Aplicación de Inteligencia Artificial Generativa y Verificación Formal en el Descubrimiento de Microservicios

Generative AI and Formal Verification in Microservice Discovery

Daniel Narváez
Universidad Abierta Interamericana (UAI), CAETI, Buenos Aires, Argentina. Keiser University Campus Latinoamericano – Facultad de Ingeniería de Software San Marcos, Departamento de Carazo, Nicaragua.

Nicolás Battaglia
Universidad Abierta Interamericana (UAI), CAETI, Buenos Aires, Argentina.

Alejandro Fernández
LIFIA, Facultad de Informática, Universidad Nacional de La Plata (UNLP), Argentina.

Gustavo Rossi
Universidad Abierta Interamericana (UAI), CAETI, Buenos Aires, Argentina. LIFIA, Facultad de Informática, Universidad Nacional de La Plata (UNLP), Argentina.

DOI *https://doi.org/10.59471/raia2025225*
Enviado: junio 2025. **Aceptado**: octubre 2025. **Publicado:** diciembre 2025
Como citar: Narváez, D., Battaglia, N., Fernández, A., & Rossi, G. (2025). Aplicación de Inteligencia Artificial Generativa y Verificación Formal en el Descubrimiento de Microservicios. Revista Abierta De Informática Aplicada, 9(1). *https://doi.org/10.59471/raia2025225*

Resumen

El diseño de microservicios a partir de requisitos textuales constituye un desafío persistente en la ingeniería de software, debido a la ambigüedad del lenguaje natural y a la ausencia de mecanismos formales que garanticen calidad arquitectónica. En el marco de una investigación doctoral en la Universidad Abierta Interamericana (UAI), se presenta ArchiGenMS, un pipeline evolutivo que combina modelos de lenguaje generativos (LLMs) con verificación formal en Lean para el descubrimiento automático de microservicios. La propuesta integra prompt engineering evolutivo, métricas estructurales de cohesión, granularidad y acoplamiento, y validación automática de restricciones arquitectónicas. Los experimentos realizados con datasets públicos de historias de usuario, como el caso g24-unibath, muestran que el enfoque permite generar arquitecturas con alta cohesión ($LCOM_{avg}$ = 0.167), granularidad controlada (SGM_{max} = 4) y bajo acoplamiento ($Coupling_{max}$ = 1). Los resultados evidencian el potencial de integrar técnicas generativas y verificación formal para construir arquitecturas mantenibles y reproducibles en escenarios greenfield.

PALABRAS CLAVES: Microservicios, descubrimiento automático, modelos de lenguaje grandes, verificación formal, Lean Theorem Prover, métricas arquitectónicas, prompt engineering evolutivo, ingeniería de software asistida por IA.

Abstract

The 1918 University Reform, initiated at the National University of Córdoba, marked a milestone in higher education in Latin America. It emerged in a context of political and social transformations, driven by students seeking to dismantle the elitist, clerical, and authoritarian university model. It promoted university autonomy, student participation in governance, academic freedom, and the extension of knowledge to society. Although the reform originated in Argentina, its ideals influenced other countries, particularly Chile, where similar movements emerged. Its legacy endures today, as evidenced by the reinstatement of student shared governance at the University of Santiago, Chile, in 2025.

KEYWORDS: University Reform, Student Movements, Autonomy, Academic Freedom, Latin American Education.

INTRODUCCIÓN

La ingeniería de software contemporánea enfrenta una tensión fundamental: la necesidad de desarrollar sistemas escalables y mantenibles a gran velocidad (Newman, 2021), frente a la exigencia de garantizar rigor arquitectónico y robustez estructural (E. M. Clarke & Wing, 1996). Las arquitecturas de microservicios se han consolidado como la respuesta predominante a los requerimientos de escalabilidad y despliegue independiente (Velepucha & Flores, 2023), impulsadas por entornos *cloud* donde las organizaciones requieren iterar rápidamente (Taibi, Lenarduzzi & Pahl, 2017).

Sin embargo, esta transición expone una contradicción metodológica central. La agilidad, como paradigma dominante, desalienta el diseño arquitectónico inicial exhaustivo (*Big Design Upfront*). No obstante, los microservicios exigen decisiones de diseño estratégicas y correctas desde el inicio para ser exitosos. La evidencia empírica confirma que la complejidad de estos sistemas demanda un esfuerzo de planificación que a menudo entra en fricción con las prácticas ágiles tradicionales (Taibi, Lenarduzzi, Pahl & Janes, 2017), llevando a los equipos a aplicar métodos de estimación subjetivos no adaptados a la naturaleza distribuida del paradigma (Ünlü et al., 2024).

El desafío fundamental no reside en la implementación de servicios individuales, sino en el proceso de descubrimiento: decidir cómo particionar una aplicación compleja en componentes autónomos. Este reto se magnifica en los escenarios *greenfield* (desarrollo desde cero), foco de este trabajo. A diferencia de la migración *brownfield*, donde existen artefactos estructurales observables como código o trazas (Kalia et al., 2021), el diseño *greenfield* depende exclusivamente de requisitos textuales fragmentados y ambiguos, tales como historias de usuario (Vera-Rivera et al., 2023).

La literatura industrial identifica los “cortes incorrectos” (*Wrong Cuts*) como una de las principales causas de fracaso. Una partición deficiente puede derivar en un “monolito distribuido”: un sistema que sufre la rigidez del acoplamiento monolítico sumada a la latencia y fragilidad de la red (Taibi, Lenarduzzi & Pahl, 2017). Estos errores tempranos se manifiestan como *olores arquitectónicos* (*architectural smells*), tales como *God Services* o granularidad subóptima, que anulan los beneficios de escalabilidad esperados (Neri et al., 2020).

