MODELO DE REFERENCIA DE SISTEMA BASADO EN INTELIGENCIA ARTIFICIAL PARA FACILITAR LA MODERNIZACIÓN DE SISTEMAS HEREDADOS

Autor:
Carlos David Roldán Vélez

Asesor:
Sergio Armando Gutiérrez

Trabajo de grado

UNIVERSIDAD EAFIT
ESCUELA DE CIENCIAS APLICADAS E INGENIERÍA
Maestría en ingeniería
Medellín
2025

Índice

Índice de figuras

Índice de cuadros

1. Introducción

Muchas de las grandes empresas comenzaron a computarizar sus operaciones en la década de 1960, utilizando distintos tipo de sistemas centrales conocidos comúnmente como "mainframe" y el software. En este contexto, el termino mainframe se refiere a una clase de computadora centralizada de alto rendimiento, diseñada específicamente para manejar grandes volúmenes de datos y soportar miles de transacciones simultáneas de forma estable. Esta capacidad operativa llevó a que una gran cantidad de estos sistemas fueran introducidos en distintas industrias [1]. Durante este tiempo, muchos de estos sistemas han sido reemplazados, debido a que, naturalmente la evolución de cualquier industria o producto es inevitable [2]. Aunque se pensó que serían reemplazados rápidamente, como el columnista Stewart Alsop predijo en 1991 que 'el último mainframe se desconectaría en 1996', no ha sido así [3]. Hoy en día muchos de estos sistemas persisten y siguen operando, soportando los servicios críticos de las organizaciones, particularmente en sectores como el bancario, gubernamental, telecomunicaciones y salud. A raíz de esta persistencia y la necesidad de mantenerlos operativos, estos sistemas reciben la denominación de sistemas heredados (legacy systems) [2, 4].

A medida que la tecnología ha evolucionado, han surgido nuevas soluciones que ofrecen ventajas significativas sobre los sistemas heredados, en términos de escalabilidad, flexibilidad y capacidad de integración [5]. Estas soluciones modernas permiten a las empresas adaptarse rápidamente a las demandas cambiantes del mercado, al facilitar la integración con tecnologías emergentes [5]. A pesar de esto, la migración de sistemas heredados a arquitecturas modernas de desarrollo de software es una tarea desafiante y compleja, que acarrea consigo un elevado costo [5].

En este contexto, la aparición de tecnologías avanzadas como la inteligencia artificial (IA) ofrece la oportunidad de automatizar y mejorar tareas inherentes al ciclo de vida del desarrollo de software, entre ellas, planificación, desarrollo, pruebas y despliegue. Para ello, se han venido usando diferentes técnicas como machine learning (ML), redes neuronales y procesamiento del lenguaje natural y, mas recientemente, sistemas mas complejos que integra múltiples agentes de IA para llevara a cabo tareas que exceden la capacidad de un solo algoritmo [6–11].

A pesar del potencial de estas tecnologías, su aplicación en el contexto especifico de la modernización de sistemas heredados ha sido poco explorada [12]. Los trabajos existentes se enfocan en casos puntuales, como la traducción automática de código entre lenguajes o la generación de documentación para sistemas heredados o aproximaciones de ingeniería inversa. Esto evidencia que se trata de un campo emergente pero aún fragmentado [13].

En este escenario, el presente trabajo propone la construcción de un modelo de referencia basado en agentes de IA para la modernización de sistemas heredados. Para lograrlo, se llevó a cabo un mapeo sistemático de la literatura que permitió identificar cómo se está utilizando la IA, tanto en el desarrollo de software como en la modernización de sistemas heredados. A partir de la cual, se busca integrar enfoques, técnicas y desafíos reportados, con el fin de construir un modelos de referencia que aporte, desde el ámbito académico, claridad conceptual, y desde el ámbito practico, insumos que faciliten la toma de decisiones estratégicas para la modernización en

las organizaciones.

2. Planteamiento del problema

Un sistema heredado se define como una infraestructura tecnológica que ha sido administrada por las organizaciones durante más de una década y que se apropia del termino heredado al haber sido utilizada por varias generaciones de usuarios y desarrolladores [2, 14]. Este tipo de infraestructura tecnológica ha evolucionado a lo largo de años de desarrollo, acumulando un volumen significativo de líneas de código, convirtiéndose en estructuras altamente complejas. Estos sistemas suelen estar fundamentados en arquitecturas, lenguajes de programación o plataformas han sido descontinuados [2, 14].

En el contexto empresarial contemporáneo, estos continúan desempeñando un papel fundamental como activos estratégicos organizacionales, pues en ellos se sustenta lógica de procesos esenciales del negocio [4, 5]. Sin embargo, el continuar en funcionamiento genera una paradoja: mientras continúan siendo elementos indispensables para la continuidad operacional, simultáneamente representan barreras significativas para la transformación digital y la adaptación a las demandas del mercado actual [15].

La literatura identifica múltiples desafíos asociados con los sistemas heredados, los cuales pueden categorizarse en dimensiones técnicas, económicas y organizacionales [5, 15]. A continuación se detallan las principales problemáticas identificadas

2.1. Problemáticas arquitectónicas y tecnológicas

Arquitecturas obsoletas y rígidas: Los sistemas heredados frecuentemente implementan paradigmas arquitectónicos desactualizados, lo que dificulta la integración con tecnologías emergentes [5].

Uso de tecnologías descontinuadas: Muchos de estos sistemas dependen de lenguajes y plataformas obsoletas, lo que implica riesgos seguridad y falta de soporte técnico [2]. Esta situación se ve agravada por la creciente escasez de profesionales especializados en dichos lenguajes y plataformas [2, 16], lo que incrementa tanto los costos de recursos humanos especializados como los tiempos de resolución de incidencias [2, 15].

Problemas de integración: La incompatibilidad con protocolos y estándares de intercambio de datos modernos dificulta significativamente la capacidad de estos sistemas para comunicarse con otra aplicaciones [17].Esta desconexión tecnológica entre sistemas heredados y plataformas actuales, genera silos de información que obstaculiza la implementación de ecosistemas tecnológicos integrados [2].

2.2. Problemáticas de rendimiento

Limitaciones de capacidad y escalabilidad: Los sistemas heredados frecuentemente presentan restricciones inherentes a la capacidad de procesamiento, almacenamiento y concurrencia de usuarios [18]. Esta situación limita a las organizaciones para responder adecuadamente la demanda creciente de procesamiento de datos, incrementos en la base de usuarios o picos de carga operacional, comprometiendo la competitividad empresarial en mercados dinámicos [18].

Degradación progresiva del rendimiento: Con el transcurso del tiempo, estas infraestructuras experimentan una degradación natural de su rendimiento debido a la acumulación de datos históricos, la fragmentación de base de datos y la ausencia de optimizaciones moderna, lo que resulta en tiempos de respuesta más altos que llevan a una degradación de la experiencia del usuario [15].

2.3. Problemáticas económicas

Costos elevados de mantenimiento: La complejidad técnica inherente a los sistemas heredados genera costos de mantenimiento desproporcionadamente altos en comparación con soluciones modernas [2, 5]. Según estimaciones especializadas, la deuda técnica asociada con el mantenimiento de sistemas heredados representa un costo superior al trillon de dólares anuales para la economía estadounidense [19]. Estos costos incluyen no solo el mantenimiento correctivo y preventivo, sino también los recursos destinados a la adaptación de interfaces y la resolución de problemas de compatibilidad [19].

2.4. Problemáticas de cumplimiento

Desafíos de cumplimiento normativo: Las infraestructuras tecnológicas desactualizadas se enfrentan a la dificultad de cumplir con regulaciones contemporáneas en materia de protección de datos, seguridad de la información y auditoria [5].

2.5. Problemáticas operacionales

Confiabilidad y riesgo operativo: La propensión a fallas e interrupciones aumenta progresivamente con la antigüedad de los sistemas, comprometiendo la continuidad operacional y generando impactos negativos en la experiencia del cliente y la reputación organizacional [5].

Limitaciones para la innovación: La rigidez estructural de estas infraestructuras dificulta significativamente la adaptación a nuevos modelos de negocio, paradigmas tecnológicos emergentes y expectativas cambiantes de los clientes, generando un estancamiento en la innovación, lo que compromete la posición competitiva de la organización [5, 15].

2.6. Problemáticas gestión del conocimiento

Déficit documental critico: La ausencia de documentación técnica actualizada, especificaciones funcionales completas y manuales de procedimiento, obliga a los desarrolladores a depender exclusivamente del código fuente y procesos de ingeniería inversa [5,20]. Esta situación aumenta exponencialmente los tiempos de diagnostico, mantenimiento y desarrollo de nuevas funcionalidades [5].

Estos factores sumados, generan una brecha significativa entre las capacidades tecnológicas de las que disponen los sistemas heredados y las demandas operacionales de las organizaciones en la actualidad [21]. El resultado es un circulo vicioso donde

los recursos que a hoy se destinan en el mantenimiento de estos sistemas crece exponencialmente [22].

3. Justificación

La modernización de los sistema heredados constituye uno de los desafíos más crítico y costosos que enfrentan las organizaciones en la actualidad, dado que gran parte de sus procesos estratégicos y operativos dependen de plataformas obsoletas que demandan altos recursos para su mantenimiento y dificultan la innovación [23]. Se estima que entre el 60 % y el 70 % de las aplicaciones relevantes en grandes empresas corresponden a sistema heredados, las cuales consumen hasta dos tercios del presupuesto en operaciones del área de tecnología [16, 23]. Por ejemplo el gobierno de los EE. UU. gasto alrededor de 90 mil millones de dólares en TI en el año fiscal de 2019, de los cuales alrededor del 80 % fue usado para operar y darle mantenimiento a sistemas heredados [24]. Este panorama ejerce una presión significativa en las organizaciones, las cuales necesitan reducir costos, aumentar su capacidad de adaptación a los cambios y minimizar su dependencia de infraestructuras rígidas [4].

En este contexto desafiante, la IA ha emergido como una tecnología particularmente prometedora para abordar la complejidad inherente a la modernización de sistemas heredados [25]. Su capacidad para procesar y analizar grandes volúmenes de código fuente, identificar patrones complejos y automatizar tareas técnicas la convierten en una herramienta ideal para superar esta barrera [25, 26].

En los últimos años se ha comenzado documentar aplicaciones experimentales y casos de estudio que demuestran la aplicabilidad de diversas técnicas de IA en el contexto de la modernización de sistemas heredados [27]. Estas implementaciones abarcan un espectro amplio de tecnologías, incluyendo algoritmos de aprendizaje automático (machine learning), técnicas de procesamiento de lenguaje natural (NLP por sus siglas en ingles), sistemas de visión artificial, y mas recientemente, modelos de lenguaje de gran escala (LLM, Large Language Models) [7, 10, 28–30]. Los propósitos de aplicación de estas tecnologías son igualmente diversos, incluyendo análisis automático de código fuente, migración asistida de datos, generación automática de documentación técnica e identificación de patrones de diseño [7, 10, 28–30].

No obstante, el potencial transformador evidenciado en estas implementaciones se ve limitado, ya que los hallazgos reportados en la literatura científica permanecen fragmentados. La ausencia de una visión unificada dificulta tanto la consolidación del conocimiento como la adopción de practicas efectivas en entornos organizacionales, restringiendo así el verdadero impacto de la IA en la modernización de sistemas heredados [13].

En respuesta a esta brecha, la presente investigación plantea la construcción de un modelo de referencia fundamentado en un mapeo sistemático de la literatura, que permita identificar, categorizar e integrar los enfoques, técnicas y desafíos reportados en este campo. Dicho modelo no solo busca aportar claridad conceptual desde el ámbito académico, sino ofrecer insumos prácticos que faciliten la toma de decisiones estratégicas en las organizaciones en procesos de modernización tecnológica.

4. Pregunta de investigación

¿Como puede un sistema multi agente basado en inteligencia artificial facilitar el proceso de modernización de sistemas heredados?

5. Objetivos

5.1. General

Diseñar un modelo de referencia de sistemas multiagentes basado inteligencia artificial que facilite la modernización de sistemas heredados.

5.2. Específicos

OE1. Identificar los componentes clave, tendencias emergentes, limitaciones y desafíos en el contexto de modernización de sistemas heredados y el uso de inteligencia artificial, a través de un mapeo sistemático de la literatura que permita establecer una base de conocimiento robusta para el diseño de un modelo de referencia.

OE2. Integrar los elementos claves identificados en el OE1 mediante el uso de un framework de armonización de múltiples modelos, con el fin de establecer una base de conocimiento unificada que servirá como punto de partida para el diseño de la arquitectura del sistema.

OE3. Diseñar una arquitectura de referencia del sistema, que incorpore los elementos claves armonizados resultado del OE2 y facilite la migración de código legado, asegurando que las soluciones resultantes sean eficientes y escalables en entornos empresariales modernos

6. Metodología

Con el fin de llevar a cabo una investigación sistemática y organizada, se planteó el uso de la metodología investigación-acción con múltiples ciclos propuesta por Harper [31], la cual es un enfoque que busca generar conocimiento a través de la acción y reflexión. Esta metodología es particularmente útil para problemas complejos y dinámicos, como la modernización de sistemas heredados, ya que permite generar iteraciones y ajustes basados en la reflexión de los resultados obtenidos en cada ciclo. Para lograrlo la metodología se basa en la ejecución de ciclos divididos en 4 etapas como se muestra en la Figura 1.

1. **Planeación:** Identificación del problema y diseño de la intervención.
2. **Ejecución:** Implementación de la acción planificada.
3. **Observación:** Recopilación de datos y evaluación de los resultados.
4. **Reflexión:** Análisis de los resultados y reflexión sobre el proceso.

Figura 1: Etapas de la metodología investigación acción.

Para el caso de esta investigación se dividió en 4 ciclos, que guiaron las actividades del proyecto. A continuación, se describe cada ciclo.

6.1. Ciclo 1 (Mapeo sistemático de literatura)

El ciclo 1 (ver Figura2) se realizó a través de un mapeo sistemático de la literatura, siguiendo los lineamientos planteados en [32]. Este es un método de investigación, que se utiliza para proporcionar una visión general de la literatura en un área de

estudio especifico. En este caso se realizó el mapeo sistemático de la literatura con el fin de identificar, explorar y clasificar la literatura relevante para establecer un marco teórico sólido en los campos de la modernización de sistemas heredados y el uso de IA.

A continuación, se presentan las etapas y actividades ejecutadas para llevar a cabo el mapeo sistemático de la literatura.

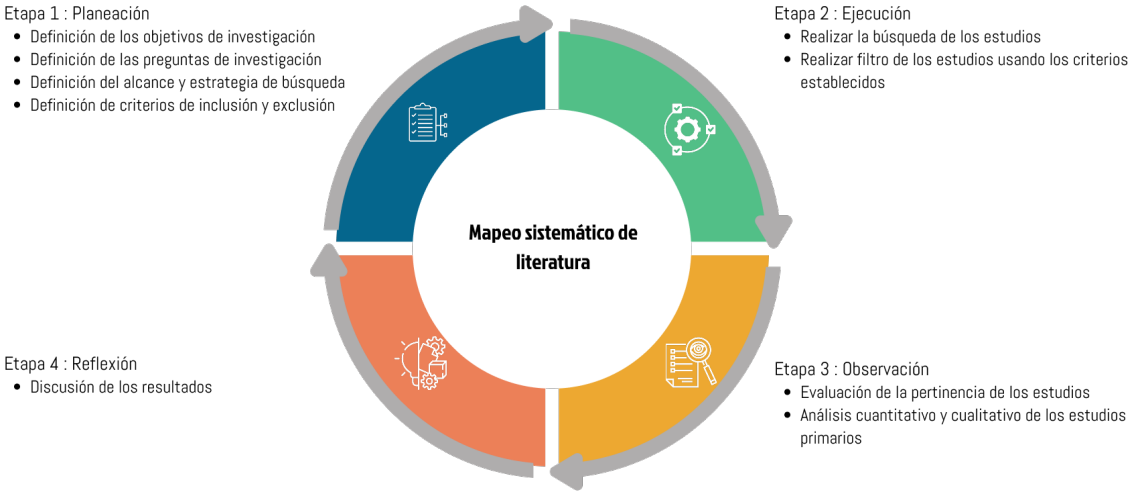

Figura 2: Ciclo 1, mapeo sistemático de literatura.

6.1.1. Etapa 1 del ciclo 1 (Planeación)

En esta fase se establecen las actividades estratégicas necesarias para obtener información valiosa y relevante para el área de interés, permitiendo ser usada en etapas posteriores.

- **Definición de los objetivos de investigación:** La definición de los objetivos de investigación es esencial para orientar la elección de la literatura y estudios relevantes, proporcionando así el marco inicial para la identificación de los componentes clave de la investigación garantizando con esto una que sea enfocada y eficiente.

- **Definición de la pregunta de investigación:** Las preguntas de investigación surgen como consecuencia lógica de los objetivos establecidos. Es a partir de estas que los objetivos se transforman en interrogantes concretas y medibles.

- **Definición del alcance y estrategia de búsqueda:** Para la definición del alcance, se delimita el ámbito de la búsqueda a partir de la selección de los motores y bases de datos más relevantes para el tema de la investigación. Además, se definió el alcance temporal y se elaboraron las ecuaciones de búsqueda, mediante un proceso de validación y refinamiento que permitió adaptar la estrategia según las características específicas de cada base de datos, basada en las siguientes actividades i) Identificación de conceptos clave, ii) Expansión de

términos, iii) Estructuración usando PICO (Population/Dominio, Intervention/Metodología, Comparison/Comparación y Outcome/Resultado), iv) Construcción incremental, v) Proceso de validación, vi) Refinamiento de términos de búsqueda vii) Adaptación por base de datos [33]. Esto garantiza que la revisión se centre en la información más pertinente y actualizada posible [33].

- **Definición de criterios de inclusión y exclusión:** Estos criterios especifican las características que deben cumplir los estudios para ser incluidos en el análisis. De esta forma, se garantizó que se analizaran aquellos estudios relevantes para la pregunta y los objetivos de investigación establecidos.

6.1.2. Etapa 2 del ciclo 1 (Ejecución)

La fase de ejecución se refiere al conjunto de actividades llevadas a cabo para obtener los resultados de interés, tras aplicar la cadena de búsqueda en cada una de las bases de datos científicas. A continuación, se describen las actividades planteadas para esta etapa.

