación y desarrollo, donde la comprensión del problema y la solución evoluciona de forma continua.

Scrum proporciona una estructura basada en roles, eventos y artefactos que facilita la organización y ejecución del trabajo colaborativo. En este proyecto, se definieron los roles esenciales de Dueño del Producto y Equipo de Desarrollo, y se establecieron los eventos clave de planificación, revisión y retrospectiva, los cuales se describen con mayor detalle en la sección 3.2.1.2, junto con los artefactos utilizados y los *stakeholders* involucrados en el proceso del desarrollo de la herramienta.

El desarrollo se organizó en una serie de *Sprints*, cada uno con una duración de dos semanas y con objetivos específicos claramente definidos. Al inicio de cada *Sprint*, se celebró una reunión de planificación en la que se seleccionaron, a partir del *Product Backlog*, las historias de usuario a implementar. Al finalizar cada iteración, se llevaron a cabo eventos de revisión para evaluar los entregables generados y sesiones de retrospectiva para analizar el proceso de trabajo y proponer mejoras. Este enfoque estructurado permitió entregar incrementos funcionales en periodos breves, favoreciendo la evaluación continua del producto, la detección oportuna de errores y la incorporación ágil de ajustes. A lo largo del desarrollo se ejecutaron cinco *Sprints*, cada uno con un enfoque particular:

- El primer *Sprint* se enfocó en la configuración inicial del sistema, sentando las bases fundamentales para el desarrollo de la herramienta. Se implementaron los elementos esenciales de la arquitectura, incluyendo la definición y creación de la base de datos, así como la construcción de las interfaces básicas para la autenticación de usuarios, la creación de nuevos proyectos y la carga de archivos. Estas funcionalidades fueron clave para habilitar una estructura mínima operativa sobre la cual se construirían los siguientes incrementos del sistema.
- El segundo *Sprint* se orientó a la implementación de un sistema de gestión de usuarios y roles, permitiendo establecer permisos diferenciados según el tipo de usuario. Esta iteración incluyó la creación de interfaces y *endpoints* para el registro, visualización y asignación de roles, así como la validación de accesos según los privilegios definidos. Gracias a estas funcionalidades, se logró mejorar la seguridad y el control de acceso dentro de la herramienta, estableciendo un modelo de interacción más robusto y personalizado.
- En el tercer *Sprint*, el foco principal fue el desarrollo de funcionalidades orientadas a mejorar la calidad del insumo textual que alimenta al LLM. Se implementó un mecanismo para validar la estructura gramatical y sintáctica de las historias de usuario, asegurando que cumplieran con criterios formales previamente establecidos. Asimismo, se incorporó la posibilidad de vincular dichas historias a proyectos específicos, lo cual permitió mantener una trazabilidad clara entre los requisitos funcionales y los diagramas generados. Este avance resultó esencial para garantizar la coherencia y precisión en el procesamiento automático posterior.
- En el cuarto *Sprint* consolidó funcionalidades clave para la gestión de archivos adjuntos en los proyectos. Se desarrollaron operaciones que permitieron a los usuarios seleccionar, descargar y eliminar documentos asociados, optimizando el manejo de recursos contextuales. Además, se introdujo un sistema de validación automática de historias de

usuario previo a su envío al LLM, el cual verifica aspectos como formato, estructura y claridad. Esta mejora fortaleció la calidad del contenido procesado, contribuyendo directamente a la precisión de los diagramas UML generados.

- Finalmente, en el quinto *Sprint* se integró el sistema RAG como componente central del proceso de generación automática de diagramas UML. Se incorporó el LLM Gemini 2.5 Pro, encargado de procesar historias de usuario que han sido previamente enriquecidas con información contextual obtenida a través del sistema RAG. Esta integración incluyó la implementación de un pipeline que realiza la validación del texto, la recuperación de información relevante y la construcción del *prompt* final enviado al modelo. Asimismo, se ajustaron los módulos del backend para gestionar eficazmente las solicitudes hacia el LLM y mantener un flujo coherente entre los diferentes componentes del sistema. Con esta mejora, la herramienta alcanzó su capacidad completa para transformar historias de usuario validadas y contextualizadas en diagramas de clases y actividades UML generados de manera automática.

4.2.1. Tecnologías utilizadas

La implementación de la herramienta de generación automática de diagramas UML se estructuró sobre una arquitectura cliente-servidor, lo que permitió separar claramente las responsabilidades entre el *frontend* (interfaz de usuario) y el *backend* (lógica del sistema y procesamiento de datos). Esta elección arquitectónica favorece la mantenibilidad, escalabilidad y evolución del sistema a lo largo del tiempo.

Para su desarrollo, se seleccionaron tecnologías actuales ampliamente utilizadas en entornos de desarrollo modernos, priorizando aquellas que ofrecieran modularidad, capacidad de integración con bibliotecas externas, facilidad de despliegue en contenedores y soporte activo por parte de la comunidad. Estas tecnologías abarcan tanto herramientas y lenguajes de programación como *frameworks*, sistemas de gestión de bases de datos y entornos de virtualización. A continuación, se describen los principales componentes tecnológicos utilizados en cada capa del sistema:

- **Frontend - React y Material UI:** La interfaz de usuario fue implementada utilizando React, una biblioteca de JavaScript orientada a la construcción de interfaces web interactivas y dinámicas. React permitió desarrollar una arquitectura basada en componentes reutilizables, facilitando la mantenibilidad y permitiendo una evolución ágil del sistema. Para el diseño visual se incorporó Material UI, un conjunto de componentes que implementan las directrices de diseño de *Google Material Design.* Esta herramienta permitió agilizar el desarrollo de la interfaz gráfica, garantizando una experiencia de usuario coherente, accesible y con una estética profesional.
- **Backend - FastAPI (Python):** El servidor de la aplicación fue desarrollado utilizando FastAPI, un *framework* moderno y de alto rendimiento para la construcción de APIs en Python. La elección de esta tecnología respondió a la necesidad de implementar una interfaz de comunicación eficiente entre el *frontend* y los servicios encargados de la generación de diagramas UML. Entre las principales razones que motivaron su adopción se destacan su soporte nativo para operaciones asíncronas, lo cual optimiza el manejo de múltiples solicitudes concurrentes, su sistema de validación automática de datos mediante Pydantic, lo que facilitó las tareas de pruebas de integración. Estas características permitieron construir una lógica de negocio robusta, mantenible y fácilmente escalable, alineada con los objetivos técnicos del sistema.

- **Base de datos – PostgreSQL:** Para la gestión y persistencia de la información generada por la herramienta, se optó por PostgreSQL, un sistema de gestión de bases de datos relacional ampliamente reconocido por su escalabilidad, confiabilidad y capacidad para manejar estructuras de datos complejas. Esta tecnología resultó especialmente adecuada para almacenar entidades clave como los usuarios, garantizando integridad y consistencia en el manejo de los datos. Además, su compatibilidad con herramientas ORM (*Object-Relational Mapping*) en Python permitió una integración fluida con el *backend* desarrollado en FastAPI, facilitando las operaciones de consulta, inserción y actualización dentro de un entorno estructurado y seguro.
- **Contenedores – Docker:** Con el objetivo de asegurar la portabilidad, replicabilidad y facilidad de despliegue del sistema, se empleó Docker como solución del uso de contenedores. Esta tecnología permite encapsular de manera aislada los distintos componentes de la herramienta (*frontend*, *backend* y base de datos) en contenedores independientes, cada uno con su propio entorno de ejecución. Esta arquitectura basada en contenedores facilitó significativamente el desarrollo local, al eliminar problemas de compatibilidad entre entornos. Asimismo, el uso de Docker contribuyó a mantener la coherencia entre versiones del sistema, favoreciendo una integración continua más confiable y una mayor eficiencia en la gestión de recursos durante el ciclo de vida del software.
- **Control de versiones - Git y GitHub:** Para la gestión del código fuente y la coordinación del desarrollo de durante los distintos *Sprints*, se utilizó Git como sistema de control de versiones, en conjunto con GitHub como plataforma de alojamiento y colaboración remota. Esta combinación facilita un seguimiento preciso de los cambios realizados en el proyecto, permitiendo mantener versiones estables para cada iteración del desarrollo y asegurando la trazabilidad de todas las modificaciones introducidas en el sistema.

4.2.2. Artefacto: Product Backlog

En las prácticas propuestas, se destacan dos roles principales: el Dueño del Producto y el Desarrollador. El primero genera los diagramas de clase iniciales y de actividades, mientras que el segundo los utiliza para mejorar la comprensión de las historias de usuario. Adicionalmente, el Desarrollador puede realizar un refinamiento según sea necesario para garantizar su alineación con las historias de usuario. A partir de los roles identificados en el conjunto de prácticas propuestas, se plantea el siguiente *Product Backlog*, diseñado específicamente para guiar el desarrollo de la herramienta de generación automática de diagramas UML haciendo uso de LLMs. Este *Product Backlog* pretende facilitar la planificación y ejecución de las actividades necesarias para crear y refinar los diagramas, asegurando que la herramienta cumpla con los objetivos del proyecto y responda a las necesidades identificadas.

Propuesta inicial del *Product Backlog*

1. **Como** Dueño del Producto, **quiero** un prototipo de interfaz inicial que permita a los usuarios (Dueño del Producto y Equipo de desarrollo) adjuntar archivos, **para** que puedan interactuar con la herramienta.
2. **Como** Dueño del Producto, **quiero** que el sistema valide las historias de usuario antes de enviarlas al LLM, **para** garantizar que el modelo reciba entradas bien estructuradas.
3. **Como** Dueño del Producto, **quiero** que la herramienta me permita generar diagramas de clase inicial válidos, **para** representar la estructura del sistema a partir de las historias de usuario.

4. **Como** Dueño del Producto, **quiero** que la herramienta me permita generar diagramas de actividades válidos, **para** visualizar el flujo de trabajo del sistema.
5. **Como** Desarrollador, **quiero** utilizar el diagrama de clase inicial generado por la herramienta, **para** comprender mejor las historias de usuario y las entidades del sistema.
6. **Como** Desarrollador, **quiero** utilizar el diagrama de actividades generado por la herramienta, **para** comprender el flujo de trabajo del sistema y sus interacciones.
7. **Como** Desarrollador, **quiero** refinar los diagramas de clase inicial generados por la herramienta, **para** alinearlos mejor con las historias de usuario.
8. **Como** Desarrollador, **quiero** refinar los diagramas de actividades generados por la herramienta, **para** alinearlos con las historias de usuario.
9. **Como** Dueño del Producto, **quiero** que la herramienta genere documentación automática basada en los diagramas de clase inicial generados **para** su documentación y futuro uso.
10. **Como** Dueño del Producto, **quiero** que la herramienta genere documentación automática basada en los diagramas de actividades generados **para** su documentación y futuro uso.
11. **Como** Desarrollador, **quiero** que la herramienta genere documentación automática basada en los diagramas de clase inicial generados **para** su documentación y futuro uso.
12. **Como** Desarrollador, **quiero** que la herramienta genere documentación automática basada en los diagramas de actividades generados **para** su documentación y futuro uso.

4.2.3. Primer *Sprint*: Implementación inicial de la herramienta

El primer *Sprint* marcó el punto de partida en el desarrollo de la herramienta de generación automática de diagramas UML a partir de historias de usuario. Este primer ciclo de trabajo tuvo como propósito establecer los cimientos funcionales y técnicas del sistema, asegurando que los elementos fundamentales estuvieran correctamente implementados para permitir un avance progresivo y sostenible en los *Sprints* posteriores.

Dado que el enfoque metodológico utilizado fue Scrum, en esta primera iteración se priorizó la selección de historias de usuario esenciales, orientadas a habilitar la interacción básica de los usuarios con el sistema, así como a preparar el entorno tecnológico para el desarrollo incremental de las funcionalidades. Las decisiones tomadas en este *Sprint* se basaron en criterios técnicos, dependencias funcionales y valor para los usuarios finales de la herramienta. En esta etapa inicial también fueron abordados aspectos estructurales clave, tales como la estructura de la base de datos, el diseño de los prototipos iniciales, definición de la estructura inicial del proyecto y la realización de la instalación de las tecnologías requeridas. Estos elementos fueron indispensables no solo para el funcionamiento básico del sistema, sino también para garantizar la coherencia y escalabilidad del proyecto a lo largo de su ciclo de desarrollo.

Planificación del *Sprint*

Durante la planificación del primer *Sprint*, se llevó a cabo una revisión detallada del *Product Backlog*, priorizado aquellas historias de usuario que permitieran alcanzar un primer nivel funcional del sistema. Esta actividad fue realizada por el Dueño del Producto, quien estableció la priorización de las historias de usuario de acuerdo con las necesidades requeridas para dar inicio con el desarrollo del proyecto. El *Sprint Backlog* resultante quedó compuesto por las siguientes historias de usuario:

1. **Como** Dueño del Producto, **quiero** un formulario de logueo que permita a los usuarios iniciar sesión **para** su acceso al sistema.

2. **Como** Dueño del Producto, **quiero** que el sistema me permita crear proyectos **para** una mejor organización de información.

3. **Como** Dueño del Producto, **quiero** una interfaz que permita a los usuarios (Dueño del Producto y Desarrollador) adjuntar archivos, para que puedan interactuar con la herramienta.

4. **Como** Dueño del Producto, **quiero** que el sistema valide las historias de usuario antes de enviarlas al LLM, **para** garantizar que el modelo reciba entradas bien estructuradas.

Estas historias de usuario fueron seleccionadas por su carácter fundamental dentro de la arquitectura de la herramienta. Su implementación no solo representaba la incorporación de funcionalidades esenciales, como lo es la autenticación de usuarios, la creación de proyectos, la carga de archivos y la validación de historias de usuario, sino que también funcionaban como punto de partida para habilitar procesos más complejos en los *Sprints* posteriores. Dichas funcionalidades constituían la base operativa del sistema, sobre la cual se integran progresivamente los módulos de procesamiento y generación de diagramas UML mediante un LLM.

Durante la reunión de planificación del *Sprint*, se analizaron los posibles desafíos técnicos asociados a esta etapa inicial, destacando entre ellos la necesidad de configurar adecuadamente el entorno de desarrollo, establecer la arquitectura del sistema, y definir las tecnologías a emplear tanto en el *backend* como en el *frontend*. Asimismo, se identificaron ciertas dependencias funcionales entre las historias de usuario seleccionadas. Sin embargo, pese a este análisis, algunas historias de usuario previas no fueron completamente identificadas en esta fase, lo que generó desfases en la ejecución del *Sprint*, al requerir realizar ajustes no contemplados en el *Sprint Backlog* inicial.

Objetivo del *Sprint*

El objetivo del primer *Sprint* fue sentar las bases funcionales del sistema, permitiendo a los usuarios interactuar con la herramienta de manera inicial y significativa. Para ellos, se definieron como prioritarias las funcionalidades relacionadas con la autenticación de usuarios, la creación y gestión de proyectos, la carga de archivos y la validación de entradas que serían procesadas por el LLM. Estas características no solo son fundamentales para garantizar un uso estructurado del sistema, sino que además actúan como componentes habilitadores de las funcionalidades que serían desarrolladas en los *Sprints* posteriores.

Ejecución del *Sprint*

La ejecución del primer *Sprint* fue se llevó a cabo con base en el *Sprint Backlog* acordado durante la reunión de planificación del *Sprint*, y estuvo a cargo de un equipo conformado por el Desarrollador y el Dueño del Producto. A lo largo de esta fase, el equipo Scrum se enfocó en transformar las historias de usuario seleccionadas en funcionalidades tangibles, siguiendo un enfoque incremental. No obstante, durante la implementación se identificaron ciertas dependencias no previstas inicialmente, lo cual obligó a replantear parcialmente la secuencia de tareas.

Durante los primeros días del *Sprint*, se evidenció que algunas de las funcionalidades seleccionadas requieren la existencia previa de elementos estructurales del sistema, como la base de datos y los servicios básicos del *backend*. Al no haber sido considerados explícitamente como parte del *Sprint Backlog* del *Sprint*, se procedió a integrar estas tareas de manera coordinada, para evitar

que interfirieran significativamente con el avance general. Entre las actividades clave realizadas durante este *Sprint* destacan las siguientes:

- **Diseño e implementación de la base de datos:** Se definió la estructura relacional necesaria para almacenar información de usuarios, proyectos y archivos. Este trabajo incluyó la creación de esquemas y tablas, así como la configuración inicial del sistema gestor de base de datos.
- **Prototipado de interfaz de usuario:** Se desarrollaron prototipos iniciales de la interfaz, que sirvieron como guía visual para la implementación de funcionalidades básicas. Estos diseños permitieron evaluar anticipadamente la experiencia de usuario prevista.
- **Configuración del entorno de desarrollo:** Se instalaron y configuraron las tecnologías base requeridas por el proyecto, incluyendo el *framework* principal, las bibliotecas para el *frontend*, y las herramientas necesarias para la conexión con la base de datos y servidores de desarrollo.
- **Desarrollo de servicios del *backend*:** Se implementaron los servicios asociados a la gestión de usuarios y roles, esenciales para la autenticación y autorización dentro del sistema. Esta funcionalidad incluyó el registro de usuarios, la verificación de credenciales, y la asignación de permisos dependiendo del rol.

A pesar de los ajustes realizados durante la ejecución del primer *Sprint*, el equipo logró conservar un ritmo de trabajo constante que permitió avanzar de forma significativa en la implementación de las funcionalidades priorizadas. Las tareas definidas en el *Sprint Backlog* fueron abordadas de manera progresiva y, en su mayoría, completadas dentro del marco temporal establecido. Este desempeño fue posible gracias a una adecuada coordinación entre los roles involucrados, así como a la capacidad del equipo para adaptarse ante imprevistos sin comprometer los objetivos generales del *Sprint*.

No obstante, durante el desarrollo surgieron algunas historias de usuario que no habían sido contempladas inicialmente, pero cuya implementación resultó necesaria para dar continuidad lógica y técnica a las funcionalidades previstas. Estas historias de usuario adicionales estaban vinculadas a aspectos estructurales del sistema que, si bien no eran evidentes en la planificación inicial, se hicieron manifiestos conforme avanzaba la construcción del producto. Su incorporación obligó al equipo a ajustar el plan de trabajo, lo cual, aunque no comprometió de forma significativa el avance del *Sprint*, sí evidenció la conveniencia de realizar un análisis más detallado de las posibles dependencias técnicas entre historias de usuario en futuras iteraciones. Este hallazgo se asumió como una oportunidad de mejora, orientada a fortalecer la preparación y reducir la incertidumbre en los próximos ciclos de desarrollo.

Esta experiencia evidenció que, si bien la metodología ágil permite adaptarse de manera flexible a cambios y nuevos requisitos, la calidad de la planificación sigue siendo un factor determinante para el éxito de cada iteración. En consecuencia, se identificó como una acción prioritaria para los próximos Sprints el fortalecimiento del análisis previo de las historias de usuario seleccionadas, poniendo especial atención en la detección de dependencias lógicas, técnicas y funcionales, con el fin de evitar desviaciones que puedan afectar el cumplimiento de los objetivos establecidos.

Revisión del *Sprint*

Al término del primer *Sprint*, se llevó a cabo una reunión de revisión entre el Desarrollador y el Dueño del Producto con el objetivo de evaluar los avances obtenidos y contrastarlos con los compromisos asumidos en el *Sprint Backlog*. Esta sesión permitió inspeccionar el incremento

desarrollado y verificar en qué medida se cumplieron los objetivos funcionales definidos para esta iteración.

Durante la revisión, se presentaron los resultados concretos de las historias de usuario implementadas. En primer lugar, se expuso la definición de la estructura de la base de datos, la cual fue diseñada para dar soporte a la gestión de usuarios, proyectos y archivos. Esta estructura fue validada tanto desde un enfoque técnico como funcional, confirmando su alineación con los requisitos establecidos en el *Product Backlog*.

En cuanto al diseño de la interfaz, se entregaron los primeros prototipos de usuario. Aunque se trataba de una versión inicial, su presentación permitió establecer una base visual sobre la que se construirá la experiencia del usuario en los siguientes ciclos de desarrollo. El Dueño del Producto realizó observaciones específicas orientadas a mejorar el proceso de generación de los diagramas UML.

También se reportó la instalación y configuración exitosa de las tecnologías necesarias para el desarrollo del sistema, incluyendo *frameworks*, servicios de base de datos y herramientas auxiliares. Esta tarea resultó esencial para garantizar que el entorno de trabajo estuviera plenamente operativo desde las etapas iniciales del proyecto.

Asimismo, se presentó el desarrollo de los servicios básicos de la base de datos, centrados principalmente en la autenticación de usuarios y en la gestión de roles. Estos servicios fueron validados mediante pruebas funcionales, las cuales confirmaron la correcta creación de usuarios, la verificación de credenciales y la asignación de permisos adecuados según el tipo de usuario.

No obstante, también se discutieron las dificultades enfrentadas durante el *Sprint*, particularmente la incorporación de historias de usuario que no habían sido identificadas durante la planificación inicial. Estas historias de usuario resultaron necesarias para la implementación adecuada de otras funcionalidades dependientes.

Retrospectiva del *Sprint*

Al término del primer *Sprint*, se llevó a cabo una sesión de revisión entre el Desarrollador y el Dueño del Producto, con el objetivo de evaluar los avances alcanzados y compararlos con los compromisos establecidos en el *Sprint Backlog*. Esta instancia permitió inspeccionar el incremento generado y verificar en qué medida se cumplieron los objetivos funcionales definidos para la iteración. Finalizada la revisión, el equipo realizó la retrospectiva del *Sprint*, orientada a reflexionar sobre el proceso seguido, identificar fortalezas, reconocer áreas de mejora y acordar acciones concretas para futuras iteraciones.

- **Aspectos positivos:** Uno de los principales logros destacados fue la capacidad del equipo para adaptarse con agilidad ante situaciones no previstas. La aparición de historias de usuario no contempladas originalmente representó un desafío tanto organizativo como técnico; sin embargo, el equipo logró reorganizar las tareas de manera eficiente, manteniendo el ritmo de trabajo y asegurando la calidad del producto. Otro aspecto valorado fue la comunicación fluida y constante entre el Desarrollador y el Dueño del Producto. Esta coordinación permitió tomar decisiones oportunas, resolver dudas con rapidez y alinear las expectativas respecto a los entregables. A pesar de tratarse de un equipo reducido, la colaboración efectiva fue clave para sortear dificultades y avanzar con claridad hacia los objetivos del *Sprint*.
- **Áreas de mejora:** Entre los aprendizajes más significativos se identificó la necesidad de fortalecer el análisis de las dependencias entre historias de usuario durante la planificación

de los *Sprints*. La detección tardía de ciertas historias previas, necesarias para implementar funcionalidades seleccionadas, evidenció una brecha en el análisis del *Product Backlog*. Esta situación afectó directamente la ejecución y obligó al equipo a introducir cambios no contemplados en el plan original.

- **Acción de mejora:** Como respuesta a lo anterior, se acordó implementar una mejora orientada a realizar un análisis más riguroso del *Product Backlog* antes de iniciar cada Sprint. Este análisis debe enfocarse especialmente en la identificación de relaciones de dependencia entre historias de usuario, con el fin de asegurar que todas las funcionalidades requeridas se encuentren contempladas en la planificación. Para ello, se propuso incorporar una revisión sistemática del *Product Backlog* durante las sesiones de planificación, centrada en detectar posibles interdependencias técnicas y funcionales. Esta práctica no solo implicó evaluar el contenido individual de cada historia, sino también su conexión dentro del flujo general de trabajo, verificando si su implementación depende de otras historias aún no abordadas.