Para asistir esta actividad crítica, la Inteligencia Artificial (IA) y los Modelos de Lenguaje de Gran Escala (LLMs) han mostrado un potencial disruptivo. Sin embargo, las soluciones actuales presentan limitaciones metodológicas significativas:

1. **La falacia de la similitud semántica:** Enfoques como SEMGROMI (Vera-Rivera et al., 2023) o GreenMicro (Bajaj et al., 2022) asumen que la similitud textual implica cohesión funcional. Investigaciones recientes demuestran que, debido a la *anisotropía* de los *embeddings*, términos funcionalmente opuestos pueden distanciarse en el espacio vectorial, llevando a particiones erróneas (Pérez et al., 2025).

2. **La brecha de la plausibilidad:** Los LLMs generan salidas que parecen correctas sintácticamente pero a menudo violan principios estructurales (referencias circulares, acoplamiento oculto). Carecen de un mecanismo de *grounding* formal (Esposito et al., 2025).

La literatura reciente, incluyendo nuestro mapeo sistemático preliminar (Narváez et al., 2024) y la revisión sistemática ampliada publicada en *Software* (Narváez et al., 2025a), advierte que aunque los LLMs muestran potencial disruptivo, carecen de mecanismos de *grounding* formal. Para abordar este vacío, esta investigación adopta el marco de *Design Science Research* (DSR) (Hevner et al., 2004) y propone ArchiGenMS, un artefacto que implementa un ciclo de *generar-verificar*. Nuestra propuesta integra la capacidad exploratoria de los LLMs con la validación rigurosa de los métodos formales. A diferencia de enfoques puramente heurísticos, ArchiGenMS utiliza el asistente de pruebas Lean 4 (Moura & Ullrich, 2021) como un oráculo formal que evalúa cada candidato mediante métricas computables de cohesión (LCOM), granularidad (SGM) y acoplamiento (Fan-Out) (Al-Debagy & Martinek, 2020). De este modo, la verificación formal no actúa como una validación tardía, sino como un motor que guía la búsqueda evolutiva hacia diseños estructuralmente consistentes.

El resto del artículo está organizado de la siguiente manera: La Sección 2 revisa críticamente el estado del arte y expone las limitaciones de los enfoques heurísticos actuales. La Sección 3 establece el marco teórico, formalizando las métricas arquitectónicas y el modelo matemático subyacente. La metodología propuesta, incluyendo la arquitectura de ArchiGenMS y los algoritmos evolutivos, se detalla en la Sección 4. El diseño experimental y el protocolo de validación se describen en la Sección 5, seguidos por el análisis de los resultados empíricos cuantitativos y cualitativos en la Sección 6. Finalmente, la Sección 7

interpreta los hallazgos en el contexto de las hipótesis planteadas, y la Sección 8 resume las aportaciones y delinea las líneas de investigación futura.

Estado del Arte y Motivación

El diseño de microservicios asistido por IA se sitúa en la intersección de la ingeniería de software empírica, la optimización metaheurística y los métodos formales. A continuación, desglosamos el estado del arte desde la práctica industrial hasta las fronteras de la investigación generativa, identificando los vacíos que motivan nuestra propuesta.

La Brecha entre Teoría y Práctica Industrial

Aunque la literatura prescribe principios claros como el desacoplamiento y la autonomía, la evidencia empírica revela que la adopción industrial de microservicios sigue siendo predominantemente manual y heurística. Estudios recientes confirman que, si bien el *Domain-Driven Design* (DDD) es el enfoque metodológico preferido, su aplicación es abstracta y depende críticamente de la pericia de arquitectos senior (Zhong et al., 2024).

En contextos ágiles, esta dependencia crea un cuello de botella: los equipos a menudo recurren a notaciones de diseño orientadas a objetos (OOAD) obsoletas o a talleres colaborativos informales como *Event Storming* (Battaglia et al., 2024). Esta falta de sistematización deriva frecuentemente en descomposiciones subóptimas, conocidas como “cortes incorrectos” (*Wrong Cuts*), que son la causa raíz de la deuda técnica arquitectónica (Taibi, Lenarduzzi, Pahl & Janes, 2017).

El Diseño como Problema de Optimización (SBSE)

Desde una perspectiva teórica, el descubrimiento de microservicios es una instancia del problema de modularización de software, el cual es computacionalmente intratable (NP-hard) (Mitchell & Mancoridis, 2008). La disciplina de la *Ingeniería de Software Basada en Búsqueda* (SBSE) aborda este desafío reformulándolo como un problema de optimización (Harman et al., 2012).

Los enfoques tradicionales de SBSE utilizan algoritmos genéticos para optimizar métricas de cohesión y acoplamiento sobre grafos de dependencia estática (código fuente). Sin embargo, estos métodos son inaplicables en escenarios *greenfield* donde no existe código. ArchiGenMS extiende este paradigma al sustituir los operadores de mutación sintáctica tradicionales por un operador de variación semántica basado en LLMs, permitiendo aplicar SBSE directamente sobre el espacio de requisitos textuales.

Limitaciones de los Enfoques Basados en NLP Clásico

Para abordar el diseño *greenfield*, han surgido propuestas que aplican Procesamiento de Lenguaje Natural (NLP) sobre historias de usuario, como SEMGROMI (Vera-Rivera et

al., 2023) o GTMicro (Bajaj et al., 2024). Estos sistemas agrupan requisitos basándose en la similitud de vectores de palabras (*embeddings*).