- **Realizar la búsqueda de los estudios usando la cadena de búsqueda:** La cadena de búsqueda obtenida durante la definición de la estrategia será aplicada en cada una de las bases de datos científicas seleccionadas. Para ello, se realizaron n-iteraciones, donde cada iteración representa la aplicación de la cadena en una base de datos.
- **Realizar el filtrado de los estudios usando los criterios establecidos:** Cada una de las iteraciones de búsqueda será sometida a tres filtros de exclusión. El primer filtro se aplicará para eliminar estudios duplicados de búsquedas previas. El segundo filtro servirá para descartar aquellos estudios que cumplan con uno o más criterios de exclusión. El resultado esperado de esta fase es un conjunto de estudios relevantes, los cuales podrían responder —o no— a las preguntas de investigación. Finalmente, el tercer filtro implicará la lectura completa del artículo para determinar su relevancia directa con los objetivos y preguntas planteados durante la fase de planeación. Los estudios que superen este filtro serán clasificados como estudios primarios.

6.1.3. Etapa 3 del ciclo 1 (Observación)

El análisis de los resultados obtenidos en la fase de ejecución se llevó a cabo a través de dos actividades principales:

- **Evaluación de la pertinencia de los estudios:** Se realizó mediante un análisis cuantitativo de los resultados obtenidos después de aplicar los criterios definidos en la fase de planeación. Estos criterios permitieron determinar la pertinencia de los estudios en relación con los objetivos planteados.
- **Análisis cuantitativo y cualitativo de los estudios primarios:** Esta actividad consistió en responder de manera sistemática cada una de las preguntas de investigación propuestas en la fase de planeación. Para lograrlo, se aplicaron métodos de análisis cuantitativos y cualitativos que permitieron obtener información valiosa y alineada con los objetivos de investigación.

6.1.4. Etapa 4 del ciclo 1 (Reflexión)

La fase de reflexión incluye todas las actividades necesarias para elaborar el reporte estructurado de los resultados, identificando brechas, desafíos y estableciendo el panorama de trabajo futuro. Esto se logró mediante un análisis profundo y crítico de los resultados obtenidos, a través de un proceso de síntesis y categorización, en el cual se identificaron patrones, tendencias y mejores prácticas.

6.2. Ciclo 2 (Armonización de múltiples modelos)

El ciclo 2 (ver Figura 3) se llevó a cabo integrando los elementos identificados en el mapeo sistemático de la literatura mediante el framework de armonización de múltiples modelos y estándares. Este framework propuesto por Pardo et al. [34], facilita la tarea de homogenización, comparación e integración, de los modelos, estándares y prácticas identificados en el mapeo sistemático de la literatura, lo cual servirá como un marco de referencia para el diseño de la arquitectura del sistema. En general, este framework define un conjunto de actividades que deben llevarse a cabo para armonizar múltiples modelos y estándares, a través de los siguientes métodos:

1. Método de identificación
2. Método de homogenización
3. Método de comparación
4. Método de integración

Figura 3: Ciclo 2, Armonización de múltiples modelos.

A continuación, estos métodos se integran a la metodología investigación-acción, y se describen con mayor detalle las actividades realizadas en cada una de las etapas.

6.2.1. Etapa 1 del ciclo 2 (Planeación)

A partir de las actividades planteadas en el método de armonización, la fase de planeación involucra todas las acciones relacionadas con el análisis y la caracterización de las soluciones que deben armonizarse. En el contexto del proyecto, este proceso se desarrolló mediante el análisis de los resultados obtenidos en el mapeo sistemático de la literatura, aplicando el método de identificación.

- **Método de identificación:** Este método, que hace parte de las fases propuestas por Pardo et al. [34], tiene como propósito reconocer y definir los elementos clave dentro de los distintos modelos y estándares del problema específico. Para ello, se utilizó como insumo el mapeo sistemático de la literatura realizado en el ciclo 1. En este método se plantearon las siguientes actividades:

  (I) Análisis estructurado de los resultados del mapeo sistemático.

  (II) Categorización de los elementos.

  (III) Identificación de patrones y prácticas.

6.2.2. Etapa 2 del ciclo 2 (Ejecución)

Para la ejecución de la armonización, el método aplicado incluye las siguientes actividades:

- **Método de homogeneización (HoMethod):** Este método busca alinear o homogeneizar los elementos de diferentes soluciones y, con ello, crear una base común que permita normalizar prácticas o conceptos que tienen un propósito similar, pero que están descritos de manera diferente en cada modelo. Para llevarlo a cabo, se aplicaron las siguientes actividades descritas en HoMethod [34]:

  (I) Adquisición de conocimiento concerniente a los modelos.

  (II) Análisis estructural y terminológico.

  (III) Identificación de los requerimientos necesarios para la homogeneización.

  (IV) Identificación de correspondencias.

  (V) Análisis de resultados.

- **Método de comparación (MaMethod):** El proceso de comparación o MaMethod (por Mapping Method) permite analizar detalladamente las similitudes, diferencias y relaciones entre los elementos de cada modelo. Este método resulta especialmente útil para identificar factores de alineación y conflicto, así como para decidir cuáles prácticas se pueden combinar, adaptar o excluir. Su aplicación se realizó mediante las siguientes actividades propuestas por [35]:

  (I) Análisis de las soluciones.

  (II) Diseño de la comparación.

  (III) Ejecución de la comparación.

6.2.3. Etapa 3 del ciclo 2 (Observación)

Método de integración: Esta etapa tiene como objetivo unificar los elementos obtenidos durante el proceso de comparación en una estructura común, genérica y agnóstica. Para ello, se aplicó el método de integración [36], el cual permite obtener de manera sistemática los elementos de proceso integrados en múltiples modelos. Este método contempla cinco actividades principales:

(I) Diseño de la integración.

(II) Definición del criterio de integración.

(III) Ejecución de la integración.

(IV) Análisis de los resultados de la integración.

(V) Presentación del modelo integrado.

6.2.4. Etapa 4 del Ciclo 2 (Reflexión)

Después de caracterizar y definir la solución de acuerdo con los alcances y objetivos propuestos en la armonización, se llevaron a cabo actividades de retroalimentación orientadas a identificar limitaciones, sesgos y a realizar análisis de correspondencia, con el fin de reducir el grado de subjetividad en la arquitectura de referencia. A continuación, se describen las actividades asociadas a este propósito:

- **Análisis de correspondencia:** El método de armonización propone realizar un análisis de correspondencia que facilite determinar el grado de cohesión entre los elementos comparados durante esta etapa. Dicho análisis permite establecer el nivel de relación entre los elementos integrados y, de esta manera, asignar un valor cuantitativo medible que facilite su evaluación futura.
- **Identificación de sesgos y limitaciones:** Con el propósito de reducir la subjetividad de la propuesta, se identificaron los sesgos asociados a la aplicación del método de armonización. Estos sesgos pueden estar relacionados con la metodología empleada, la participación del grupo investigador o el grado de subjetividad introducido durante la ejecución de cada actividad.

6.3. Ciclo 3 (Diseño de arquitectura)

Este ciclo (ver Figura 4) se enfocó en la construcción detallada de la arquitectura del sistema, la cual requería un marco metodológico que facilitara la creación de una representación sistemática y comunicable de sistemas complejos. En este contexto, y con el fin de cumplir dicho propósito, en el proyecto se adoptó el modelo C4 (Context, Containers, Components, Code) propuesto por Simon Brown [37]. Este modelo surge como una aproximación metodológica para documentar y visualizar arquitecturas de software en diferentes niveles de abstracción. El enfoque resulta especialmente útil para sistemas complejos, ya que organiza las representaciones

en vistas jerárquicas, lo que permite tanto a audiencias técnicas como no técnicas comprender cómo interactúan los distintos componentes del sistema.

Para la adopción de C4 en este ciclo, se tomó como insumo principal los resultados de la armonización de múltiples modelos desarrollada en el ciclo 2, la cual proporcionó un marco de referencia basado en mejores prácticas y estándares.

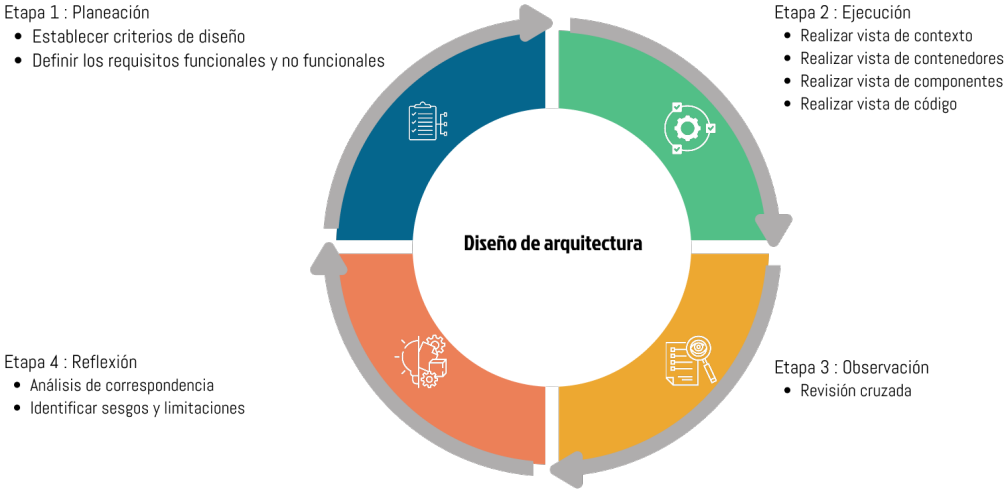

Figura 4: Ciclo 3, Diseño de la arquitectura.

6.3.1. Etapa 1 del ciclo 3 (Planeación)

En esta etapa se definieron los fundamentos arquitectónicos que guiaron el diseño del sistema, asegurando que las decisiones estuvieran alineadas con los objetivos del proyecto. Para ello, se llevaron a cabo las siguientes actividades:

- **Establecer criterios de diseño:** Esta es una actividad fundamental en la planeación de la arquitectura, ya que define los lineamientos y restricciones que guían la construcción del sistema. Estos criterios se desagregan en dos dimensiones principales:
  - **Principios arquitectónicos:** Son reglas o recomendaciones de alto nivel que garantizan que el diseño sea consistente, escalable y sostenible. Algunos ejemplos de estos principios son la modularidad, la interoperabilidad, la simplicidad y la seguridad.
  - **Restricciones de diseño:** En esta subactividad se identifican las limitaciones técnicas, organizacionales y/o regulativas que influyen en el diseño de la arquitectura y que derivan del análisis de armonización. Estas restricciones permiten garantizar que el diseño sea realista y factible dentro del contexto del proyecto.
- **Definir los requisitos funcionales y no funcionales:** Esta actividad constituye la base del diseño arquitectónico, ya que determina qué debe hacer el sistema (requisitos funcionales) y cómo debe hacerlo (requisitos no funcionales).

6.3.2. Etapa 2 del ciclo 3 (Ejecución)

Esta etapa se enfocó en desarrollar cada una de las vistas arquitectónicas del modelo C4, proporcionando así diferentes niveles de detalle que facilitan la comprensión y comunicación de la estructura del sistema. Cada vista aborda un aspecto específico de la arquitectura, desde una visión general hasta los detalles más técnicos. Para este proyecto, las actividades relacionadas con las vistas del modelo se desarrollaron de la siguiente forma:

- **Vista de contexto:** Es la representación de más alto nivel de abstracción, mostrando cómo el sistema interactúa con usuarios y sistemas externos. Además, define los límites y el propósito del sistema.
- **Vista de contenedores:** Esta vista descompone el sistema en contenedores tecnológicos y detalla los elementos principales, agrupándolos en funcionalidades soportadas por tecnologías específicas.

6.3.3. Etapa 3 del ciclo 3 (Observación)

El objetivo principal de esta etapa fue evaluar el diseño arquitectónico propuesto, asegurando su consistencia, claridad y alineación con los objetivos y criterios definidos previamente. Para ello, se desarrolló la siguiente actividad:

- **Revisión cruzada (Peer Review):** Es un proceso colaborativo que involucra tanto a participantes técnicos como no técnicos, con el fin de evaluar el diseño arquitectónico. Esta actividad permite verificar que el diseño cumple con los requisitos y que sus componentes (vistas C4) son coherentes y están alineados con los principios y restricciones previamente establecidos. Para llevarla a cabo, se realizaron entrevistas estructuradas con los participantes, a fin de recopilar percepciones sobre:
  - La claridad y comprensibilidad de las vistas arquitectónicas.
  - La alineación del diseño con los objetivos del proyecto.
  - Las fortalezas y debilidades percibidas en cada vista.

6.3.4. Etapa 4 del Ciclo 3 (Reflexión)

Esta etapa se centró en analizar los resultados de la observación para identificar los aprendizajes claves y las oportunidades de mejora en el diseño. Se empleó un análisis sistemático para transformar los hallazgos en conocimiento práctico, para esto en primer lugar se revisaron los comentarios y percepciones recopiladas en la etapa anterior, para así entender las causas de las brechas identificadas.

7. Marco teórico

En este capítulo se presentan los fundamentos conceptuales y teóricos que sustentan la presente investigación sobre la modernización de sistemas heredados utilizando inteligencia artificial. En primer lugar, se define y caracteriza el concepto de sistemas heredados, identificando sus principales problemáticas desde el punto de vista técnico y organizacional. A continuación, se aborda el papel de la inteligencia artificial en el ciclo de vida del software, con énfasis en enfoques recientes como el uso de agentes inteligentes y modelos de lenguaje. Finalmente, se revisan modelos y marcos teóricos relevantes para guiar procesos de modernización tecnológica. Este marco servirá como base para el diseño del modelo de referencia propuesto.

7.1. Sistemas heredados

Los sistemas heredados también conocidos como legacy systems, son aplicaciones o infraestructuras tecnológicas desarrolladas con paradigmas, lenguajes de programación o plataformas, aunque hoy son consideradas obsoletas, continúan en operación debido a que soportan procesos críticos para las organizaciones [2, 5, 14, 38]. En la literatura estos sistemas suelen caracterizarse por haber estado en funcionamiento durante un largo periodos de tiempo (generalmente más de una década) y por concentrar lógica de negocio y volúmenes de datos cuya relevancia sigue siendo clave para la continuidad operativa [2].

Desde un punto de vista teórico, el envejecimiento del software ha sido ampliamente estudiado por Lehman a través de sus leyes de evolución del software, postuladas en la década de 1970 [39]. En particular, tres de ellas permiten comprender por qué muchos sistemas se convierten en heredados:

- La **ley de cambio continuo** establece que un sistema en uso debe adaptarse constantemente a su entorno, o dejaran progresivamente de cumplir con los requerimientos de los usuarios [40].
- La **ley de complejidad creciente** sostiene que todo sistema tiende a aumentar su complejidad estructural con el tiempo, a menos que se invierta explícitamente en su control, lo que explica la alta complejidad que caracteriza los sistemas heredados [5, 40]
- La **ley del crecimiento continuo** señala que los sistemas deben ampliar regularmente su funcionalidad, lo que muchas veces genera sobrecarga estructural, deuda técnica y rigidez [40].

Estas leyes ayudan a explicar por qué los sistemas heredados no solo son antiguos, sino también complejos, costosos de mantener y difíciles de transformar. En términos técnicos y organizacionales, se presentan las siguientes características distintivas:

- **Antigüedad tecnológica:** Fueron desarrollados en lenguajes de programación y arquitecturas que que actualmente tienen escaso soporte (Ejm COBOL, FORTRAN, JCL, mainframes) [5].

- **Dependencia organizacional:** Concentran funciones esenciales del negocio, lo que dificulta su sustitución sin generar riesgos operativos significativos [2].
- **Alta complejidad estructural:** Contienen millones de líneas de código y múltiples dependencias, lo que dificulta su comprensión y mantenimiento [5].
- **Rigidez y baja capacidad de interoperatividad:** Su diseño no está orientado a estándares modernos de integración, lo que limita su capacidad para interactuar con nuevas aplicaciones y plataformas [5].
- **Costos crecientes de mantenimiento:** Debido tanto a su complejidad como a la escasez de profesionales capacitados en las tecnologías utilizadas [2].
- **Persistencia estratégica:** A pesar de sus limitaciones, muchas empresas optan por mantenerlos en producción debido al riesgo, costo o complejidad asociados con su reemplazo [4].

Los sistemas heredados se agrupan en una amplia tipología, que refleja tanto la evolución histórica de la informática como las decisiones arquitectónicas que se adoptaron en dichos contextos históricos [24]. Entre los más representativos se encuentran los mainframes, los cuales eran computadoras de gran capacidad, que permitían procesar grandes volúmenes de datos y transacciones con alta fidelidad [3]. Gracias a su robustez, en su época lograron garantizar durante décadas la operación de procesos críticos en sectores como el bancario, los seguros y el gubernamental [3]. Asimismo se destacan los sistemas monolíticos, caracterizados por una arquitectura rígida y acoplada, lo que dificulta su escalabilidad y mantenibilidad frente a las demandas de flexibilidad en la actualidad. A estas se suman variantes, como los sistemas cliente servidor heredados y aplicaciones construidas sobre lenguajes y entornos de desarrollo que a hoy se consideran obsoletos [2, 5].

7.2. Modernización de sistemas heredados

El termino “modernización de software” surgió como un tema de investigación a principios de los años 2000 [24]. Comprender la modernización implica entender que va mucho mas allá de una simple actualización tecnológica. Se trata en esencia, de un proceso de transformación de la arquitectura y las organizaciones, cuyo objetivo es restablecer la capacidad de evolución de los sistemas que han caído en una rigidez estructural crítica [15].

La modernización se fundamenta teóricamente en reconocer que estos sistemas, a pesar de ser técnicamente funcionales, sufren lo que Parnas [41] definió como obsolescencia de diseño. Esta condición se da cuando las decisiones arquitectónicas originales, aunque válidas en su momento, se convierten en restricciones que impiden la adaptación a nuevos paradigmas de negocio y tecnología [41]. Esta conceptualización explica porque la modernización no es solo un capricho técnico sino un imperativo estratégico para ofrecer a los usuarios productos y servicios innovadores y competitivos [24].