La retrospectiva concluyó con una valoración positiva del trabajo realizado durante el *Sprint*, reconociendo tanto los logros alcanzados como los aprendizajes obtenidos. El equipo manifestó su compromiso con la mejora continua, acordando aplicar los ajustes identificados para perfeccionar los procesos de desarrollo y elevar la calidad del producto en los próximos ciclos.

4.2.4. Segundo *Sprint*: Desarrollo de funcionalidades para la gestión de usuarios

El segundo *Sprint* marcó una etapa clave en el desarrollo de la herramienta de generación automática de diagramas UML, ya que sentó las bases funcionales necesarias para iniciar la interacción con el sistema. En esta iteración, el equipo se enfocó en implementar las funcionalidades esenciales de autenticación y gestión de usuarios, con el objetivo de establecer un entorno seguro, estructurado y accesible para los futuros módulos de la aplicación. Este conjunto de tareas representó el primer paso hacia la consolidación de una arquitectura capaz de manejar el flujo de usuarios y sus respectivas operaciones dentro del sistema. La planificación del Sprint se llevó a cabo en estrecha colaboración entre el Dueño del Producto y el Desarrollador, quienes priorizaron aquellas historias de usuario orientadas a garantizar un acceso confiable y una administración eficiente de la información de los usuarios registrados.

Planificación del *Sprint*

Durante la planificación del segundo *Sprint*, el equipo procedió a seleccionar, a partir del *Product Backlog* priorizado, las historias de usuario que serían abordadas en esta iteración. La atención se centró en desarrollar funcionalidades esenciales para la gestión de cuentas de usuario, una capacidad indispensable para garantizar una interacción segura y personalizada con la herramienta. Este enfoque respondía a la necesidad de dotar al sistema de una estructura básica de control de acceso, que permitiera a los usuarios autenticarse, modificar su información personal y, en el caso de los administradores, gestionar las cuentas registradas. Las historias de usuario seleccionadas para conformar el *Sprint Backlog* en este segundo *Sprint* fueron las siguientes:

1. **Como** Dueño del Producto, **quiero** que los usuarios puedan registrarse en el sistema mediante un formulario de creación de cuenta, **para** que puedan obtener acceso.

2. **Como** Dueño del Producto, **quiero** un formulario de inicio de sesión que valide las credenciales de los usuarios, **para** que solo los usuarios registrados puedan acceder al sistema.

3. **Como** Dueño del Producto, **quiero** que los usuarios puedan actualizar su información (como nombre, contraseña), **para** mantener sus datos actualizados.

4. **Como** Dueño del Producto, **quiero** que el usuario con rol "Administrador" tenga la opción de eliminar las cuentas, **para** gestionar y dar de baja cuentas cuando sea necesario.

Objetivo del *Sprint*

El objetivo principal de este segundo *Sprint* fue implementar las funcionalidades necesarias para habilitar el registro y la gestión de cuentas de usuario dentro de la herramienta. Concretamente, se buscó desarrollar un formulario de registro que permitiera la creación de cuentas, un formulario de inicio de sesión con validación de credenciales, un módulo de actualización de información personal, y una interfaz que otorgue a los administradores la capacidad de eliminar cuentas cuando sea necesario.

Estas funcionalidades representan un avance significativo hacia la consolidación de una plataforma operativa, al sentar las bases de un sistema de autenticación robusto y flexible que garantice la integridad y seguridad de los datos de los usuarios.

Ejecución del *Sprint*

Durante el desarrollo del segundo *Sprint*, se abordaron las historias de usuario previstas en el Sprint Backlog, centradas en la implementación de funcionalidades para la gestión de cuentas dentro del sistema. El trabajo se orientó a construir una base sólida de autenticación y administración de usuarios, elementos esenciales para garantizar la seguridad y la personalización de la experiencia dentro de la herramienta.

En primer lugar, se desarrolló el módulo de registro de usuarios, el cual permite a nuevos usuarios crear una cuenta a través de un formulario interactivo. Este componente fue conectado a la base de datos para asegurar el almacenamiento persistente de la información registrada y verificar automáticamente que los datos sensibles, como el correo electrónico, sean únicos dentro del sistema.

Posteriormente, se implementó el formulario de inicio de sesión, encargado de verificar las credenciales proporcionadas por los usuarios. Este módulo garantiza que solo los usuarios autenticados puedan acceder a las funcionalidades del sistema, y actúa como puerta de entrada al entorno de trabajo personalizado. Asimismo, se incorporó una funcionalidad que permite a los usuarios actualizar su información personal, como el nombre de usuario o la contraseña. Estas actualizaciones se gestionan mediante validaciones tanto del lado del cliente como del servidor, asegurando la integridad de los datos y reflejando los cambios en tiempo real dentro del sistema.

Finalmente, se habilitó una interfaz específica para que los usuarios con rol de "Administrador" puedan eliminar cuentas registradas. Esta capacidad resulta crucial para la administración del sistema, ya que permite ejercer control sobre el ciclo de vida de los usuarios, facilitando la gestión de accesos y el mantenimiento de un entorno seguro. La ejecución de estas tareas se desarrolló sin mayores inconvenientes técnicos, y permitió avanzar de manera significativa en la construcción de la infraestructura básica del sistema, consolidando los cimientos sobre los cuales se implementarán funcionalidades más complejas en los siguientes *Sprints*.

Revisión del *Sprint*

Al concluir el segundo *Sprint*, se realizó la reunión de revisión con la participación del Desarrollador y el Dueño del Producto, siguiendo lo establecido en la metodología Scrum. Esta reunión tuvo como propósito principal inspeccionar el incremento desarrollado, evaluar el cumplimiento de los objetivos del Sprint y adaptar el *Product Backlog* en caso de ser necesario.

Durante la sesión, se presentó el conjunto de funcionalidades completadas, las cuales fueron puestas en funcionamiento para demostrar su operatividad. El Dueño del Producto verificó que cada historia de usuario implementada cumpliera con los criterios de aceptación previamente definidos. Las funcionalidades evaluadas incluyeron el registro de usuarios, la autenticación mediante inicio de sesión, la edición de datos personales y la eliminación de cuentas por parte del rol de administrador.

Se constató que todas las funcionalidades comprometidas en el *Sprint Backlog* fueron finalizadas correctamente, sin necesidad de ajustes adicionales. El sistema respondió adecuadamente ante distintos escenarios de uso, lo que permitió validar su correcto comportamiento y su integración con los componentes previamente desarrollados.

Como resultado de la revisión, se confirmó que el incremento del producto cumple con los estándares de calidad definidos por el equipo, y puede considerarse un avance potencialmente entregable. A modo de evidencia, se presentan a continuación capturas de pantalla que ilustran el funcionamiento de las funcionalidades incorporadas durante este *Sprint*.

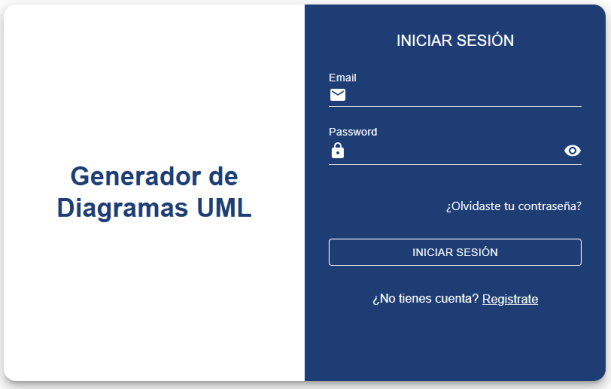


Figura 45 Formulario de inicio de sesión

La Figura 4.1 muestra la pantalla de autenticación de usuarios, la cual se encarga de verificar las credenciales ingresadas y permitir el acceso únicamente a aquellos usuarios que estén registrados y validados en el sistema.

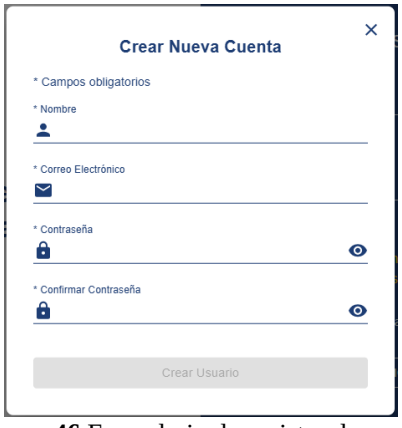


Figura 46 Formulario de registro de usuario

La Figura 4.2 muestra la interfaz destinada al registro de nuevos usuarios en la herramienta. Este formulario constituye uno de los mecanismos de entrada al sistema y está diseñado para garantizar la integridad y validez de los datos capturados.

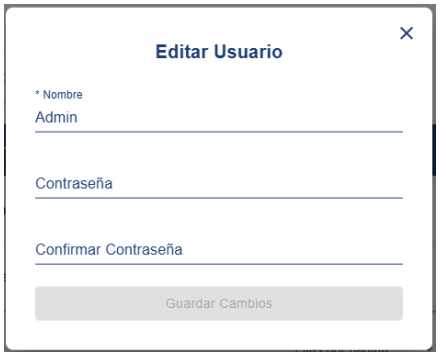


Figura 47 Actualización de información del usuario

La Figura 4.3 muestra la interfaz correspondiente a la edición de la información personal del usuario. A través de esta sección del sistema, los usuarios autenticados pueden modificar datos como su nombre y contraseña, lo que les permite mantener actualizado su perfil conforme a sus necesidades.

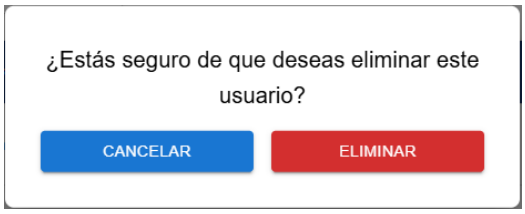


Figura 48 Eliminación de cuentas por parte del Administrador

La Figura 4.4 presenta la interfaz diseñada para el rol de Administrador, mediante la cual se habilita la gestión de las cuentas de usuarios registrados en el sistema. Esta funcionalidad permite, entre otras acciones, la eliminación definitiva de cuentas.

Retrospectiva del *Sprint*

Una vez concluida la revisión del incremento, se llevó a cabo la reunión de retrospectiva correspondiente al segundo *Sprint*. Esta sesión tuvo como propósito reflexionar sobre el desarrollo del trabajo realizado, identificar los elementos que funcionaron correctamente, detectar áreas de mejora y establecer acciones concretas que contribuyan al perfeccionamiento del proceso en los próximos ciclos. Durante la retrospectiva, el equipo analizó tanto los aspectos positivos como los desafíos enfrentados durante el *Sprint*. Entre los elementos que se destacaron como fortalezas, se señaló la claridad en la planificación inicial, la cual permitió mantener el enfoque durante toda la iteración y alcanzar los objetivos propuestos. Asimismo, la estructuración del *Sprint Backlog* en historias de usuario bien definidas facilitó la organización del trabajo y la implementación progresiva de las funcionalidades. La comunicación constante y fluida entre el Desarrollador y el Dueño del Producto fue otro de los factores clave que favorecieron el avance sostenido del desarrollo, permitiendo tomar decisiones oportunas y resolver dudas sin dilaciones.

No obstante, también se identificó una oportunidad de mejora relacionada con la lógica de asignación de roles durante el proceso de registro de nuevos usuarios. En la versión actual del sistema, al momento de crear una cuenta de usuario, se asigna automáticamente un rol predeterminado. Sin embargo, durante el *Sprint* se concluyó que esta asignación automática no responde a la lógica

funcional deseada. En su lugar, se acordó que los roles deben asignarse en el momento de creación de un proyecto, lo cual permite una gestión más controlada y contextualizada de los permisos de acceso, en función de las necesidades específicas del proyecto en cuestión.

- **Acción a implementar:** Como resultado de esta reflexión, se definió una acción concreta para los siguientes *Sprints*: modificar el flujo actual de registro de usuarios para que no se asigne ningún rol de forma automática. En adelante, el rol será definido únicamente durante la creación de un proyecto, y estará a cargo del administrador o del Dueño del Producto, según las condiciones particulares del proyecto y los perfiles de los usuarios involucrados. Esta medida busca mejorar la gestión de accesos y promover una asignación de roles más flexible y segura, alineada con la arquitectura general de la herramienta.

La retrospectiva concluyó con una percepción positiva sobre el avance logrado durante el *Sprint*. El equipo valoró no solo el cumplimiento de los objetivos técnicos, sino también la manera en que se llevó a cabo la colaboración y la toma de decisiones. La experiencia acumulada en esta iteración permitió identificar aprendizajes valiosos que serán aplicados en los siguientes ciclos de desarrollo. Se alcanzó un consenso claro sobre la necesidad de mantener una actitud abierta y receptiva al cambio, reconociendo que la mejora continua es un componente esencial en la metodología ágil. El equipo acordó que perfeccionar constantemente los procesos de trabajo, así como incorporar mejoras progresivas basadas en la experiencia adquirida durante cada iteración, permitirá no solo aumentar la eficiencia del desarrollo sino también elevar la calidad del producto final. Esta disposición para el aprendizaje constante y la adaptación se consolidó como uno de los pilares fundamentales para enfrentar los desafíos de los siguientes ciclos de trabajo, asegurando un progreso sostenible y alineado con los objetivos del proyecto.

4.2.5. Tercer *Sprint*: Desarrollo de funcionalidades para la gestión de proyectos

El tercer *Sprint* tuvo como principal objetivo ampliar las capacidades del sistema mediante la incorporación del módulo de gestión de proyectos, el cual permitiría a los usuarios crear, visualizar, editar y eliminar proyectos, consolidando así una estructura que facilitara la organización de la información. Esta iteración respondió a la necesidad de dotar al sistema de herramientas que permitieran a los usuarios interactuar de forma más dinámica y personalizada con su entorno de trabajo. Durante este ciclo de desarrollo, se buscó construir una funcionalidad coherente, integrando operaciones CRUD (crear, leer, actualizar y eliminar) que sirvieran de base para futuras funcionalidades relacionadas con la administración de requisitos y diagramas UML. La planificación del *Sprint* priorizó aquellas historias de usuario que aportaban mayor valor al uso cotidiano del sistema y reforzaban su propósito como una herramienta práctica para el análisis y documentación de software.

Planificación del *Sprint*

Durante la sesión de planificación correspondiente al tercer *Sprint*, el Dueño del Producto y el Desarrollador definieron como prioridad la implementación de las operaciones esenciales para la gestión de proyectos dentro del sistema. En este sentido, se estableció como objetivo principal habilitar las funcionalidades básicas de creación, consulta, modificación y eliminación de proyectos, así como la funcionalidad de cierre de sesión para los usuarios. Para alcanzar dicho objetivo, se seleccionaron del *Product Backlog* las siguientes historias de usuario, que fueron integradas al *Sprint Backlog*:

1. **Como** Dueño del Producto, **quiero** que el sistema me permita crear nuevos proyectos, **para** organizar mejor la información.

2. **Como** Dueño del Producto, **quiero** que el sistema muestre una lista de los proyectos creados con sus detalles, **para** acceder fácilmente a su información.
3. **Como** Dueño del Producto, **quiero** poder modificar la información de un proyecto, **para** actualizar datos como el nombre o la descripción del proyecto.
4. **Como** Dueño del Producto, **quiero** que el sistema me permita eliminar proyectos, **para** mantener organizada la información y descartar proyectos innecesarios.
5. **Como** Dueño del Producto, **quiero** que el sistema me permita cerrar sesión, **para** que los usuarios puedan finalizar su acceso al sistema.

Estas historias de usuario fueron priorizadas en función de su relevancia para la estructura general del sistema y por su vínculo directo con los flujos de trabajo fundamentales que sustentaban el uso de la herramienta. La gestión de proyectos representaba un componente central dentro de la funcionalidad de la aplicación, ya que constituía el punto de partida para la generación automática de diagramas UML, objetivo principal de la herramienta. Por esta razón, contar con operaciones sólidas para crear, visualizar, modificar y eliminar proyectos resultaba indispensable para garantizar una experiencia de usuario fluida y una organización eficiente de la información.

Asimismo, la inclusión de la funcionalidad para cerrar sesión respondió a la necesidad de asegurar una interacción segura por parte de los usuarios, en concordancia con las buenas prácticas de control de acceso y protección de datos. Al integrar estas funcionalidades en esta etapa del desarrollo, se buscó consolidar la base operativa del sistema, estableciendo así las condiciones necesarias para abordar módulos de mayor complejidad en los *Sprints* posteriores.

Objetivo del *Sprint*

El objetivo de este *Sprint* consistió en desarrollar las funcionalidades esenciales para la gestión de proyectos dentro del sistema, incluyendo la creación, visualización, modificación y eliminación de proyectos. Estas operaciones resultaban fundamentales para estructurar de manera organizada la información generada por los usuarios, permitiendo así una administración más eficiente de los recursos del sistema. Asimismo, se incorporó la funcionalidad de cierre de sesión, con el fin de completar el flujo de autenticación de usuarios y reforzar la seguridad en el acceso. En conjunto, estas implementaciones sentaron las bases funcionales necesarias para la consolidación del módulo de proyectos, sobre el cual se podrán integrar capacidades más complejas en los siguientes *Sprints*.

Ejecución del *Sprint*

Durante la ejecución del tercer *Sprint*, el equipo de desarrollo centró sus esfuerzos en la implementación del módulo de gestión de proyectos, cuyo propósito fue permitir a los usuarios realizar operaciones fundamentales como la creación, consulta, modificación y eliminación de proyectos. Esta funcionalidad buscó asegurar una administración eficiente y estructurada de la información dentro del sistema.

Desde el inicio de la iteración, se definieron objetivos concretos y se asignaron tareas específicas, basadas en las historias de usuario priorizadas durante la planificación. Para mantener una visión clara del progreso y detectar posibles impedimentos de forma oportuna, se utilizaron herramientas de gestión que facilitaron el seguimiento continuo del trabajo.

El desarrollo se llevó a cabo de manera iterativa, con la construcción progresiva de componentes que se integraron adecuadamente al sistema existente. Se puso especial atención en garantizar la coherencia entre la interfaz de usuario y la base de datos, asegurando que toda la información capturada se almacenara correctamente y se reflejara en tiempo real en la plataforma. Para ello, se

diseñaron formularios funcionales y vistas dinámicas que respondieron de forma eficiente a las interacciones del usuario.

La comunicación constante entre el Desarrollador y el Dueño del Producto resultó clave a lo largo de esta fase. Este intercambio permitió resolver dudas, ajustar requisitos en función de nuevas necesidades técnicas y priorizar tareas conforme avanzaba el desarrollo. Gracias a este diálogo activo, se logró mantener la alineación con los objetivos del *Sprint* y garantizar la calidad del producto entregado.

Al finalizar el *Sprint*, todas las funcionalidades contempladas en la planificación fueron desarrolladas, probadas e integradas exitosamente al sistema. Cada componente fue sometido a validaciones funcionales que confirmaron su correcto comportamiento y su coherencia con los requisitos definidos. Como resultado, se obtuvo un módulo estable, funcional y completamente operativo para la gestión de proyectos, que permite a los usuarios crear, consultar, actualizar y eliminar registros de manera eficiente. Este avance no solo cumplió con los objetivos del *Sprint*, sino que también sentó las bases para futuras ampliaciones del sistema, al establecer una arquitectura sólida sobre la cual se podrán construir nuevas funcionalidades. En este sentido, el módulo quedó listo para ser presentado en el evento de revisión del *Sprint*, donde se evaluaría su alineación con las expectativas del Dueño del Producto y su aporte al desarrollo incremental de la herramienta.

Revisión del *Sprint*

Como parte del evento de revisión, se llevó a cabo una reunión entre el Dueño del Producto y el Desarrollador, en la cual se presentaron y evaluaron las funcionalidades desarrolladas durante el *Sprint*. Cada historia de usuario fue analizada conforme a los criterios de aceptación previamente establecidos.

El Dueño del Producto validó que las funcionalidades implementadas cumplían con los requisitos definidos y expresó su conformidad con los avances logrados. En particular, las funcionalidades de creación, visualización, modificación y eliminación de proyectos fueron consideradas completas y operativas, facilitando una gestión eficiente de la información dentro del sistema. Además, se confirmó el correcto funcionamiento de la opción para cerrar sesión, la cual proporciona a los usuarios un método seguro para finalizar su acceso y regresar a la pantalla de inicio, fortaleciendo así la seguridad y usabilidad de la herramienta. A continuación, se presentan capturas de pantalla que ilustran las funcionalidades implementadas:

Figura 49 Formulario de Creación de Proyecto.

La Figura 4.5 muestra la interfaz correspondiente al formulario de creación de proyectos dentro del sistema. Esta funcionalidad permite a los usuarios registrar nuevos proyectos mediante la introducción de datos esenciales, como el nombre del proyecto y una descripción general.

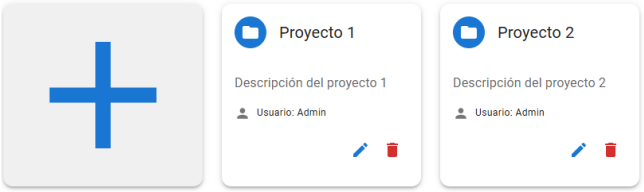


Figura 50 Pantalla de Listado de Proyectos.

La Figura 4.6 presenta la interfaz de listado de proyectos registrados en el sistema. Esta pantalla permite visualizar de manera ordenada y accesible la información esencial de cada proyecto, incluyendo el nombre, la descripción y el usuario responsable del proyecto.

Figura 51 Pantalla de Edición de Proyecto.

La Figura 4.7 muestra la interfaz destinada a la edición de proyectos previamente registrados en el sistema. Esta funcionalidad permite a los usuarios actualizar la información de un proyecto, incluyendo campos como el nombre y la descripción, con el objetivo de reflejar cambios en su alcance, especificaciones o enfoque.

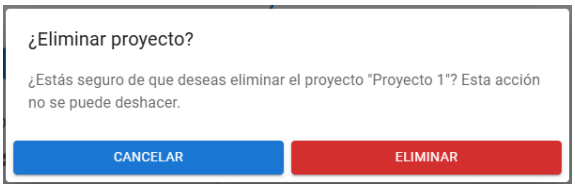


Figura 52 Pantalla de Eliminación de Proyecto.

La Figura 4.8 presenta la interfaz que permite la eliminación de proyectos registrados en el sistema. Esta funcionalidad brinda al usuario la opción de eliminar de forma definitiva un proyecto, asegurando que tanto los datos almacenados en la base de datos como su representación en la interfaz sean removidos.

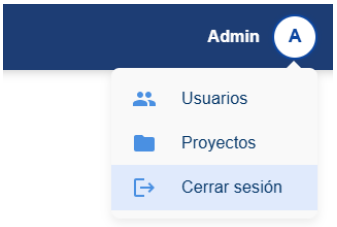


Figura 53 Pantalla de Cierre de Sesión.

La Figura 4.9 muestra la interfaz que permite a los usuarios cerrar su sesión de forma segura en el sistema. Al seleccionar la opción de cierre de sesión, se finaliza la sesión activa del usuario,

garantizando la protección de la información y la privacidad. Esta acción redirige al usuario a la pantalla de autenticación, impidiendo el acceso no autorizado hasta que se inicie una nueva sesión.

Retrospectiva del *Sprint*

Una vez concluida la revisión del incremento, se llevó a cabo la reunión de retrospectiva correspondiente al tercer *Sprint*. Esta sesión tuvo como propósito reflexionar sobre el desarrollo del trabajo realizado, identificar los aspectos que funcionaron adecuadamente, detectar oportunidades de mejora y establecer acciones concretas que permitan optimizar el proceso en los próximos ciclos.