Sin embargo, estos enfoques se fundamentan en una heurística problemática: asumen que la *similitud semántica* es un proxy suficiente de la *cohesión funcional*. Como demuestran Pérez et al. (Pérez et al., 2025), esta asunción es defectuosa debido a la anisotropía en los espacios de *embeddings*. En un espacio vectorial anisotrópico, las representaciones de términos funcionalmente opuestos (ej. “Crear” vs. “Borrar”) pueden distanciarse significativamente, llevando a algoritmos de clustering a separar operaciones que deberían pertenecer al mismo microservicio (alta cohesión lógica), o a agrupar erróneamente términos léxicamente cercanos pero funcionalmente independientes.

IA Generativa y la Brecha de Verificabilidad

La irrupción de los LLMs ha permitido superar la rigidez del clustering. Estudios explo¬ratorios con ChatGPT muestran que los LLMs pueden realizar razonamiento abductivo para proponer arquitecturas complejas (Stojanovic & Lazarevié, 2023). No obstante, estos modelos operan bajo un paradigma de plausibilidad probabilística, no de corrección lógica. Revisiones recientes advierten que los LLMs son propensos a “alucinaciones arquitectóni¬cas”: invención de dependencias, violación de granularidad o creación de referencias circulares (Esposito et al., 2025). Existe, por tanto, una brecha de verificabilidad: la capacidad de ge¬nerar diseños ha superado la capacidad de validarlos automáticamente. ArchiGenMS aborda este vacío integrando un oráculo formal que evalúa la corrección estructural de las propuestas generativas antes de que sean aceptadas.

Justificación de la Verificación Formal Estática

Para mitigar los riesgos de la generación estocástica, es necesario un mecanismo de validación riguroso. En la literatura de métodos formales, existen dos familias principales: el *Model Checking* (ej. TLA+, Alloy), orientado a verificar propiedades temporales y de concurrencia; y los *Asistentes de Prueba* (ej. Lean, Coq), orientados a la demostración matemática y el cálculo funcional (E. Clarke et al., 1999).

Dado que nuestro objetivo es validar la calidad estructural estática (cohesión, acoplamiento, granularidad) y no el comportamiento dinámico en tiempo real, adoptamos Lean 4 como validador computable (Moura & Ullrich, 2021). A diferencia de un *model checker* que explora espacios de estados exponenciales, Lean permite definir las métricas arquitectónicas como funciones totales y verificar invariantes (como la irreflexividad de las llamadas) de manera determinista y eficiente ($< 4s$ por individuo), haciéndolo ideal para integrarse en el bucle interno de un algoritmo evolutivo.

Fundamentación Teórica y Formalización Matemática

La validación de arquitecturas generadas por IA requiere un marco de referencia riguroso que trascienda las descripciones en lenguaje natural. En esta sección, establecemos las bases matemáticas del modelo arquitectónico, formalizamos las métricas de calidad como funciones computables y definimos las propiedades algebraicas que garantizan la consistencia de la evaluación.

Preliminares y Notación

Para garantizar la precisión en la definición de las métricas, fijamos la siguiente notación basada en la teoría de conjuntos elemental y teoría de grafos (Gross et al., 2018):

- Denotamos por $\mathscr{S}$ al conjunto finito de microservicios y por O_s al conjunto finito d operaciones del microservicio $s \in \mathscr{S}$.
- Usamos $|X|$ para denotar la cardinalidad de un conjunto finito X.
- Definimos las funciones de agregación sobre el conjunto de servicios como:

$$\text{avg}(\mu) = \frac{1}{|\mathscr{S}|} \sum_{s \in \mathscr{S}} \mu(s), \quad \max(\mu) = \max_{s \in \mathscr{S}} \mu(s), \quad \text{sum}(\mu) = \sum_{s \in \mathscr{S}} \mu(s).$$

Modelo Formal de la Arquitectura

Modelamos una arquitectura de microservicios candidata como una estructura de grafo dirigido y rotulado. Definimos la tupla $\mathscr{A}$ como:

$$\mathscr{A} = (S,E,O,P) \qquad (1)$$

Donde:

- S es el conjunto de vértices que representan los servicios.
- $E \subseteq \{(s_i, s_j) \in S \times S \mid s_i \neq s_j\}$ es el conjunto de aristas dirigidas, representando llamadas remotas.
- $O : S \to \mathscr{P}(Op)$ asigna a cada servicio su conjunto de operaciones.
- $P : Op \to \mathscr{P}(Param)$ asigna a cada operación su firma de parámetros.

Formalización de Métricas Estructurales

Basándonos en los marcos de evaluación de microservicios establecidos por (Taibi & Systä, 2019) y (Al-Debagy & Martinek, 2020), formalizamos las métricas como funciones

totales computables sobre el grafo arquitectónico A . Esta formalización es necesaria para garantizar que el cálculo sea determinista dentro del asistente de pruebas Lean 4.

Cohesión (LCOM Normalizado)

Adaptamos la métrica Lack of Cohesion of Methods para evaluar la cohesión interna. Para un servicio s, sean Ps y Pns los conjuntos de pares de operaciones que comparten y no comparten parámetros, respectivamente. Definimos:

$$LCOM(s) = \begin{cases} 0.0 & \text{si } |O(s)| < 2 \\ \frac{|P_{ns}|}{|P_{ns}|+|P_s|} & \text{en otro caso} \end{cases} \quad (2)$$

Esta definición normalizada $LCOM \in [0,1]$ es consistente con la implementación en Lean 4, donde se manejan explícitamente los casos degenerados para garantizar la totalidad de la función.

Granularidad y Dispersión (SGM)

Evaluamos el tamaño funcional (SGM) y la consistencia interna de las interfaces (SGMSD) utilizando la desviación estándar poblacional:

$$SGM(s) = |O(s)| \quad (3)$$

$$SGM_{SD}(s) = \sqrt{\frac{1}{|O(s)|}\sum_{o \in O(s)} (|P(o)| - \mu_p)^2} \quad (4)$$

Donde μ_p es el promedio de parámetros por operación en el servicio s.