El modelo de ciclo de vida extendido propuesto por Bennet y Rajlich [42] ofrece el marco clave para comprender cuándo se vuelve indispensable la modernización.

Según este modelo, el software atraviesa fases predecibles: Desarrollo, evolución, mantenimiento (servicing), desfase (phase-out) y retiro (close-down). Segun el autor, la modernización por lo tanto es una intervención estratégica diseñada para revertir la transición hacia fases terminales, permitiendo que el sistema recupere un estado donde la evolución sea nuevamente posible [42].

7.3. Estrategias de modernización

La literatura ofrece distintos marcos conceptuales para guiar las estrategias de modernización. Uno de los mas influyentes es el Horseshoe Model (Modelo de la herradura) de Byrne [43], que establece que existe un relación teórica fundamental entre el nivel de transformación y el conocimiento del sistema. En otras palabras, cuanto más drástica sea la transformación del sistema, menos conocimiento del sistema heredado necesitamos, y viceversa. El modelo recibe su nombre por la forma de herradura (U) que toma cuando se gráfica su relación:

- En el extremo izquierdo están los cambios mínimos (como mantenimiento o refactorización), donde se requiere un alto conocimiento del sistema heredado para no alterar su comportamiento.
- En el centro están los cambios radicales, como la reescritura completa o sustitución, donde el conocimiento del sistema existente pierde relevancia, ya que este se diseña prácticamente desde cero.
- En el el extremo derecho de la herradura, donde los cambios son intermedios (Por ejemplo, reingeniería parcial o migración de plataforma), se requiere el máximo conocimiento del sistema, ya que el éxito depende de comprender y adaptar estructuras existentes sin destruirlas completamente.

Este modelo explica por qué los contextos organizacionales favorecen distintas estrategias. Esto ocurre especialmente en sistemas con escasa documentación y alta complejidad, los cuales tienden hacia enfoques preservativos. En cambio, las arquitecturas que están irremediablemente degradadas o que están bien comprendidas justifican intervenciones reconstructivas más drásticas [43].

7.4. Principales enfoques de modernización

Los enfoques de la modernización de sistemas heredados dependen en gran medida, del valor que aún aporta el sistema, de la criticidad de su funcionalidad y la viabilidad de la intervención técnica [38].

Estos son los principales enfoques identificados en la literatura académica, desde la mayor hasta la menor intervención:

7.4.1. Intervención mínima y adaptación externa

En este extremo, el objetivo es extender la vida útil del sistema sin modificar su código principal.

- **Encapsulamiento:** Consisten en envolver el sistema heredado con una capa de software (como una API o servicio web) para exponer sus funcionalidades al mundo moderno sin alterar su lógica interna [44].
- **Rehosting (Lift-and-shift):** Implica trasladar el sistema a una nueva infraestructura (por ejemplo, mover un mainframe a una nube) sin modificar una sola linea de código. Aunque no mejora el diseño interno, ofrece beneficios inmediatos como la reducción de costos operativos y facilita las tareas de soporte [45].

7.4.2. Modificación interna y estructural

Estas estrategias se centran en mejorar el código o el diseño del sistema para facilitar el mantenimiento.

- **Refactorización:** Se enfoca en mejorar el código fuente interno del sistema sin cambiar su comportamiento externo. Esta técnica es ideal cuando el sistema funciona bien, pero su mantenimiento se ha vuelto costoso y arriesgado debido al desorden o la deuda técnica acumulada en su estructura [46].
- **Reingenieria:** Representa un cambio más profundo. Requiere un análisis estructural exhaustivo (a menudo mediante ingeniería inversa) para rediseñar partes claves de la lógica de negocio o de la arquitectura técnica. Se aplica cuando el sistema tiene valor, pero necesita una transformación significativa para mantener su relevancia [47].

7.4.3. Reemplazo completo

En el extremo de mayor intervención, se opta por sustituir el sistema existente, asumiendo el mayor riesgo a cambio de máxima flexibilidad.

- **Reescritura:** Esta es la opción más flexible, ya que implica descartar el sistema actual y desarrollar uno nuevo desde cero. Sin embargo también es la más costosa y que trae más riesgos, especialmente si el sistema original carece de documentación o su lógica de negocio crítica es difícil de replicar con precisión [48].
- **Sustitución:** Se da cuando el sistema es reemplazado por una solución comercial ya existente (como un paquete ERP o CRM). Es la opción mas viable cuando no se justifica mantener o reescribir una solución propia , aunque a menudo se requiere que la organización adapte sus procesos internos al software nuevo [49]

7.5. Inteligencia artificial

La inteligencia artificial se define como una disciplina que se enfoca en el desarrollo de sistemas capaces de ejecutar tareas que tradicionalmente han requerido el uso de la inteligencia humana, tales como el razonamiento, la percepción, el aprendizaje

o la toma de decisiones [50]. En el contexto específico de la ingeniería de software, el uso de la IA ha evolucionado significativamente, pasando de los sistemas expertos tradicionales y la lógica difusa a enfoques contemporáneos basados en aprendizaje automático (machine learning) y los modelos de gran escala (LLMs) [**?**].

Desde una perceptiva técnica, la IA abarca varias ramas fundamentales:

- **Machine learning (ML):** Consiste en un conjunto de técnicas que permiten a los sistemas aprender de los datos, sin una programación explicita para cada escenario. Abarca algoritmos de aprendizaje supervisado, no supervisado y por refuerzo [51].

- **Deep learning (DL):** Es una subcampo avanzado del ML que utiliza redes neuronales para resolver tareas complejas de clasificación, predicción y crucialmente generacion de contenido [51].

- **Modelos de lenguaje de gran escala (LLMs):** Son sistemas masivamente entrenados con grandes volúmenes de texto y código. Su función es generar, completar o traducir tanto el lenguaje natural como el código [52].

- **Sistemas multiagente SMA**: Son sistemas con arquitecturas distribuidas formadas por agentes inteligentes. Su valor reside en su capacidad para cooperar, coordinarse y negociar mediante la descomposición en agentes autónomos, permitiendo la paralización y especialización de tareas en sistemas de gran escala [11, 53].

Estos avances han posicionado a la IA no solo como una herramienta de automatización, sino como un mecanismo para amplificar las capacidades humanas. Hablando específicamente de la ingeniería de software, la IA se utiliza cada vez más para mitigar la complejidad descriptiva del código y del software, facilitando la comprensión de sistemas robustos y frecuentemente deficientes en documentación [6, 54]

Desde una perspectiva teórica, el creciente uso de IA en el desarrollo de software se logra comprender a través del marco AI-Augmented Software Engineering (AI4SE), donde los sistemas de IA actúan como amplificadores de capacidades cognitivas humanas, trascendiendo el rol unicamente de ejecutores automatizados [55]. Esta distinción es clave: mientras la automatización tradicional replica procesos bien definidos, los enfoques basados en IA generan artefactos a partir de patrones aprendidos de grandes fuentes de conocimiento [55].

Con todo esto se ha encontrado en la modernización de sistemas heredados un campo fértil de aplicación de la IA. Dado que muchos sistemas carecen de documentación actualizada, presentan complejidad estructural y fueron desarrollados en tecnologías obsoletas, la compresión y transformación manual se vuelve costosa y riesgosa [56].

Desde un punto de vista teórico, la IA especialmente los LLMs ofrecen una solución a estas barreras cognitivas al ser capaces de:

- **Aprender representaciones semánticas del código:** A partir de grandes volúmenes de ejemplos, lo que permite generar traducciones, sugerencias y reestructuraciones que conservan la lógica de negocio original [57].

- **Inferir modelos de arquitectura:** Mediante ingeniería inversa asistida por IA, es posible recuperar diagramas de componentes, modelos de dominio o flujos lógicos, facilitando la planificación de la modernización [58].
- **Optimizar decisiones de transformación:** A través del uso de ML para clasificar componentes críticos, predecir riesgos o priorizar fases del proceso [59].

8. Antecedentes

8.1. Inteligencia artificial en la ingeniería de software

Los diferentes estudios empíricos de la revisión sistemática de la literatura muestran que la inteligencia artificial (IA) está transformando de forma significativa la ingeniería de software. Lo que antes era un campo impulsado por procesos manuales [1], ahora está integrando mecanismos de automatización, asistencia inteligente y análisis avanzado en prácticamente todas las etapas del ciclo de vida del desarrollo de software [**?**].

Entre los principales casos de uso identificados se encuentran:

- **Ingeniería de requisitos y gestión del conocimiento tácito:** Los LLMs han demostrado capacidad para analizar, clasificar y generar requerimientos funcionales y no funcionales a partir de especificaciones en lenguaje natural [29, 57, 60].
- **Analisis y generación de código:** La efectividad de modelos como GPT, LlaMa, Claude o Copilot en la generación de código ha sido validada empíricamente en múltiples lenguajes de programación (Go, Java, Python, etc.) [7, 10, 28, 61, 62].
- **Aseguramiento de calidad y mantenimiento:** La aplicación de la IA se ha validado en la generación automática de pruebas, la detección de vulnerabilidades y la corrección de defectos es un área de rápido crecimiento [9,30,62–70].
- **Gestión de proyectos y metodologías ágiles:** En entornos ágiles, la IA ha sido integrada en flujos de planificación, facilitando el seguimiento de tareas y la toma de decisiones basada en el historial organizacional [71, 72].

8.2. Inteligencia artificial aplicada a la modernización de sistemas heredados

El análisis sistemático de la literatura revela que la IA se ha explorado también para resolver problemas específicos en cada etapa del ciclo de modernización, desde el análisis de código fuente hasta la transformación de la arquitectura [4, 19, 73, 74], usando para ellos distintas técnicas de IA [56].

8.2.1. Aplicación de modelos de lenguaje de gran escala (LLMs)

Los LLMs han emergido cómo una tecnología prometedora para este campo. Entre las técnicas de modernización que implementan LLMs en la literatura encontramos:

- **Traducción y preservación semántica:** La aplicación de LLMs a la traducción entre lenguajes de programación es una extensión natural de sus habilidades lingüísticas [57, 75]. Estudios como el trabajo de transformación de Fortran a Python demuestran que estos modelos no solo realizan la traducción sintáctica, sino que generan código idiomático en el lenguaje destino, a menudo complementado con artefactos de calidad como pruebas unitarias [19, 74–80].
- **Ingeniería inversa asistida y recuperación de conocimiento:** La capacidad de los LLMs para extraer modelos de arquitectura del código heredado representa una innovación conceptual significativa. Investigaciones en el área de la ingeniería inversa dirigida por modelos (Model-Driven Reverse Engineering) demuestran que estos sistemas pueden inferir abstracciones de arquitectura (como modelos de dominio o diagramas de componentes) a partir de la implementación y codigo fuente [58, 72, 76, 80–83].

8.2.2. Enfoques híbridos y optimización del proceso

La literatura también documenta el valor de otros enfoques de inteligencia artificial que abordan problemas específicos de comprensión y priorización.

- **Clasificación y priorización con Machine Learning (ML):** Los algoritmos de ML tradicionales se fundamentan en la teoría de aprendizaje supervisado para clasificar o predecir características del software. Esto implementa una optimización guiada por datos, donde el ML identifica los componentes de alto riesgo o alto valor, permitiendo una priorización estratégica de los esfuerzos en lugar de un análisis manual exhaustivo [6, 20, 59].
- **Arquitecturas multiagente:** Una tendencia avanzada es la combinación de IA con patrones de arquitectura modernos, proponiendo sistemas multiagente. Aquí, agentes inteligentes especializados se encargan de la identificación, extracción y encapsulamiento de funcionalidades críticas [11, 26, 27].

9. Resultados

9.1. Ciclo 1 (Mapeo sistemático de literatura)

El primer ciclo de esta investigación consistió en llevar a cabo el mapeo sistemático de la literatura. A continuación se presentan cada una de las definiciones generadas en la etapa de planeación y que fueron usadas para la ejecución del mapeo:

Objetivos de investigación

OI1. Identificar y analizar las aplicaciones actuales de técnicas de inteligencia artificial en los procesos de modernización de sistemas heredados, con el fin de establecer un panorama amplio de herramientas, enfoques y capacidades, y su alineación con los desafíos comunes en este tipo de procesos.

OI2. Analizar y categorizar las metodologías y estrategias utilizadas en la modernización de sistemas heredados tradicionales, identificando factores críticos de éxito, patrones de fracaso y mejores practicas documentadas en la literatura, con el fin de establecer un marco comparativo

OI3. Identificar y clasificar los principales usos de la inteligencia artificial en la asistencia en el ciclo de vida del desarrollo de software, con el fin de establecer un marco de referencia sobre estas tecnologías.

Preguntas de investigación

PI1. ¿Qué técnicas, herramientas y frameworks basados en inteligencia artificial se están utilizando actualmente para abordar los desafíos técnicos, organizacionales y de calidad en la modernización de sistemas heredados?

PI2. ¿Que metodologías de modernización de sistemas heredados han demostrado mayor efectividad en términos de tiempo, costos y calidad? ¿cuales son sus factores críticos de éxito y fracaso?

PI3. ¿Cuales son los principales casos de uso de la inteligencia artificial en el desarrollo de software y cómo esta se ha integrado en las distintas etapas del ciclo de vida del software?

Alcance y estrategia de búsqueda

Alcance temporal

Para el mapeo sistemático de la literatura se definieron dos rangos temporales según las preguntas investigación. Para la PI1 y PI3 relacionadas con IA, se estableció un rango de 5 años (2019 - 2025) considerando que esta es una tecnología emergente y con muy rápida evolución por lo que se planteó analizar información actualizada.

Por otro lado, para la PI2 relacionada con las metodologías de modernización de sistemas heredados, se estableció un rango de 10 años (2014 -2025) debido a que

el tema tiene mayor trayectoria histórica, dando la oportunidad así para identificar enfoques metodológicos y validados, documentar lecciones aprendidas y analizar su efectividad en diferentes contextos.

Bases de datos

La selección de bases de datos se realizo considerando la necesidad de obtener literatura altamente especializada y relevante que pudieran dar respuesta a las preguntas de investigación planteadas. Por esto se seleccionaron las siguientes bases de datos:

- **Primarias**
  - IEEE Xplore
  - ACM Digital Library
  - Scopus
  - Science Direct
  - Web of Science
- **Secundarias**
  - arXiv
  - Google Scholar
  - ResearchGate
  - Citaciones relevantes

Criterios de inclusión (CI) y exclusión (CE)

En las Tablas 1 y 2 se definen los criterios de inclusión y exclusión (CE) usados para el mapeo sistemático de la literatura respectivamente.

Estrategia de búsqueda y formulación PICO

Para garantizar una búsqueda exhaustiva y sistemática, se empleó la estructura PICO (Población, Intervención, Comparación, Resultado), adaptada al contexto del desarrollo de software y la inteligencia artificial [33]. La Tabla 3 presenta la descomposición conceptual utilizada para cada pregunta de investigación.

Ecuaciones de búsqueda

Las ecuaciones de búsqueda se construyeron a partir del análisis PICO realizado para cada pregunta de investigación, considerando sinónimos, términos relacionados y operadores booleanos. A continuación, se presentan las ecuaciones principales empleadas:

PI1. Aplicaciones de IA en la modernización de sistemas heredados

Id	Descripción	Motivación
CI1	Estudios que incluyan en título, resumen o palabras clave términos relacionados con inteligencia artificial, modernización de sistemas heredados o desarrollo de software.	Garantizar que los estudios seleccionados estén alineados con el alcance de la investigación y puedan contribuir efectivamente a responder las preguntas planteadas. Este primer filtro es crucial para asegurar la relevancia inicial de los estudios.
CI2	Estudios publicados en inglés o español.	El inglés es el idioma predominante en publicaciones científicas de tecnología e ingeniería de software, permitiendo acceder a una base más amplia de investigación internacional. El español se incluye para capturar experiencias relevantes en el contexto latinoamericano y casos de estudio regionales.
CI3	Para estudios sobre LLMs (PI1, PI3): publicaciones entre 2019–2024. Para estudios sobre modernización (PI2): publicaciones entre 2014–2024.	Los rangos temporales diferenciados permiten equilibrar la necesidad de actualidad en el campo emergente de los LLMs con la captura de metodologías y prácticas maduras en modernización de sistemas. Esto asegura una cobertura temporal apropiada para cada aspecto de la investigación.
CI4	Estudios que describan, propongan o evalúen aplicaciones de técnicas de inteligencia artificial (LLMs, ML, DL, sistemas expertos, RPA, etc.) en el ciclo de desarrollo de software o en procesos de modernización.	Asegurar que los estudios seleccionados proporcionen información práctica y aplicable sobre el uso de la IA y metodologías de modernización. Esto facilita la identificación de patrones, mejores prácticas y lecciones aprendidas en ambos campos.
CI5	Estudios con evidencia empírica (casos de estudio, prototipos, validaciones, experimentos, etc.).	La inclusión de estudios con validación práctica permite evaluar la efectividad real de las soluciones propuestas y su aplicabilidad en contextos del mundo real. Esto es crucial para identificar factores de éxito y fracaso.
CI6	Estudios que presenten contribuciones primarias y originales.	Asegurar la recopilación de contribuciones originales que aporten nuevo conocimiento en el campo de estudio.

Tabla 1: Criterios de inclusión utilizados en el mapeo sistemático

Id	Descripción	Motivación
CE1	Estudios que solo mencionan IA o modernización de forma tangencial, sin aportar aplicaciones prácticas o análisis detallado.	Evitar la inclusión de estudios superficiales que no aportan conocimiento sustancial sobre la aplicación práctica de la IA o metodologías de modernización. Esto ayuda a mantener el enfoque en contribuciones significativas al campo.
CE2	Publicaciones en formato no académico o de baja extensión (posters, abstracts extendidos, presentaciones sin paper completo).	Los formatos reducidos generalmente no proporcionan el nivel de detalle necesario para una evaluación completa. Se buscan publicaciones completas que permitan un análisis exhaustivo de la metodología y resultados.
CE3	Literatura gris sin respaldo institucional, académico o industrial.	Mantener la calidad y credibilidad de las fuentes incluidas en el estudio. La literatura sin respaldo formal puede contener información no verificada o sesgada que podría afectar la validez de las conclusiones.
CE4	Estudios duplicados o versiones preliminares de trabajos ya incluidos.	Evitar la redundancia en el análisis y asegurar que solo se considere la versión más completa y actualizada de cada investigación. Esto mejora la eficiencia del análisis y previene el sesgo por sobre-representación.
CE5	Estudios secundarios y terciarios como: revisiones sistemáticas de literatura, mapeos sistemáticos previos, meta-análisis, etc.	La exclusión de estudios secundarios y terciarios evita la duplicación en el análisis y ayuda a mantener el foco en las contribuciones originales y primarias. Si bien estos estudios pueden ser valiosos como referencia y para contrastar resultados, su inclusión podría distorsionar el mapa del estado actual de la investigación primaria en el campo.