Durante la retrospectiva, el equipo analizó tanto las fortalezas como los desafíos afrontados durante el Sprint. Entre los aspectos positivos, se destacó la completa finalización de las historias de usuario planificadas dentro del plazo establecido, así como la integración efectiva de las funcionalidades con la base de datos, lo que garantizó la estabilidad y coherencia del sistema. La comunicación constante y fluida entre el Desarrollador y el Dueño del Producto contribuyó de manera significativa a mantener el enfoque y a tomar decisiones oportunas frente a los retos técnicos que surgieron.

Por otro lado, se identificaron áreas de mejora, entre las cuales sobresale la necesidad de implementar validaciones rigurosas en las direcciones de correo electrónico durante el registro de usuarios, con el objetivo de asegurar la calidad y seguridad de los datos ingresados al sistema. Además, se evaluó la historia de usuario relacionada con la recuperación de cuenta, la cual fue considerada de baja prioridad para la etapa actual del proyecto, por lo que se decidió posponer su desarrollo para futuras iteraciones.

- **Acciones a implementar:** Como resultado de esta reflexión, se definieron acciones concretas para los siguientes *Sprints*: priorizar la incorporación de la validación de correos electrónicos en el proceso de registro de usuarios, y planificar la implementación de la funcionalidad de recuperación de cuenta en etapas posteriores. Estas medidas buscan fortalecer la seguridad y usabilidad del sistema, al tiempo que aseguran que las prioridades del proyecto se mantengan alineadas con los objetivos definidos.

La retrospectiva concluyó con una valoración positiva sobre el progreso alcanzado durante el Sprint. El equipo reconoció la importancia de mantener una actitud abierta y flexible ante los cambios, entendiendo que la mejora continua es un principio fundamental dentro de la metodología ágil. Se reafirmó el compromiso de perfeccionar constantemente los procesos de trabajo y de incorporar mejoras progresivas basadas en la experiencia adquirida, con el fin de aumentar la eficiencia del desarrollo y elevar la calidad del producto final. Esta disposición para el aprendizaje y la adaptación se consolidó como un pilar esencial para afrontar con éxito los desafíos de las próximas iteraciones, garantizando un avance sostenido y alineado con los objetivos del proyecto.

4.2.6. Cuarto *Sprint*: Desarrollo de funcionalidades para la gestión de archivos

El cuarto *Sprint* representó un avance significativo en la evolución de la herramienta, orientado a fortalecer la gestión de los proyectos desde una perspectiva más funcional e integrada. Durante esta iteración, el equipo centró sus esfuerzos en desarrollar capacidades que permitieran a los usuarios administrar los archivos adjuntos asociados a cada proyecto, incluyendo su selección, descarga y eliminación. Estas funcionalidades resultaban fundamentales para enriquecer el entorno de trabajo y facilitar el uso de información contextual relevante durante la generación de diagramas UML, especialmente en sistemas RAG. De forma complementaria, se dio inicio a la implementación de un mecanismo de validación automática para las historias de usuario, con el propósito de asegurar que los datos enviados al LLM cumplieran con una estructura clara, coherente y alineada a los estándares

del sistema. Las funcionalidades priorizadas en este *Sprint* fueron seleccionadas tanto por su impacto directo en la experiencia del usuario como por su papel estratégico en la consolidación de una arquitectura robusta y escalable para el sistema.

Planificación del *Sprint*

Durante la planificación del cuarto *Sprint*, el equipo conformado por el Dueño del Producto y el Desarrollador definió como objetivo principal la implementación de funcionalidades orientadas a la gestión de archivos adjuntos dentro de los proyectos registrados en el sistema. Estas funcionalidades permitirían a los usuarios seleccionar, descargar y eliminar archivos asociados a sus proyectos, facilitando así una administración más eficiente y personalizada de los recursos documentales. Además, se decidió iniciar el desarrollo de un mecanismo de validación automática para las historias de usuario antes de ser enviadas al LLM, con el fin de asegurar que las entradas estén correctamente estructuradas y cumplan con los estándares definidos para una generación precisa de diagramas UML.

Estas dos líneas de trabajo fueron consideradas prioritarias, ya que abordaban necesidades funcionales identificadas en ciclos anteriores. Por un lado, la gestión de archivos adjuntos era fundamental para mejorar la interacción de los usuarios con la documentación del proyecto, aportando mayor flexibilidad en su uso dentro del sistema. Por otro lado, la validación previa de historias de usuario representaba un paso clave hacia la mejora continua en la calidad de los datos que alimentan al modelo de lenguaje, incrementando la fiabilidad de las salidas generadas. Para alcanzar dicho objetivo, se seleccionaron del *Product Backlog* las siguientes historias de usuario, que fueron integradas al *Sprint Backlog*:

1. **Como** usuario del sistema *"Dueño del Producto"*, **quiero** poder seleccionar/marcar los archivos adjuntos de un proyecto, **para** su uso en la herramienta.

2. **Como** usuario del sistema *"Dueño del Producto"*, **quiero** poder descargar los archivos adjuntos de un proyecto, **para** su visualización.

3. **Como** usuario del sistema *"Dueño del Producto"*, **quiero** poder eliminar los archivos adjuntos de un proyecto, **para** su no visualización en el sistema.

4. **Como** Dueño del Producto, **quiero** que el sistema valide las historias de usuario antes de enviarlas al LLM, **para** garantizar que el modelo reciba entradas bien estructuradas.

Durante esta etapa de planificación, se desglosaron las tareas técnicas necesarias para el desarrollo de cada funcionalidad, se estimaron los tiempos de implementación y se definieron los criterios de aceptación correspondientes. Asimismo, se acordó mantener una comunicación fluida entre los integrantes del equipo, lo que permitiría realizar ajustes oportunos y garantizar que el desarrollo se mantuviera alineado con los objetivos del Sprint y del proyecto en general.

Objetivo del *Sprint*

El objetivo principal de esta iteración fue ampliar las capacidades del sistema mediante la incorporación de funciones que facilitaran la gestión de archivos adjuntos en los proyectos, permitiendo su selección, descarga y eliminación de forma controlada. Asimismo, se abordó el desarrollo inicial de un mecanismo de validación automática de historias de usuario, con el fin de asegurar que las entradas al LLM estén correctamente estructuradas y cumplan con los estándares requeridos, mejorando así la calidad del procesamiento posterior.

Ejecución del *Sprint*

Durante el desarrollo del cuarto Sprint, se implementaron de forma progresiva las funcionalidades correspondientes a las historias de usuario seleccionadas, abordando dos áreas clave: la gestión de archivos adjuntos en los proyectos y la validación automática de historias de usuario. El enfoque adoptado permitió construir e integrar estas características al sistema de manera iterativa, garantizando su correcta funcionalidad y alineación con los requisitos definidos.

En cuanto a la gestión de archivos adjuntos, se desarrollaron tres operaciones principales: selección, descarga y eliminación. La funcionalidad de selección de archivos permite a los usuarios marcar documentos específicos dentro de un proyecto, facilitando su posterior inclusión en procesos como la generación de diagramas UML mediante el sistema RAG. Por su parte, la descarga de archivos fue implementada con una interfaz intuitiva que permite obtener los documentos asociados a un proyecto de manera rápida y confiable, sin afectar la estructura de datos. Finalmente, la opción de eliminación de archivos garantiza que los elementos removidos desaparezcan tanto de la interfaz como del sistema de almacenamiento, asegurando la integridad y actualización de la información en tiempo real.

En paralelo, se avanzó en la implementación de un sistema de validación previa para historias de usuario, cuyo propósito es asegurar que los textos enviados al LLM estén bien estructurados y cumplan con los criterios formales establecidos. Esta validación verifica aspectos como la presencia de formato adecuado, coherencia gramatical y alineación con los estándares definidos para el procesamiento automático. Con esta funcionalidad, se sientan las bases para mejorar la calidad de los insumos utilizados en la generación de diagramas UML y reducir posibles errores derivados de entradas mal formuladas.

Revisión del *Sprint*

Durante el evento de revisión del cuarto *Sprint*, el Dueño del Producto y el Desarrollador sostuvieron una sesión de evaluación centrada en validar el incremento generado. En esta reunión se presentó el conjunto de funcionalidades desarrolladas, verificando su comportamiento y asegurando que cumplieran con los criterios de aceptación previamente definidos para cada historia de usuario.

En primer lugar, se constató que el sistema permite seleccionar y marcar archivos adjuntos asociados a un proyecto, habilitando su posterior uso dentro del sistema RAG o su eliminación. Esta funcionalidad proporciona al usuario una forma ágil de identificar y gestionar los documentos relevantes, facilitando el flujo de trabajo durante el modelado de requisitos y diagramas UML.

Después, se evaluó la funcionalidad de descarga de archivos, la cual permite obtener los documentos almacenados de forma sencilla a través de la interfaz del sistema. Las pruebas realizadas evidenciaron que el proceso de descarga se lleva a cabo de manera eficiente, sin errores ni pérdidas de información, y con una experiencia de usuario fluida.

Respecto a la eliminación de archivos adjuntos, se comprobó que los documentos eliminados se retiran correctamente tanto de la base de datos como de la interfaz gráfica del sistema. Esta funcionalidad se comportó de manera consistente y demostró una adecuada integración con la lógica interna de la herramienta.

Finalmente, se revisó la validación automática de historias de usuario antes de su envío al LLM. Esta funcionalidad fue diseñada para identificar inconsistencias de formato o estructura, asegurando que las entradas procesadas por el sistema cumplan con los estándares definidos. Durante la sesión, se aplicaron distintos casos de prueba que confirmaron la utilidad de esta validación como paso previo para mejorar la calidad de los resultados generados.

El Dueño del Producto expresó su conformidad con el trabajo realizado, destacando que las funcionalidades entregadas fortalecen la capacidad del sistema para gestionar información de manera eficiente y confiable, y representan un avance significativo en la consolidación de la herramienta.

Retrospectiva del *Sprint*

Durante la retrospectiva del cuarto *Sprint*, el equipo realizó un análisis profundo y detallado del proceso de desarrollo e implementación de las funcionalidades trabajadas a lo largo del ciclo. Este ejercicio reflexivo permitió identificar con claridad tanto los éxitos obtenidos como las áreas que requieren atención y mejora para optimizar el rendimiento en futuras iteraciones.

- **Aspectos positivos:** En esta iteración, el equipo logró completar de manera exitosa todas las historias de usuario planificadas. Entre los logros más relevantes se encuentran la implementación eficaz de las funcionalidades para la gestión de archivos adjuntos, que incluye la capacidad para seleccionar, descargar y eliminar documentos asociados a los proyectos. Estas funcionalidades contribuyen significativamente a mejorar la interacción de los usuarios con la herramienta, facilitando un manejo más dinámico y organizado de la información. Además, se logró implementar una validación previa de las historias de usuario antes de enviarlas al LLM, lo cual representa un avance importante para asegurar que el sistema procese datos estructurados y coherentes, incrementando la calidad y precisión de los diagramas UML generados. Cabe destacar que la comunicación constante y fluida entre el Dueño del Producto y el Desarrollador jugó un papel fundamental durante todo el *Sprint*, ya que permitió identificar de manera oportuna ajustes necesarios, aclarar dudas y tomar decisiones rápidas, lo que contribuyó a mantener un desarrollo alineado con los objetivos establecidos y a evitar retrasos.
- **Aspectos a mejorar:** No obstante, durante la retrospectiva también se señalaron oportunidades para perfeccionar ciertos aspectos del sistema. En particular, se detectó la necesidad de optimizar la interfaz de usuario en el proceso de selección y marcado de archivos adjuntos. La experiencia mostró que esta funcionalidad puede resultar poco intuitiva y podría beneficiarse de una mejora que facilite a los usuarios la identificación rápida y precisa de los documentos relevantes. La implementación de esta mejora contribuiría a incrementar la eficiencia en el uso del sistema, reduciendo el tiempo requerido para gestionar los archivos y evitando posibles errores o confusiones. Por esta razón, se acordó que esta optimización será priorizada en próximas iteraciones, con el objetivo de ofrecer una experiencia de usuario más amigable y efectiva, acorde con los estándares de calidad que se desean para la herramienta.

La retrospectiva del cuarto *Sprint* evidenció no solo el avance técnico y funcional alcanzado durante la iteración, sino también el compromiso del equipo con los principios de mejora continua y adaptación progresiva. Este ejercicio de reflexión permitió consolidar aprendizajes clave y reforzar la importancia de mantener una comunicación efectiva, una planificación clara y una orientación constante hacia la calidad del producto. La práctica reiterada de revisar y ajustar el proceso en función de la experiencia adquirida demuestra la madurez del equipo en la aplicación de la metodología ágil, asegurando que cada *Sprint* contribuya de manera significativa al crecimiento del sistema y a la satisfacción de las necesidades de los usuarios finales.

4.2.7. Quinto *Sprint*: Integración del sistema RAG y los LLMs seleccionados

El quinto *Sprint* representó un punto clave en el desarrollo de la herramienta, al integrar el sistema RAG y consolidar el flujo completo de la generación de los diagramas UML. Durante este

Sprint se integraron los modelos Gemini 2.5 Pro y Claude Sonnet 4 como componentes centrales del procesamiento, permitiendo transformar historias de usuario en diagramas UML tanto de clases como de actividades. En esta iteración se fortaleció de manera significativa la arquitectura de la herramienta, al integrar procesos de validación, recuperación de información contextual, la construcción del *prompt* y la comunicación con los LLMs.

Planificación del *Sprint*

Durante la planificación del quinto *Sprint*, el Dueño del Producto y el Desarrollador definieron como objetivo principal la integración del sistema RAG dentro del flujo de generación de diagramas UML, con el propósito de enriquecer las historias de usuario mediante información contextual proveniente de glosarios, documentos y recursos asociados a cada proyecto. Este enriquecimiento permitiría generar *prompts* más completos y precisos, optimizando la calidad de las salidas producidas por los LLMs.

Asimismo, se determinó necesario desarrollar un módulo encargado de construir el *prompt* final a partir de la combinación entre la historia de usuario validada y la información recuperada por el sistema RAG. Dicho *prompt* sería posteriormente enviado al LLM correspondiente, por lo que también se consideró fundamental adaptar el *backend* para gestionar adecuadamente las solicitudes hacia los distintos LLMs, manejar sus respuestas y garantizar la coherencia de su respuesta. Para alcanzar dicho objetivo, se seleccionaron del *Product Backlog* las siguientes historias de usuario, que fueron integradas al *Sprint Backlog*:

1. **Como** usuario del sistema *"Dueño del Producto"*, **quiero** que el sistema recupere información contextual mediante un sistema RAG, **para** complementar las historias de usuario.
2. **Como** usuario del sistema *"Dueño del Producto"*, **quiero** que el sistema construya un *prompt* enriquecido, **para** generar una entrada estructurada y completa.
3. **Como** usuario del sistema *"Dueño del Producto"*, **quiero** que el sistema envíe correctamente el *prompt* al LLM seleccionado, **para** obtener una respuesta generada acorde al contexto del proyecto.
4. **Como** usuario del sistema *"Dueño del Producto"*, **quiero** visualizar el diagrama UML generado por los modelos, **para** analizar su contenido dentro del flujo de trabajo del proyecto.
5. **Como** usuario del sistema *"Dueño del Producto"*, **quiero** poder descargar el diagrama UML generado, **para** almacenarlo localmente o utilizarlo fuera de la herramienta.
6. **Como** usuario del sistema *"Dueño del Producto"*, **quiero** que el sistema almacene el diagrama generado dentro del proyecto correspondiente, **para** mantener un registro accesible de los diagramas asociados a cada historia de usuario.

Para avanzar con estas funcionalidades, se identificaron las actividades técnicas necesarias y se realizó una estimación preliminar del esfuerzo requerido para cada una. También se establecieron los criterios mínimos que permitirían considerar cada historia de usuario como completada. Finalmente, se acordó un esquema de seguimiento continuo durante la iteración, con el fin de ajustar prioridades y resolver cualquier incidencia que surgiera durante el desarrollo.

Objetivo del *Sprint*

El objetivo de esta iteración fue completar la integración del sistema RAG dentro del flujo de generación automática de diagramas UML, incorporando los procesos de recuperación contextual, construcción del *prompt* enriquecido y comunicación con los modelos de lenguaje seleccionados.

Asimismo, se buscó habilitar la visualización, descarga y almacenamiento del diagrama generado dentro del proyecto correspondiente, con el fin de consolidar un ciclo de trabajo continuo que permitiera transformar historias de usuario validadas en artefactos UML listos para su consulta y gestión dentro de la herramienta.

Ejecución del *Sprint*

Durante la ejecución del quinto *Sprint*, se realizó la integración de los componentes que conforman el flujo completo de generación automática de diagramas UML. El trabajo se organizó en torno a tres áreas principales: la recuperación contextual mediante el sistema RAG, la construcción del *prompt* enriquecido y la comunicación con los modelos de lenguaje seleccionados.

En primer lugar, se implementó y dejó completamente operativo el módulo responsable de la recuperación de información mediante el sistema RAG. Este proceso permitió enriquecer las historias de usuario con definiciones, conceptos y referencias extraídas de los recursos del proyecto, proporcionando un contexto más completo para la generación automática de los diagramas UML. Después, se desarrolló el componente encargado de generar el *prompt* final, combinando la historia de usuario validada con especificaciones para la generación de los diagramas UML. El resultado fue una entrada estructurada y coherente, adecuada para ser procesada por Gemini-2.5-Pro y Claude Sonnet 4. También se concluyó la integración del backend con los modelos de lenguaje, implementando los procesos necesarios para enviar el *prompt*, recibir la respuesta generada y transformarla en un formato apto para la herramienta. Esto incluyó la normalización del contenido devuelto y su conversión en el diagrama UML correspondiente. Finalmente, se habilitó la visualización del diagrama dentro del proyecto, junto con las funciones de descarga y almacenamiento interno. Con ello, el sistema completó el ciclo de generación automática, permitiendo que el usuario obtenga, consulte y gestione los diagramas generados directamente en la interfaz.

Revisión del *Sprint*

Como parte del evento de revisión del quinto *Sprint*, el Dueño del Producto y el Desarrollador evaluaron el incremento obtenido, verificando que todas las funcionalidades planificadas estuvieran finalizadas y operando conforme a los criterios establecidos. La sesión permitió examinar de manera integral el flujo completo de generación de diagramas UML, desde la preparación de la entrada hasta la obtención del resultado final. El Dueño del Producto confirmó que las funcionalidades desarrolladas cumplían con los requisitos establecidos y expresó su conformidad con los resultados alcanzados. Entre los aspectos revisados se verificó el funcionamiento del sistema RAG para la recuperación de información contextual, la correcta construcción del *prompt* enriquecido y la interacción con el modelo Gemini 2.5 Pro. Asimismo, se constató el adecuado postprocesamiento de las respuestas generadas por los modelos, lo cual permitió extraer únicamente el diagrama UML en sintaxis PlantUML. También se verificó que el diagrama pueda visualizarse en la interfaz del proyecto, descargarse sin inconvenientes y almacenarse correctamente dentro del sistema, completando así el flujo de generación automática previsto para esta iteración. A continuación, se presentan capturas de pantalla que ilustran las funcionalidades implementadas:

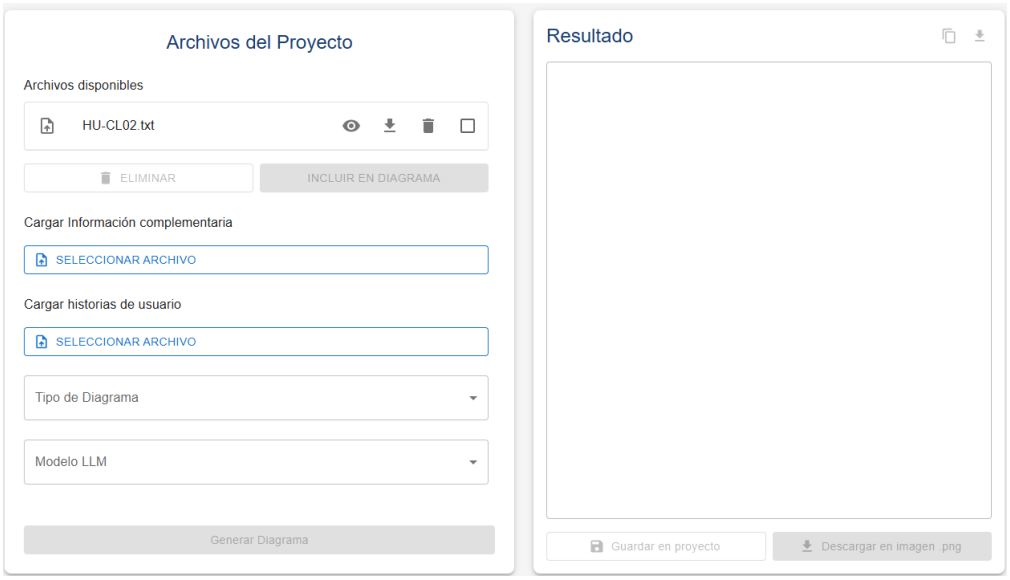


Figura 54 Pantalla de gestión de archivos y generación de diagramas UML en el proyecto.

La Figura 4.10 muestra la pantalla principal del proyecto, donde el usuario puede visualizar los archivos disponibles, cargar información complementaria y subir historias de usuario. Desde esta interfaz se selecciona el tipo de diagrama a generar, el LLM a utilizar y se accede a las opciones de visualización, descarga y almacenamiento del diagrama UML generado.

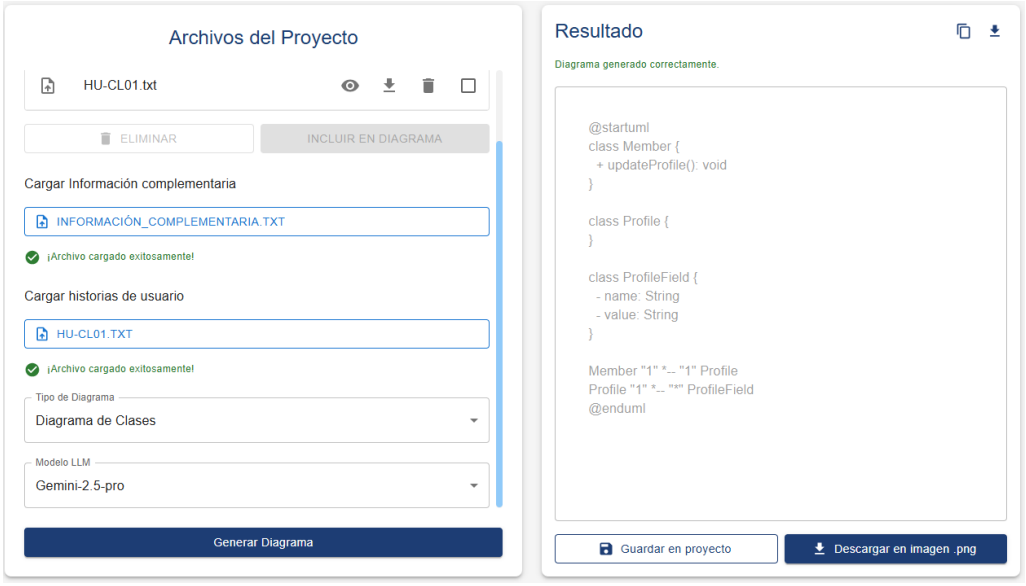


Figura 55 Generación exitosa de un diagrama de clases a partir de una historia de usuario.