Acoplamiento (Fan-Out)

El acoplamiento se define estrictamente como el grado de salida (dout) en el grafo de llamadas inducido:

$$Coupling(s) = |\{t \in S \mid (s,t) \in E\}| \quad (5)$$

Función de Aptitud Agregada

La búsqueda evolutiva minimiza una función de aptitud escalar F (G) que agrega las mé-tricas individuales. Para garantizar la comparabilidad con los experimentos previos (Narváez et al., 2025b), definimos los operadores de agregación como:

$$\text{avg}(\mu) = \frac{1}{|S|}\sum_{s \in S} \mu(s), \quad \max(\mu) = \max_{s \in S} \mu(s), \quad \text{sum}(\mu) = \sum_{s \in S} \mu(s)$$

La función objetivo final se compone linealmente sin pesos arbitrarios, facilitando el análisis de sensibilidad:

$$\mathscr{F}(\mathscr{G}) = \text{avg}(LCOM) + \max(SGM) + \text{sum}(SGM_{SD}) + \max(Coupling) \tag{6}$$

Trazabilidad del Cómputo: Un Ejemplo Constructivo

Para ilustrar la naturaleza determinista del oráculo formal, presentamos un micro-caso de trazabilidad completa, desde la representación del genotipo hasta el cálculo de la aptitud.

1. Entrada (Genotipo $\mathscr{G}$): Considérese un sistema con tres servicios definido por el LLM:

- `UserService` (S_1): 3 operaciones (o_{1a}, o_{1b}, o_{1c}).
  - $o_{1a}(user, pass)$, $o_{1b}(user, data)$ comparten param 'user'.
  - $o_{1c}(ticket)$ no comparte nada con los anteriores.
- `AuthService` (S_2): 2 operaciones que comparten 'user'.
- `DataService` (S_3): 1 operación aislada.
- Llamadas: $S_1 \to S_2$, $S_1 \to S_3$, $S_2 \to S_3$.

2. Cómputo en Lean:

- Para S_1: $|O_{S1}| = 3$. Pares posibles: 3. Comparten: 1 (o_{1a}, o_{1b}). No comparten: 2.

$$LCOM(S_1) = \frac{2}{2+1} = 0.667$$

- Para S_2: $|O_{S2}| = 2$. Pares: 1. Comparten: 1. No comparten: 0.

$$LCOM(S_2) = \frac{0}{0+1} = 0.000$$

- Para S_3: $|O_{S3}| = 1$. Caso degenerado.

$$LCOM(S_3) = 0.000$$

[1] *Si bien en este artículo se aborda el caso particular de la Universidad de Santiago de Chile (USACH), existen otras universidades chilenas que han incursionado en la participación triestamental en la elección de sus autoridades durante los últimos años.*

3. Agregación:

$$LCOM_{avg} = \frac{0.667 + 0.0 + 0.0}{3} = 0.222$$

$$Coupling_{max} = \max(d^+(S_1), d^+(S_2), d^+(S_3)) = \max(2, 1, 0) = 2$$

Este proceso demuestra que el cálculo de la función de aptitud F es una transformación determinista y auditable del grafo arquitectónico, libre de la subjetividad de las métricas basadas en similitud semántica pura.

Metodología: El Pipeline ArchiGenMS

Para abordar los desafíos de plausibilidad y corrección estructural, diseñamos ArchiGenMS, un artefacto que implementa un ciclo de *Generar-Verificar* iterativo. La metodología se fundamenta en la Ingeniería de Software Basada en Búsqueda (SBSE) (Harman et al., 2012), tratando el diseño arquitectónico como un problema de optimización combinatoria NP-hard (Mitchell & Mancoridis, 2008).

A diferencia de enfoques previos que utilizan algoritmos genéticos tradicionales sobre código existente, ArchiGenMS opera sobre requisitos textuales y utiliza un Modelo de Lenguaje (LLM) como operador de variación semántica.

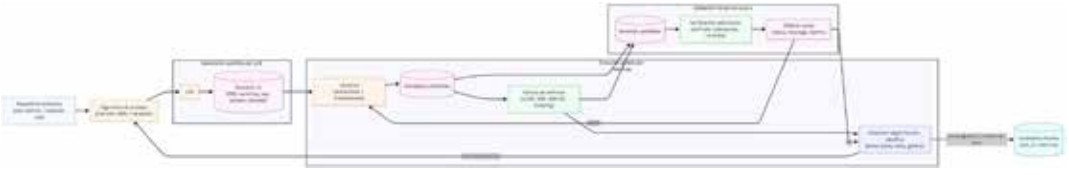
Arquitectura del Sistema

El sistema se compone de dos subsistemas desacoplados (Ver Figura 1):

1. **El Orquestador (Python):** Gestiona el ciclo de vida evolutivo, mantiene el estado de la población y construye los prompts dinámicos.

2. **El Verificador (Lean 4):** Un componente *stateless* compilado que actúa como oráculo determinista. Recibe un genotipo (JSON), valida invariantes y computa métricas con precisión de punto flotante.

FIGURA 1: Arquitectura del Pipeline Evolutivo. Los LLMs generan candidatos que son filtrados y evaluados por el verificador formal Lean 4 antes de la selección.



Estrategia de Prompt Engineering Evolutiva

La calidad de la generación depende críticamente del contexto provisto al LLM. Implementamos una estrategia de *Prompting Dual* para mitigar la deriva semántica:

Fase 1: Inicialización (Zero-Shot)

Para la generación $g = 0$, se inyectan los requisitos crudos R y se impone una restricción gramatical estricta (JSON Schema) para garantizar que la salida sea serializable. El objetivo es maximizar la diversidad inicial del espacio de búsqueda.