Tabla 2: Criterios de exclusión utilizados en el mapeo sistemático

PI	Población	Intervención	Resultado esperado
PI1	Sistemas heredados	Técnicas de IA (LLMs, ML, etc.)	Soluciones a desafíos técnicos/organizacionales
PI2	Procesos de modernización	Metodologías o estrategias	Mejores prácticas, factores críticos
PI3	Ciclo de vida del software	Uso de IA	Casos de uso, herramientas, automatización

Tabla 3: Descomposición PICO para cada pregunta de investigación

```
(("Legacy*" OR "legacy system*" OR "legacy code" OR "
   mainframe" OR "legacy application*" OR "legacy software"
   OR "legacy architecture") AND ("large language model" OR "
   LLM" OR "GPT" OR "ChatGPT" OR "artificial intelligence" OR
    "machine learning" OR "deep learning" OR "AI") AND ("
   modernization" OR "migration" OR "refactor" OR "
   transformation" OR "reverse engineering"))
```

PI2. Metodologías de modernización de sistemas heredados

```
("legacy system*" OR "legacy code" OR "legacy application*"
   OR "legacy software" OR "legacy architecture" OR "legacy
   platform*" OR "mainframe" OR "monolithic system*" OR "
   obsolete system*" OR "aging system*") AND ("moderniz*" OR
   "migration" OR "transform*" OR "replatform*" OR "rehost*"
   OR "refactor*" OR "reengine*" OR "rewrite" OR "repackag*"
   OR "reverse engineer*" OR "restructur*" OR "renewal" OR "
   upgrad*" OR "evolut*") AND ("methodolog*" OR "strateg*" OR
    "approach*" OR "framework*" OR "process*" OR "technique*"
    OR "method*" OR "pattern*" OR "practice*" OR "roadmap" OR
    "guideline*")
```

PI3. Usos de IA en el ciclo de vida del desarrollo de software

```
(("large language model*" OR "LLM*" OR "foundation model*" OR
    "language model*" OR "transformer model*" OR "GPT" OR "
   ChatGPT" OR "Codex" OR "Copilot")  AND ("software
   development" OR "software engineering" OR "programming" OR
    "software life cycle" OR "SDLC")  AND ("application*" OR
   "implementation*" OR "integration" OR "automation" OR "
   tool*" OR "support" OR "enhancement"))
```

Resultados etapa de ejecución

Como resultado de este proceso, se identificaron inicialmente 96 estudios que cumplían con los criterios de inclusión. Tras aplicar el primer filtro de eliminación de duplicados, el conjunto se redujo a 82 artículos únicos. posteriormente se aplicaron los criterios de exclusión, que permitieron descartar algunos estudios. El conjunto resultante fue de 51 artículos, clasificados de la siguiente manera:

- PI1 – Aplicaciones de IA en modernización: 18 artículos
- PI2 – Metodologías de modernización tradicionales: 9 artículos
- PI3 – Casos de uso de IA en el ciclo de vida del software: 24 artículos

Resultados etapas de observación

Durante la etapa de observación se llevó a cabo el análisis sistemático de los estudios primarios seleccionados durante el mapeo, con el fin de dar respuesta a las

preguntas de investigación (PI1, PI2, PI3) definidas en la fase de planeación.

PI1. Aplicaciones de IA en la modernización de sistemas heredados

El conjunto de estudios seleccionados para PI1 incluyó 18 artículos, en los cuales se lograron identificar distintas estrategias de modernización, destancando:

- Traducción automática de código.
- Refactorización automatizada.
- Reversión de ingeniería basada en modelos (MDRE)
- Reconstrucción de documentación técnica
- Análisis semántico y extracción de conocimiento oculto en código heredado.
- Apoyo en tareas de análisis y mantenimiento

El análisis cualitativo permitió identificar una clara predominancia del uso de modelos de lenguaje de gran escala (LLMs) como GPT-4, Codex y Copilot. Sin embargo, no se limitó a estos, como se observa en la Tabla 4.

Tipo de IA aplicada	**Número de artículos**
Large Language Models (LLMs)	8
Machine Learning tradicional	4
Deep Learning	3
IA simbólica / heurística	2
Híbrida (combinación de enfoques)	1

Tabla 4: Distribución de los artículos según el tipo de IA aplicada.

Entre las herramientas y frameworks documentados se destacan CARGO, gPROFIT, CodeTrans, FortranAnalyser y GPT-Code-Search. Estas herramientas abordan tanto desafíos técnicos como organizacionales, entre ellos deuda técnica acumulada, obsolescencia tecnológica, y pérdida de conocimiento por rotación de personal.

Entre las técnicas mas utilizadas de IA, destacan:

- Transformers y LLMs fine-tuned.
- Clasificación supervisada (MLP, redes neuronales).
- Aprendizaje no supervisado (clustering).
- Análisis estático de código.
- Ingeniería de características para extracción de conocimiento.
- Sistemas de reglas y análisis semántico.

Esto sugiere que la elección de la técnica de IA depende del objetivo puntual del artículo: por ejemplo, los trabajos enfocados en generación de documentación tienden a usar LLMs, mientras que los centrados en ingeniería inversa aplican análisis estructurales más tradicionales o técnicas simbólicas.

A nivel cuantitativo, los estudios se concentran entre 2021 y 2024, lo que refleja un interés creciente y actual en el uso de LLMs para tareas de modernización de sistemas heredados. Ademas, el análisis permitió clasificar los enfoques observados en el marco teórico propuesto.

PI2. Metodologías de modernización de sistemas heredados

Para responder a la PI2, se analizaron 9 estudios primarios que presentan propuestas metodológicas, estudios de caso o evaluaciones empíricas de técnicas tradicionales de modernización. El objetivo fue identificar los enfoques más comunes utilizados en la práctica para transformar sistemas heredados, estos enfoques ya fueron expuestos en el desarrollo del marco teórico y clasificados desde mayor a menor grado de intervención de la siguiente forma:

- Intervención mínima y adaptación externa: Encapsulamiento y Rehosting (Lift-and-shift)
- Modificación Interna y estructural: Refactorización y Reingenieria
- Reemplazo completo: Reescritura y Sustitución

PI3. Usos de IA en el ciclo de vida del desarrollo de software

Los estudios vinculados a la PI3 sumaron 24 artículos, en los cuales los casos de uso documentados se distribuyen en prácticamente todas las etapas del ciclo de vida del software, como se evidencia en la Tabla 5

Etapa del ciclo de vida	**Frecuencia**
Desarrollo de software	10
Mantenimiento / Refactorización	4
Documentación / Requisitos	3
Seguridad / Pruebas / QA	4
Gestión de proyectos / Agile	2
Gestión de riesgos (FMEA)	1

Tabla 5: Distribución de artículos según la etapa del ciclo de vida abordada.

Las herramientas más mencionadas incluyen GPT-3.5, GPT-4, Gemini, Alpaca, CodeBERT, y CodeT5, lo que muestra alta dependencia de modelos fundacionales para tareas cognitivas.

A nivel experimental, se reportan métricas como F1-score, precisión, recall, BLEU, MOS, throughput, y métricas cualitativas como la satisfacción del usuario o la facilidad de integración. Muchos estudios combinaron evaluaciones automáticas con validaciones por expertos humanos.

Ademas, el análisis permitió clasificar los enfoques observados según el marco teórico propuesto.

9.2. Ciclo 2 (Armonización de múltiples modelos)

El segundo ciclo de la investigación estuvo centrado en la construcción de un marco de referencia unificado para la modernización de sistemas heredados, integrando los resultados del mapeo sistemático con base en el framework de armonización de múltiples modelos propuesto por Pardo et al [36]. Este proceso permitió extraer, comparar, estructurar e integrar un conjunto de practicas y conceptos relevantes provenientes de los distintos enfoques, estándares, herramientas y metodologías, con el objetivo de sentar las bases para el diseño de una arquitectura de modernización de sistemas heredados usando IA.

Resultados etapa de planeación

Durante esta fase se aplicó el método de identificación, con el propósito de reconocer y organizar los elementos clave presentes en los estudios primarios seleccionados en el mapeo sistemático. Como resultado, se identificaron:

- **36 elementos clave**, organizados en tres categorías principales:
  - **Prácticas de modernización asistidas por IA**, incluyendo traducción automática de código, refactorización semántica, análisis arquitectónico asistido por modelos de lenguaje (LLMs), recomendación automatizada de pruebas y generación de documentación técnica.
  - **Modelos y metodologías** aplicados a procesos de migración y reingeniería, tales como ADM (Architecture-Driven Modernization), MDA (Model-Driven Architecture), modernización incremental y pipelines de refactorización automatizada.
  - **Herramientas y frameworks especializados**, tales como CodeTrans, GPT-Code-Search, CARGO, Copilot, FortranAnalyser, entre otros.
- A nivel cuantitativo:
  - El **61 % de los estudios** (31 de 51) hace referencia explícita al uso de modelos de lenguaje de gran escala (LLMs) en tareas relacionadas con la modernización y el mantenimiento de software.
  - El **35 %** (18 de 51) incorpora herramientas concretas como parte de las soluciones analizadas, lo que evidencia una orientación práctica y aplicada del campo.
  - El **57 %** (29 de 51) describe metodologías estructuradas o enfoques sistemáticos, lo cual subraya la necesidad de marcos de trabajo formales para enfrentar los desafíos de la modernización.

Este proceso permitió construir una base terminológica y conceptual homogénea, facilitando la trazabilidad entre prácticas, herramientas y estudios.

Resultados etapa observación

En esta etapa se implementó el método de integración, cuyo objetivo fue consolidar un marco estructurado, genérico y reutilizable que sirviera como base para el diseño posterior del modelos de referencia. Los resultados principales fueron:

- La definición de un modelo de procesos, compuesto por siete actividades fundamentales en un proceso de modernización asistido y que pueden simplificarse a capacidades cognitivas de los agentes:

  1. Análisis del sistema heredado asistido por LLMs
  2. Generación de modelos de requerimientos y arquitectónicos
  3. Clasificación y selección de estrategias de modernización
  4. Transformación asistida por IA (traducción semántica, refactorización, análisis sintáctico)
  5. Validación automática y evolución de calidad
  6. Generación y documentación del conocimiento
  7. Monitoreo y trazabilidad del sistema modernizado

- El modelo fue diseñado para ser agnóstico respecto a herramientas específicas, priorizando la reutilización de conceptos, la escalabilidad y la adaptabilidad tecnológica frente a distintos contextos organizacionales.

Resultados etapa de reflexión

Como parte del proceso de reflexión de la armonización de múltiples modelos, se llevó a cabo un análisis de correspondencia entre los agentes definidos para el modelo de referencia y los estudios identificados durante el mapeo sistemático de la literatura. Esto con el fin de establecer el grado de soporte empírico y conceptual que sustenta la inclusión de cada agente. Este análisis permitió validar la alineación de los agentes propuestos con las tendencias, capacidades y desafíos documentados en la literatura especializada.

Para ello, se establecieron correspondencias entre las funcionalidades específicas de cada agente y los hallazgos obtenidos en las tres dimensiones del mapeo:

- PI1: Técnicas y frameworks basados en inteligencia artificial aplicados a la modernización.
- PI2: Metodologías de modernización tradicionales, factores críticos de éxito y patrones de fracaso.
- PI3: Casos de uso y capacidades de los modelos de lenguaje (LLMs) en el ciclo de vida del desarrollo de software.

En la Tabla 6 se presentan los vínculos, evidenciado que cada agente del modelo cuenta con un respaldo documentado que justifica su inclusion. Si bien algunos agentes presentan mayor densidad bibliográfica (e.g., AASH, ATC, AVV), aquellos con menor número de referencias (e.g., AMA, AGC) corresponden a áreas emergentes donde la literatura aún es incipiente, lo cual sugiere líneas de investigación futura.

9.3. Ciclo 3 (Diseño de arquitectura)

Definición de principios arquitectónicos

La arquitectura se fundamenta en cinco pilares que garantizan la precisión, la resiliencia y la evolución continua del proceso de modernización.

1. **Modularidad contractual y especialización cognitiva** El sistema está diseñado como una red de agentes altamente especializados que trabajan juntos.
    - **Especialización clara:** Cada agente tiene una capacidad cognitiva única (analizar, transformar, validar o diseñar). Esto garantiza que la cohesión se base en la especialización de su conocimiento y no en artefactos tecnológicos, lo que facilita el mantenimiento y la evolución independiente.
    - **Contratos semánticos:** La comunicación no es solo técnica, sino también contractual. Los agentes intercambian datos y resultados de acuerdo con pre-condiciones claras. Además, las dependencias entre ellos son siempre unidireccionales.
2. **Control centralizado y resiliencia** El Agente Orquestador (AO) garantiza que el proceso sea coherente, auditable y capaz de recuperarse ante fallos.
    - **Orquestación híbrida:** El AO no ejecuta tareas, sino que verifica los contratos y coordina el flujo. Su modelo de control es proactivo (secuencia planificada) y reactivo (responde a fallos y bloqueos).
    - **Diseño a prueba de fallos:** La resiliencia es obligatoria. Todos los agentes deben ser idempotentes (capaces de re-ejecutarse sin efectos secundarios) y el sistema debe mantener la persistencia de los estados, para poder recuperarse y reanudar el proceso exactamente donde falló.
    - **Escalada explícita al humano:** Se deben establecer puntos de decisión claros para involucrar a expertos humanos. Esto debe ocurrir solo cuando hay ambigüedad semántica (múltiples interpretaciones), inconsistencias entre agentes, o cuando el riesgo operacional es alto.
3. **Trazabilidad estratificada y explicabilidad Justificada** Cada paso de la modernización debe ser transparente, auditable y justificado, especialmente las decisiones asistidas por IA.
    - **Niveles de trazabilidad:** La intensidad del registro de la información auditable es proporcional a la criticidad de la decisión:
        - **Alta fidelidad:** Para decisiones estratégicas (Arquitectura, Estrategia de migración).
        - **Media Fidelidad:** Para la ejecución táctica (Transformación de código, Generación de tests).
    - **Explicabilidad:** Toda decisión que amerite tener una traza debe incluir su contexto, justificación y las distintas alternativas existentes.

- **Auditabilidad y reconstrucción:** Los registros son inmutables y lo suficientemente detallados para que un tercero pueda reconstruir y verificar el proceso de modernización completo desde cero.

4. **Gestión del conocimiento** El sistema debe estar diseñado para aprender de cada modernización, mejorando su rendimiento con cada proyecto.

5. **Mejora continua** La modernización no es un evento único, sino un ciclo de vida gestionado que prioriza la estabilidad.

Definición restricciones de diseño

Se identificaron diversas restricciones derivadas tanto del análisis previo (armonización de modelos) como del contexto práctico del sistema:

- **Restricción técnica:** El sistema debe operar sobre sistemas heredados que utilizan tecnologías obsoletas (e.g., COBOL, Fortran), lo que impone limitaciones sobre el tipo de análisis automatizable por IA.
- **Restricción de infraestructura:** Algunos entornos de despliegue son on-premise y no permiten conexión directa con servicios LLM en la nube, lo que obligó a diseñar una arquitectura desacoplada de la plataforma y habilitar modelos locales donde sea necesario.
- **Restricción organizacional:** No todos los roles involucrados (como el Líder de modernización) tienen conocimiento técnico profundo, por lo que la interfaz del sistema debe abstraer la complejidad interna y permitir una interacción basada en validaciones y aprobaciones visuales.
- **Restricción regulativa:** En proyectos que afectan sectores regulados (e.g., financiero o salud), el sistema debe registrar y justificar todas las transformaciones y decisiones mediante artefactos trazables.

Definición de requisitos funcionales y no funcionales

Requisitos funcionales

Los requisitos funcionales definen las capacidades específicas que debe proporcionar cada agente del sistema de modernización.

Requisitos del Agente Orquestador (AO)

- **RF-AO-01:** El sistema debe coordinar la ejecución secuencial y paralela de agentes según las dependencias definidas en el pipeline de modernización.
- **RF-AO-02:** El sistema debe validar el cumplimiento de pre-condiciones antes de activar cada agente y verificar post-condiciones al finalizar su ejecución.
- **RF-AO-03:** El sistema debe gestionar fallos mediante reintentos automáticos y re-planificación dinámica del flujo de trabajo.

- **RF-AO-04:** El sistema debe escalar decisiones a expertos humanos cuando detecte ambigüedad semántica, inconsistencias entre agentes o alto riesgo operacional.
- **RF-AO-05:** El sistema debe mantener un registro detallado e inmutable de trazabilidad de todo el proceso de modernización.
- **RF-AO-06:** El sistema debe monitorear el estado, progreso, fallos y métricas de cada agente en tiempo real.
- **RF-AO-07:** El sistema debe permitir la reanudación del proceso de modernización desde el último punto de fallo exitoso.

Agente de Análisis del Sistema Heredado (AASH)

- **RF-AASH-01:** El sistema debe analizar código fuente en tecnologías heredadas (COBOL, Fortran, entre otras).
- **RF-AASH-02:** El sistema debe extraer y documentar la lógica de negocio embebida en el código heredado.
- **RF-AASH-03:** El sistema debe generar diagramas UML y de componentes del sistema heredado.
- **RF-AASH-04:** El sistema debe identificar y mapear dependencias técnicas entre componentes.
- **RF-AASH-05:** El sistema debe extraer estructuras de datos y modelos de dominio.
- **RF-AASH-06:** El sistema debe identificar patrones arquitectónicos existentes en el sistema heredado.
- **RF-AASH-07:** El sistema debe detectar código problemático, antipatrones y áreas de alto riesgo.
- **RF-AASH-08:** El sistema debe enriquecer el código fuente con anotaciones y metadatos de análisis.