La Figura 4.11 muestra la interfaz del proyecto después de cargar los archivos necesarios y ejecutar el proceso de generación del diagrama de clases. En el panel izquierdo se visualizan la información complementaria y la historia de usuario incorporadas correctamente, junto con la selección del tipo de diagrama y del LLM utilizado. En el panel derecho se presenta el resultado obtenido, donde se despliega la sintaxis PlantUML correspondiente al diagrama de clases generado, además de las opciones para almacenarlo dentro del proyecto o descargarlo como imagen.

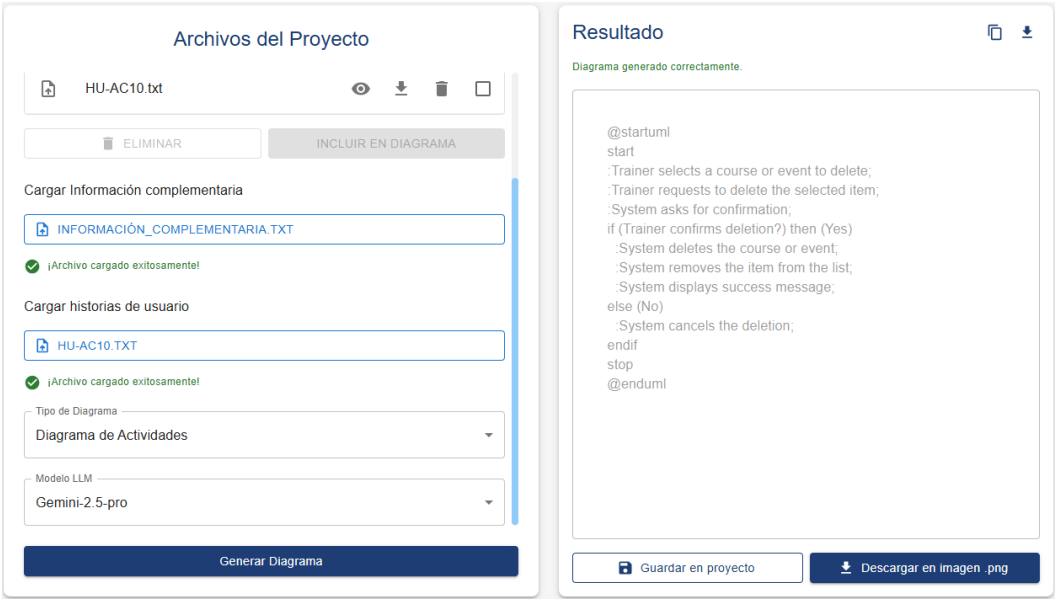


Figura 56 Generación de un diagrama de actividades a partir de una historia de usuario.

La Figura 4.12 muestra la interfaz del proyecto después de cargar la información complementaria y la historia de usuario correspondiente. Tras seleccionar el tipo de diagrama y el LLM, el sistema genera el diagrama de actividades en sintaxis PlantUML, mostrado en el panel derecho. Desde esta sección, el usuario puede guardar el diagrama dentro del proyecto o descargarlo como imagen.

Retrospectiva del *Sprint*

Durante la retrospectiva del quinto *Sprint*, el equipo realizó un análisis sobre el desarrollo de las actividades y los resultados obtenidos en esta iteración. Este espacio permitió reflexionar sobre aquello que funcionó adecuadamente y los aspectos que podrían optimizarse en proyectos futuros con características similares.

- **Aspectos positivos:** Se destacó que todas las funcionalidades previstas para este *Sprint* fueron completadas de acuerdo con lo planeado, permitiendo consolidar el flujo completo de generación automática de diagramas UML. La integración del sistema RAG, la construcción del *prompt* enriquecido y la interacción con los modelos de lenguaje se incorporaron sin afectar la operatividad general de la herramienta. De igual forma, la visualización, descarga y almacenamiento del diagrama generado se integraron adecuadamente en la interfaz del proyecto, ofreciendo un proceso continuo y claro para el usuario. La coordinación entre el Dueño del Producto y el Desarrollador facilitó la toma oportuna de decisiones y permitió mantener un ritmo constante de trabajo durante la iteración.
- **Aspectos a mejorar:** Se identificó que la presentación visual del diagrama generado podría beneficiarse de ajustes adicionales, especialmente en lo referente a la claridad y organización del contenido mostrado en la interfaz. Aunque el sistema cumple con su función, un refinamiento en la forma de despliegue podría hacer más accesible la interpretación del diagrama para el usuario final. Este punto fue registrado como un área de oportunidad en caso de que se retomen mejoras posteriores.

La retrospectiva permitió cerrar la iteración con una comprensión clara del trabajo realizado y del nivel de madurez alcanzado por la herramienta, confirmando que el flujo de generación de diagramas está completamente integrado y funcionando dentro del sistema. Además, este momento de análisis ayudó a valorar el avance conseguido a lo largo del desarrollo y la coherencia del proceso aplicado. En conjunto, los resultados de este *Sprint* consolidan la funcionalidad principal de la herramienta y marcan el cierre de esta fase del proyecto.

5. Caso de estudio

En entornos de desarrollo ágil, particularmente bajo el marco de trabajo Scrum, las historias de usuario constituyen el principal mecanismo para expresar requisitos funcionales desde la perspectiva del usuario final. Sin embargo, debido a su naturaleza deliberadamente breve y conversacional, las historias de usuario pueden presentar ambigüedades o interpretaciones distintas entre los miembros del equipo. Esta situación puede derivar en discrepancias durante el diseño, la comunicación interna y la implementación del *software*.

Si bien Scrum no establece ni exige el uso de diagramas UML, estudios previos han señalado que el modelado visual puede complementar de manera significativa las historias de usuario (Madanayake et al., 2017; Mornie et al., 2023), ya que aporta una representación estructurada del sistema, mejora la claridad conceptual y facilita la comunicación entre *Product Owner*, Equipo de Desarrollo y las partes interesadas. En este sentido, UML actúa como una herramienta de apoyo que contribuye al entendimiento compartido del producto, sin contradecir los principios ágiles de documentación mínima y colaboración continua.

Sin embargo, la construcción manual de diagramas UML sigue dependiendo del criterio y experiencia de los desarrolladores, lo que introduce variabilidad y posibles inconsistencias entre equipos o incluso dentro del mismo proyecto. Además, el proceso puede resultar costoso en tiempo y en esfuerzo, especialmente cuando es necesario actualizar el modelado UML conforme evolucionan las historias de usuario.

Ante este escenario, se propuso el desarrollo de una herramienta web basada en el uso de un Sistema RAG y LLMs, capaz de generar de manera automática diagramas UML de clases y de actividades a partir de historias de usuario. El propósito de la herramienta es agilizar la elaboración de los diagramas UML, de manera que tanto el Dueño del Producto como el Desarrollador puedan comprender con mayor claridad la historia de usuario y su representación en el modelo.

5.1. Identificación y selección de expertos

Para el desarrollo del caso de estudio se contempla la participación de expertos con experiencia en modelado UML, análisis y diseño de software, así como con participación activa en la industria del desarrollo de *software*, quienes serán responsables de realizar la evaluación cualitativa de los diagramas generados por la herramienta. La definición de este perfil responde a la necesidad de contar con evaluadores con formación técnica sólida y con experiencia práctica, capaces de emitir juicios fundamentados sobre la calidad visual, cognitiva, estructural y funcional de los diagramas UML en contextos reales de trabajo. La selección de los expertos se realizó con base en los siguientes criterios:

- Contar con formación profesional en Ingeniería de Software, Ciencias Computacionales o áreas afines.

- Desempeñarse actualmente en la industria del desarrollo de software o tener experiencia profesional reciente en este ámbito.
- Haber participado en proyectos de desarrollo de software en los que el modelado UML se utilice como herramienta de especificación, análisis o diseño.
- Poseer experiencia previa en la interpretación, creación o evaluación de diagramas UML, particularmente diagramas de clases y diagramas de actividades.
- Mostrar disposición para revisar los diagramas generados y emitir valoraciones con base en los criterios establecidos para la evaluación cualitativa.

La definición de este perfil de expertos permitirá asegurar que la evaluación cualitativa se realice desde una perspectiva técnica, práctica y comunicativa adecuada, orientada a valorar la correspondencia entre las historias de usuario y los diagramas generados, la claridad visual de los modelos, su coherencia semántica y su utilidad como apoyo para la comprensión del sistema. De este modo, se refuerza la validez cualitativa del estudio, al respaldar las observaciones y conclusiones en el conocimiento teórico y la experiencia profesional aplicada en entornos reales de desarrollo de *software*.

5.2. Diseño del caso de estudio

El diseño del caso de estudio se estructuró con el propósito de analizar de manera sistemática el comportamiento de la herramienta propuesta y valorar su utilidad para mejorar la comprensión de las historias de usuario mediante la generación automática de diagramas UML. Para ello, se definieron los elementos fundamentales que guiaron su aplicación: el objetivo del estudio, el proceso de selección del conjunto de historias de usuario, la participación de los expertos, los materiales y artefactos empleados, así como el procedimiento seguido para llevar a cabo la evaluación. Sobre esta base, se presentan los resultados obtenidos tanto para los diagramas de clases como para los diagramas de actividades, junto con las conclusiones derivadas de su análisis.

Con el fin de ofrecer una visión sintética del proceso seguido, la Figura 5.1 presenta el flujo general del diseño, ejecución y análisis-validación del caso de estudio.

5.2.1. Objetivo

El presente caso de estudio tuvo como objetivo evaluar la eficacia de la herramienta a partir de los diagramas UML generados automáticamente con base en las historias de usuario ingresadas en ella, los cuales fueron valorados por doce expertos en SE con conocimientos en modelado UML. La evaluación se realizó empleando los doce criterios de la evaluación cualitativa y sus correspondientes valores de ponderación, previamente definidos, lo que permitió analizar de manera integral tanto aspectos estructurales, como la coherencia, la correspondencia con la historia de usuario y la completitud, como aspectos perceptuales y cognitivos, entre ellos la claridad gráfica, la expresividad visual, la legibilidad y la comprensibilidad. Con ello, se buscó determinar en qué medida la herramienta contribuye a facilitar la comprensión y la comunicación de las historias de usuario mediante la generación automática de diagramas UML.

En consecuencia, este caso de estudio no solo valida el funcionamiento técnico de la herramienta, sino que también aporta evidencia empírica sobre el papel del modelado UML como práctica complementaria en Scrum, así como sobre el potencial de los modelos de lenguaje para apoyar la comprensión, el análisis y la elaboración de representaciones de software en entornos reales de trabajo.

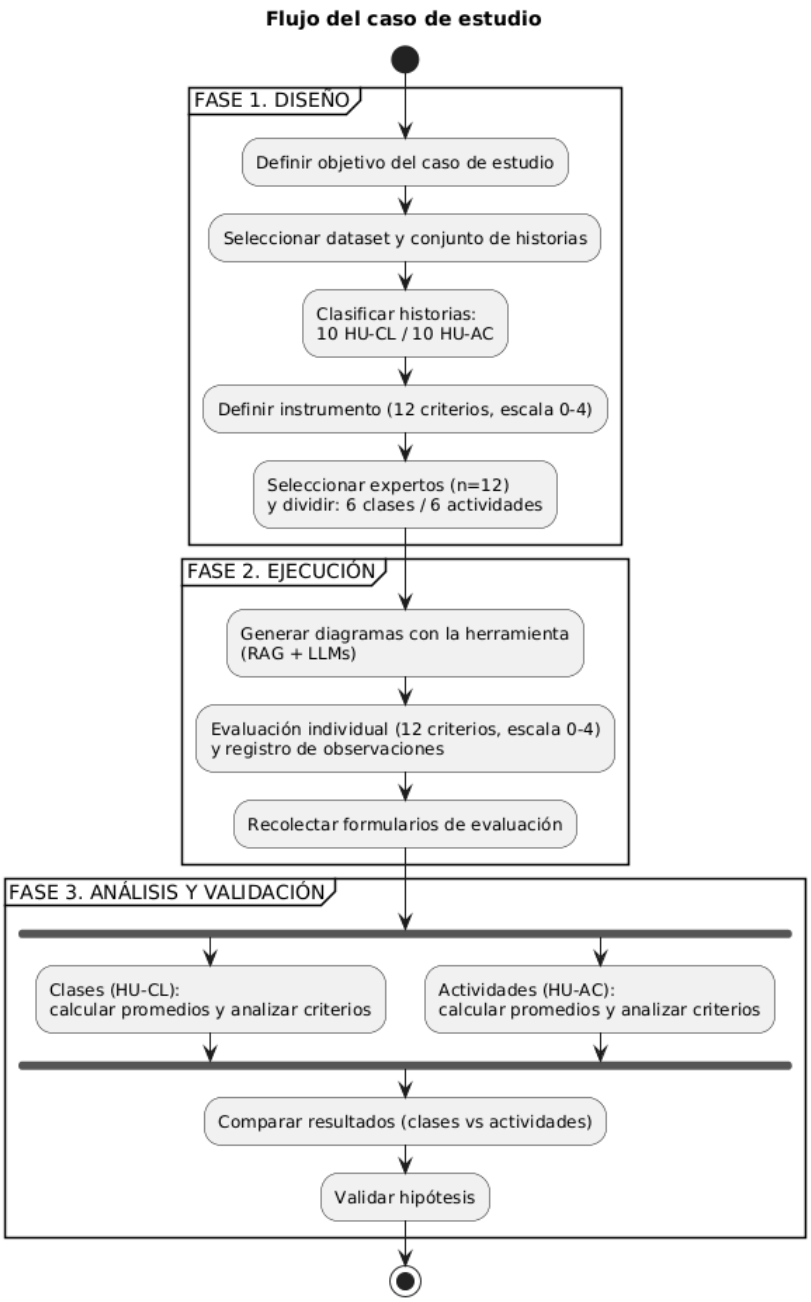


Figura 57 Flujo del diseño y ejecución del caso de estudio organizado por fases.

5.2.2. Selección del conjunto de historias de usuario

Las historias de usuario seleccionadas para este caso de estudio provienen del *dataset* "*User Stories Datasets for Agile Software Development Research*" (Felizardo et al., 2018), publicado en el repositorio *Mendeley Data*. Este conjunto reúne historias de usuario reales empleadas en proyectos de desarrollo ágil, organizadas por dominio y contexto de aplicación, lo que lo convierte en un recurso adecuado para estudios relacionados con análisis de requisitos y modelado. De dicho *dataset* se seleccionó el archivo ***g10-scrumalliance.txt***, el cual corresponde a funcionalidades del sitio web de la Scrum Alliance, una organización ampliamente reconocida en la difusión y certificación de prácticas asociadas al marco de trabajo Scrum. Este archivo fue elegido debido a que sus historias de usuario presentan las siguientes características deseables para el proceso de modelado UML:

- **Claridad semántica:** Las historias están redactadas siguiendo la estructura estándar *"As a [*rol*], I want to [*acción*], so that [*objetivo*]".*
- **Coherencia y propósito funcional definido:** Cada historia describe una funcionalidad concreta vinculada a la interacción de usuarios con el sistema.
- **Adecuación para el modelado estructural y de comportamiento:** Permiten identificar entidades, relaciones y flujos de actividades de forma natural.
- **Lenguaje accesible y no técnico:** Facilita el análisis sin ambigüedades adicionales.

Para este caso de estudio se seleccionaron 20 historias de usuario, distribuidas de la siguiente manera. La elección no fue arbitraria: se buscó incluir historias que cubrieran distintos tipos de requisitos y niveles de detalle, de modo que la herramienta pudiera evaluarse ante situaciones variadas y comparables con las que enfrentaría un equipo ágil en un proyecto real.

- Diez historias de usuario para la generación de diagramas de clases (véase Tabla 32).
- Diez historias de usuario para la generación de diagramas de actividades (véase Tabla 33).

La clasificación HU-CL y HU-AC facilita el análisis diferenciado de los dos tipos de diagramas generados y permite evaluar la herramienta considerando tanto la perspectiva estructural correspondiente al modelo estático como la perspectiva dinámica asociada al modelo de comportamiento. A continuación, se presentan las historias de usuario seleccionadas para cada categoría.

Tabla 32. Historias de usuario seleccionadas para diagramas de clases.

ID	Historia de usuario
HU-CL01	***As** a site member, **I want** to describe myself on my own page in a semi-structured way, **so that** others can learn about me.*
HU-CL02	***As** a Practitioner, **I want** to include additional details about me in my profile page, **so that** I can showcase my experience.*
HU-CL03	***As** a Trainer, **I want** to include additional details in my profile page about me, **so that** others can learn about me and decide if I am the right trainer for them.*
HU-CL04	***As** a site member, **I want** to view the profiles of other members, **so that** I can find others I might want to connect with.*
HU-CL05	***As** a site member, **I want** to search for profiles based on a few fields, **so that** I can find others I might want to connect with.*
HU-CL06	***As** a site member, **I want** to mark my profile as private, **so that** no one can learn things about me I don't want shared.*
HU-CL07	***As** a site member, **I want** to mark my email address as private even if the rest of my profile is not, **so that** no one can contact me.*
HU-CL08	***As** a trainer, **I want** to list my upcoming classes in my profile and include a link to a detailed page about each, **so that** prospective attendees can find my courses.*
HU-CL09	***As** a site visitor, **I want** to see a list of all upcoming Certification Courses, **so that** I can choose the best course for me.*
HU-CL10	***As** a site visitor, **I want** to see a list of all upcoming Events, **so that** I can decide if I want to attend any.*

Tabla 33. Historias de usuario seleccionadas para diagramas de actividades.

ID	Historia de usuario
HU-AC01	*As a site member, I want to fill out an application to become a Certified Scrum Practitioner, so that I can earn that designation.*
HU-AC02	*As a site member, I want to fill out an application to become a Certified Scrum Trainer, so that I can teach CSM and CSPO courses and certify others.*
HU-AC03	*As a site administrator, I want to read practicing and training applications and approve or reject them, so that only applicants who qualify can become CSPs or CSTs.*
HU-AC04	*As a site visitor, I want to read current news on the home page, so that I stay current on agile news.*
HU-AC05	*As a site visitor, I want to access old news that is no longer on the home page, so that I can access things I remember from the past.*
HU-AC06	*As a site member, I want to subscribe to an RSS feed of news, so that I remain sufficiently and easily informed.*
HU-AC07	*As a site visitor, I want to see a list of all upcoming Other Courses, so that I can choose the best course for me.*
HU-AC08	*As a trainer, I want to create a new course or event, so that site visitors can see it.*
HU-AC09	*As a trainer, I want to update one of my existing courses or events, so that it reflects accurate information.*
HU-AC10	*As a trainer, I want to delete one of my courses or events, so that it's no longer listed if I cancel for some reason.*

5.2.3. Expertos participantes

Para la evaluación cualitativa de los diagramas generados se contó con la participación de doce expertos en SE, todos con experiencia activa en la industria del desarrollo de *software*. Con el fin de asegurar una valoración especializada para cada tipo de diagrama, los expertos se dividieron en dos grupos independientes: seis expertos para la evaluación de los diagramas de clases y seis expertos para la evaluación de los diagramas de actividades.

Esta distribución permitió que cada conjunto de diagramas fuera analizado por profesionales con perfiles acordes a la naturaleza del modelo evaluado, fortaleciendo así la validez técnica y práctica de los resultados obtenidos. Todos los expertos desempeñan roles directamente vinculados con el desarrollo de *software* y cuentan con experiencia en el análisis, diseño e interpretación de modelos UML en contextos reales de trabajo.

La Tabla 34 presenta la caracterización de los seis expertos que participaron en la evaluación de los diagramas de clases, considerando sus años de experiencia profesional y su rol principal en la industria.

Tabla 34. Datos de los expertos que participaron en la evaluación de los diagramas de clases.

Experto	Años de experiencia	Rol principal
E1	7	Desarrollador de *software*
E2	5	Desarrollador *Frontend*

Experto	Años de experiencia	Rol principal
E3	8	Desarrollador de *software*
E4	4	Desarrollador Java *Backend*
E5	3	Desarrollador de *software*
E6	5	Desarrollador de *software*

Como se observa, los expertos en diagramas de clases presentan una experiencia profesional que oscila entre 3 y 8 años, con una mayor concentración en perfiles de desarrolladores de *software*, complementados por un perfil de *frontend* y uno de *backend* Java. Esta diversidad de roles permitió analizar los diagramas tanto desde la perspectiva estructural del modelo como desde su utilidad práctica para distintos tipos de desarrollo. La participación de estos expertos permitió valorar aspectos clave como la correcta identificación de clases, atributos y relaciones, la aplicación adecuada de la notación UML y la coherencia estructural de los modelos generados.

La Tabla 35 muestra la caracterización de los seis expertos que evaluaron los diagramas de actividades, considerando igualmente los años de experiencia y el rol principal desempeñado en la industria.

Tabla 35. Datos de los expertos que participaron en la evaluación de los diagramas de actividades.

Experto	Años de experiencia	Rol principal
E7	10	Desarrollador *Frontend*
E8	8	Desarrollador de *software*
E9	4	Desarrollador *Backend*
E10	6	Desarrollador de *software*
E11	7	Desarrollador de *software*
E12	4	Desarrollador de *software*

En este grupo, los expertos presentan una experiencia profesional comprendida también entre 4 y 10 años, con predominio nuevamente de desarrolladores de software, lo que garantizó un enfoque práctico en la evaluación de los flujos de actividades, la secuencia de acciones y la representación de decisiones dentro de los procesos modelados. Los perfiles de *frontend* y *backend* permitieron además valorar la utilidad de los diagramas desde la perspectiva de la implementación y la interpretación del comportamiento del sistema en escenarios reales de desarrollo.

La selección de expertos con experiencia activa en la industria del *software* permitió que la evaluación cualitativa no se limitara a un análisis teórico de los diagramas, sino que incorporara criterios basados en su uso real como herramientas de apoyo para la comprensión, comunicación y análisis de sistemas. La combinación de distintos roles técnicos aportó una visión equilibrada tanto desde el desarrollo *frontend*, backend como desde el desarrollo general de *software*. La experiencia profesional acumulada de los expertos participantes garantizó una valoración fundamentada sobre la claridad visual, la coherencia estructural, la consistencia semántica y la utilidad funcional de los

diagramas generados por la herramienta, fortaleciendo así la confiabilidad de los resultados del análisis cualitativo.