Fase 2: Variación (Few-Shot con Contexto)

Para $g > 0$, el sistema construye un *Inspiration Prompt*. En lugar de realizar un cruce de bits aleatorio (que destruiría la semántica), inyectamos:

- **Genotipo Padre (Gparent):** La arquitectura seleccionada en la ronda anterior.
- **Inspiraciones (Ibest):** Un conjunto de k = 3 soluciones de alta aptitud recuperadas de la historia evolutiva.

El LLM actúa como un operador de mutación inteligente: "lee" la solución padre y los ejemplos de excelencia, y genera una nueva variante que intenta optimizar la estructura respetando el dominio.

Algoritmo de Orquestación ($\mu + \lambda$)

El núcleo del proceso es un algoritmo evolutivo $(\mu + \lambda)$ donde μ es el tamaño de la población de padres y λ el número de descendientes. El ciclo se formaliza en el Algoritmo 1

```
def evolucionar(requisitos, max_gen, mu, lam):
    # 1. Inicialización
    P = [LLM.generate(requisitos) for _ in range(mu)]
    evaluar_poblacion(P) # Invoca a Lean

    for g in range(max_gen):
        # 2. Selección de Elites
        padres = seleccionar_mejores(P, mu)
        descendencia = []

        # 3. Variación Asistida por LLM
        while len(descendencia) < lam:
            padre = random.choice(padres)
            prompt = construir_prompt(requisitos, padre, inspiraciones=
                padres[:3])

            hijo_json = LLM.generate(prompt, temperature=0.5)

            # 4. Verificación Formal
            valido, metricas = Lean.validate(hijo_json)
```

```
            if valido:
                hijo.fitness = calcular_fitness(metricas)
            else:
                hijo.fitness = PENALIZACION_MAX # Soft constraint

            descendencia.append(hijo)

        # 5. Reemplazo (Mu + Lambda)
        P = seleccionar_mejores(padres + descendencia, mu)
        persistir_generacion(P, g)

    return P[0] # Mejor individuo
```

Listing 1: *Pseudocódigo del Algoritmo de Orquestación.*

Lean 4 como Validador Computable

La validación de calidad en esta propuesta se centra en la estructura estática de la arquitectura. Bajo este foco, adoptamos Lean 4 como un componente *operativo* del pipeline: un ejecutable que formaliza y calcula métricas, y verifica invariantes estructurales mínimas de forma determinista (Moura & Ullrich, 2021). Esta elección metodológica contrasta con herramientas orientadas al análisis temporal de estados (como TLA+), alineando la verificación con el lenguaje matemático de las métricas empleadas.

Cálculo de Métricas (LCOM)

El Listado 2 presenta la implementación en Lean de la métrica de cohesión, tal como se define en el módulo ServiceMetrics.lean de la tesis. Se observa el uso de tipos de punto flotante para el cálculo de precisión.

```
def LCOM (s : Service) : Float :=
  let ops := s.ops
  -- Caso degenerado: menos de 2 operaciones
  if ops.length < 2 then 0.0 else

  let listOfLists := ops.map (fun o1 => ops.map (fun o2 => (o1, o2)))
  let allPairs := listOfLists.foldr List.append []
  -- Filtrar pares únicos
  let pairs := allPairs.filter (fun (o1, o2) => o1.name < o2.name)

  -- Partición: comparten vs no comparten parámetros
  let (sharing, notSharing) := pairs.partition
    (fun (o1, o2) => o1.params.any (fun p => o2.params.contains p))

  let p := Float.ofNat notSharing.length
  let q := Float.ofNat sharing.length

  if p == 0.0 || q == 0.0 then 0.0 else p / (p + q)
```

Listing 2: *Definición de estructuras y cálculo de LCOM en Lean 4.*

Validación de Invariantes Estructurales

Además del cálculo métrico, el sistema impone restricciones duras mediante la función validateInvariants. Como se muestra en el Listado 3, esta función descarta arquitecturas-que contengan auto-llamadas o referencias a servicios inexistentes antes de proceder a la evaluación.

```
def validateInvariants (gt : Genotype) : Except String Unit := do
  -- 1. Construir conjunto de nombres válidos
  let serviceNames := (gt.microservices.map (fun s => s.name)).
     toHashSet

  -- 2. Verificar cada llamada
  for c in gt.calls do
    -- I1: No auto-llamadas (Self-call)
    if c.caller == c.callee then
      throw s!"Error␣[I1]:␣Auto-llamada␣en␣'{c.caller}'"

    -- I2: Integridad de referencias
    if ! (serviceNames.contains c.caller) then
      throw s!"Error␣[I2]:␣Caller␣'{c.caller}'␣no␣existe"
    if ! (serviceNames.contains c.callee) then
      throw s!"Error␣[I2]:␣Callee␣'{c.callee}'␣no␣existe"

  pure ()
```

Listing 3: *Validación de invariantes estructurales en Lean 4.*

Esta validación actúa como un filtro de consistencia en el bucle evolutivo, asegurando que solo los genotipos estructuralmente válidos sean considerados para la selección.

Implementación y Reproducibilidad

La reproducibilidad es un desafío central en la IA generativa (Esposito et al., 2025). Para garantizarla, ArchiGenMS implementa:

- Determinismo en Validación: El verificador Lean es una función pura; ante el mismo JSON, siempre retorna las mismas métricas.
- Control de Semillas: Aunque los LLMs son estocásticos, fijamos la temperatura (τ = 0.5) y las semillas del generador de números pseudoaleatorios de Python.
- Artefactos Persistentes: Cada ejecución genera un archivo .jsonl con la traza completa (prompts, respuestas, métricas), permitiendo auditoría ex-post.