Agente de Análisis de Dependencias y Deuda Técnica (ADDT)

- **RF-ADDT-01:** El sistema debe identificar y cuantificar la deuda técnica del sistema heredado.
- **RF-ADDT-02:** El sistema debe generar un mapa de dependencias enriquecido con métricas de acoplamiento.
- **RF-ADDT-03:** El sistema debe clasificar la deuda técnica por categorías (código obsoleto, redundancia, complejidad ciclomática, etc.).
- **RF-ADDT-04:** El sistema debe realizar análisis de riesgo por componente considerando impacto y probabilidad.

- **RF-ADDT-05:** El sistema debe priorizar módulos para modernización según impacto, urgencia y costo.
- **RF-ADDT-06:** El sistema debe generar recomendaciones de refactorización justificadas técnicamente.

Agente de Gestión de Requisitos (AGR)

- **RF-AGR-01:** El sistema debe derivar requisitos funcionales a partir del análisis técnico del código heredado.
- **RF-AGR-02:** El sistema debe derivar requisitos no funcionales considerando restricciones técnicas y de negocio.
- **RF-AGR-03:** El sistema debe generar una matriz de trazabilidad entre requisitos y componentes del sistema heredado.
- **RF-AGR-04:** El sistema debe permitir la validación interactiva de requisitos con stakeholders.
- **RF-AGR-05:** El sistema debe generar historias de usuario con criterios de aceptación.
- **RF-AGR-06:** El sistema debe construir una ontología de dominio del sistema.
- **RF-AGR-07:** El sistema debe identificar y documentar restricciones técnicas y de negocio.

Agente de Estrategia de Migración (AEM)

- **RF-AEM-01:** El sistema debe evaluar múltiples estrategias de modernización (*rehosting*, *replatforming*, *refactoring*, *rearchitecting*, *rebuilding*, *replacing*).
- **RF-AEM-02:** El sistema debe seleccionar la estrategia óptima por componente considerando complejidad, costo y prioridades.
- **RF-AEM-03:** El sistema debe generar un plan de migración faseado con hitos claros y dependencias.
- **RF-AEM-04:** El sistema debe realizar análisis de *trade-offs* entre diferentes estrategias de migración.
- **RF-AEM-05:** El sistema debe identificar y documentar riesgos asociados a cada fase de migración.
- **RF-AEM-06:** El sistema debe generar cronogramas estimados por módulo y fase.

Agente de Definición Arquitectónica (ADA)

- **RF-ADA-01:** El sistema debe diseñar la arquitectura objetivo del sistema modernizado.

- **RF-ADA-02:** El sistema debe generar especificaciones de componentes, APIs, bases de datos y protocolos de comunicación.
- **RF-ADA-03:** El sistema debe producir diagramas arquitectónicos (modelo C4, UML, ERD).
- **RF-ADA-04:** El sistema debe definir guías de diseño y lineamientos tecnológicos.
- **RF-ADA-05:** El sistema debe asegurar que la arquitectura propuesta sea escalable, mantenible y cumpla con estándares modernos.
- **RF-ADA-06:** El sistema debe documentar decisiones arquitectónicas con justificación técnica.

Agente de Transformación de Código (ATC)

- **RF-ATC-01:** El sistema debe transformar código heredado al lenguaje y framework objetivo.
- **RF-ATC-02:** El sistema debe preservar la semántica y funcionalidad del código original durante la transformación.
- **RF-ATC-03:** El sistema debe generar código que cumpla con los estándares y lineamientos definidos por ADA.
- **RF-ATC-04:** El sistema debe mantener trazabilidad entre código original y código transformado a nivel de línea/función.
- **RF-ATC-05:** El sistema debe documentar decisiones de implementación y cambios realizados.
- **RF-ATC-06:** El sistema debe aplicar las estrategias de migración definidas por AEM durante la transformación.

Agente de Verificación y Validación (AVV)

- **RF-AVV-01:** El sistema debe generar conjuntos de pruebas automatizadas (unitarias, integración, regresión).
- **RF-AVV-02:** El sistema debe ejecutar pruebas y generar reportes de resultados con métricas de cobertura.
- **RF-AVV-03:** El sistema debe detectar desviaciones funcionales entre sistema heredado y modernizado.
- **RF-AVV-04:** El sistema debe identificar casos que requieren validación humana adicional.
- **RF-AVV-05:** El sistema debe calcular métricas de confiabilidad y calidad del código transformado.

- **RF-AVV-06:** El sistema debe validar el cumplimiento de criterios de aceptación definidos por AGR.

Agente de Despliegue (ADD)

- **RF-ADD-01:** El sistema debe automatizar el despliegue en entornos de staging y producción.
- **RF-ADD-02:** El sistema debe implementar estrategias de despliegue progresivo (blue-green, canary,).
- **RF-ADD-03:** El sistema debe generar configuraciones de monitoreo y alertas para el sistema desplegado.
- **RF-ADD-04:** El sistema debe documentar procedimientos de *rollback* automático.
- **RF-ADD-05:** El sistema debe generar logs completos de despliegue con trazabilidad.
- **RF-ADD-06:** El sistema debe validar el ambiente objetivo antes del despliegue.

Agente de Monitoreo y Adaptación (AMA)

- **RF-AMA-01:** El sistema debe recolectar métricas de rendimiento en tiempo real (latencia, *throughput*, errores).
- **RF-AMA-02:** El sistema debe analizar logs y trazas para detectar anomalías.
- **RF-AMA-03:** El sistema debe generar reportes de rendimiento y confiabilidad.
- **RF-AMA-04:** El sistema debe realizar análisis predictivo de tendencias y problemas potenciales.
- **RF-AMA-05:** El sistema debe generar recomendaciones de optimización continua.
- **RF-AMA-06:** El sistema debe activar *triggers* para iteraciones de mejora cuando se detecten oportunidades.

Agente de Gestión de Conocimiento (AGC)

- **RF-AGC-01:** El sistema debe mantener una base de conocimiento versionada de todos los proyectos de modernización.
- **RF-AGC-02:** El sistema debe almacenar conocimiento técnico (patrones, tecnologías, soluciones).
- **RF-AGC-03:** El sistema debe almacenar conocimiento de dominio (ontologías, lógica de negocio).

- **RF-AGC-04:** El sistema debe almacenar conocimiento procedimental (estrategias exitosas, lecciones aprendidas).
- **RF-AGC-05:** El sistema debe almacenar conocimiento contextual (decisiones, justificaciones, alternativas).
- **RF-AGC-06:** El sistema debe proporcionar capacidades de búsqueda y recuperación (RAG) para todos los agentes.
- **RF-AGC-07:** El sistema debe actualizar automáticamente la base de conocimiento con cada proyecto completado.

Interacción Humana

- **RF-IH-01:** El sistema debe proporcionar interfaces de validación para stakeholders no técnicos.
- **RF-IH-02:** El sistema debe permitir aprobaciones visuales de requisitos, arquitectura y estrategias.
- **RF-IH-03:** El sistema debe solicitar intervención humana en puntos de decisión críticos.
- **RF-IH-04:** El sistema debe abstraer la complejidad técnica en interfaces orientadas a decisiones de negocio.

Requisitos No Funcionales

Los requisitos no funcionales establecen las cualidades y restricciones del sistema que garantizan su operación efectiva, segura y sostenible.

Disponibilidad y Confiabilidad

- **RNF-DC-01:** El sistema debe ser capaz de recuperarse automáticamente ante fallos de agentes individuales sin pérdida de datos.
- **RNF-DC-02:** Todos los agentes deben ser idempotentes, permitiendo reejecución sin efectos secundarios.
- **RNF-DC-03:** El sistema debe mantener persistencia de estado para permitir recuperación exacta desde puntos de fallo.

Escalabilidad y Rendimiento

- **RNF-ER-01:** El sistema debe soportar la modernización de sistemas heredados de hasta 1 millón de líneas de código.
- **RNF-ER-02:** El análisis inicial (AASH) no debe tomar más de 24 horas para sistemas de hasta 100,000 líneas de código.
- **RNF-ER-03:** El sistema debe escalar horizontalmente para soportar múltiples proyectos de modernización simultáneos.

Seguridad

- **RNF-SEG-01:** El sistema debe cifrar en tránsito todas las comunicaciones entre agentes.
- **RNF-SEG-02:** El sistema debe cifrar en reposo toda la información sensible del sistema heredado.
- **RNF-SEG-03:** El sistema debe implementar control de acceso basado en roles (RBAC) para diferentes stakeholders.
- **RNF-SEG-04:** El sistema debe auditar todas las acciones de usuarios y agentes con marcas de tiempo y usuario responsable.
- **RNF-SEG-05:** El sistema debe cumplir con regulaciones aplicables al sector (financiero, salud, etc.).

Trazabilidad y Auditabilidad

- **RNF-TA-01:** El sistema debe mantener registros inmutables de todas las decisiones y transformaciones.
- **RNF-TA-02:** Los registros deben incluir contexto completo, justificación y alternativas consideradas.
- **RNF-TA-03:** El sistema debe permitir la reconstrucción completa del proceso de modernización por terceros auditores.
- **RNF-TA-04:** La trazabilidad debe ser estratificada: alta fidelidad para decisiones estratégicas, media para ejecución táctica.
- **RNF-TA-05:** El sistema debe mantener versionado completo de todos los artefactos generados.

Explicabilidad

- **RNF-EXP-01:** Todas las decisiones asistidas por IA deben incluir explicaciones en lenguaje natural.
- **RNF-EXP-02:** Las explicaciones deben incluir el razonamiento, evidencia utilizada y nivel de confianza.
- **RNF-EXP-03:** El sistema debe identificar explícitamente cuando una decisión requiere validación humana.
- **RNF-EXP-04:** Las recomendaciones deben incluir pros, contras y *trade-offs* de alternativas.

Mantenibilidad y Extensibilidad

- **RNF-ME-01:** El sistema debe seguir una arquitectura modular que permita la sustitución o actualización de agentes individuales.

- **RNF-ME-02:** Los contratos entre agentes deben estar formalmente documentados y versionados.
- **RNF-ME-03:** El sistema debe permitir la adición de nuevos agentes especializados sin modificar agentes existentes.
- **RNF-ME-04:** El código del sistema debe mantener una cobertura de pruebas mínima del 80 %.

Portabilidad e Interoperabilidad

- **RNF-PI-01:** El sistema debe soportar despliegue tanto en nube pública como en infraestructura *on-premise*.
- **RNF-PI-02:** El sistema debe permitir la utilización de modelos LLM en la nube o locales según restricciones de infraestructura.
- **RNF-PI-03:** El sistema debe integrarse con sistemas de control de versiones estándar (Git, SVN).
- **RNF-PI-04:** El sistema debe exportar resultados en formatos estándar de la industria (JSON, XML, Markdown, PDF).

Usabilidad

- **RNF-USA-01:** Las interfaces de validación deben ser comprensibles para usuarios sin conocimiento técnico profundo.
- **RNF-USA-02:** El sistema debe proporcionar dashboards visuales del progreso de modernización.
- **RNF-USA-03:** Los reportes generados deben incluir resúmenes ejecutivos y secciones técnicas detalladas.
- **RNF-USA-04:** El sistema debe proporcionar notificaciones proactivas cuando requiera intervención humana.

Aprendizaje y Mejora Continua

- **RNF-AMC-01:** El sistema debe mejorar su precisión con cada proyecto completado mediante aprendizaje continuo.
- **RNF-AMC-02:** La base de conocimiento debe actualizarse automáticamente con patrones exitosos identificados.
- **RNF-AMC-03:** El sistema debe medir y reportar métricas de mejora en eficiencia y calidad entre proyectos.

Documentación

- **RNF-DOC-01:** El sistema debe generar documentación técnica completa del sistema modernizado.

- **RNF-DOC-02:** El sistema debe generar documentación de usuario final cuando sea aplicable.
- **RNF-DOC-03:** El sistema debe mantener documentación actualizada de decisiones arquitectónicas (ADR - *Architecture Decision Records*).
- **RNF-DOC-04:** Toda la documentación generada debe estar disponible en formatos accesibles y consultables.

Resiliencia ante Fallos

- **RNF-RF-01:** El sistema debe implementar *circuit breakers* para aislar fallos de agentes individuales.
- **RNF-RF-02:** El sistema debe tener mecanismos de *retry* con *backoff* exponencial para operaciones transitorias.
- **RNF-RF-03:** El sistema debe notificar proactivamente a administradores ante fallos que requieran intervención.
- **RNF-RF-04:** El sistema debe mantener métricas de salud de cada agente y del pipeline completo.

Cumplimiento y Gobernanza

- **RNF-CG-01:** El sistema debe cumplir con políticas de retención de datos aplicables.
- **RNF-CG-02:** El sistema debe generar evidencia auditable para procesos de *compliance*.
- **RNF-CG-03:** El sistema debe permitir la configuración de políticas de gobernanza específicas por organización.
- **RNF-CG-04:** El sistema debe registrar y justificar todas las transformaciones para sectores regulados.

Resultados etapa de ejecución

Esta etapa crucial de la metodología se centró en la definición y formalización de la arquitectura a través del Modelo C4. El objetivo fue crear una representación unificada de la estructura del sistema.

Modelo de agentes propuesto

A partir del modelo de procesos integrado resultante en la etapa de armonización de múltiples modelos y basado en los principios arquitectónicos y restricciones del sistema, se diseñó un pipeline de modernización basado en agentes inteligentes, en el que cada agente representa una unidad funcional autónoma, especializada en una etapa del ciclo de modernización.

La Figura 5 ilustra el flujo general del sistema, coordinado por un Agente Orquestador (AO) encargado de gestionar dependencias, fallos y sincronización entre componentes y un agente encargado de la gestión del conocimiento resultante de las salidas de los demás agentes. Cada agente fue formalmente definido en cuanto a:

- Función principal
- Entradas y salidas esperadas
- Justificación de su rol en el proceso

Agentes

- **AO – Agente Orquestador**
  Función: Coordina, supervisa y valida la interacción entre todos los agentes del sistema de modernización. Controla el ciclo de vida de ejecución, resuelve conflictos, gestiona fallos y asegura que la secuencia de actividades cumpla con las pre-condiciones y post condiciones definidas para cada agente. Actúa como núcleo central de gobernanza del pipeline de modernización.
  Entradas:
  - Estado actual de cada agente (progreso, fallos, métricas).
  - Resultados parciales generados por los agentes previos en la secuencia.
  - Condiciones de control definidas por la metodología (pre-condiciones y post-condiciones).
  - Interacciones humanas requeridas (decisiones en caso de ambigüedad, conflictos o excepciones).
  - Parámetros globales de ejecución (umbrales de confianza, políticas de reintento, prioridades).

  Salidas:
  - Activación secuencial o paralela de agentes según dependencias.
  - Gestión de fallos, reintentos y re-planificación dinámica.
  - Señales de escalamiento para decisiones humanas cuando hay ambigüedades.
  - Registro detallado de trazabilidad del proceso de modernización.
  - Validación del cumplimiento de pre-condiciones y post-condiciones por agente.
  - Activación de iteraciones de mejora (cuando provienen de AMA u otros agentes).

  Por qué es crítico: El AO asegura la coherencia operativa del sistema multi-agente. Sin él, los agentes funcionarían de manera independiente, generando resultados desalineados, ejecuciones fuera de orden, pérdidas de trazabilidad y errores acumulativos en el proceso de modernización. El AO garantiza control, gobernanza, estabilidad, coordinación y capacidad de iteración continua, habilitando un flujo confiable de modernización extremo a extremo.

- **AASH – Agente de Análisis del Sistema Heredado**
  Función: Realiza el análisis inicial y comprehensivo del sistema heredado, identificando sus componentes, estructura, tecnologías y lógica de negocio. Prepara el camino para los demás agentes mediante la extracción sistemática de conocimiento del código fuente.
  Entradas:

  - Código fuente del sistema heredado.
  - Documentación (si está disponible).
  - Configuraciones y scripts de despliegue.
  - Esquemas de bases de datos.

  Salidas:

  - Código fuente enriquecido.
  - Información básica del sistema (lenguajes, frameworks, versiones).
  - Lógica de negocio base.
  - Diagramas del sistema (UML, diagrama de componentes).
  - Mapa de dependencias técnicas.
  - Estructuras de datos y modelos de dominio.
  - Patrones arquitectónicos identificados.
  - Posibles problemas, código heredado problemático y áreas de riesgo

  Por qué es crítico: Sin un entendimiento profundo del sistema heredado, la modernización se realiza a ciegas. AASH establece la base de conocimiento sobre la cual todos los demás agentes construyen sus análisis y decisiones.

- **ADDT – Agente de Análisis de Dependencias y Deuda Técnica**
  Función: Evalúa la complejidad de la modernización mediante la identificación y cuantificación de dependencias y deuda técnica. Identifica componentes obsoletos, redundantes y puntos críticos de acoplamiento.
  Entradas:

  - Dependencias técnicas identificadas (AASH).
  - Estructuras de datos y diagramas base (AASH).
  - Código fuente y métricas de calidad (AASH).
  - Patrones arquitectónicos documentados (AASH)

  Salidas:

  - Mapa de dependencias enriquecido.
  - Informe de deuda técnica clasificada y cuantificada.
  - Análisis de riesgo por componente.
  - Priorización de módulos según impacto y urgencia.

- Recomendaciones de refactorización con justificación.

Por qué es crítico: Sin esta visión, la modernización se realiza sin priorización clara. ADDT provee la base para entender qué partes del sistema son más costosas o riesgosas de migrar, orientando la toma de decisiones estratégicas del AEM.

- **AGR – Agente de Gestión de Requisitos**
  Función: Interpreta los hallazgos técnicos del AASH y ADDT para derivar los requerimientos funcionales y no funcionales del sistema modernizado. Traduce el conocimiento técnico en especificaciones del negocio, garantizando que la modernización preserve la funcionalidad crítica y mejore las capacidades operativas.
  Interacción con stakeholders: Este agente interactúa directamente con usuarios y stakeholders para validar los requisitos construidos, asegurando que la interpretación técnica se alinee con las expectativas y necesidades del negocio.
  Entradas:

  - Resultados del análisis de código (AASH).
  - Lógica de negocio extraída (AASH).
  - Mapa de dependencias y análisis de riesgos (ADDT).
  - Priorización de módulos (ADDT).