5.2.4. Materiales y artefactos

Para la ejecución del caso de estudio se emplean los siguientes materiales y artefactos, los cuales permitieron llevar a cabo de manera sistemática tanto la generación automática de los diagramas UML como su posterior evaluación cualitativa por parte de los expertos. Estos materiales fueron seleccionados con el propósito de asegurar la correcta implementación de la herramienta, la adecuada gestión de las historias de usuario, la generación de los diagramas de clases y de actividades, así como la recolección ordenada de los datos derivados del proceso de evaluación. En conjunto, los artefactos utilizados constituyeron el soporte técnico y metodológico necesario para garantizar la validez y confiabilidad de los resultados obtenidos en el desarrollo del caso de estudio. A continuación, se enlistan los materiales y artefactos utilizados para esta evaluación:

- **Herramienta web para la generación automática de diagramas UML:** La herramienta web desarrollada en este trabajo permite generar automáticamente diagramas UML a partir de historias de usuario redactadas en lenguaje natural. Para ello, integra un enfoque RAG, con el fin de mejorar la coherencia y precisión del diagrama resultante. Los diagramas se representan en sintaxis PlantUML, lo que facilita su visualización y edición. Esta herramienta web constituye el objeto central de análisis en el caso de estudio, ya que la evaluación se centra en determinar la calidad, claridad y utilidad de los diagramas que es capaz de generar.
- **Conjunto de historias de usuario:** El conjunto de historias de usuario empleado en este caso de estudio corresponde a las 20 historias seleccionadas previamente, organizadas en dos grupos de acuerdo con el tipo de diagrama UML a generar. Las historias orientadas a diagramas de clases se presentan en la Tabla 32, mientras que las historias destinadas a la generación de diagramas de actividades se muestran en la Tabla 33. Este conjunto constituye el insumo principal que se utiliza para la generación automática y posterior evaluación cualitativa de los diagramas UML.
- **Instrumento de evaluación cualitativa:** El instrumento de evaluación cualitativa se estructuró a partir de los 12 criterios y sus valores de ponderación, definidos previamente, organizados en las cuatro dimensiones de análisis establecidas. Este instrumento permitió registrar de manera ordenada y homogénea las valoraciones emitidas por los expertos sobre los diagramas UML generados. Este instrumento asegura que la evaluación sea sistemática, comparable y fundamentada, permitiendo obtener resultados coherentes y trazables durante el análisis, y fortaleciendo la validez interpretativa de los hallazgos del caso de estudio. A partir del instrumento descrito, se procede a la aplicación del proceso de evaluación cualitativa. En la siguiente sección se detalla el procedimiento seguido por los evaluadores, incluyendo la forma en que se presentaron los diagramas, la modalidad de revisión y el registro de las valoraciones y observaciones. Este procedimiento asegura la consistencia en la aplicación de los criterios y permite obtener resultados comparables entre los participantes.
- **Formularios elaborados en *Google Forms***: Para el registro estructurado de las evaluaciones cualitativas y de las observaciones emitidas por cada experto, se emplearon dos formularios: uno destinado a los diagramas de clases y otro a los diagramas de actividades. El formulario utilizado para la evaluación de diagramas de clases puede consultarse en:

https://docs.google.com/forms/d/e/1FAIpQLSftVymLccFmJ0kHbSpHCmYP8MmBB-KHIc5-4ntWOpZITIO_fw/viewform?usp=dialog.

Asimismo, el formulario utilizado para la evaluación de diagramas de actividades puede consultarse en:

https://docs.google.com/forms/d/e/1FAIpQLSfvizF9ZdTX3J9xLwC64O4MiJJG1RBJSIdZbSVhc2jOG4e4YA/viewform?usp=dialog.

5.2.5. Aplicación de la evaluación

La evaluación se llevó a cabo de manera individual y contempló tanto el uso de la herramienta web como la valoración cualitativa de los diagramas obtenidos. El procedimiento seguido fue el siguiente:

1. **Familiarización con la herramienta:** Cada experto recibió una explicación sobre la funcionalidad de la herramienta y la forma de ingresar historias de usuario para generar diagramas UML.
2. **Generación de diagramas:** Cada experto ingresa las historias de usuario correspondientes (HU-CL y HU-AC) en la herramienta y genera los diagramas de clases y diagramas de actividades asociados. La intención es observar la interpretación que la herramienta realiza a partir de la historia de usuario ingresada.
3. **Revisión de los diagramas generados:** Los profesionales examinan los diagramas obtenidos, considerando tanto la estructura como el contenido del modelo, y lo contrastan con la interpretación esperada de la historia de usuario.
4. **Aplicación de la matriz de criterios:** Con base en la matriz de evaluación cualitativa, cada experto asigna un valor entre 0 y 4 para cada criterio dentro de las cuatro dimensiones analíticas.
5. **Registro de observaciones cualitativas:** Los expertos registran comentarios que describen su percepción sobre el diagrama, señalando aspectos positivos, inconsistencias o mejoras necesarias dentro del formulario correspondiente.
6. **Entrega de resultados:** Una vez finalizada la evaluación, cada experto envía el formulario con los valores de la evaluación cualitativa realizada. Dado que la actividad es individual, se asegura la independencia del juicio y se evita la influencia entre evaluadores.

Este procedimiento permitió evaluar no solo el funcionamiento de la herramienta en un escenario de uso real, sino también la calidad, claridad y utilidad comunicativa de los diagramas generados. Al involucrar a profesionales con experiencia en modelado UML y solicitarles que utilicen directamente la herramienta, es posible obtener una valoración fundamentada tanto sobre la interpretación que la herramienta realiza de las historias de usuario, como sobre la pertinencia y precisión de los modelos producidos. De esta manera, la evaluación refleja condiciones auténticas del proceso de análisis y diseño en entornos ágiles, permitiendo identificar fortalezas, limitaciones y oportunidades de mejora en la herramienta para su uso en contextos reales de desarrollo de *software*.

5.2.6. Resultados obtenidos de los diagramas de clases

En esta sección se presentan los resultados correspondientes a la evaluación cualitativa aplicada a los diagramas de clases generados por la herramienta durante el caso de estudio. Los diagramas elaborados por los expertos participantes mediante el uso de la herramienta pueden consultarse en:

https://github.com/LoreMartinez/Repositorio_Tesis_DiagramasUML_Archivos/tree/main/Archivos_evaluacion_Caso_Estudio/Diagramas_clases_generados_expertos_E1-E6.

La valoración fue realizada por seis expertos con experiencia profesional en el desarrollo de *software* y modelado UML, quienes analizaron un conjunto de diez diagramas de clases que se enlistan en la Tabla 32 a partir de los doce criterios mostrados en la Tabla 20 definidos en el instrumento de evaluación, con su correspondiente valor de ponderación. Los registros completos de las evaluaciones realizadas por los seis expertos, incluyendo las valoraciones asignadas a cada criterio para cada uno de los diagramas de clases analizados, se encuentran disponibles para su consulta en: https://github.com/LoreMartinez/Repositorio_Tesis_DiagramasUML_Archivos/blob/main/Archivos_evaluacion_Caso_Estudio/Resultados_evaluacion_cualitativa_diagramas_clases.xlsx.

Asimismo, con el propósito de ilustrar detalladamente el proceso de generación realizado por la herramienta, en el Apéndice A se presenta un ejemplo completo de generación de un diagrama de clases, incluyendo la historia de usuario original, el *prompt* utilizado, la respuesta generada por el modelo de lenguaje y el diagrama UML resultante. Con base en estas valoraciones, expresadas en una escala de 0 a 4 previamente descrita (véase Tabla 21), se calculó el promedio global por evaluador, cuyos resultados se presentan en la Tabla 36.

Tabla 36. Promedio global por experto para los diagramas de clases generados.

Experto	Promedio
E1	**3.93**
E2	1.97
E3	2.63
E4	2.97
E5	**3.91**
E6	2.68

Los promedios globales obtenidos por cada uno de los expertos evidencian diferencias relevantes en los patrones de valoración del conjunto de diagramas analizados. En particular, los expertos E1 y E5 alcanzan los valores más altos, con promedios de 3.93 y 3.91 respectivamente, lo que refleja una percepción global favorable respecto a la calidad de los diagramas de clases generados por la herramienta. Este comportamiento sugiere que, desde la perspectiva de ambos evaluadores, los modelos presentan un grado elevado de adecuación técnica y claridad estructural. En contraste, el experto E2 presenta el promedio más bajo, con un valor de 1.97, lo que indica una postura más estricta o conservadora en la aplicación de los criterios de valoración. Esta tendencia puede estar asociada a un enfoque analítico más crítico, a una mayor exigencia en términos de consistencia semántica o a una interpretación más rigurosa de los estándares de calidad del modelado UML.

Por su parte, los expertos E3, E4 y E6 registran valores intermedios, con promedios de 2.63, 2.97 y 2.68 respectivamente. Estos resultados reflejan una percepción moderada de la calidad de los diagramas, en la que se reconocen tanto fortalezas formales como limitaciones en aspectos conceptuales y de utilidad práctica. En conjunto, este grupo de evaluadores contribuye a equilibrar la tendencia general de la evaluación, reduciendo la influencia de valoraciones extremas. Con el objetivo de analizar el desempeño de los diagramas desde una perspectiva más detallada, la Tabla 37 presenta

los promedios obtenidos por cada criterio de evaluación, organizados de acuerdo con las dimensiones definidas en el instrumento de evaluación cualitativa.

Tabla 37. Promedio global por criterio para los diagramas de clases generados.

Dimensión	Criterio	Media
Claridad y consistencia visual	Claridad semiótica	3.08
	Discriminabilidad perceptiva	3.43
	Expresividad visual	3.08
	Simplicidad	2.97
Comprensión cognitiva	Comprensibilidad	2.88
	Carga cognitiva	2.98
	Memorabilidad	3.05
Calidad estructural y semántica	Corrección sintáctica	3.35
	Consistencia semántica	3.02
	Completitud	2.77
Integración y utilidad	Integración cognitiva	2.97
	Adecuación al usuario	2.55

Los resultados obtenidos por criterio, a partir de las evaluaciones realizadas por los seis expertos, permitieron identificar con claridad las principales fortalezas y debilidades de los diagramas de clases generados por la herramienta, considerando las cuatro dimensiones definidas en el instrumento de evaluación.

En la dimensión de claridad y consistencia visual, los resultados evidenciaron un desempeño favorable. La discriminabilidad perceptiva destacó con el valor más alto de 3.43, lo que indicó que los elementos de los diagramas generados pudieron distinguirse con claridad, facilitando si identificación. Asimismo, la claridad semiótica y la expresividad visual alcanzaron valores de 3.08, lo que sugirió que los símbolos y recursos gráficos empleados representaron de manera adecuada los conceptos del dominio. En contraste, la simplicidad registró un valor de 2.97, lo que, si bien reflejó un nivel adecuado de organización visual, también sugirió que existieron oportunidades para mejorar la disposición de los elementos y hacer los diagramas más claros.

En la dimensión de comprensión cognitiva, la memorabilidad presentó el valor más alto de 3.05, lo que indicó que los diagramas generados resultaron relativamente fáciles de recordar. Le siguió la carga cognitiva con un valor de 2.98, lo que sugirió que el esfuerzo requerido para interpretarlos fue moderado y acorde con la estructura presentada. Por su parte, la comprensibilidad alcanzó 2.88, mostrando que, aunque los diagramas pudieron entenderse en términos generales, todavía fue posible reducir ciertas ambigüedades para facilitar su interpretación.

Con respecto a la calidad estructural y semántica, la corrección sintáctica obtuvo el valor más alto de 3.35, lo que confirmó el uso adecuado de la notación UML en los diagramas generados. En un nivel cercano, la consistencia semántica alcanzó el valor de 3.02, lo cual evidenció que los elementos representados mantuvieron una correspondencia generalmente coherente con el dominio del problema. Sin embargo, la completitud presentó un valor de 2.77, lo que indicó que, aunque se representaron los componentes principales del sistema, aún se omitieron algunos elementos relevantes. En la dimensión de integración y utilidad, la integración cognitiva alcanzó un valor de 2.97, lo que reflejó que los diagramas generados permitieron una comprensión adecuada de la

estructura global del sistema. En cambio, la adecuación al usuario obtuvo el valor más bajo de 2.55, lo que indicó que un nivel moderado de ajuste a las necesidades y características del usuario final, evidenciando un área de mejora en este aspecto.

En conjunto, los resultados mostraron un comportamiento consistente entre las distintas dimensiones evaluadas, con un desempeño favorable en los aspectos visuales y sintácticos, y niveles adecuados en términos de comprensión y coherencia semántica. Los valores más bajos se concentraron en la completitud de los modelos y en la adecuación al usuario, lo que evidenció áreas específicas que requieren fortalecimiento. En este sentido, los hallazgos permitieron identificar que la herramienta cumple con su propósito como apoyo en la generación inicial de diagramas, aunque aún resulta necesario mejorar la cobertura de los elementos representados y su ajuste al contexto de uso. Estos resultados aportan evidencia sobre el potencial de la propuesta, al mismo tiempo que orientan futuras mejoras para incrementar su utilidad y precisión.

5.2.6.1. Observaciones cualitativas de los expertos sobre los diagramas de clases generados

Las observaciones emitidas por los expertos coincidieron en señalar que la herramienta de generación automática de diagramas de clases ofreció un apoyo valioso durante las etapas iniciales del análisis y diseño, principalmente por su capacidad para transformar de manera rápida las descripciones en lenguaje natural en representaciones estructuradas de entidades y relaciones. Como parte de la evaluación, se les planteó la pregunta, *"¿En qué medida considera que el uso de la herramienta de generación automática de diagramas de clases UML facilita la comprensión de las historias de usuario en el contexto de metodologías ágiles?"*

Las respuestas dieron como resultado un promedio de **2.83** en una escala de 0 a 4, lo cual refleja una percepción moderadamente positiva en cuanto a su utilidad para comprender las historias de usuario, aunque con reservas importantes sobre la calidad conceptual de los modelos generados.

Desde el punto de vista técnico, los expertos señalaron que la herramienta permitió acelerar de forma notable el proceso de prototipado y el análisis inicial de soluciones, resultando especialmente útil durante las primeras etapas del diseño. Asimismo, indicaron que la generación automática de los diagramas ayudó a transformar las descripciones en lenguaje natural de las historias de usuario en estructuras técnicas concretas, lo que facilitó la comprensión de los requisitos y la discusión entre los miembros del equipo de desarrollo.

No obstante, las observaciones también pusieron en evidencia limitaciones importantes en la calidad conceptual de los diagramas generados. De manera recurrente, los expertos señalaron la presencia de clases innecesarias, especialmente en los diagramas relacionados con la gestión de privacidad del perfil, así como la omisión de atributos fundamentales, como el identificador de las clases. Estas carencias afectaron la completitud del modelo y redujeron su utilidad directa como artefacto de diseño definitivo.

Uno de los aspectos críticos señalados por los expertos fue el fenómeno identificado como *"falsa sensación de corrección"*. A pesar de que los diagramas presentaron una apariencia visualmente correcta y una notación UML limpia, varios evaluadores advirtieron que esta perfección formal podía inducir a desarrolladores con menor experiencia a asumir erróneamente que la lógica subyacente también era correcta. Se documentaron casos en los que la herramienta incurrió en sobreingeniería, al generar múltiples clases y relaciones complejas cuando la solución arquitectónica real podía resolverse mediante un simple atributo, lo que implicaría un potencial riesgo de generación de deuda técnica si los resultados no fueran revisados por un especialista.

Como sugerencias de mejora, los expertos señalaron la conveniencia de permitir al usuario proporcionar mayor contexto técnico como entrada, con el fin de refinar los resultados, reducir

soluciones excesivamente literales y evitar la generación de estructuras innecesarias. Del mismo modo, destacaron la importancia de incorporar mecanismos que favorezcan la generación de modelos más completos, detallados y conceptualmente coherentes, incluyendo atributos relevantes en cada clase. En conjunto, las observaciones de los expertos permitieron concluir que la herramienta presentó un alto potencial como acelerador de prototipado y apoyo al análisis preliminar, pero que, en su estado actual, no sustituye la labor del analista o arquitecto de *software*. Los diagramas generados requirieron siempre una revisión, validación y refinamiento humano para asegurar su alineación con el dominio del problema, evitar sobreingeniería y garantizar su utilidad como documentación técnica confiable.

5.2.6.2. Análisis de errores identificados en los diagramas de clases generados

Las observaciones realizadas por los expertos permitieron identificar algunos aspectos susceptibles de mejora en determinados diagramas de clases generados durante el caso de estudio. Con el propósito de complementar los resultados de la evaluación cualitativa, se analizaron algunos casos representativos en los que se observaron diferencias relevantes respecto a los diagramas de referencia. Este análisis permitió identificar tres tipos principales de errores: omisiones, alucinaciones y errores semánticos. Las omisiones corresponden a elementos relevantes del dominio que no fueron representados en el diagrama generado. Las alucinaciones se refieren a la incorporación de clases, atributos, métodos o relaciones que no se derivan directamente de la historia de usuario o del diagrama de referencia. Por su parte, los errores semánticos ocurren cuando el modelo interpreta de forma incorrecta la intención del requisito y genera una representación conceptual distinta a la esperada.

Ejemplo 1. Omisión y alucinación de elementos en la HU-CL04

La HU-CL04 tiene como objetivo permitir que los miembros consulten perfiles y establezcan conexiones con otros usuarios. La Figura 5.2 presenta la comparación entre el diagrama de referencia y el diagrama generado por la herramienta para esta historia de usuario. En el diagrama de referencia se incluyen las clases *Member*, *Profile* y *ProfileDirectory*. Sin embargo, el diagrama generado omitió completamente la clase *ProfileDirectory* e incorporó una nueva clase denominada *PrivacySetting*. Debido a que esta clase no forma parte del modelo de referencia, su inclusión constituye una alucinación generada por el modelo. De igual forma, la ausencia de la clase *ProfileDirectory* representa una omisión de un elemento relevante para la funcionalidad descrita. Asimismo, se observa que el diagrama generado enfatiza aspectos relacionados con la privacidad de los perfiles, desviándose parcialmente del objetivo principal de la historia de usuario. Este comportamiento evidencia cómo los modelos de lenguaje pueden incorporar conceptos semánticamente plausibles, aunque estos no correspondan necesariamente a la solución esperada.

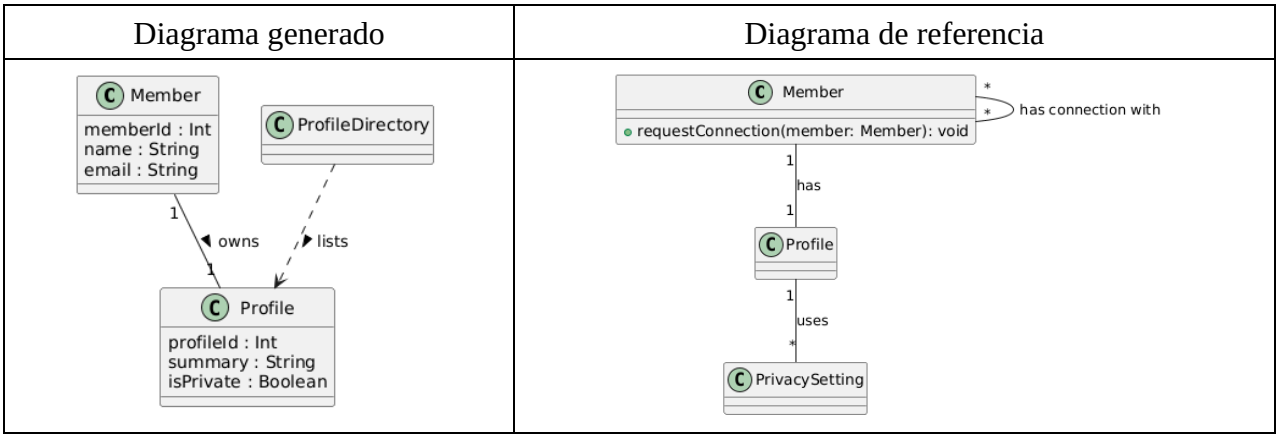


Figura 58 Comparación entre el diagrama de referencia y el diagrama generado para la HU-CL04.

Ejemplo 2. Error semántico en la HU-CL09

La Figura 5.3 presenta la comparación entre el diagrama de referencia y el diagrama generado para la historia de usuario HU-CL09. En el modelo de referencia se incluyen las clases *CertificationCourse* y *CourseCatalog*, donde esta última representa el mecanismo utilizado para organizar y publicar los cursos de certificación disponibles. No obstante, el diagrama generado omitió completamente la clase *CourseCatalog* e incorporó las clases *Visitor* y *Course*. Como resultado, la representación obtenida se centra en la interacción de un visitante con los cursos disponibles, en lugar de modelar el catálogo de cursos descrito en el modelo de referencia. Este caso constituye un ejemplo de error semántico, ya que el modelo generó una interpretación alternativa del requisito y modificó la representación conceptual esperada del dominio. Aunque el diagrama generado mantuvo coherencia interna y una sintaxis UML válida, la estructura conceptual obtenida difiere significativamente de la propuesta en el modelo de referencia.

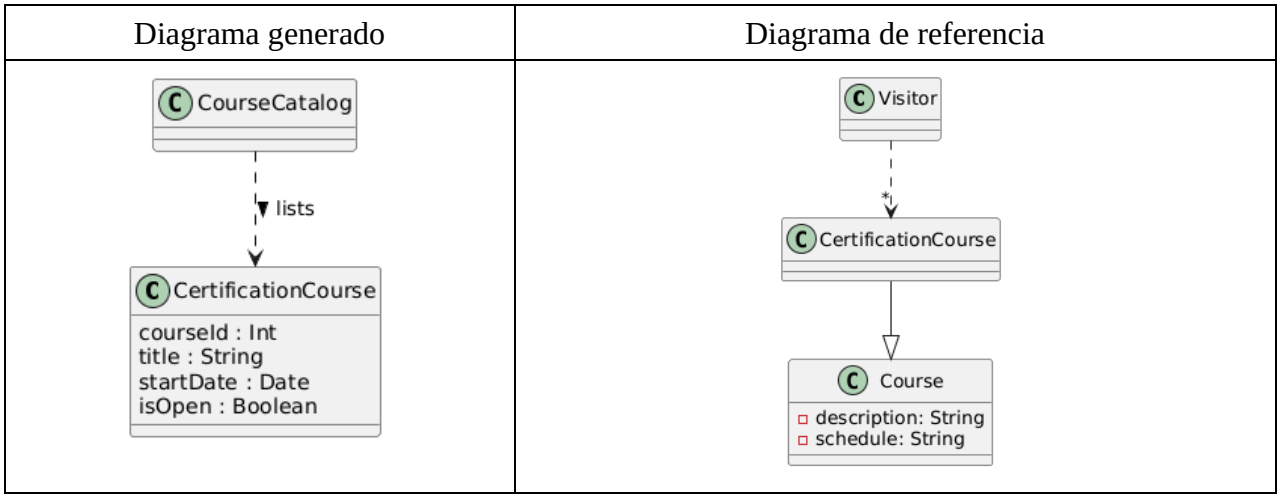


Figura 59 Comparación entre el diagrama de referencia y el diagrama generado para la HU-CL09.

Los ejemplos anteriores muestran algunas de las situaciones que influyeron en las valoraciones emitidas por los expertos durante la evaluación cualitativa. En particular, la omisión de elementos relevantes del dominio, la incorporación de conceptos no contemplados en el modelo de referencia y las diferencias en la interpretación de ciertos requisitos pueden afectar criterios como la completitud, la consistencia semántica y la adecuación al usuario. No obstante, estos casos representaron situaciones puntuales dentro del conjunto de diagramas evaluados y permiten identificar áreas de oportunidad para futuras mejoras de la herramienta.

5.2.7. Resultados obtenidos de los diagramas de actividades

En esta sección se presentan los resultados correspondientes a la evaluación cualitativa aplicada a los diagramas de actividades generados por la herramienta durante el caso de estudio. Los diagramas elaborados por los expertos participantes mediante el uso de la herramienta pueden consultarse en:

https://github.com/LoreMartinez/Repositorio_Tesis_DiagramasUML_Archivos/tree/main/Archivos_evaluacion_Caso_Estudio/Diagramas_actividades_generados_expertos_E7-E12

La valoración fue realizada por seis expertos con experiencia profesional en el desarrollo de software y en el uso de UML, quienes analizaron el conjunto de diagramas de actividades generados por la herramienta, a partir de los doce criterios mostrados en la Tabla 20, definidos en el instrumento de evaluación, con su correspondiente valor de ponderación. Los registros completos de las

evaluaciones realizadas por los seis expertos, correspondientes a cada diagrama de actividades y criterio, se encuentran disponibles en:

https://github.com/LoreMartinez/Repositorio_Tesis_DiagramasUML_Archivos/blob/main/Archivos_evaluacion_Caso_Estudio/Resultados_evaluacion_cualitativa_diagramas_actividades.xlsx

Con el fin de ejemplificar de manera detallada el proceso de generación implementado por la herramienta, el Apéndice B presenta un caso completo de generación de un diagrama de actividades, incluyendo la historia de usuario de entrada, el *prompt* empleado, la respuesta producida por el modelo de lenguaje y el diagrama UML resultante. A partir de estas valoraciones, expresadas en una escala de 0 a 4 previamente descrita (véase Tabla 21), se calculó el promedio global por evaluador, cuyos resultados se muestran en la Tabla 38.