El código fuente completo y los scripts de orquestación están disponibles en el paquete de replicación (Narváez, 2025).

Diseño y Configuración Experimental

La evaluación empírica se diseñó siguiendo los lineamientos de la metodología DSR para completar el ciclo de validación del artefacto. El objetivo central es determinar si la integración de un oráculo formal en el bucle generativo produce mejoras estadísticamente significativas en la calidad arquitectónica respecto a una línea base estocástica.

A continuación, se detallan las preguntas de investigación, las hipótesis planteadas, el protocolo de ejecución y los datasets utilizados.

Preguntas e Hipótesis de Investigación

Para operacionalizar la validación, formulamos las siguientes preguntas de evaluación (PE) y sus correspondientes hipótesis experimentales (H):

- **PE1:** ¿Produce ArchiGenMS arquitecturas con mayor cohesión interna que los enfoques puramente generativos?
    - **H1:** La métrica de cohesión promedio (LCOMavg) disminuirá monótonamente a través de las generaciones, alcanzando valores inferiores a 0.30 (umbral de buena cohesión (Al-Debagy & Martinek, 2020)).
- **PE2:** ¿Es capaz el sistema de mantener el acoplamiento bajo control sin intervención humana?
    - **H2:** El acoplamiento máximo (Couplingmax) convergerá hacia valores unitarios (1.0), minimizando las dependencias cíclicas o excesivas.
- **PE3:** ¿Son los resultados reproducibles a pesar de la estocasticidad de los LLMs?
    - **H3:** La varianza de las métricas entre múltiples corridas independientes con semillas controladas será despreciable, confirmando la estabilidad del método.

Dataset de Referencia

Utilizamos el corpus de historias de usuario de Dalpiaz et al. (Dalpiaz, 2018), considerado el estándar de facto para la evaluación de tareas de ingeniería de requisitos automatizada. Este dataset fue seleccionado por su:

1. **Heterogeneidad:** Incluye 22 proyectos de dominios diversos (e-commerce, gestión académica, gobierno, ciencia ciudadana).

2. **Realismo:** Los requisitos presentan ambigüedades típicas del lenguaje natural que desafían a los parsers tradicionales.

3. **Disponibilidad:** Al ser de acceso abierto, garantiza la replicabilidad de nuestro estudio. La Tabla 1 resume las características de los proyectos seleccionados para la prueba.

TABLA 1: Subconjunto representativo de los proyectos del dataset de Dalpiaz ut lizados en la evaluación.

Identificador	Dominio	# Historias
g04-recycling	*Gestión de reciclaje*	52
g24-unibath	*Repositorio institucional (Educación)*	48
g02-federalspending	*Transparencia financiera (Gobierno)*	65
g05-openspending	*Presupuestos abiertos*	38
g28-zooniverse	*Ciencia ciudadana*	42
g13-planningpoker	*Herramientas ágiles*	35
...	...	...

Protocolo y Configuración

El experimento se ejecutó en un entorno controlado (Intel Core i7, 12GB RAM) utilizando el orquestador Python 3.11 y el verificador Lean 4 v4.22. Para garantizar la validez estadística, se aplicó el siguiente protocolo riguroso:

1. **Aislamiento:** Cada proyecto del dataset se trató como un escenario independiente.

2. **Repetición:** Se realizaron 5 corridas completas para cada escenario, variando la semilla aleatoria (*SEED* {42,101,...}) para mitigar el sesgo estocástico del LLM.

3. **Parámetros** Evolutivos: Se configuraron para equilibrar la exploración y la explotación:

   - Población (μ): 10 individuos.
   - Descendencia (λ): 10 hijos por generación.
   - Generaciones: 5 (se observó convergencia temprana en pruebas piloto).
   - Modelo LLM: gpt-4o-mini con temperatura τ = 0.5. Una temperatura media permite variación creativa en la mutación sin degradar la coherencia sintáctica del JSON.

Resultados y Análisis

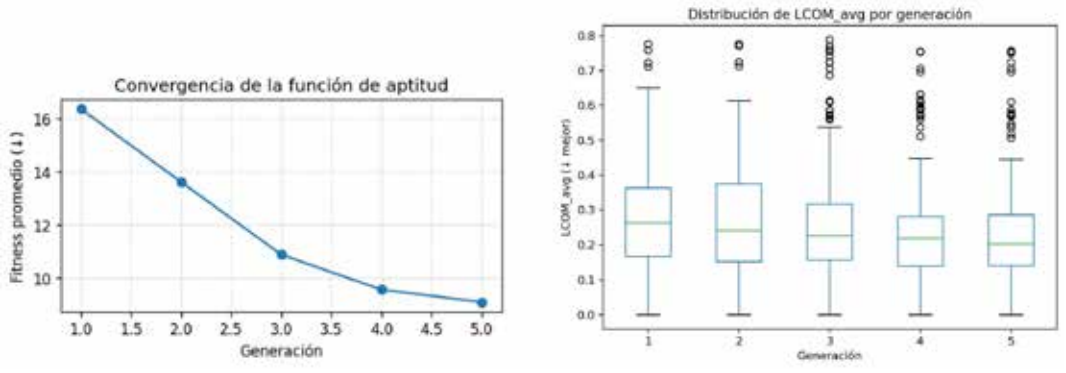
En esta sección se presentan los hallazgos empíricos derivados de la ejecución del pipeline sobre los 22 escenarios de prueba del dataset de (Dalpiaz, 2018). El análisis se estructura en dos niveles: primero, una evaluación **cuantitativa** global que examina el rendimiento del algoritmo evolutivo y la significancia estadística de las mejoras; y segundo, un examen **cualitativo** detallado que ilustra la coherencia lógica de las soluciones arquitectónicas generadas mediante un caso de estudio representativo.