  Salidas:

  - Documento de requisitos (funcionales y no funcionales).
  - Matriz de trazabilidad entre requerimientos y componentes del sistema heredado.
  - Identificación de restricciones técnicas y de negocio.
  - Historias de usuario y criterios de aceptación.
  - Ontología de dominio del sistema

  Por qué es crítico: Establece el puente entre lo técnico y lo estratégico, garantizando que la modernización responda a las verdaderas necesidades del negocio y no solo a decisiones tecnológicas.

- **AEM – Agente de Estrategia de Migración**
  Función: Selecciona la estrategia óptima de modernización por componente, considerando complejidad, costo, prioridades y restricciones organizacionales.
  Entradas:

  - Complejidad y deuda técnica (ADDT).
  - Priorización de módulos (ADDT).
  - Requisitos y restricciones (AGR).
  - Matriz de trazabilidad (AGR).

Salidas:

- Plan de migración faseado con hitos y dependencias.
- Estrategia específica por módulo o componente.
- Análisis de trade-offs.
- Cronograma e identificación de riesgos.

Por qué es crítico: Define el *cómo* de la modernización, evitando soluciones genéricas y costosas.

- **ADA – Agente de Definición Arquitectónica**
  Función: Define la arquitectura objetivo del sistema modernizado con base en los requisitos validados y las estrategias de migración definidas. Garantiza que el diseño resultante sea escalable, mantenible y alineado con estándares organizacionales modernos.
  Entradas:

  - Documento de requerimientos validados (AGR).
  - Estrategia de modernización por componente (AEM).
  - Restricciones técnicas y de negocio (AGR).
  - Patrones arquitectónicos del sistema herdado (AASH)

  Salidas:

  - Especificación de arquitectura objetivo (componentes, APIs, bases de datos, comunicación).
  - Diagramas arquitectónicos (C4 model, UML, ERD).
  - Guías de diseño y lineamientos tecnológicos.

  Por qué es crítico: Proporciona la estructura de referencia que guía la transformación del sistema y asegura coherencia entre los equipos de desarrollo y los agentes posteriores en el pipeline de modernización.

- **ATC – Agente de Transformación de Código**
  Función: Ejecuta la conversión del código heredado hacia la nueva arquitectura definida, aplicando las estrategias de migración decididas por el AEM y siguiendo las directrices arquitectónicas del ADA.
  Entradas:

  - Código fuente enriquecido (AASH).
  - Directrices arquitectónicas y lineamientos (ADA).
  - Estrategias de migración por módulo (AEM).
  - Conocimiento del sistema heredado (AASH).
  - Patrones y estructuras identificadas (ADDT).

  Salidas:

- Código modernizado y estructurado conforme a estándares actuales.
- Reporte de trazabilidad entre código original y código transformado.
- Evidencia de preservación semántica y funcional.
- Documentación de cambios y decisiones de implementación.

Por qué es crítico: Constituye el núcleo operativo de la modernización: traduce el conocimiento analizado y el diseño arquitectónico en un artefacto funcional y moderno que reemplazará al sistema heredado.

- **AVV – Agente de Verificación y Validación**
  Función: Genera y ejecuta pruebas automatizadas para validar que la funcionalidad del sistema original se preserve tras la modernización, detectando regresiones funcionales y garantizando la integridad del sistema transformado.
  Entradas:

  - Código transformado (ATC).
  - Trazabilidad de transformaciones (ATC).
  - Documentación de requerimientos y criterios de aceptación (AGR).
  - Comportamiento observado del sistema heredado (AASH).

  Salidas:

  - Suite completa de pruebas automatizadas (unitarias, integración, regresión).
  - Reporte de resultados de pruebas con métricas de cobertura.
  - Identificación de desviaciones funcionales.
  - Métricas de confiabilidad y calidad del código.
  - Casos que requieren validación humana.

  Por qué es crítico: Garantiza que la modernización no rompa la lógica del negocio ni degrade la calidad del sistema. Actúa como red de seguridad antes del despliegue.

- **ADD – Agente de Despliegue**
  Función: Automatiza el despliegue del sistema modernizado en entornos de prueba o producción, implementando estrategias que minimizan riesgos y permiten rollback rápido en caso de problemas.
  Entradas:

  - Código validado y probado (AVV).
  - Métricas de calidad y cobertura (AVV).
  - Especificaciones de infraestructura (ADA).

  Salidas:

- Sistema modernizado desplegado en ambiente objetivo (staging o producción).
- Reporte de despliegue con logs completos y trazabilidad.
- Configuración de monitoreo y alertas.
- Documentación de procedimientos de rollback.

Por qué es crítico: Reduce el riesgo humano en la liberación del software y permite iteraciones continuas sin interrupción del servicio, facilitando la entrega incremental de valor.

- **AMA – Agente de Monitoreo y Adaptación**
  Función: Supervisa el rendimiento, estabilidad y comportamiento del sistema modernizado en producción, detectando anomalías y proponiendo mejoras continuas que retroalimentan el proceso de modernización.
  Entradas:

  - Métricas de producción (latencia, throughput, errores).
  - Logs y trazas del sistema.
  - Comportamiento de usuarios.
  - Incidentes y alertas.

  Salidas:

  - Reportes de rendimiento y confiabilidad.
  - Análisis de tendencias y predicción de problemas.
  - Recomendaciones de optimización continua.
  - Triggers de iteraciones de mejora.

  Por qué es crítico: Transforma el proceso de modernización en un ciclo continuo de mejora, garantizando que el sistema no solo se moderniza una vez, sino que evoluciona continuamente para mantener su relevancia y calidad.

- **AGC – Agente de Gestión de Conocimiento**
  Función: Mantiene y actualiza la base de conocimiento global del proceso de modernización. Centraliza documentación, patrones descubiertos, decisiones arquitectónicas y lecciones aprendidas, sirviendo como fuente de conocimiento para sistemas de recuperación (RAG) utilizados por los demás agentes.
  Estructura del conocimiento:

  - Conocimiento técnico.
  - Conocimiento del dominio.
  - Conocimiento procedimental.
  - Conocimiento contextual.

  Entradas:

- Salidas de todos los agentes previos (informes, métricas, decisiones).
- Retroalimentación de iteraciones (AMA).
- Validaciones y aprobaciones de stakeholders.

Salidas:

- Base de conocimiento versionada.
- Repositorio de patrones y buenas prácticas.
- Documentación de decisiones y justificación.
- Actualización continua de la memoria organizacional RAG.

Por qué es crítico: Asegura aprendizaje organizacional y evita la pérdida de conocimiento entre proyectos. Permite que los agentes evolucionen y mejoren con el tiempo al tener acceso a experiencias previas, convirtiendo cada proyecto de modernización en una fuente de conocimiento para futuros proyectos.

Vista de contexto

La Vista de Contexto que se muestra en la Figura 6 es la representación de más alto nivel, sirviendo como el mapa del ecosistema. Esta define los límites del sistema (SMAM), cómo interactúa con los actores humanos y los sistemas esenciales para el proceso de modernización. En este se posiciona al SMAM como la plataforma central, se establecen claramente los puntos de validación humana (Ingeniero de Software) y los puntos de decisión estratégica (Líder de Modernización).

Para asegurar la claridad de sistema y definir los límites, se han documentado las interacciones clave entre del sistema Multi-Agente de Modernización (SMAM) y los actores externos principales. En la Tabla 7 se resumen las responsabilidades, destacando dónde recae la decisión estratégica y dónde la validación técnica.

Vista de Contenedores

La vista de contenedores, mostrada en la Figura 7 descompone el SMAM en sus contenedores tecnológicos principales, representados en las fases de modernización. El diagrama ilustra claramente el pipeline de valor dividido en tres fases principales:

- Fase 1: Análisis y diagnóstico (AASH, ADDT, AGR): Se enfoca en la extracción de conocimiento y la transformación de hallazgos técnicos en requisitos.
- Fase 2: Planeación y Diseño (AEM, ADA): Se enfoca en la toma de decisiones estratégicas sobre la migración y la arquitectura objetivo.
- Fase 3: Ejecución y Validación (ATC, AVV, ADD, AMA): Se enfoca en la transformación del código y la entrega continua.

Vista de Componentes

Finalmente, la Figura 8 proporciona una Vista de Componentes detallada del SMAM. Este diagrama presenta los agentes alineados por fase, incluyendo sus relaciones internas y externas. Esta vista permite observar con mayor precisión la secuencia de ejecución del pipeline, así como los puntos de integración con sistemas externos.

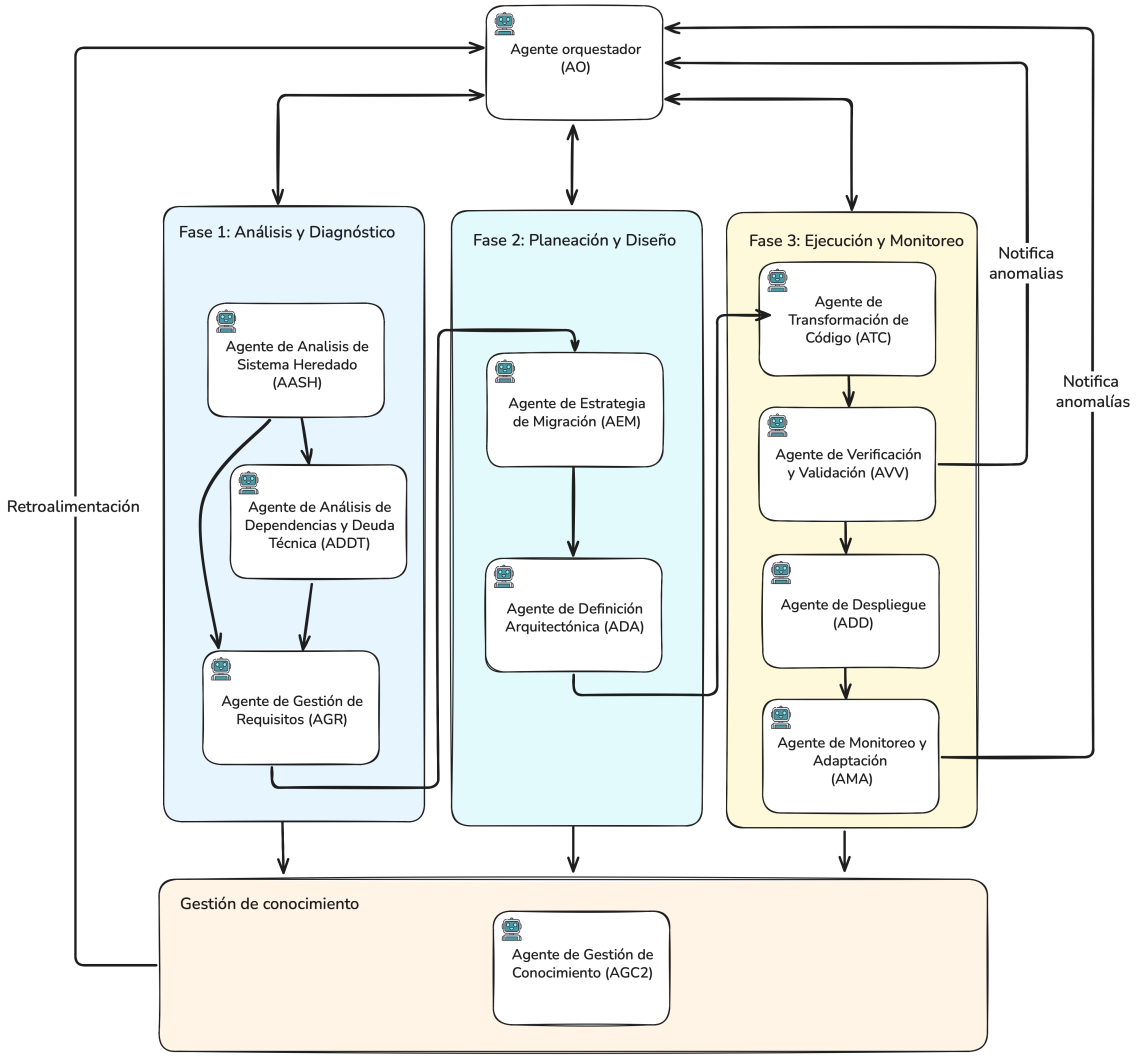

Figura 5: Flujo del sistema propuesto basado en agentes inteligentes para la modernización de sistemas heredados.

Resultados etapa de observación

Durante la etapa de observación, se llevó a cabo la validación del modelo propuesto mediante varias secciones de revisión cruzada (peer review) con expertos del dominio. La muestra estuvo compuesta por N=6 expertos, todos ellos profesionales del área de tecnología y arquitectura empresarial del sector financiero, enmarcados dentro del contexto organizacional de esta investigación. Esta validación tuvo como propósito asegurar la aplicabilidad, robustez y alineación del sistema multiagente propuesto (SMAM) con las necesidades reales de entornos productivos, mediante el análisis de las vistas arquitectónicas desarrolladas con el modelo C4.

La revisión se desarrolló en dos fases complementarias:

Fase 1 – Sesiones de retroalimentación no estructurada

En una primera fase, de carácter exploratorio y no estructurado, se realizaron entrevistas abiertas con los expertos, lo que permitió obtener observaciones cualitativas fundamentales para el refinamiento del diseño. Entre los hallazgos más relevantes se

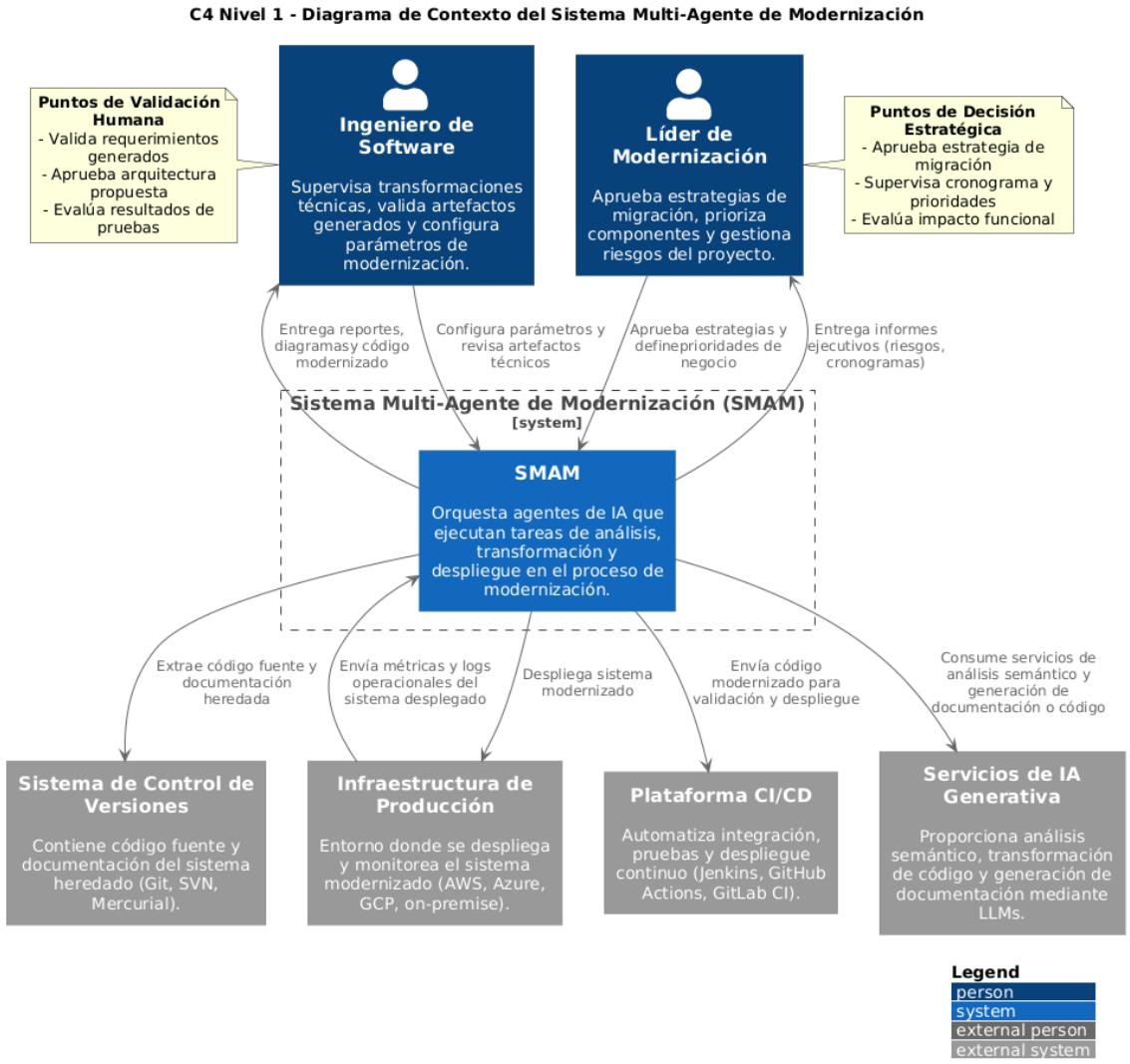

Figura 6: C4 Nivel 1 - Diagrama de Contexto del Sistema Multi-Agente de Modernización (SMAM)

destacan:

- La necesidad de incluir un mecanismo de orquestación que garantizara la coordinación eficiente entre los agentes del sistemas. En respuesta a esta observación, se diseñó e incorporó un agente orquestador (AO), el cual es el responsable de la planificación y distribución de tareas entre los agentes especializados, lo cual le apunta a mejorar la coherencia interna del modelo, así como su capacidad de adaptación ante los distintos escenarios en la modernización de sistemas heredados.
- Se abordó con énfasis la seguridad de la información, dado que los procesos de modernización implican la manipulación código fuente que puede contener procesos sensibles y datos críticos, lo cual debe alinearse con normativas como SOX y políticas internas de confidencialidad. En consecuencia, dentro de los requisitos no funcionales se abordan temas de seguridad relevantes.