Tabla 38. Promedio global por experto para los diagramas de actividades generados.

Experto	Promedio
E7	3.43
E8	2.80
E9	3.60
E10	3.98
E11	**3.99**
E12	2.63

Los promedios globales obtenidos por cada uno de los expertos evidenciaron diferencias relevantes en los patrones de valoración del conjunto de diagramas de actividades analizados. En particular, los expertos E11 y E10 alcanzaron los valores más altos, con promedios de 3.99 y 3.98 respectivamente, lo que reflejó una percepción global altamente favorable respecto a la calidad de los diagramas de actividades generados por la herramienta. Este comportamiento sugirió que, desde la perspectiva de ambos evaluadores, los modelos presentaron un grado elevado de claridad en la representación de los flujos de trabajo y de las decisiones del sistema. En contraste, el experto E12 presentó el promedio más bajo, con un valor de 2.63, seguido del experto E8 con 2.80, lo que indicó una postura más estricta en la aplicación de los criterios de valoración. Esta tendencia se asoció a una mayor exigencia en aspectos relacionados con la completitud de los procesos y la fidelidad semántica de los flujos representados respecto a las historias de usuario. Por su parte, los expertos E7 y E9 registraron valores intermedios de 3.43 y 3.60 respectivamente, lo que reflejó una percepción favorable pero más moderada de la calidad de los diagramas de actividades. En conjunto, este comportamiento contribuyó a equilibrar la tendencia general de la evaluación, reduciendo la influencia de valoraciones extremas.

Con el objetivo de analizar el desempeño de los diagramas de actividades desde una perspectiva más detallada, la Tabla 39 presenta los promedios obtenidos por cada criterio de evaluación, organizados de acuerdo con las dimensiones definidas en el instrumento de evaluación cualitativa.

Tabla 39. Promedio global por criterio para los diagramas de actividades generados.

Dimensión	Criterio	Media
Claridad y consistencia visual	Claridad semiótica	3.52
	Discriminabilidad perceptiva	3.58

Dimensión	Criterio	Media
	Expresividad visual	3.20
	Simplicidad	3.60
Comprensión cognitiva	Comprensibilidad	3.55
	Carga cognitiva	3.57
	Memorabilidad	3.52
Calidad estructural y semántica	Corrección sintáctica	3.45
	Consistencia semántica	3.27
	Completitud	3.12
Integración y utilidad	Integración cognitiva	3.42
	Adecuación al usuario	3.08

Los resultados obtenidos por criterio, a partir de las evaluaciones realizadas por los seis expertos, permitieron identificar con claridad las principales fortalezas y debilidades de los diagramas de actividades generados por la herramienta, considerando las cuatro dimensiones definidas en el instrumento de evaluación. En la dimensión de claridad y consistencia visual, la claridad semiótica alcanzó un valor de 3.52, lo que indicó que, desde la percepción de los expertos, los símbolos, flujos y nodos de decisión fueron representados de manera clara en la mayoría de los diagramas de actividades generados. La discriminabilidad perceptiva obtuvo un valor de 3.58, lo que evidenció una adecuada diferenciación entre los distintos elementos del diagrama. La expresividad visual presentó un valor de 3.20, lo que reflejó un uso aceptable de los recursos gráficos, aunque aún perfectible en escenarios con flujos más complejos. Finalmente, la parsimonia gráfica alcanzó un valor de 3.60, indicando un equilibrio favorable entre simplicidad visual y cantidad de información presentada.

En la dimensión de comprensión cognitiva, la comprensibilidad obtuvo un valor de 3.55, lo que mostró que los flujos principales pudieron ser comprendidos sin dificultades relevantes por los expertos. La carga cognitiva presentó un valor de 3.57, lo que indicó que el esfuerzo mental requerido para interpretar los diagramas fue bajo en términos generales. La memorabilidad alcanzó un valor de 3.52, lo que sugirió que los diagramas resultaron relativamente fáciles de recordar tras su análisis, favoreciendo su utilidad como apoyo para la comprensión de los procesos.

En relación con la calidad estructural y semántica, la corrección sintáctica alcanzó un valor de 3.45, lo que confirmó que la notación UML fue empleada de manera adecuada en la mayoría de los diagramas de actividades. La consistencia semántica obtuvo un valor de 3.27, lo que indicó una correspondencia generalmente correcta entre las actividades representadas y el dominio de las historias de usuario. No obstante, la completitud presentó un valor de 3.12, lo que evidenció la ausencia de algunos escenarios alternativos, excepciones o detalles operativos en parte de los modelos generados.

Finalmente, en la dimensión de integración y utilidad, la integración cognitiva alcanzó un valor de 3.42, lo que mostró que los diagramas permitieron una comprensión global adecuada del comportamiento de los procesos. La adecuación al usuario obtuvo un valor de 3.08, lo que indicó que los diagramas se ajustaron de forma aceptable a las necesidades del usuario final, aunque persistieron oportunidades de mejora en función del perfil del analista o desarrollador que los utilizó. En conjunto, las valoraciones realizadas por los expertos evidenciaron que los diagramas de actividades generados por la herramienta presentaron una alta calidad global, especialmente en términos de claridad visual, comprensibilidad y baja carga cognitiva. Asimismo, se observó una consistencia adecuada en la

estructura de los flujos y en la aplicación de la notación UML. No obstante, también se identificaron áreas de mejora en la completitud de los procesos y en la adecuación al usuario, particularmente en la incorporación de escenarios alternativos y excepciones, lo que puso de manifiesto la necesidad de realizar ajustes manuales antes de su uso como documentación técnica definitiva.

5.2.7.1. Observaciones cualitativas de los expertos sobre los diagramas de actividades generados

Las observaciones emitidas por los expertos coincidieron en señalar que la herramienta demostró una alta solidez lógica en la generación de los diagramas de actividades. Como parte de la evaluación, se les planteó la misma pregunta que en el caso de los diagramas de clases.

Las respuestas obtenidas de los expertos arrojaron un promedio global de **3.33** en una escala de 0 a 4, lo que refleja una valoración ampliamente positiva respecto a la utilidad de los diagramas para apoyar la interpretación inicial de los requisitos. Este resultado cuantitativo se alineó con los comentarios emitidos por los evaluadores, quienes destacaron que los flujos generados fueron correctos, coherentes y fáciles de seguir. En la mayoría de los casos, las secuencias permitieron comprender con claridad el comportamiento general del sistema y las transiciones entre actividades, favoreciendo una lectura rápida e intuitiva de las historias de usuario.

Desde el punto de vista de la claridad visual y sintáctica, los expertos indicaron que, aunque algunos elementos de la sintaxis original de los diagramas de actividades UML variaron como la forma de representar al usuario y al sistema, la notación empleada fue suficientemente clara para permitir la comprensión inmediata de cada actividad. En términos generales, la mayoría de los diagramas fueron considerados concretos, simples y fáciles de entender, manteniendo una adecuada correspondencia con la sintaxis UML estándar. Varios expertos señalaron que los diagramas generados facilitaron la comprensión general de las historias de usuario, al presentar flujos claros, sin saturación de información y con una estructura bien definida. Asimismo, se reconoció que la herramienta contribuyó a reducir ambigüedades y a estandarizar la comunicación. No obstante, también se indicó que los diagramas no capturaron toda la complejidad real de los procesos, particularmente en lo relativo al manejo detallado de errores y escenarios alternativos, lo que limitó su uso como documentación definitiva sin una revisión posterior.

Desde una perspectiva de utilidad en entornos de desarrollo ágil, los expertos coincidieron en que la herramienta resultó especialmente valiosa como apoyo para las etapas iniciales de análisis y diseño, al proporcionar un borrador visual que tradujo de manera eficaz el lenguaje de las historias de usuario a una representación formal en UML. Este enfoque permitió acelerar el proceso de diseño ágil, facilitar la discusión técnica y servir como una base sólida para el posterior refinamiento de los modelos. Asimismo, se señaló que la herramienta sería de gran utilidad para diseñadores, desarrolladores y testers dentro de equipos Scrum, al reducir el tiempo requerido para la elaboración inicial de diagramas. Los expertos destacaron que la generación automática de los diagramas de actividades permitió modelar de manera adecuada la interacción Actor-Sistema, detallando de forma clara quién ejecuta cada acción en el flujo del proceso. Se reconoció como un valor añadido la posibilidad de incorporar puntos de decisión y lógica de seguridad, ya que esto permitió formalizar reglas de negocio que, en muchos casos, se encontraban implícitas en las historias de usuario.

Finalmente, los evaluadores coincidieron en que la herramienta permitió detectar de manera temprana lagunas en la lógica de negocio y necesidades de arquitectura, como la posible inclusión de servicios intermedios, incluso antes de iniciar la etapa de codificación. Este aspecto fue considerado especialmente relevante, ya que contribuyó a reducir la deuda técnica potencial y a fortalecer la calidad del diseño desde las fases iniciales del desarrollo. En conjunto, las observaciones realizadas por los expertos permitieron concluir que la herramienta presentó un desempeño altamente favorable en la generación de diagramas de actividades como apoyo inicial al análisis de procesos, destacándose por

su claridad, coherencia funcional y utilidad en contextos ágiles. No obstante, también se enfatizó que los diagramas generados requirieron validación y refinamiento humano para su uso como documentación formal, especialmente cuando se demandó un mayor nivel de precisión, control de errores y detalle operativo.

5.2.7.2. Análisis de errores identificados en los diagramas de actividades generados

Las observaciones realizadas por los expertos también permitieron identificar algunas limitaciones en los diagramas de actividades generados durante el caso de estudio. Con el propósito de complementar los resultados de la evaluación cualitativa, se analizaron algunos ejemplos representativos en los que se observaron diferencias relevantes respecto a los diagramas de referencia. Este análisis permitió identificar principalmente errores asociados con la omisión de actividades y la interpretación parcial de los flujos descritos en las historias de usuario.

Ejemplo 1. Omisión de actividades en la HU-AC01

La Figura 5.4 muestra la comparación entre el diagrama de referencia y el diagrama generado para la historia de usuario HU-AC01. El diagrama de referencia describe un flujo completo para el registro de una solicitud, incluyendo la captura de información, la carga de evidencias, la vista previa de la solicitud, la validación de los datos ingresados y la gestión de posibles errores. Sin embargo, el diagrama generado únicamente representa tres actividades generales: acceder al formulario, completar la solicitud y enviarla. Como consecuencia, se omitieron actividades relevantes relacionadas con la validación de la información, la carga de archivos, la visualización previa de la solicitud y la notificación del resultado del proceso. Este caso evidencia una simplificación excesiva del flujo de trabajo, lo que afecta la completitud del diagrama generado y reduce el nivel de detalle representado respecto al modelo de referencia.

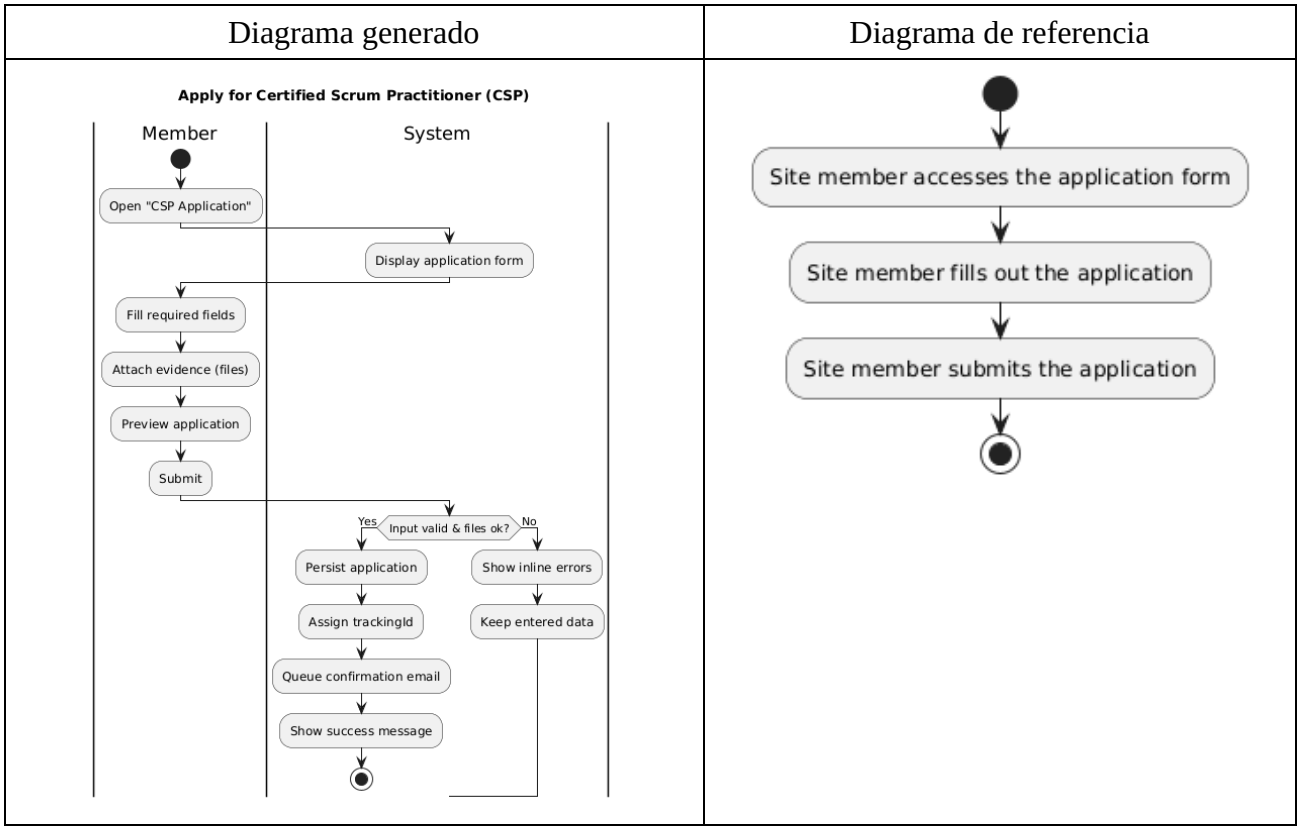


Figura 60 Comparación entre el diagrama de referencia y el diagrama generado para la HU-AC01.

Ejemplo 2. Error semántico en la HU-AC04

La Figura 5.5 muestra la comparación entre el diagrama de referencia y el diagrama generado para la historia de usuario HU-AC04. En el modelo de referencia, el usuario accede a la página principal, consulta los titulares disponibles, selecciona una noticia específica y posteriormente procede a leer su contenido. Por el contrario, el diagrama generado elimina la actividad de selección de la noticia y representa únicamente el acceso a la página principal, la visualización de noticias por parte del sistema y la lectura de las mismas. Aunque el flujo generado mantiene coherencia lógica, modifica parcialmente la interacción descrita en el requisito original. Este comportamiento constituye un error semántico, ya que la secuencia de actividades obtenida no refleja completamente el proceso de navegación previsto en el modelo de referencia. La omisión de la actividad de selección altera la representación de la interacción principal del usuario con el sistema.

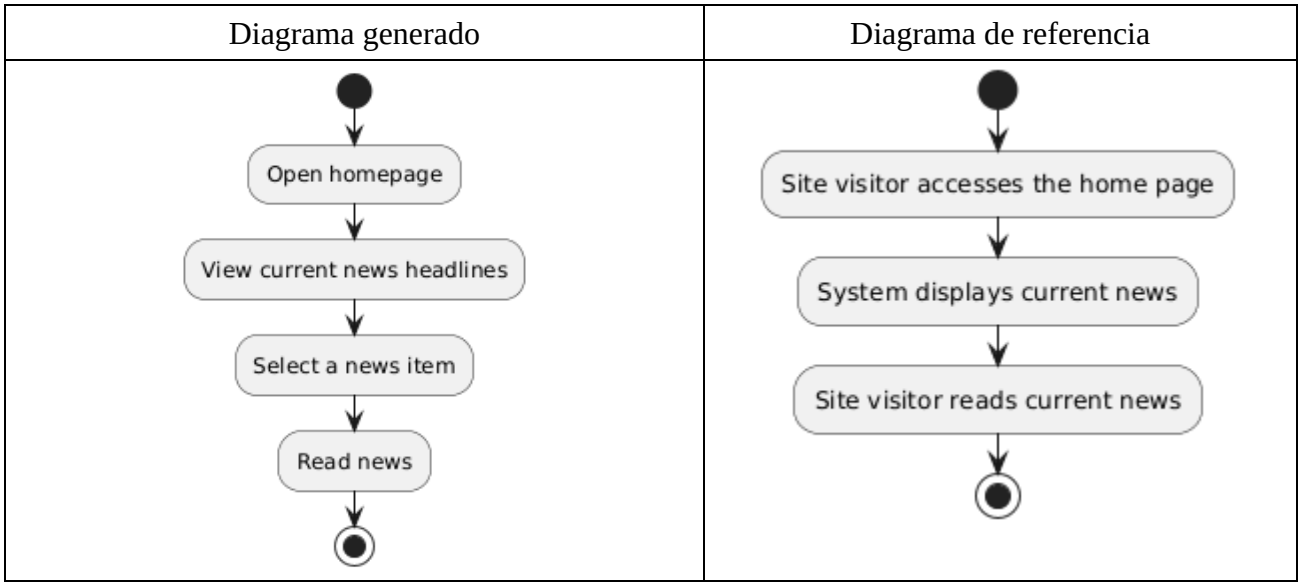


Figura 61 Comparación entre el diagrama de referencia y el diagrama generado para la HU-AC04.

Los ejemplos anteriores muestran algunas de las situaciones que influyeron en las valoraciones emitidas por los expertos durante la evaluación cualitativa. En particular, la omisión de actividades relevantes y las diferencias en la interpretación de determinados flujos pueden afectar criterios relacionados con la completitud, la consistencia semántica y la adecuación del diagrama al requisito descrito. No obstante, estos casos representaron situaciones específicas dentro del conjunto de diagramas evaluados y permiten identificar oportunidades de mejora para futuras versiones de la herramienta.

5.2.8. Conclusiones del caso de estudio

Considerando las valoraciones y comentarios de los expertos, se concluye que el caso de estudio permitió llevar a cabo una evaluación cualitativa de la utilidad de los diagramas UML generados automáticamente a partir de historias de usuario. En términos generales, los resultados evidencian que la herramienta es capaz de producir diagramas visualmente claros y comprensibles, lo que la posiciona como un recurso adecuado para apoyar las etapas iniciales del análisis y diseño de *software*. Durante la evaluación se identificaron dificultades asociadas a interpretaciones literales de los requisitos, las cuales derivaron en omisiones o en la inclusión de elementos innecesarios. No obstante, en los diagramas de clases se destacó la corrección sintáctica y la claridad estructural de los elementos representados. Por su parte, los diagramas de actividades obtuvieron valoraciones más favorables,

principalmente debido a la claridad del flujo y a la coherencia lógica de las acciones, aunque persistieron limitaciones en la representación de situaciones excepcionales y escenarios alternativos. De manera general, los expertos coincidieron en que la herramienta resulta especialmente útil como apoyo al prototipado inicial y a la discusión técnica. Sin embargo, señalaron que los diagramas generados no deben considerarse artefactos finales, dado que requieren revisión y validación por parte de un analista y/o arquitecto de *software*. Por lo tanto, el caso de estudio confirma el potencial de la herramienta como punto de partida en el diseño de sistemas, así como la necesidad de continuar mejorando la precisión y la completitud de las representaciones del dominio.

6. Conclusiones y trabajo futuro

En el desarrollo de *software* ágil, comprender con claridad las necesidades del usuario continúa siendo un desafío. Aunque las historias de usuario ofrecen una manera accesible y colaborativa de expresar requisitos, su naturaleza narrativa puede dar lugar a distintas interpretaciones entre los roles que participan en el proyecto de *software*. Este trabajo de investigación surgió precisamente de esa necesidad: encontrar una forma de apoyar esa comprensión inicial y ayudar a los equipos a tener una visión más clara desde las etapas iniciales.

En la SE, la comprensión de los requisitos se construye iterativamente y diálogo, lo que en entornos ágiles resulta especialmente crítico. Esta tesis aborda la brecha entre el lenguaje natural de las historias de usuario y la necesidad del uso de modelos estructurados compartidos por el equipo. Para ello, se propuso un conjunto de prácticas apoyadas por una herramienta para la generación automática de diagramas UML, con el objetivo de reducir ese vacío interpretativo. Los resultados mostraron que los modelos visuales pueden integrarse como un recurso ligero que aporta claridad inicial y favorece conversaciones más productivas.

Un hallazgo relevante fue que la utilidad de los diagramas no depende solo de su exactitud técnica, sino de su capacidad para generar reflexión en el equipo. Incluso con imperfecciones, ayudan a detectar omisiones y aclarar aspectos no definidos, funcionando como apoyos para el análisis y el diálogo en entornos ágiles.

En relación con la hipótesis planteada que el uso de estas prácticas apoyadas por una herramienta facilitaría la comprensión de las historias de usuario en metodologías ágiles, los resultados del caso de estudio muestran evidencia favorable. En los diagramas de clases, el promedio global fue de 3.01 y la valoración sobre su aporte a la comprensión alcanzó 2.83, lo que refleja una percepción moderadamente positiva. En los diagramas de actividades, los resultados fueron más altos: el promedio global fue de 3.41 y la valoración de utilidad llegó a 3.33. Estos valores indican que la herramienta contribuye de manera clara a mejorar la interpretación inicial de los requisitos, especialmente cuando se modela el comportamiento del sistema.

De acuerdo con los resultados mencionados, la comparación entre ambos tipos de diagramas permitió comprender mejor su alcance. Los diagramas de actividades mostraron mayor facilidad de interpretación al alinearse con el flujo narrativo de la historia de usuario, mientras que los diagramas de clases evidenciaron mayores dificultades al traducir el lenguaje natural en estructuras conceptuales más abstractas. Esto confirma que aún existen retos en la interpretación semántica profunda, particularmente en el modelado estructural.

Más allá de los valores cuantitativos, las valoraciones de los expertos mostraron que los diagramas generados facilitaron la comprensión inicial, ayudaron a reducir ambigüedades y sirvieron

como punto de partida para el análisis. Aunque señalaron la necesidad de revisión crítica, coincidieron en que los modelos aportan claridad y favorecen discusiones más informadas.

El cumplimiento de la hipótesis no se reflejó únicamente en los valores obtenidos, sino también en los comentarios de los expertos. Ellos coincidieron en que los diagramas generados facilitaron la comprensión inicial de las historias de usuario, ayudaron a reducir ambigüedades y permitieron entender con mayor claridad lo que la historia buscaba comunicar. Esto confirma que la herramienta realmente aporta valor: convierte una historia de usuario escrita en lenguaje natural en una representación visual más fácil de interpretar y de analizar. Esta capacidad de organizar la información de manera más clara ha sido una de las razones por las que los diagramas UML han sido tan útiles en la SE, y el hecho de generarlos de manera automática abre nuevas oportunidades para apoyar el trabajo, la comunicación y la colaboración dentro de los equipos ágiles.