Resultados Cuantitativos

A continuación, se presentan los hallazgos empíricos derivados de la ejecución sistemática del pipeline sobre los 22 escenarios del dataset de referencia. El análisis se enfoca en evaluar la eficacia del mecanismo evolutivo mediante dos dimensiones clave: la capacidad de optimización global de la función de aptitud y la distribución estadística de las métricas estructurales resultantes.

Convergencia del Fitness

La Figura 2a muestra la evolución promedio de la función de aptitud *f* (G) a lo largo de las generaciones. Se observa un descenso monótono y sostenido, pasando de una media de 16.34 en la Generación 1 a 9.10 en la Generación 5. Esto representa una mejora global del **44.3%** en la calidad arquitectónica según las métricas formalizadas.



(a) *Convergencia de la Función de Aptitud f (G).* **(b)** *Distribución de Cohesión (LCOM) por Gen.*

Figura 2: Análisis de convergencia a lo largo de 5 generaciones. La reducción en la media y la dispersión confirma la eficacia del operador de variación basado en LLM.

En paralelo, la cohesión promedio (LCOMavg) mejoró significativamente, bajando de 0.289 a 0.230 (Figura 2b). Esto indica que el sistema aprendió a agrupar operaciones que comparten parámetros, reduciendo la fragmentación funcional sin intervención humana directa.

Resumen de Resultados por Proyecto

La Tabla 2 presenta el detalle completo de los mejores genotipos obtenidos para cada uno de los 22 escenarios. Estos datos permiten observar la consistencia del enfoque a través de dominios diversos.

Significancia Estadística

Para validar si la mejora es producto del azar, aplicamos la prueba no paramétrica de Kruskal-Wallis. Los resultados ($H = 153.08$, $p \approx 4.4 \times 10^{-32}$) confirman diferencias estadísticamente significativas entre las distribuciones de las distintas generaciones. El análisis post-hoc de Dunn con corrección de Holm confirmó que la diferencia entre la Generación 1 y la Gene-ración 5 es significativa ($p < 0.001$). .

TABLA 2: Resumen de genotipos óptimos por proyecto. Se detallan las métricas estructurales y el valor de aptitud (fitness) alcanzado por el mejor individuo validado en la última generación para cada uno de los 22 escenarios.

Proyecto	#Serv.	LCOM	Coup	SGM	SD	Fit
g02-federalspending.txt	7	0.000	1.000	3.000	0.471	4.471
g17-cask.txt	4	0.000	1.000	5.000	0.000	6.000
g22-rdadmp.txt	4	0.000	2.000	5.000	0.471	7.471
g19-alfred.txt	7	0.095	1.000	4.000	1.366	6.461
g25-duraspace.txt	5	0.120	1.000	5.000	1.633	7.753
g11-nsf.txt	6	0.133	1.000	3.000	0.000	4.133
g14-datahub.txt	6	0.140	1.000	5.000	0.980	7.120
g13-planningpoker.txt	5	0.140	1.000	5.000	2.708	8.848
g18-neurohub.txt	8	0.146	1.000	4.000	2.376	7.522
g08-frictionless.txt	6	0.150	2.000	5.000	0.400	7.550
g24-unibath.txt	**6**	**0.167**	**1.000**	**4.000**	**2.853**	**8.020**
g27-culrepo.txt	6	0.200	1.000	5.000	2.555	8.755
g26-racdam.txt	4	0.213	1.000	5.000	0.871	7.084
g16-mis.txt	8	0.219	1.000	8.000	2.644	11.863
g03-loudoun.txt	7	0.238	1.000	5.000	1.804	8.042
g12-camperplus.txt	6	0.250	1.000	5.000	1.470	7.720
g21-badcamp.txt	7	0.286	1.000	4.000	0.971	6.257
g04-recycling.txt	6	0.294	1.000	5.000	1.690	7.984
g23-archivesspace.txt	6	0.298	1.000	**12.000**	0.844	14.142
g10-scrumalliance.txt	6	0.356	1.000	5.000	1.766	8.122
g05-openspending.txt	5	0.507	1.000	5.000	2.138	8.645
g28-zooniverse.txt	5	0.538	1.000	8.000	0.681	10.219

Análisis de Casos Atípicos y Heterogeneidad

Si bien la convergencia global fue positiva, la Tabla 2 revela comportamientos divergentes que iluminan las limitaciones del enfoque.

Outliers de Cohesión. En dominios con alta heterogeneidad funcional intrínseca, como g05-openspending y g28-zooniverse, el sistema convergió a valores de LCOM > 0.50. Esto sugiere que las historias de usuario en estos datasets describen operaciones disjuntas que

difícilmente pueden agruparse en servicios cohesivos sin violar restricciones de acoplamiento. En estos casos, ArchiGenMS priorizó mantener un bajo acoplamiento (Coupling = 1) a costa de una menor cohesión interna.

Outliers de Granularidad. El caso g23-archivesspace (resaltado en la tabla) presentó un servicio con SGMmax = 12. Aunque el validador formal no rechazó esta configuración (ya que no violaba invariantes duros), este valor sugiere la presencia de un God Service residual. Este hallazgo indica que la función de aptitud podría beneficiarse de penalizaciones no lineales para granularidades extremas (SGM > 10).

Análisis Cualitativo: Caso de Estudio g24-unibath

Para ilustrar la ïnteligencia"del diseño, analizamos el caso g24-unibath (Repositorio Institucional).