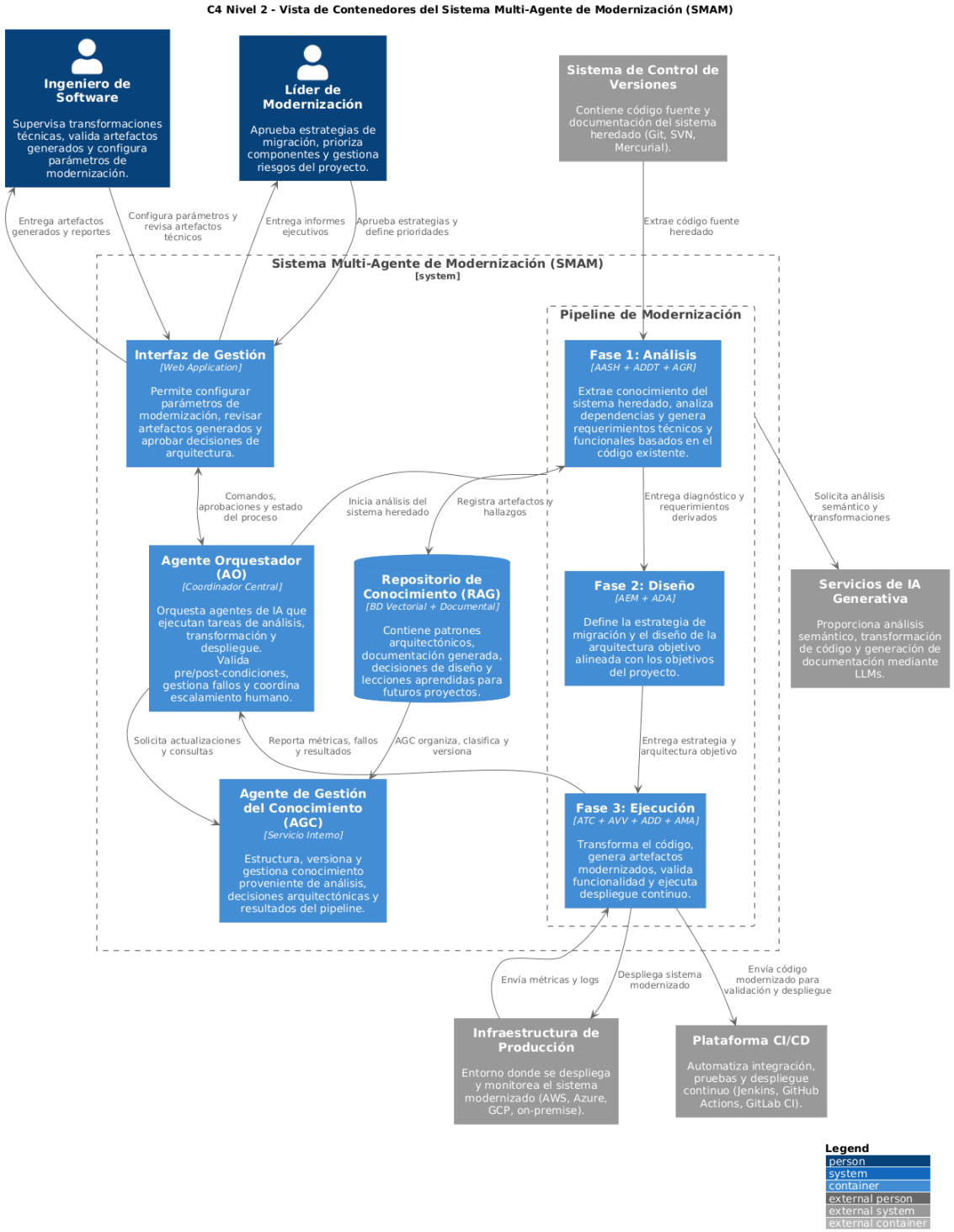

Figura 7: C4 Nivel 2 - Diagrama de contenedores del Sistema Multi-Agente de Modernización (SMAM)

- Se abordó la necesidad de tener trazabilidad en cada una de las etapas de modernización, permitiendo que cada ejecución del sistema pueda se auditada y rastreada. Como resultado se implementó el agente de gestión de conocimiento

(AGC), con el fin de garantizar una correcta cadena de documentación de cada uno de los pasos del proceso. Ademas, cada uno de los agentes tiene definidas salidas especificas que facilitan la trazabilidad y auditoria del sistema, que el AGC se encarga de gestionar.

- Se enfatizó la conveniencia de que el sistema modernizado operara en ambientes preproductivos que estuvieran alineados con estándares tecnológicos de la organización. Por lo tanto se definió el RF-ADD-01 el cual es un requisito funcional del agente de desligues (ADD), que implicada tener un entorno de staging.

- En cuanto a la representación arquitectónica, se sugirió reorganizar la vista de contenedores agrupando por fases del ciclo de vida del software (Análisis, diseño y ejecución). Esta recomendación fue implementada, mejorando la legibilidad del modelo y facilitando su adopción tanto por equipos técnicos como por responsables de negocio.

- Otro de los aspectos críticos surgió durante la presentación de cada uno de los agentes del sistema. Algunos de los expertos manifestaron confusión respecto a las entradas y salidas esperadas de cada agente, específicamente en cuanto al origen y destino de los artefactos que se procesaban. Para solucionar esto, se ajustaron las descripciones de los agentes incluyendo explícitamente, para cada entrada, el agente de origen del artefacto. Esta mejora facilitó la comprensión global del flujo de información dentro del sistema y al eliminarse dicha información de los diagramas simplificó su entendimiento.

- En varias sesiones se discutió el carácter agnóstico del modelo frente a tecnologías especificas. Al tratarse de un modelo de referencia, se acordó que su arquitectura debía mantenerse neutral respecto a lenguajes, frameworks o herramientas concretas, permitiendo que cada organización pueda implementar el modelo con base en sus capacidades y restricciones tecnológicas. Este principio se integró formalmente al diseño, estableciendo que el modelo define roles, flujos y responsabilidades, pero no impone tecnologías específicas para su implementación. Sin embargo se enfatiza la importancia del mapeo sistemático de la literatura al fundamentar cada uno de los agentes en estudios académicos relevantes.

Fase 2 – Evaluación estructurada del diseño

Posteriormente, se aplicó una segunda fase de validación con enfoque estructurado. En esta etapa, se presentó la versión mejorada del modelo a los expertos, quienes valoraron seis criterios globales del diseño utilizando una escala ordinal (adecuado / poco adecuado / no adecuado). Los aspectos evaluados fueron: diseño general, claridad, pertinencia del modelo, potencial de escalabilidad, alineación con los problemas reales, y aplicabilidad práctica.

Los resultados evidencian un alto nivel de aceptación en todos los criterios evaluados, como se muestra en la Tabla 8.

Este ejercicio estructurado complementa las observaciones cualitativas de la fase anterior, permitiendo consolidar la validación del marco desde una perspectiva tanto

técnica como contextual. La retroalimentación obtenida respalda la idea de que el modelo SMAM presenta un diseño sólido y adaptable, con un enfoque práctico orientado a los desafíos reales de modernización de sistemas heredados.

Resultados etapa de Reflexión

Como resultado de este ejercicio reflexivo, se identificaron dos dimensiones clave de aprendizaje:

1. Coherencia y robustez del modelo arquitectónico: la inclusión de componentes como el agente orquestador (AO) y el agente de gestión de conocimiento (AGC) responden directamente a observaciones criticas sobre la necesidad de coordinación y trazabilidad en el proceso de modernización. Estos ajustes no solo reforzaron la arquitectura en términos funcionales, sino también, mejorando su alineación con los requerimientos operativos y normativos de sectores como el financiero, elevando así la posibilidad de aplicarlo en contextos reales con altas exigencias regulatorias.

2. Claridad y usabilidad del modelo: Cambios como la reorganización de la vista de contenedores por etapas del ciclo de vida del desarrollo del software y la definición de las entradas y salidas para cada agente, facilitaron significativamente la comprensión del sistema por parte de los evaluadores.

En términos metodológicos, esta etapa reafirmó la importancia de incorporar la revisión por pares como una práctica sistemática dentro del ciclo de diseño arquitectónico. La retroalimentación cualificada de expertos no solo valida decisiones técnicas, sino que enriquece el modelo al aportar diversas perspectivas desde la experiencia práctica.

Agente	Función	Artículos Relacionados
AO (Agente Orquestador)	Controla el ciclo de vida de los agentes, asegurando el orden lógico de ejecución, la sincronización entre ellos y la trazabilidad del proceso.	[83]
AASH (Análisis de Sistema Heredado)	Realiza el análisis inicial y riguroso del sistema heredado, identificando sus componentes, estructura, tecnologías y lógica de negocio.	[73], [20], [74], [76], [84], [85], [81], [82], [56]
ADDT (Análisis de Dependencias y Deuda Técnica)	Evalúa la complejidad de la modernización mediante la identificación, clasificación y cuantificación de dependencias y deuda técnica.	[85], [82]
AGR (Gestión de Requisitos)	Interpreta los hallazgos técnicos del AASH y ADDT para derivar los requerimientos funcionales y no funcionales del sistema modernizado.	[82], [57], [71]
AEM (Estrategia de Migración)	Decide la estrategia óptima de modernización para cada componente del sistema, considerando factores de complejidad, prioridad, costo, criticidad del negocio y restricciones organizacionales.	[86], [83], [56]
ADA (Definición Arquitectónica)	Define la arquitectura objetivo del sistema modernizado con base en los requisitos validados y las estrategias de migración definidas.	[76], [84], [85]
ATC (Transformación de Código)	Ejecuta la conversión del código heredado hacia la nueva arquitectura definida, aplicando las estrategias de migración y siguiendo las directrices arquitectónicas.	[17], [19], [74], [75], [79], [80], [56]
AVV (Verificación y Validación)	Genera y ejecuta pruebas automatizadas para validar que la funcionalidad del sistema original se preserve tras la modernización.	[74], [85], [84], [83], [56], [63], [64], [9]
ADD (Despliegue Automatizado)	Automatiza el despliegue del sistema modernizado en entornos de prueba o producción, implementando estrategias que minimizan riesgos y permiten rollback.	[76], [84]
AMA (Monitoreo y Adaptación)	Supervisa el rendimiento, estabilidad y comportamiento del sistema modernizado, detectando anomalías y proponiendo mejoras.	[19]
AGC (Gestión de Conocimiento)	Mantiene y actualiza la base de conocimiento del proceso de modernización, centralizando documentación, patrones, decisiones y lecciones aprendidas.	[73], [20], [19], [58], [71]

Tabla 6: Correspondencia entre agentes del modelo y evidencia documental (PI1, PI2, PI3).

Actor/Sistema	Interacción con SMAM	Rol Principal
Líder de modernización (Persona)	Aprueba estrategias, cronogramas y prioridades.	Decisión Estratégica.
Ingeniero de Software (Persona)	Valida artefactos generados (código, requisitos, pruebas).	Validación Técnica (El Cómo).
Control de Versiones (Externo)	SMAM extrae código fuente y documentación.	Fuente autoritativa del sistema heredado.
Servicios de IA Generativa (Externo)	SMAM consume servicios para análisis semántico y transformación de código.	Capacidades de Inteligencia Cognitiva para los agentes.
Plataforma CI/CD (Externo)	SMAM le envía el código modernizado para automatizar pruebas y despliegue.	Automatización y ejecución de pipeline.

Tabla 7: Interacciones Clave del Sistema Multi-Agente de Modernización (SMAM)

Criterio Evaluado	Frecuencia (de 6)
Diseño general del modelo	5
Claridad en la representación del modelo	5
Pertinencia del modelo frente al problema	6
Potencial de escalabilidad	4
Alineación con problemáticas reales del sector	5
Aplicabilidad en contextos organizacionales reales	6

Tabla 8: Evaluación estructurada del diseño del modelo SMAM por parte de los expertos.

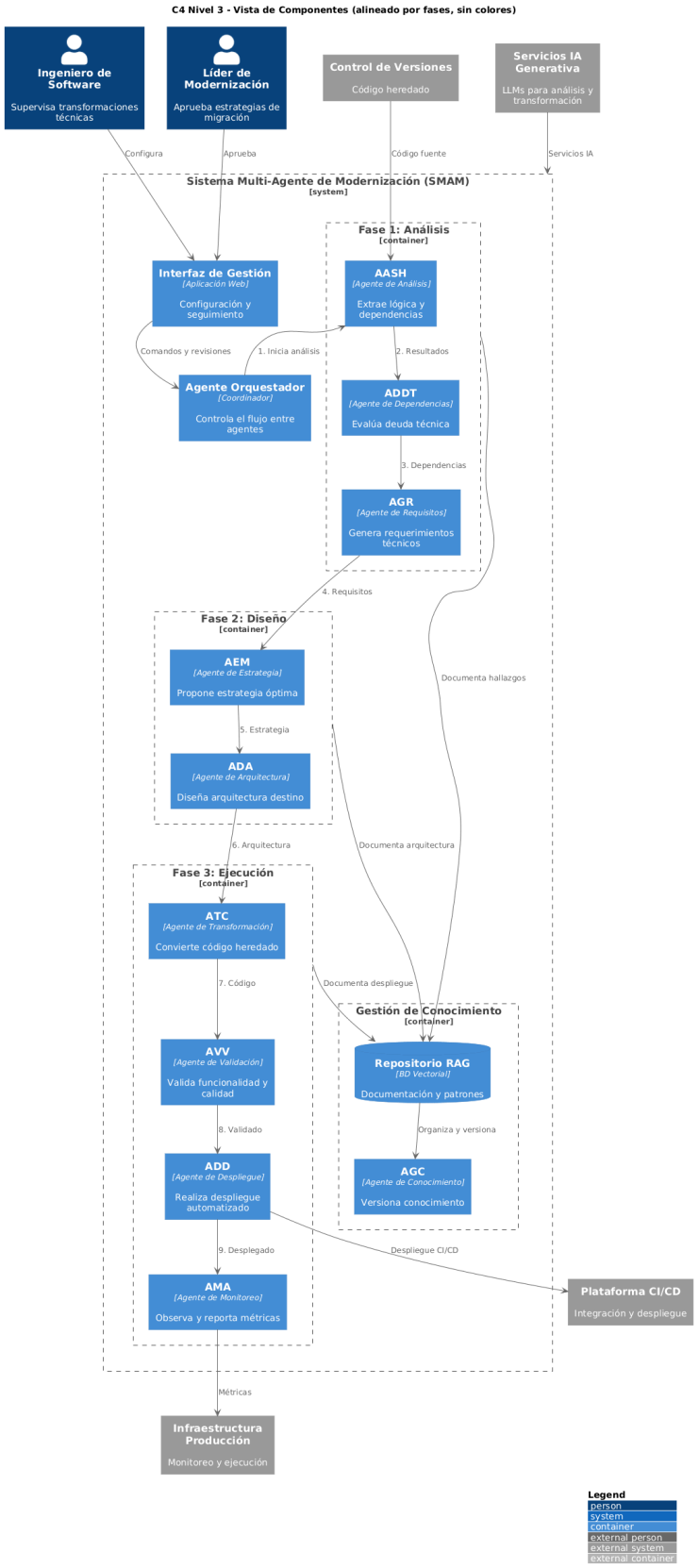

Figura 8: C4 Nivel 3 - Diagrama de componentes del Sistema Multi-Agente de Modernización (SMAM)

10. Conclusiones

- El mapeo sistemático de la literatura mostró que, aunque la modernización de sistemas ha sido estudiada durante décadas, la integración de técnicas de IA en este contexto es aún un campo en consolidación. Sin embargo, la evidencia recopilada indica un crecimiento acelerado desde el 2021, con aplicaciones que abarcan traducción automática de código, ingeniería inversa, generación de documentación, análisis semántico y exteriorización inteligente. Estos resultados confirman que la IA no solo es viable para apoyar la modernización, sino que está empezando a transformar profundamente la forma en que se aborda este proceso.

- El análisis documental evidenció que los modelos de lenguaje de gran escala (LLMs) son la técnica dominante para automatizar tareas históricamente complejas en el mantenimiento y modernización de sistemas, tales como la comprensión de código, recuperación de arquitectura y traducción entre lenguajes. La capacidad de estos modelos para inferir patrones, estructuras y relaciones semánticas los convierte en herramientas críticas para enfrentar problemas asociados a la pérdida de conocimiento, la ausencia de documentación y la complejidad estructural de los sistemas heredados.

- Los estudios analizados muestran que estrategias como refactorización, reingeniería, encapsulamiento, rehosting y reescritura continúan siendo la base de los procesos de modernización. No obstante, la evidencia sugiere que su efectividad se ve limitada cuando se aplican de manera manual en contextos donde existen miles o millones de líneas de código, múltiples dependencias y arquitecturas rígidas. Por ello, el trabajo propone que el futuro de la modernización radica en combinar prácticas tradicionales con capacidades inteligentes, logrando procesos más seguros, reproducibles y escalables.

- La aplicación del framework de armonización de múltiples modelos permitió integrar prácticas, herramientas, enfoques metodológicos y capacidades de IA extraídas del mapeo sistemático. Este proceso llevó a un conjunto de siete actividades fundamentales que representan las capacidades necesarias en un proceso de modernización asistido:

  1. Análisis del sistema heredado
  2. Generación de modelos de requerimientos y arquitectónicos
  3. Clasificación y selección de estrategias de modernización
  4. Transformación asistida por IA (traducción semántica, refactorización, análisis sintáctico)
  5. Validación automática y evolución de calidad
  6. Generación y documentación del conocimiento
  7. Monitoreo y trazabilidad del sistema modernizado

  Esta estandarización proporciona una base teórica sólida para arquitecturas futuras y reduce la subjetividad en la toma de decisiones.

- El diseño final, representado mediante vistas C4, establece un sistema multiagente modular, escalable y orientado a la cooperación. Cada agente fue definido con funciones, entradas, salidas y responsabilidades claras, y su incorporación fue validada mediante un análisis de correspondencia con la literatura. El resultado es un pipeline conceptual de modernización, capaz de coordinar procesos complejos, distribuir tareas cognitivas y gestionar el conocimiento generado durante la transformación del sistema

- Los ejercicios de peer review permitieron validar la claridad, pertinencia y consistencia del diseño arquitectónico. La retroalimentación recibida evidenció fortalezas como la modularidad del sistema, la alineación con tendencias actuales y la claridad en la representación C4. También permitió identificar ajustes necesarios en la formulación de agentes y relaciones, lo que enriqueció la calidad final del modelo.