El estudio también evidenció que la automatización puede integrarse de manera efectiva en entornos ágiles cuando se utiliza como complemento y no como sustituto del juicio experto. Lejos de introducir burocracia, los diagramas generados contribuyeron a mejorar la comunicación y la alineación del equipo desde las primeras etapas del *Sprint*.

Las evaluaciones realizadas por los expertos evidenciaron los límites actuales de la técnica. Si bien los diagramas de actividades obtuvieron resultados positivos, los diagramas de clases mostraron que la traducción del lenguaje natural a estructuras conceptuales representa un desafío que requiere mayor refinamiento. La herramienta avanzó en esa dirección, pero aún necesita mejorar su capacidad para distinguir qué elementos son realmente relevantes dentro del dominio y representar relaciones con mayor precisión semántica. Más que debilidades, estas observaciones señalan áreas de mejora y oportunidades para continuar desarrollando enfoques más sofisticados.

El diseño del conjunto de prácticas, el desarrollo de la herramienta y el análisis cualitativo permitieron construir una perspectiva sólida sobre cómo los diagramas UML generados automáticamente pueden integrarse en un entorno ágil como lo es Scrum. Lejos de representar un reemplazo de actividades tradicionales, emergen como aliados que permiten iniciar conversaciones más informadas, descubrir ambigüedades ocultas y alinear expectativas entre los miembros del equipo. Aunque su aporte pueda parecer modesto a simple vista, su impacto en la claridad y la comunicación técnica resulta favorable.

Este trabajo de investigación muestra el papel creciente de las tecnologías de NLP y los LLMs en la SE, especialmente en la interpretación de requisitos. La capacidad de transformar lenguaje natural en representaciones visuales útiles contribuye a mejorar la comprensión inicial del sistema, reducir ambigüedades y facilitar discusiones más informadas en equipos ágiles. Aunque aún en evolución, estas tecnologías ofrecen una base sólida para seguir integrándolas de manera responsable y efectiva en la práctica diaria de la SE.

Los resultados del caso de estudio muestran que la generación automática de diagramas UML facilita la comprensión inicial de las historias de usuario, enriquece el análisis y reduce la incertidumbre en etapas tempranas del desarrollo. La herramienta no sustituye al analista, sino que actúa como apoyo para aportar claridad en entornos ágiles. En este sentido, los hallazgos refuerzan la importancia de seguir desarrollando mecanismos que mejoren la comprensión del sistema antes de su construcción, fortaleciendo la comunicación y la toma de decisiones en la SE.

En este sentido, el trabajo realizado permitió consolidar un conjunto de prácticas orientadas a favorecer la comprensión de historias de usuario mediante la generación automática de diagramas UML en entornos ágiles basados en Scrum. Para apoyar la implementación de dichas prácticas, se desarrolló una herramienta que integró técnicas de NLP, modelos LLM y sistemas RAG, permitiendo transformar historias de usuario escritas en lenguaje natural en representaciones visuales

estructuradas. Asimismo, la evaluación cuantitativa y cualitativa realizada sobre los diagramas generados aportó evidencia sobre el potencial de este tipo de enfoques como apoyo para las actividades de análisis y diseño de *software*, particularmente en escenarios donde la claridad y la comunicación de los requisitos representan un desafío constante. Más allá de la automatización de diagramas, los hallazgos obtenidos mostraron que las representaciones visuales generadas pueden contribuir a facilitar la interpretación inicial de los requisitos, promover discusiones más informadas entre los miembros del equipo y apoyar la identificación temprana de ambigüedades en las historias de usuario. De esta manera, los resultados obtenidos permitieron evidenciar que el conjunto de prácticas propuesto representa una alternativa viable para apoyar la comprensión de requisitos en entornos ágiles, fortaleciendo la comunicación, el análisis y la construcción compartida del entendimiento del sistema durante las etapas iniciales del desarrollo de *software*.

En términos de trabajo futuro, los resultados obtenidos en el caso de estudio permitieron identificar líneas de mejora asociadas principalmente con los criterios que presentaron valoraciones relativamente menores dentro de la evaluación cualitativa. En este sentido, aunque la consistencia semántica alcanzó un nivel adecuado con un valor de 3.02, la completitud con 2.77 y la adecuación al usuario con 2.55 evidenciaron un desempeño moderado, lo que sugiere la necesidad de fortalecer la cobertura de los elementos del dominio y mejorar la adaptación de los diagramas a las necesidades del usuario final.

Una primera línea de trabajo consiste en incorporar mecanismos de retroalimentación directa dentro de la herramienta, permitiendo que los equipos validen, ajusten o señalen omisiones en los diagramas generados. Esta interacción podría utilizarse para refinar el contexto recuperado o ajustar el proceso generativo, avanzando hacia un sistema más interactivo y alineado con la realidad del proyecto.

Asimismo, se propone fortalecer el sistema RAG mediante un enriquecimiento contextual más robusto, incorporando reglas de negocio explícitas, definiciones formales del dominio o ejemplos estructurados adicionales que permitan mejorar la coherencia conceptual y reducir interpretaciones literales del lenguaje natural.

También resulta pertinente integrar mecanismos de validación automática que permitan abordar de manera más sistemática las limitaciones detectadas. Si bien la corrección sintáctica alcanzó valores máximos (4.00), la evaluación cualitativa reveló el riesgo de una “falsa sensación de corrección”, donde un diagrama formalmente válido puede no reflejar adecuadamente la lógica de negocio. Integrar validaciones orientadas no solo a la sintaxis UML, sino también a la detección de omisiones estructurales o relaciones inconsistentes, permitiría reducir este riesgo antes de la revisión humana.

Por otra parte, dado que los diagramas de actividades obtuvieron valoraciones significativamente más altas en consistencia semántica y completitud, futuras investigaciones podrían analizar con mayor profundidad por qué este tipo de modelo se adapta mejor a la interpretación automática del lenguaje natural. Este análisis comparativo podría orientar estrategias específicas para fortalecer los diagramas de clases, aprovechando patrones exitosos observados en la generación de modelos de comportamiento.

Otra línea relevante consiste en evaluar la herramienta con versiones más recientes de LLMs, considerando la rápida evolución del estado del arte. Dado que el desempeño mostró variaciones significativas entre modelos, especialmente en la representación conceptual del dominio, la integración de modelos más actuales, con mejoras en comprensión contextual y razonamiento estructurado, podría contribuir a mitigar las limitaciones identificadas. La arquitectura modular propuesta facilita esta actualización y permitiría replicar el experimento para analizar el impacto de los avances tecnológicos en los resultados.

Finalmente, la ampliación hacia otros tipos de diagramas UML representa una oportunidad de expansión natural del sistema. No obstante, esta evolución debería incorporar desde el inicio los aprendizajes derivados del caso de estudio, integrando mecanismos de validación humana y control semántico que eviten reproducir las debilidades observadas.

Estas líneas de trabajo futuro no solo amplían el alcance técnico de la herramienta, sino que responden directamente a las áreas de mejora identificadas empíricamente. La evolución del sistema debe mantener un equilibrio claro: aprovechar la automatización como acelerador del análisis inicial, sin sustituir el juicio experto necesario para asegurar la validez conceptual y la coherencia del diseño en contextos ágiles.

6.1. Productos científicos derivados

Como resultado de la investigación desarrollada en esta tesis, se generaron productos científicos orientados a la difusión de los hallazgos obtenidos.

En primer lugar, se publicó el artículo científico titulado "Comparación del desempeño de LLMs en la generación de diagramas de clases UML mediante un sistema RAG", en la Revista de Investigación en Tecnologías de la Información, Vol. 13, Núm. 31, pp. 4–17. Este trabajo presentó una evaluación comparativa del desempeño de diferentes LLMs en la generación automática de diagramas de clases UML mediante la incorporación de un sistema RAG, constituyendo una parte fundamental de los resultados obtenidos durante esta investigación.

Asimismo, derivado de los resultados obtenidos en esta tesis, se elaboró el artículo científico titulado *"Automatic Generation of UML Diagrams from Agile User Stories Using Large Language Models and Retrieval-Augmented Generation: A Practice Framework and Empirical Evaluation"*, el cual fue sometido para su evaluación en el número especial *Rethinking Requirements Engineering in the Age of Large Language Models* de la revista *Requirements Engineering*. Este trabajo presenta el conjunto de prácticas propuesto para apoyar la generación automática de diagramas UML a partir de historias de usuario en Scrum, así como la integración de LLMs y técnicas de RAG. Además, reporta los resultados obtenidos mediante las evaluaciones cuantitativas y cualitativas realizadas durante la investigación, proporcionando evidencia empírica sobre la viabilidad y utilidad de la propuesta. Al momento de la redacción de este documento, el artículo se encuentra en proceso de revisión.

La publicación y sometimiento de estos artículos científicos evidencian la contribución académica de la investigación realizada y favorecen la difusión del conocimiento generado en torno al uso de LLMs para apoyar actividades de análisis y diseño de *software*.

7. Referencias bibliográficas

Abbas, M., Rioboo, R., Ben-Yelles, C.-B., & Snook, C. F. (2021). Formal modeling and verification of UML Activity Diagrams (UAD) with FoCaLiZe. *Journal of Systems Architecture*, *114*, 101911. https://doi.org/10.1016/j.sysarc.2020.101911

Abrahamsson, P., Salo, O., & Ronkainen, J. (2002). *Agile Software Development Methods: Review and Analysis*.

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2017). *Agile Software Development Methods: Review and Analysis* (arXiv:1709.08439). arXiv. http://arxiv.org/abs/1709.08439

Afiansyah, T., Sidik, A., Hakim, Z., & Sofia, D. (2023). Applying the SCRUM Framework in Managing the Development of Candidate Profiling Project (A Case Study of Bagidata). *JURNAL SISFOTEK GLOBAL*, *13*(1), 35. https://doi.org/10.38101/sisfotek.v13i1.3503

Ahmed, I., & Islam, R. (2024). *Gemini-the most powerful LLM: Myth or Truth*. https://doi.org/10.36227/techrxiv.171177477.70151414/v1

Alberto, C., Beltrán, E., & Scrum, C. A. (2020). *SCRUM, Un enfoque práctico de metodología ágil para la ingeniería de software*. *8*(2).

Alsaqqa, S., Sawalha, S., & Abdel-Nabi, H. (2020). Agile *Software* Development: Methodologies and Trends. *International Journal of Interactive Mobile Technologies (iJIM)*, *14*(11), Article 11. https://doi.org/10.3991/ijim.v14i11.13269

Altameem, E. (2015). Impact of Agile Methodology on *Software* Development. *Computer and Information Science*, *8*(2), p9. https://doi.org/10.5539/cis.v8n2p9

Amna, A. R., & Poels, G. (2022a). Ambiguity in user stories: A systematic literature review. *Information and Software Technology*, *145*, 106824. https://doi.org/10.1016/j.infsof.2022.106824

Amna, A. R., & Poels, G. (2022b). Systematic Literature Mapping of User Story Research. *IEEE Access*, *10*, 51723–51746. https://doi.org/10.1109/ACCESS.2022.3173745

Arefeen, M. A., Debnath, B., & Chakradhar, S. (2024). LeanContext: Cost-efficient domain-specific question answering using LLMs. *Natural Language Processing Journal*, *7*, 100065. https://doi.org/10.1016/j.nlp.2024.100065

Ayodele, T. O. (2010). Machine Learning Overview. *New Advances in Machine Learning*.

Ayyadevara, V. K. (2018). *Pro Machine Learning Algorithms*. Apress. https://doi.org/10.1007/978-1-4842-3564-5

Azanzi, J., Tapamo, H., & Camara, G. (2023). Combining Scrum and Model Driven Architecture for the development of an epidemiological surveillance *software*. *Revue Africaine de Recherche en Informatique et Mathématiques Appliquées, Volume 39-2023*. https://doi.org/10.46298/arima.9873

Badillo, S., Banfai, B., Birzele, F., Davydov, I. I., Hutchinson, L., Kam-Thong, T., Siebourg-Polster, J., Steiert, B., & Zhang, J. D. (2020). An Introduction to Machine Learning. *Clinical Pharmacology & Therapeutics*, *107*(4), 871–885. https://doi.org/10.1002/cpt.1796

Badreddin, O., Rahad, K., Forward, A., & Lethbridge, T. (2021). The Evolution of *Software* Design Practices Over a Decade: A Long Term Study of Practitioners. *The Journal of Object Technology*, *20*(2), 1:1. https://doi.org/10.5381/jot.2021.20.2.a1

Behutiye, W., Seppänen, P., Rodríguez, P., & Oivo, M. (2020). Documentation of Quality Requirements in Agile *Software* Development. *Proceedings of the Evaluation and Assessment in Software Engineering*, 250–259. https://doi.org/10.1145/3383219.3383245

Bergström, G., Hujainah, F., Ho-Quang, T., Jolak, R., Rukmono, S. A., Nurwidyantoro, A., & Chaudron, M. R. V. (2022). Evaluating the layout quality of UML class diagrams using machine learning. *Journal of Systems and Software*, *192*, 111413. https://doi.org/10.1016/j.jss.2022.111413

Bhatt, B., & Nandu, M. (2021). An overview of structural uml diagrams. *International Research Journal of Engineering and Technology*, *8*(8), 1577-1583.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., … Amodei, D. (2020). Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, *33*, 1877–1901. https://proceedings.neurips.cc/paper_files/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

Calo, K. M., Estevez, E., & Fillottrani, P. (2010). *Evaluación de Metodologías Ágiles para Desarrollo de Software*.

Cámara, J., Troya, J., Burgueño, L., & Vallecillo, A. (2023). On the assessment of generative AI in modeling tasks: An experience report with ChatGPT and UML. *Software and Systems Modeling*, *22*(3), 781–793. https://doi.org/10.1007/s10270-023-01105-5

Carlà, M. M., Gambini, G., Baldascino, A., Boselli, F., Giannuzzi, F., Margollicci, F., & Rizzo, S. (2024). Large language models as assistance for glaucoma surgical cases: A ChatGPT vs. Google Gemini comparison. *Graefe's Archive for Clinical and Experimental Ophthalmology*. https://doi.org/10.1007/s00417-024-06470-5

Chen, S., Li, Y., Lu, S., Van, H., Aerts, H. J., Savova, G. K., & Bitterman, D. S. (2024). Evaluating the ChatGPT family of models for biomedical reasoning and classification. *Journal of the American Medical Informatics Association*, *31*(4), 940-948.

Chen, F., Zhang, L., Lian, X., & Niu, N. (2022). Automatically recognizing the semantic elements from UML class diagram images. *Journal of Systems and Software*, *193*, 111431. https://doi.org/10.1016/j.jss.2022.111431

Chen, P.-S., Chen, G. Y.-H., Lien, S.-F., & Huang, W.-T. (2019). Using Scrum and unified modelling language to analyze and design an automatic course scheduling system. *Journal of the Chinese Institute of Engineers*, *42*(6), 534–543. https://doi.org/10.1080/02533839.2019.1613930

Chowdhury, G. G. (2020). *Natural Language Processing*.

Clason, D. L., & Dormody, T. J. (1994). Analyzing Data Measured By Individual Likert-Type Items. *Journal of Agricultural Education*, *35*(4), 31–35. https://doi.org/10.5032/jae.1994.04031

Cohn, M. (2004). *User stories applied: For agile software development*. Addison-Wesley Professional.

Cruz, L. M. H., Turriza, J. L. L., Huh, Y. P., Turriza, J. M. L., Salgado, F. Á. Á., & Álvarez, D. C. M. (2023). Los roles del marco de trabajo Scrum: un análisis de competencias y habilidades. *Revista Ibérica de Sistemas e Tecnologias de Informação*, (E58), 450-458.

Dada, O. A., & Sanusi, I. T. (2022). The adoption of *Software* Engineering practices in a Scrum environment. *African Journal of Science, Technology, Innovation and Development*, *14*(6), 1429–1446. https://doi.org/10.1080/20421338.2021.1955431

Darrin, M. A. G., & Devereux, W. S. (2017). The Agile Manifesto, design thinking and systems engineering. *2017 Annual IEEE International Systems Conference (SysCon)*, 1–5. https://doi.org/10.1109/SYSCON.2017.7934765

de Pinho, D. R. (2020). *Effective Communication in Agile Teams: A Pattern Language*. https://repositorio-aberto.up.pt/bitstream/10216/128599/2/412429.pdf

Dias, R. P., Vidanapathirana, C. S. L., Weerasinghe, R., Manupiya, A., Bandara, R. M. S. J., & Ranasinghe, Y. P. H. W. (2023). *Automated use case diagram generator using NLP and ML* (arXiv:2306.06962). arXiv. http://arxiv.org/abs/2306.06962

Diebold, P., & Dahlem, M. (2014). Agile practices in practice: A mapping study. *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, 1–10. https://doi.org/10.1145/2601248.2601254

Dimitrijević, S., Jovanović, J., & Devedžić, V. (2015). A comparative study of *software* tools for user story management. *Information and Software Technology*, *57*, 352–368.

Elallaoui, M., Nafil, K., & Touahni, R. (2015). Automatic generation of UML sequence diagrams from user stories in Scrum process. *2015 10th international conference on intelligent systems: theories and applications (SITA)*, 1–6. https://ieeexplore.ieee.org/abstract/document/7358415/

Elallaoui, M., Nafil, K., & Touahni, R. (2018). Automatic transformation of user stories into UML use case diagrams using NLP techniques. *Procedia computer science*, *130*, 42–49.

Elmansouri, R., Meghzili, S., & Chaoui, A. (2021). A UML 2.0 Activity Diagrams/CSP Integrated Approach for Modeling and Verification of *Software* Systems. *Computer Science*, *22*(2). https://doi.org/10.7494/csci.2021.22.2.3478

Estrada-Velasco, M. V., Núñez-Villacis, J. A., Saltos-Chávez, P. R., & Cunuhay-Cuchipe, W. C. (2021). *Revisión Sistemática de la Metodología Scrum para el Desarrollo de Software Systematic review of the SCRUM methodology for software development Revisão Sistemática da Metodologia Scrum para Desenvolvimento de Software*. *7*.

Falah, B., Karroumi, S., & Lahkim, O. (2020). *UTS: A new modeling methodology with UML and Test case generation, applied in conjunction with Scrum framework.*

Felderer, M., & Travassos, G. H. (2020). The Evolution of Empirical Methods in *Software* Engineering. En M. Felderer & G. H. Travassos (Eds.), *Contemporary Empirical Methods in Software Engineering* (pp. 1–24). Springer International Publishing. https://doi.org/10.1007/978-3-030-32489-6_1

Felizardo, K. R., et al. (2018). *User Stories Datasets for Agile Software Development Research*. Mendeley Data. https://doi.org/10.17632/r8gvz82g5f.2

Fernández-Sáez, A. M., Genero, M., Chaudron, M. R. V., Caivano, D., & Ramos, I. (2015). Are Forward Designed or Reverse-Engineered UML diagrams more helpful for code maintenance?: A family of experiments. *Information and Software Technology*, *57*, 644–663. https://doi.org/10.1016/j.infsof.2014.05.014

Gallud, J. A., & Fardoun, H. M. (2018). UML and Agile Methods: Looking for an Agreement: *Proceedings of the 13th International Conference on Software Technologies*, 780–785. https://doi.org/10.5220/0006940907800785

Ge, Y., Tan, J., Hua, W., Xu, S., Mei, K., Li, Z., Ji, J., & Zhang, Y. (2024). *OpenAGI: When LLM Meets Domain Experts*.

Gemini Team, Anil, R., Borgeaud, S., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., Silver, D., Johnson, M., Antonoglou, I., Schrittwieser, J., Glaese, A., Chen, J., Pitler, E., Lillicrap, T., Lazaridou, A., … Vinyals, O. (2024). *Gemini: A Family of Highly Capable Multimodal Models* (arXiv:2312.11805). arXiv. http://arxiv.org/abs/2312.11805

Ghimire, D., & Charters, S. (2022). The impact of Agile development practices on project outcomes. *Software*, *1*(3), 265–275.

Gilson, F., & Irwin, C. (2018). From user stories to use case scenarios towards a generative approach. *2018 25th Australasian Software Engineering Conference (ASWEC)*, 61–65. https://ieeexplore.ieee.org/abstract/document/8587287/

Gómez-Luna, E., Navas, D. F., Aponte-Mayor, G., & Betancourt-Buitrago, L. A. (2014). Literature review methodology for scientific and information management, through its structuring and systematization. *Dyna*, *81*(184), 158–163.

Grau, X. F., & Segura, M. I. S. (2008). *Desarrollo Orientado a Objetos con UML*.

Gupta, A., Poels, G., & Bera, P. (2022). Using Conceptual Models in Agile *Software* Development: A Possible Solution to Requirements Engineering Challenges in Agile Projects. *IEEE Access*, *10*, 119745–119766. https://doi.org/10.1109/ACCESS.2022.3221428

Gutiérrez, E. G., Guevara, M. M. M., & López, N. R. (2020). *METODOLOGÍAS ÁGILES PARA EL DESARROLLO DE PROYECTOS*.

Helmy, W., Kamel, A., & Hegazy, O. (2012). *Requirements Engineering Methodology in Agile Environment*. *9*(5).

Hema, V., Thota, S., Naresh Kumar, S., Padmaja, C., Rama Krishna, C. B., & Mahender, K. (2020). Scrum: An Effective *Software* Development Agile Tool. *IOP Conference Series: Materials Science and Engineering*, *981*(2), 022060. https://doi.org/10.1088/1757-899X/981/2/022060

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). *Design Science in Information Systems Research*.

Hron, M., & Obwegeser, N. (2022). Why and how is Scrum being adapted in practice: A systematic review. *Journal of Systems and Software*, *183*, 111110.

Jain, A., Patel, H., Nagalapatti, L., Gupta, N., Mehta, S., Guttula, S., Mujumdar, S., Afzal, S., Sharma Mittal, R., & Munigala, V. (2020). Overview and Importance of Data Quality for Machine Learning Tasks. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 3561–3562. https://doi.org/10.1145/3394486.3406477

Jain, P., Sharma, A., & Ahuja, L. (2018). The Impact of Agile *Software* Development Process on the Quality of *Software* Product. *2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, 812–815. https://doi.org/10.1109/ICRITO.2018.8748529

Jalali, S., & Wohlin, C. (2012). Global *software* engineering and agile practices: A systematic review. *Journal of Software: Evolution and Process*, *24*(6), 643–659. https://doi.org/10.1002/smr.561

Janiesch, C., Zschech, P., & Heinrich, K. (2021). Machine learning and deep learning. *Electronic Markets*, *31*(3), 685–695. https://doi.org/10.1007/s12525-021-00475-2

Javaid, M., Haleem, A., Singh, R. P., & Suman, R. (2022). Artificial Intelligence Applications for Industry 4.0: A Literature-Based Study. *Journal of Industrial Integration and Management*, *07*(01), 83–111. https://doi.org/10.1142/S2424862221300040

Joshi, A., Kale, S., Chandel, S., & Pal, D. (2015). Likert Scale: Explored and Explained. *British Journal of Applied Science & Technology*, *7*(4), 396–403. https://doi.org/10.9734/BJAST/2015/14975

Kahng, M., Tenney, I., Pushkarna, M., Liu, M. X., Wexler, J., Reif, E., Kallarackal, K., Chang, M., Terry, M., & Dixon, L. (2024). *LLM Comparator: Visual Analytics for Side-by-Side Evaluation of Large Language Models* (arXiv:2402.10524). arXiv. http://arxiv.org/abs/2402.10524

Kalyan, K. S. (2024). A survey of GPT-3 family large language models including ChatGPT and GPT-4. *Natural Language Processing Journal*, *6*, 100048. https://doi.org/10.1016/j.nlp.2023.100048

Kannan, V., Basit, M. A., Bajaj, P., Carrington, A. R., Donahue, I. B., Flahaven, E. L., Medford, R., Melaku, T., Moran, B. A., Saldana, L. E., Willett, D. L., Youngblood, J. E., & Toomay, S. M. (2019). User stories as lightweight requirements for agile clinical decision support development. *Journal of the American Medical Informatics Association*, *26*(11), 1344–1354. https://doi.org/10.1093/jamia/ocz123

Kasneci, E., Seßler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., Gasser, U., Groh, G., Günnemann, S., & Hüllermeier, E. (2023). ChatGPT for good? On opportunities and challenges of large language models for education. *Learning and individual differences*, *103*, 102274.

Kassab, M. (2015). *The Changing Landscape of Requirements Engineering Practices Over The Past Decade*. https://doi.org/10.1109/EmpiRE.2015.7431299

Kevian, D., Syed, U., Guo, X., Havens, A., Dullerud, G., Seiler, P., Qin, L., & Hu, B. (2024). *Capabilities of Large Language Models in Control Engineering: A Benchmark Study on GPT-4, Claude 3 Opus, and Gemini 1.0 Ultra* (arXiv:2404.03647). arXiv.

http://arxiv.org/abs/2404.03647

Kharmoum, N., Bouchti, K. E., Laaz, N., Rhalem, W., & Rhazali, Y. (2020). Transformations' Study Between Requirements Models and Business Process Models in MDA Approach. *Procedia Computer Science, 170*, 819–824. https://doi.org/10.1016/j.procs.2020.03.150

Kirchenbauer, J., Geiping, J., Wen, Y., Katz, J., Miers, I., & Goldstein, T. (2023). A watermark for large language models. *International Conference on Machine Learning*, 17061–17084. https://proceedings.mlr.press/v202/kirchenbauer23a.html

Koç, H., Erdoğan, A. M., Barjakly, Y., & Peker, S. (2021). UML Diagrams in *Software* Engineering Research: A Systematic Literature Review. *The 7th International Management Information Systems Conference*, 13. https://doi.org/10.3390/proceedings2021074013

Kochbati, T., Li, S., Gérard, S., & Mraidha, C. (2021). From User Stories to Models: A Machine Learning Empowered Automation: *Proceedings of the 9th International Conference on Model-Driven Engineering and Software Development*, 28–40. https://doi.org/10.5220/0010197800280040

Kulkarni, R., & Srinivasa, C. (2021). Novel approach to transform UML Sequence diagram to Activity diagram. *Journal of University of Shanghai for Science and Technology, 23*, 1247–1255. https://doi.org/10.51201/JUSST/21/07300

Kussunga, F., & Ribeiro, P. (2019). Proposal of a Visual Environment to Support Scrum. *Procedia Computer Science, 164*, 491–497. https://doi.org/10.1016/j.procs.2019.12.211

Kussunga, F., Ribeiro, P., & Santos, N. (2019). *Characterization of a Visual Environment to Support SCRUM ceremonies*.

Lee, G.-G., Latif, E., Shi, L., & Zhai, X. (2023). *Gemini Pro Defeated by GPT-4V: Evidence from Education* (arXiv:2401.08660). arXiv. http://arxiv.org/abs/2401.08660

Li, H., Leung, J., Wang, H., & Shen, Z. (2023). *Prompting LLMs to Solve Complex Tasks: A Review*.

Lima, L., Tavares, A., & Nogueira, S. C. (2019). *A framework for verifying deadlock and nondeterminism in UML activity diagrams based on CSP* (arXiv:1910.13638). arXiv. http://arxiv.org/abs/1910.13638

Lin, Z. (2024). How to write effective prompts for large language models. *Nature Human Behaviour*, 1–5.

Lina, J., & de Mello, R. M. (2015). *Guidelines for Conducting Surveys in Software Engineering v. 1.*

Lopes, A., Steinmacher, I., & Conte, T. (2019). UML Acceptance: Analyzing the Students' Perception of UML Diagrams. *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, 264–272. https://doi.org/10.1145/3350768.3352575

Lucassen, G., Dalpiaz, F., Werf, J. M. E. M. van der, & Brinkkemper, S. (2016). The Use and Effectiveness of User Stories in Practice. En M. Daneva & O. Pastor (Eds.), *Requirements Engineering: Foundation for Software Quality* (pp. 205–222). Springer International Publishing. https://doi.org/10.1007/978-3-319-30282-9_14

M. Maatuk, A., & A. Abdelnabi, E. (2021). Generating UML Use Case and Activity Diagrams Using NLP Techniques and Heuristics Rules. *International Conference on Data Science, E-Learning and Information Systems 2021*, 271–277. https://doi.org/10.1145/3460620.3460768

Madanayake, R., Dias, K., & Kodikara, N. D. (2017). Transforming Simplified Requirement in to a UML Use Case Diagram Using an Open Source Tool. *International Journal of Computer Science and Software Engineering*, *6*, 2409–4285.

Mahmood, W., Usmani, N., Ali, M., & Farooqui, S. (2020). Benefits to organizations after migrating to Scrum. *Proc. 29th Int. Bus. Inf. Manag. Assoc. Conf.-Educ. Excell. Innov. Manag. through Vis*, 38153828. https://www.researchgate.net/profile/Waqas-Mahmood-2/publication/317066055_Benefits_to_Organizations_after_Migrating_to_Scrum/links/5b83e63992851c1e1234fff2/Benefits-to-Organizations-after-Migrating-to-Scrum.pdf

Malinova, M., & Mendling, J. (2021). Cognitive Diagram Understanding and Task Performance in Systems Analysis and Design. *MIS Quarterly*, *45*(4), 2101–2158. https://doi.org/10.25300/MISQ/2021/15262

McCrae, J. P., Rademaker, A., Rudnicka, E., & Bond, F. (2020). English WordNet 2020: Improving and Extending a WordNet for English using an Open-Source Methodology. En T. Declerk, I. Gonzalez-Dios, & G. Rigau (Eds.), *Proceedings of the LREC 2020 Workshop on Multimodal Wordnets (MMW2020)* (pp. 14–19). The European Language Resources Association (ELRA). https://aclanthology.org/2020.mmw-1.3

Melouk, M., Rhazali, Y., & Youssef, H. (2020). An Approach for Transforming CIM to PIM up To PSM in MDA. *Procedia Computer Science*, *170*, 869–874. https://doi.org/10.1016/j.procs.2020.03.122

Menzinsky, A., López, G., Palacio, J., Sobrino, M. Á., Álvarez, R., & Rivas, V. (2018). Historias de usuario. *Ingeniería de requisitos ágil*. https://www.scrummanager.com/files/scrum_manager_historias_usuario.pdf

Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., & Gao, J. (2024). *Large Language Models: A Survey* (arXiv:2402.06196). arXiv. http://arxiv.org/abs/2402.06196

Molla, M. M. I., Ahmad, J., & Kadir, W. M. N. W. (2024). A comparison of transforming the user stories and functional requirements into UML use case diagram. *International Journal of Innovative Computing*, *14*(1), 29-36.

Moody, D. (2009). The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Transactions on Software Engineering*, *35*(6), 756–779. https://doi.org/10.1109/TSE.2009.67

Mornie, M. N., Jali, N., Junaini, S. N., Mit, E., Shiang, C. W., & Saee, S. (2023). Visualisation of User Stories in UML Models: A Systematic Literature Review. *Acta Informatica Pragensia*, *12*(2), 419–438. https://doi.org/10.18267/j.aip.212

Mornie, M. N., Jali, N., Shiang, C. W., Mit, E., Saee, S., Jali, S. K., & Greer, D. (2026). Visualisation of user stories to UML use case diagram. *Journal of Advanced Research in Applied Sciences and Engineering Technology*, *63*(3), 68-80.

Nasiri, S., Adadi, A., & Lahmer, M. (2023). Automatic generation of business process models from user stories. *International Journal of Electrical and Computer Engineering (IJECE)*, *13*(1), 809. https://doi.org/10.11591/ijece.v13i1.pp809-822

Nasiri, S., Rhazali, Y., Lahmer, M., & Chenfour, N. (2020). Towards a Generation of Class Diagram from User Stories in Agile Methods. *Procedia Computer Science*, *170*, 831–837. https://doi.org/10.1016/j.procs.2020.03.148

Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Akhtar, N., Barnes, N., & Mian, A. (2024). *A Comprehensive Overview of Large Language Models* (arXiv:2307.06435). arXiv. http://arxiv.org/abs/2307.06435

Nikiforova, O., Sejans, J., & Cernickins, A. (2011). Role of UML Class Diagram in Object-Oriented *Software* Development. *J. Riga Technical University*, *44*, 65–74. https://doi.org/10.2478/v10143-011-0023-4

N.Vithana, V. (2015). Scrum Requirements Engineering Practices and Challenges in Offshore *Software* Development. *International Journal of Computer Applications*, *116*(22), 43–49. https://doi.org/10.5120/20472-2649

Ohst, D., Welle, M., & Kelter, U. (2003). *Differences between Versions of UML Diagrams*.

Ojha, M., Gupta, S., & Sharma, R. (2025, March). Towards Class Diagram Generation from User Stories Using LLMs. In *2025 International Conference on Next Generation Information System Engineering (NGISE)* (Vol. 1, pp. 1-6). IEEE.

Oladipupo, T. (2010). Types of Machine Learning Algorithms. En Y. Zhang (Ed.), *New Advances in Machine Learning*. InTech. https://doi.org/10.5772/9385

Ozkaya, M. (2019). Are the UML modelling tools powerful enough for practitioners? A literature review. *IET Software*, *13*(5), 338–354. https://doi.org/10.1049/iet-sen.2018.5409

Páez, J. A., Cortes, J. A., Simanca, F. A., & Blanco, F. (2021). Aplicación de UML y SCRUM al desarrollo del *software* sobre control de acceso. *Información tecnológica*, *32*(5), 57–66. https://doi.org/10.4067/S0718-07642021000500057

Pereira, T. F., Morais, F., Salgado, C. E., Lima, A., Silva, A., Pereira, M., Oliveira, J., & Machado, R. J. (2023). The Adoption of 4Step-Rule-Set Method for Ontological Design: Application in a Real Industrial Project. *Procedia Computer Science*, *219*, 405–415. https://doi.org/10.1016/j.procs.2023.01.306

Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., & Still, J. (2008). The impact of agile practices on communication in *software* development. *Empirical Software Engineering*, *13*(3), 303–337. https://doi.org/10.1007/s10664-008-9065-9

Pokharel, P., & Vaidya, P. (2020). *A Study of User Story in Practice* (p. 5). https://doi.org/10.1109/ICDABI51230.2020.9325670

Pressman, R. S., & Maxim, B. R. (2020). *Software engineering: A practitioner's approach* (Ninth edition). McGraw-Hill Education.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020). *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*.

Raharjana, I. K., Siahaan, D., & Fatichah, C. (2021). User Stories and Natural Language Processing: A Systematic Literature Review. *IEEE Access*, *9*, 53811–53826. https://doi.org/10.1109/ACCESS.2021.3070606

Recker, J., & Green, P. F. (2019). How do Individuals Interpret Multiple Conceptual Models? A Theory of Combined Ontological Completeness and Overlap. *Journal of the Association for Information Systems*, 1210–1241. https://doi.org/10.17705/1jais.00565

Reggio, G., Leotta, M., & Ricca, F. (2014). Who Knows/Uses What of the UML: A Personal Opinion Survey. En J. Dingel, W. Schulte, I. Ramos, S. Abrahão, & E. Insfran (Eds.), *Model-Driven Engineering Languages and Systems* (pp. 149–165). Springer International Publishing. https://doi.org/10.1007/978-3-319-11653-2_10

Reggio, G., Leotta, M., Ricca, F., & Clerissi, D. (2013). *What are the used UML diagrams? A Preliminary Survey*.

Rodríguez, P., Mäntylä, M., Oivo, M., Lwakatare, L. E., Seppänen, P., & Kuvaja, P. (2019). Advances in using agile and lean processes for *software* development. En *Advances in computers* (Vol. 113, pp. 135–224). Elsevier. https://www.sciencedirect.com/science/article/pii/S0065245818300299

Rumbaugh, J., Jacobson, I., & Booch, G. (1999). *The unified modeling language reference manual*. Addison-Wesley.

Sampietro, M. (2016). The adoption and evolution of agile practices. *PM World Journal*, *5*(6), 1–16.

Sandsto R. & Reme-Ness C. (2021). Agile Practices and Impacts on Project Success. *Journal of Engineering, Project, and Production Management*.

Sassa, A. C., Almeida, I. A. D., Pereira, T. N. F., & Oliveira, M. S. D. (2023). Scrum: A Systematic Literature Review. *International Journal of Advanced Computer Science and Applications*, *14*(4). https://doi.org/10.14569/IJACSA.2023.0140420

Sayin, A., & Gierl, M. (2024). Using OpenAI GPT to Generate Reading Comprehension Items. *Educational Measurement: Issues and Practice*, *43*(1), 5–18. https://doi.org/10.1111/emip.12590

Scanniello, G., Gravino, C., Genero, M., Cruz-Lemus, J. A., Tortora, G., Risi, M., & Dodero, G. (2018). Do *software* models based on the UML aid in source-code comprehensibility? Aggregating evidence from 12 controlled experiments. *Empirical Software Engineering*, *23*(5), 2695–2733. https://doi.org/10.1007/s10664-017-9591-4

Schmidt, C. (2016). Agile *Software* Development. En C. Schmidt, *Agile Software Development Teams* (pp. 7–35). Springer International Publishing. https://doi.org/10.1007/978-3-319-26057-0_2

Schön, E.-M., Thomaschewski, J., & Escalona, M. J. (2017). Agile Requirements Engineering: A systematic literature review. *Computer Standards & Interfaces*, *49*, 79–91. https://doi.org/10.1016/j.csi.2016.08.011

Shafiq, M., & sman Waheed, U. (2018). Documentation in agile development a comparative analysis. *2018 IEEE 21st International Multi-Topic Conference (INMIC)*, 1–8. https://ieeexplore.ieee.org/abstract/document/8595625/

Shaikh, A., Hafeez, A., Wagan, A. A., Alrizq, M., Alghamdi, A., & Reshan, M. S. A. (2021). More Than Two Decades of Research on Verification of UML Class Models: A Systematic Literature Review. *IEEE Access*, *9*, 142461–142474. https://doi.org/10.1109/ACCESS.2021.3121222

Sharifani, K., & Amini, M. (2023). *Machine Learning and Deep Learning: A Review of Methods and*

Applications. *10*, 3897–3904.

Sidky, A., Arthur, J., & Bohner, S. (2007). A disciplined approach to adopting agile practices: The agile adoption framework. *Innovations in Systems and Software Engineering*, *3*(3), 203–216. https://doi.org/10.1007/s11334-007-0026-z

Singh, A. (2023). Book review: Jeff Sutherland. 2014. Scrum The Art of Doing Twice the Work in Half the Time. *Metamorphosis*, *22*(2), 193–194. https://doi.org/10.1177/09726225241227488

Sommerville, I. (2020). *Engineering software products: An introduction to modern software engineering* (First edition). Pearson.

Sonbol, R., Rebdawi, G., & Ghneim, N. (2022). The Use of NLP-Based Text Representation Techniques to Support Requirement Engineering Tasks: A Systematic Mapping Review. *IEEE Access*, *10*, 62811–62830. https://doi.org/10.1109/ACCESS.2022.3182372

Sonoda, Y., Kurokawa, R., Nakamura, Y., Kanzawa, J., Kurokawa, M., Ohizumi, Y., Gonoi, W., & Abe, O. (2024). *Diagnostic Performances of GPT-4o, Claude 3 Opus, and Gemini 1.5 Pro in "Diagnosis Please" Cases*. https://doi.org/10.1101/2024.05.26.24307915

Srivastava, A., Bhardwaj, S., & Saraswat, S. (2017). SCRUM model for agile methodology. *2017 International Conference on Computing, Communication and Automation (ICCCA)*, 864–869. https://doi.org/10.1109/CCAA.2017.8229928

Storrle, H. (2011). On the impact of layout quality to understanding UML diagrams. *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 135–142. https://doi.org/10.1109/VLHCC.2011.6070390

Storrle, H. (2018). On the impact of size to the understanding of UML diagrams. *Software & Systems Modeling*, *17*(1), 115–134. https://doi.org/10.1007/s10270-016-0529-x

Suaza, K. V., García, J. J. T., & Jaramillo, C. M. Z. (2015). Mejora de historias de usuario y casos de prueba de metodologías ágiles con base en TDD. *Cuaderno activa*, *7*, 41–53.

Sundararajan, S., Bhasi, M., & Vijayaraghavan, P. K. (2014). Case study on risk management practice in large offshore-outsourced Agile *software* projects. *IET Software*, *8*(6), 245–257. https://doi.org/10.1049/iet-sen.2013.0190

Thamrongchote, C., & Vatanawood, W. (2016). Business process ontology for defining user story. *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, 1–4. https://doi.org/10.1109/ICIS.2016.7550829

Trkman, M., Mendling, J., & Krisper, M. (2016). Using business process models to better understand the dependencies among user stories. *Information and Software Technology*, *71*, 58–76. https://doi.org/10.1016/j.infsof.2015.10.006

Tsai, H. C., Huang, Y. F., & Kuo, C. W. (2024). Comparative analysis of automatic literature review using mistral large language model and human reviewers.

Vallon, R., da Silva Estácio, B. J., Prikladnicki, R., & Grechenig, T. (2018). Systematic literature review on agile practices in global *software* development. *Information and Software Technology*, *96*, 161–180. https://doi.org/10.1016/j.infsof.2017.12.004

Wang, X., Conboy, K., & Pikkarainen, M. (2012). Assimilation of agile practices in use. *Information Systems Journal*, *22*(6), 435–455. https://doi.org/10.1111/j.1365-2575.2011.00393.x

Wang, Y., Bogicevic, I., & Wagner, S. (2017). A study of safety documentation in a Scrum development process. *Proceedings of the XP2017 Scientific Workshops*, 1–5. https://doi.org/10.1145/3120459.3120482

Wilbanks, D., Mondal, D., Tandon, N., & Gray, K. (2024). *Large Language Models as Moral Experts? GPT-4o Outperforms Expert Ethicist in Providing Moral Guidance*. https://doi.org/10.31234/osf.io/w7236

Xue, L., Constant, N., Roberts, A., Kale, M., Al-Rfou, R., Siddhant, A., Barua, A., & Raffel, C. (2021). *mT5: A massively multilingual pre-trained text-to-text transformer* (arXiv:2010.11934). arXiv. http://arxiv.org/abs/2010.11934

Yao, Y., Duan, J., Xu, K., Cai, Y., Sun, Z., & Zhang, Y. (2024). A Survey on Large Language Model (LLM) Security and Privacy: The Good, The Bad, and The Ugly. *High-Confidence Computing*, *4*(2), 100211. https://doi.org/10.1016/j.hcc.2024.100211

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., … Wen, J.-R. (2023). *A Survey of Large Language Models* (arXiv:2303.18223). arXiv. http://arxiv.org/abs/2303.18223

8. Anexo A.- Acrónimos

DT	Equipo de Desarrollo
IA	Inteligencia Artificial
LLM	Modelo de Lenguaje Largo
NLP	Procesamiento del Lenguaje Natural
PO	Dueño del Producto
UML	*Lenguaje Unificado de Modelado*
XP	Programación Extrema
RAG	Generación Aumentada por Recuperación
SE	Ingeniería de Software

9. APÉNDICES

Los siguientes apéndices presentan ejemplos representativos de la generación automática de diagramas de clases y diagramas de actividades obtenidos durante la aplicación del caso de estudio. Cada ejemplo incluye la historia de usuario original, el prompt proporcionado al modelo de lenguaje, la respuesta generada en sintaxis PlantUML y el diagrama UML resultante.

Apéndice A. Ejemplo de la generación automática de un diagrama de clases

HU-CL01

As a site member, I want to describe myself on my own page in a semi-structured way, so that others can learn about me.

LLM utilizado:

Gemini 2.5 Pro

***Prompt* proporcionado al LLM:**

You are an expert software analyst and UML modeler.

Your task is to generate a valid PlantUML clases diagram based solely on the user story and additional context provided.

Your task: Generate a UML class diagram (analysis level).

User story (English): As a site member, I want to describe myself on my own page in a semi-structured way, so that others can learn about me.

STRICT RULES:

- *Output ONLY valid PlantUML code.*
- *Must start with @startuml and end with @enduml.*
- *Classes must follow: class ClassName { attributes and methods }.*
- *No , interface, abstract, or constructors like + ClassName(...).*
- *No placeholder classes (e.g., 'System', 'Entity').*
- *No comments or explanations.*
- *Use only +, -, # for visibility.*
- *Use meaningful names based on the user story.*

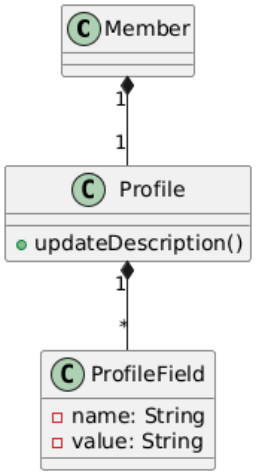
Modeling guidelines:

- *Focus on user/business responsibilities, not technical implementation.*
- *Exclude infrastructure (DB, controllers, services).*
- *Use English names for classes, attributes, methods, or actions.*
- *Keep the diagram clean and consistent.*

FORMAT: Output ONLY valid PlantUML code between @startuml and @enduml.

Respuesta generada por el LLM:

Sintaxis PlantUML generada	Diagrama de clases generado

@startuml
class Member {
}
class Profile {
+ updateDescription()
}
class ProfileField {
- name: String
- value: String
}
Member "1" *-- "1" Profile
Profile "1" *-- "*" ProfileField
@enduml
	

Apéndice B. Ejemplo de la generación automática de un diagrama de actividades

HU-AC06

As a site member, I want to subscribe to an RSS feed of news, so that I remain sufficiently and easily informed.

LLM utilizado:

Gemini 2.5 Pro

Prompt proporcionado al LLM:

You are an expert software analyst and UML modeler.

Your task is to generate a valid PlantUML actividades diagram based solely on the user story and additional context provided.

Your task: Generate a UML activity diagram showing the workflow.

User story: As a site member, I want to subscribe to an RSS feed of news, so that I remain sufficiently and easily informed.

STRICT RULES:

- *Output ONLY valid PlantUML code.*
- *Must start with @startuml and end with @enduml.*
- *Use start, stop, actions (;), and if/else blocks correctly.*
- *Do NOT use class syntax.*
- *No comments or explanations.*
- *Use clear action names from the user story.*

Modeling guidelines:

- *Focus on user/business responsibilities, not technical implementation.*
- *Exclude infrastructure (DB, controllers, services).*
- *Use English names for classes, attributes, methods, or actions.*
- *Keep the diagram clean and consistent.*

FORMAT: Output ONLY valid PlantUML code between @startuml and @enduml.

Respuesta generada por el LLM:

Sintaxis PlantUML generada	Diagrama de actividades generado
tartuml start :Site member navigates to news section; :Site member selects option to subscribe to RSS feed; :System provides the RSS feed link; :Site member adds the feed link to their RSS reader; stop @enduml