Referencias

[1] I. Sommerville, *Software Engineering*, M. Horton and M. Hirsch, Eds. Global Edition, 2016, vol. 10, es un libro, es un referencia importante. [Online]. Available: www.pearsonglobaleditions.com

[2] G. Salvatierra, C. Mateos, M. Crasso, A. Zunino, and M. Campo, "Legacy system migration approaches," *IEEE Latin America Transactions*, vol. 11, pp. 840–851, 2013.

[3] G. Barnett, "The future of the mainframe," 2005.

[4] H. K. A. Bakar, R. Razali, and D. I. Jambari, "Implementation phases in modernisation of legacy systems," in *2019 6th International Conference on Research and Innovation in Information Systems (ICRIIS)*, 2019, pp. 1–6.

[5] B. Althani and S. Khaddaj, "Systematic review of legacy system migration," in *2017 16th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES)*, 2017, pp. 154–157.

[6] H. Sofian, N. A. M. Yunus, and R. Ahmad, "Systematic mapping: Artificial intelligence techniques in software engineering," *IEEE Access*, vol. 10, pp. 51 021–51 040, 2022.

[7] M. A. Shehab, M. Wardat, S. Omari, and Y. Jararweh, "Evaluating large language models for code generation: Assessing accuracy, quality, and performance," in *2024 2nd International Conference on Foundation and Large Language Models (FLLM)*, 2024, pp. 407–416.

[8] J. Stümpfle, S. Baum, D. Dittler, N. Jazdi, and M. Weyrich, "Automating software product line adoption based on feature models using large language models," in *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2024, pp. 1–4.

[9] A. Lops, F. Narducci, A. Ragone, and M. Trizio, "Agonetest: Automated creation and assessment of unit tests leveraging large language models," in *2024 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2024, pp. 2440–2441.

[10] Z. Cai, J. Chen, W. Chen, W. Wang, X. Zhu, and A. Ouyang, "F-codellm: A federated learning framework for adapting large language models to practical software development," in *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2024, pp. 416–417.

[11] C. Qian, W. Liu, H. Liu, N. Chen, Y. Dang, J. Li, C. Yang, W. Chen, Y. Su, X. Cong, J. Xu, D. Li, Z. Liu, and M. Sun, "Chatdev: Communicative agents for software development," 6 2024.

[12] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang, "Large language models for software engineering: Survey and open problems," in *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*, 2023, pp. 31–53.

[13] E. de Vargas Agilar, R. B. de Almeida, and E. D. Canedo, "A systematic mapping study on legacy system modernization," in *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, 2016, pp. 345–350.

[14] H. Sherrington, "How to migrate legacy systems from mainframe to open systems technology," in *IEE Coilloquium on Legacy Information System-Barriers to Business Process Re-engineering (Digest no. 1994/246)*, 1994, pp. 9/1–9/3.

[15] H. Knoche and W. Hasselbring, "Using microservices for legacy software modernization," *IEEE Software*, vol. 35, pp. 44–49, 2018.

[16] S. Rosenkranz, D. Staegemann, M. Volk, and K. Turowski, "Explaining the business-technological age of legacy information systems," *IEEE Access*, vol. 12, pp. 84 579–84 611, 2024.

[17] N. Somogyi and G. Kövesdán, "Software modernization using machine learning techniques," in *2021 IEEE 19th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, 2021, pp. 361–365.

[18] M. H. Hasan, M. H. Osman, N. I. Admodisastro, and M. S. Muhammad, "Legacy systems to cloud migration: A review from the architectural perspective," *Journal of Systems and Software*, vol. 202, p. 111702, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121223000973

[19] R. Pietrini, M. Paolanti, and E. Frontoni, "Bridging eras: Transforming fortran legacies into python with the power of large language models," in *2024 IEEE 3rd International Conference on Computing and Machine Intelligence (ICMI)*, 2024, pp. 1–5.

[20] J. A. García-García, C. A. Maldonado, A. Meidan, E. Morillo-Baro, and M. J. Escalona, "gprofit: A tool to assist the automatic extraction of business knowledge from legacy information systems," *IEEE Access*, vol. 9, pp. 94 934–94 952, 2021.

[21] P. Joshi, A. Akbari, and R. B. Svensson, "Impact of usability on process lead-time in information systems: A case study," *Journal of Systems and Software*, vol. 148, pp. 148–169, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121218302383

[22] A. Procter, "Why organizations are moving away from legacy systems | okoone," 3 2024. [Online]. Available: https://www.okoone.com/spark/strategy-transformation/why-organizations-are-moving-away-from-legacy-systems/

[23] Oracle, "Modernize it infrastructure: Oracle mainframe rehosting," 2012. [Online]. Available: https://www.oracle.com/technetwork/middleware/tuxedo/overview/mf-rehost-solution-art-brief-1722399.pdf

[24] W. K. G. Assunção, L. Marchezan, A. Egyed, and R. Ramler, "Contemporary software modernization: Perspectives and challenges to deal with legacy systems," 7 2024.

[25] C. Diggs, M. Doyle, A. Madan, S. Scott, E. Escamilla, J. Zimmer, N. Nekoo, P. Ursino, M. Bartholf, Z. Robin, A. Patel, C. Glasz, W. Macke, P. Kirk, J. Phillips, A. Sridharan, D. Wendt, S. Rosen, N. Naik, J. F. Brunelle, and S. Thaker, "Leveraging llms for legacy code modernization: Challenges and opportunities for llm-generated documentation," 11 2024.

[26] S. Siddeeq, Z. Rasheed, M. A. Sami, M. Hasan, M. Waseem, J. Rasku, M. Saari, K.-K. Kemell, and P. Abrahamsson, "Distributed approach to haskell based applications refactoring with llms based multi-agent systems," 2 2025.

[27] V. Ala-Salmi, Z. Rasheed, A. M. Sami, Z. Zhang, K.-K. Kemell, J. Rasku, S. Siddeeq, M. Saari, and P. Abrahamsson, "Autonomous legacy web application upgrades using a multi-agent system," 1 2025.

[28] Y. Li, Y. Peng, Y. Huo, and M. R. Lyu, "Enhancing llm-based coding tools through native integration of ide-derived static context," in *2024 IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code)*, 2024, pp. 70–74.

[29] R. Ramírez-Rueda, E. Benítez-Guerrero, C. Mezura-Godoy, and E. Bárcenas, "Transforming software development: A study on the integration of multi-agent systems and large language models for automatic code generation," in *2024 12th International Conference in Software Engineering Research and Innovation (CONISOFT)*, 2024, pp. 11–20.

[30] T. He, M. Yang, W. Hu, and Y. Chen, "Analysis of the effectiveness of large language model feature in source code defect detection," in *2024 3rd International Conference on Artificial Intelligence and Computer Information Technology (AICIT)*, 2024, pp. 1–4.

[31] T. W. Harper, *Research Methods in Information Systems: Using Action Research*, 1985.

[32] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*. BCS Learning & Development Ltd., 2008, pp. 68–77.

[33] C. Schardt, M. B. Adams, T. Owens, S. Keitz, and P. Fontelo, "Utilization of the pico framework to improve searching pubmed for clinical questions," *BMC Medical Informatics and Decision Making*, vol. 7, p. 16, 2007. [Online]. Available: https://doi.org/10.1186/1472-6947-7-16

[34] C. Pardo, "A framework to support the harmonization between multiple models and standards," Ph.D. dissertation, 7 2012.

[35] F. J. Pino, M. T. Baldassarre, M. Piattini, and G. Visaggio, "Harmonizing maturity levels from cmmi-dev and iso-iec 15504," *J. Softw. Maint. Evol.*, vol. 22, pp. 279–296, 6 2010.

[36] C. Pardo, F. Garcia, M. Piattini, F. J. Pino, S. Lemus, and M. T. Baldassarre, "Integrating multiple models for definition of it governance model for banking itgsm," *International Business Management*, vol. 10, pp. 4644–4653, 2016.

[37] S. Brown, "The c4 model for software architecture," 2018, accessed: 2025-11-27. [Online]. Available: https://www.infoq.com/articles/C4-architecture-model/

[38] X. Franch and A. Susi, "Risk assessment in open source systems," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, 2016, pp. 896–897.

[39] M. M. Lehman, "Programs, life cycles, and laws of software evolution," in *Proceedings of the IEEE*, 1974.

[40] N. Drouin and M. Badri, "Investigating the applicability of lehman's laws of software evolution using metrics: An empirical study on open source software," in *8th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*. SCITEPRESS, 2013.

[41] D. L. Parnas, "Software aging," in *Proceedings of the 16th International Conference on Software Engineering*. IEEE Computer Society Press, 1994, pp. 279–287.

[42] K. Bennett, *Software Maintenance and Evolution: a Roadmap*, 5 2000.

[43] E. J. Byrne, "Software reverse engineering: A case study," *Software Practice and Experience*, vol. 22, pp. 865–886, 1992. [Online]. Available: https://doi.org/10.1002/spe.4380211206

[44] C.-C. Chiang, "Wrapping legacy systems for use in heterogeneous computing environments," *Information and Software Technology*, vol. 43, pp. 497–507, 2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584901001604

[45] K. S. Ahmad, N. Ahmad, H. Tahir, and S. Khan, "Fuzzy moscow: A fuzzy based moscow method for the prioritization of software requirements," in *2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)*, 2017, pp. 433–437.

[46] A. A. B. Baqais and M. Alshayeb, "Automatic software refactoring: a systematic literature review," *Software Quality Journal*, vol. 28, pp. 459–502, 2020. [Online]. Available: https://doi.org/10.1007/s11219-019-09477-y

[47] G. Canfora and M. D. Penta, "New frontiers of reverse engineering," in *Future of Software Engineering (FOSE '07)*, 2007, pp. 326–341.

[48] H. Sneed and C. Verhoef, "Re-implementing a legacy system," *Journal of Systems and Software*, vol. 155, pp. 162–184, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121219301050

[49] H. A. Bakar, R. Razali, and D. I. Jambari, "A qualitative study of legacy systems modernisation for citizen-centric digital government," *Sustainability*, vol. 14, 2022. [Online]. Available: https://www.mdpi.com/2071-1050/14/17/10951

[50] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Pearson, 2016.

[51] I. H. Sarker, "Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions," *SN Computer Science*, vol. 2, p. 420, 11 2021.

[52] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large language models: A survey," 3 2025.

[53] M. Kim, T. Zimmermann, and N. Nagappan, "An empirical study of refactoring challenges and benefits at microsoft," *IEEE Transactions on Software Engineering*, vol. 40, pp. 633–649, 2014.

[54] R. Feldt, F. G. de Oliveira Neto, and R. Torkar, "Ways of applying artificial intelligence in software engineering," in *2018 IEEE/ACM 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, 2018, pp. 35–41.

[55] I. K. Schieferdecker, "Augmenting software engineering with ai and developing it further towards ai-assisted model-driven software engineering," 7 2025.

[56] S. Agarwal, S. Chimalakonda, S. Krishnan, V. Kanvar, and S. Shah, "Tutorial report on legacy software modernization: A journey from non-ai to generative ai approaches," in *Proceedings of the 17th Innovations in Software Engineering Conference*. Association for Computing Machinery, 2024. [Online]. Available: https://doi-org.ezproxy.eafit.edu.co/10.1145/3641399.3641434

[57] Y. Zhang and Y. Li, "The classification of software requirements using large language models: Emerging results and future directions," in *2024 3rd International Conference on Cloud Computing, Big Data Application and Software Engineering (CBASE)*, 2024, pp. 746–749.

[58] A. Boronat and J. Mustafa, "Mdre-llm: A tool for analyzing and applying llms in software reverse engineering," in *2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2025, pp. 850–854.

[59] K. Bhandari, K. Kumar, and A. L. Sangal, "Artificial intelligence in software engineering: Perspectives and challenges," in *2023 Third International Conference on Secure Cyber Computing and Communication (ICSCCC)*, 2023, pp. 133–137.

[60] G. Antal, R. Vozár, and R. Ferenc, "Toward a new era of rapid development: Assessing gpt-4-vision's capabilities in uml-based code generation," in *2024 IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code)*, 2024, pp. 84–87.

[61] N. J. Al-Khafaji and B. K. Majeed, "Evaluating large language models using arabic prompts to generate python codes," in *2024 4th International Conference on Emerging Smart Technologies and Applications (eSmarTA)*, 2024, pp. 1–5.

[62] X. She, Y. Zhao, and H. Wang, "Wadec: Decompiling webassembly using large language model," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. Association for Computing Machinery, 2024, pp. 481–492. [Online]. Available: https://doi-org.ezproxy.eafit.edu.co/10.1145/3691620.3695020

[63] T. Yang, Z. Jiang, and Y. Wang, "Llmsqli: A black-box web sqli detection tool based on large language model," in *2024 5th International Conference on Big Data & Artificial Intelligence & Software Engineering (ICBASE)*, 2024, pp. 629–633.

[64] Y. Xia, N. Jazdi, and M. Weyrich, "Enhance fmea with large language models for assisted risk management in technical processes and products," in *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2024, pp. 1–4.

[65] X. Yu, W. K. Wong, and S. Wang, "Emi testing of large language model (llm) compilers," in *2024 IEEE 35th International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2024, pp. 187–190.

[66] J. L. Crandall and A. S. Crandall, "Large language model-supported software testing with the cs matrix taxonomy," *J. Comput. Sci. Coll.*, vol. 40, pp. 49–58, 10 2024.

[67] Z. Zhu, S. Du, and Y. Qiu, "Research on the application of large language models in programming tutoring," in *2024 IEEE 16th International Conference on Advanced Infocomm Technology (ICAIT)*, 2024, pp. 314–318.

[68] M. Kim, T. Stennett, D. Shah, S. Sinha, and A. Orso, "Leveraging large language models to improve rest api testing," in *2024 IEEE/ACM 46th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, 2024, pp. 37–41.

[69] F. N. Meem, "Effective integration and use of non-development llms in software development," in *2024 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2024, pp. 374–375.

[70] R. Feldt, S. Kang, J. Yoon, and S. Yoo, "Towards autonomous testing agents via conversational large language models," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2023, pp. 1688–1693.

[71] J. A. Chudziak and K. Cinkusz, "Towards llm-augmented multiagent systems for agile software engineering," in *2024 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2024, pp. 2476–2477.

[72] C. Qian, W. Liu, H. Liu, N. Chen, Y. Dang, J. Li, C. Yang, W. Chen, Y. Su, X. Cong, J. Xu, D. Li, Z. Liu, and M. Sun, "Chatdev: Communicative agents for software development." [Online]. Available: https://github.com/OpenBMB/ChatDev.

[73] H. A. Siala, "Enhancing model-driven reverse engineering using machine learning," in *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2024, pp. 173–175.

[74] V. Nitin, "Using ai to automate the modernization of legacy software applications," in *2024 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2024, pp. 2514–2517.

[75] H. Zhang, C. David, M. Wang, B. Paulsen, and D. Kroening, "Scalable, validated code translation of entire projects using large language models," *Proc. ACM Program. Lang.*, vol. 9, 6 2025. [Online]. Available: https://doi-org.ezproxy.eafit.edu.co/10.1145/3729315

[76] V. Singh, C. Korlu, and W. K. G. Assunção, "Experiences on using large language models to re-engineer a legacy system at volvo group," in *2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2025, pp. 102–112.

[77] M. Macedo, Y. Tian, F. Cogo, and B. Adams, "Exploring the impact of the output format on the evaluation of large language models for code

translation," in *Proceedings of the 2024 IEEE/ACM First International Conference on AI Foundation Models and Software Engineering*. Association for Computing Machinery, 2024, pp. 57–68. [Online]. Available: https://doi-org.ezproxy.eafit.edu.co/10.1145/3650105.3652301

[78] M. S. Abid, M. Pawagi, S. Adhikari, X. Cheng, R. Badr, M. Wahiduzzaman, V. Rathi, R. Qi, C. Li, L. Liu, R. S. Naidu, L. Lin, Q. Liu, A. Z. Palak, M. Haque, X. Chen, D. Marinov, and S. Dutta, "Gluetest: Testing code translation via language interoperability," in *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2024, pp. 612–617.

[79] J. Hong and S. Ryu, "Type-migrating c-to-rust translation using a large language model," *Empirical Software Engineering*, vol. 30, p. 10573, 2024. [Online]. Available: https://doi.org/10.1007/s10664-024-10573-2

[80] D. Busch, A. Bainczyk, S. Smyth, and B. Steffen, "Llm-based code generation and system migration in language-driven engineering," *International Journal on Software Tools for Technology Transfer*, vol. 27, pp. 137–147, 2025. [Online]. Available: https://doi.org/10.1007/s10009-025-00798-x

[81] H. A. Siala and K. Lano, "Towards using llms in the reverse engineering of software systems to object constraint language," in *2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2025, pp. 1–6.

[82] S. A. Rukmono, L. Ochoa, and M. Chaudron, "Deductive software architecture recovery via chain-of-thought prompting," in *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*. Association for Computing Machinery, 2024, pp. 92–96. [Online]. Available: https://doi.org/10.1145/3639476.3639776

[83] H. L. Truong, M. Vukovic, and R. Pavuluri, "On coordinating llms and platform knowledge for software modernization and new developments," in *2024 IEEE International Conference on Software Services Engineering (SSE)*, 2024, pp. 188–193.

[84] G. C. P. Reddy and R. S. Kadali, "Automated refactoring of monolithic applications to cloud-native containers: Application modernization using genai and agentic frameworks," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 12, pp. 2424–2437, 2024. [Online]. Available: https://www.ijisae.org/index.php/IJISAE/article/view/7356

[85] R. Chindanuru, "Modernizing legacy applications: Strategies and emerging trends," *International Journal of Information Technology & Management Information System (IJITMIS)*, vol. 16, pp. 143–158, 2025.

[86] K. Y. Tan, S. C. Tan, and T. C. Chuah, "A multi-phase drl-driven sdn migration framework addressing budget, legacy service compatibility, and dynamic traffic," *IEEE Access*, vol. 13, pp. 33 202–33 219, 2025.