Evolución del Diseño. En la Generación 1, el LLM propuso un diseño ingenuo con un *God Service* llamado 'SystemManager' (12 operaciones, *LCOM* = 0.85) y acoplamiento cíclico entre 'Auth' y 'User'. El verificador Lean penalizó estas estructuras inválidas. Hacia la Generación 3, el mecanismo de inspiraciones guio al modelo a descomponer el monolito, pero surgieron "nano-servicios"de una sola operación, lo que elevó la penalización por acoplamiento.

Finalmente, en la Generación 5 (ver Figura 3), el sistema convergió en una topología estable de 6 servicios con LCOM = 0.16 y $\text{Coupling}_{\text{max}}$ = 1. Cualitativamente, observamos la emergencia del patrón **Database-per-Service** virtual: el servicio 'DataDeposit' encapsuló exclusivamente las operaciones de escritura (ingesta), mientras que 'DataRetrieval' centralizó las lecturas. Este refinamiento no fue programado explícitamente, sino que emergió de la presión selectiva ejercida por las métricas formales.

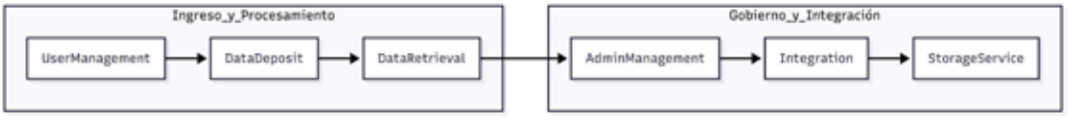


Figura 3: *Arquitectura generada para g24-unibath. Nótese la clara separación de responsabi-lidades entre Ingesta, Recuperación y Gestión.*

Discusión

Los resultados obtenidos confirman la viabilidad del enfoque híbrido y permiten validar las hipótesis planteadas al inicio del estudio. En esta sección, interpretamos la evidencia empírica a la luz de las cuatro hipótesis de investigación, contrastamos el desempeño de ArchiGenMS con el estado del arte y analizamos las amenazas a la validez.

Validación de Hipótesis e Interpretación

Sobre la Cohesión (H1). La reducción sostenida del *LCOMavg* (de 0.289 a 0.230) confirma la hipótesis **H1**. A diferencia de los enfoques basados puramente en similitud semántica como SEMGROMI (Vera-Rivera et al., 2023), que son vulnerables a la anisotropía de los *embeddings* (Pérez et al., 2025), nuestro enfoque evolutivo agrupó operaciones basándose en la coherencia de sus firmas de parámetros. El oráculo formal actuó como un discriminador efectivo, penalizando servicios que, aunque semánticamente afines, carecían de cohesión estructural.

Sobre la Granularidad (H2). Los datos respaldan parcialmente la hipótesis **H2**. La mayoría de los escenarios convergieron hacia servicios con un tamaño funcional equilibrado (*SGM* □ [2,5]), evitando la proliferación de microservicios anémicos. Sin embargo, la detección del caso atípico en g23-archivesspace (*SGMmax* = 12) indica que, sin una penalización no lineal estricta, el sistema puede tolerar "God Services" locales si estos contribuyen a minimizar el acoplamiento global. Esto sugiere una oportunidad para refinar la función de aptitud en trabajos futuros.

Sobre el Acoplamiento (H3). La convergencia del acoplamiento máximo hacia valores unitarios (Couplingmax ≈ 1.15) valida robustamente **H3**. ArchiGenMS demostró capacidad para descubrir topologías con dependencias mínimas, superando a los diseños manuales que a menudo introducen acoplamientos accidentales. Las excepciones observadas (como en g22-rdadmp) corresponden a dominios con interdependencias funcionales irreducibles, donde un mayor acoplamiento es consecuencia de la lógica del negocio y no de un defecto del algoritmo.

Sobre la Reproducibilidad (H4). Los intervalos de confianza estrechos observados en las curvas de convergencia (Figura 2) confirman la hipótesis **H4**. A pesar de la estocasticidad inherente a los LLMs, la varianza entre corridas independientes fue despreciable en las generaciones finales. Esto demuestra que la estrategia de Prompting con inspiraciones y la selección elitista logran estabilizar el proceso generativo, garantizando resultados reproducibles.

Posicionamiento en la Ingeniería de Software Basada en Búsqueda (SBSE)

Desde la perspectiva de la SBSE (Harman et al., 2012), ArchiGenMS representa una innovación metodológica: sustituye los operadores de mutación sintáctica aleatoria (típicos de los algoritmos genéticos clásicos) por un operador de variación semántica basado en LLMs. Esto permite explorar el espacio de diseño mediante "saltosïnteligentes y contextualizados, en lugar de caminatas aleatorias, lo que explica la rápida convergencia en apenas 5 generaciones frente a las cientos requeridas por métodos tradicionales.

Amenazas a la Validez

El rigor científico exige reconocer las limitaciones del diseño experimental.

Validez de Constructo. Existe una brecha ontológica entre la calidad estructural y la calidad operacional. Las métricas empleadas (LCOM, SGM, Fan-Out) son proxies estáticos consolidados en la literatura (Al-Debagy & Martinek, 2020), pero no capturan atributos dinámicos como la latencia, el throughput o la tolerancia a fallos. ArchiGenMS valida la solidez del "plano. ª rquitectónico, pero no garantiza el comportamiento del sistema bajo carga real.

Validez Interna. Aunque el control de semillas mitiga la variabilidad, la dependencia de un modelo de lenguaje específico (GPT-4o-mini) introduce un sesgo tecnológico. Cambios en la versión del modelo subyacente podrían alterar la eficacia del operador de mutación. No obstante, la arquitectura modular del pipeline permite la sustitución del motor generativo sin invalidar el método de verificación.

Validez Externa. Los experimentos se limitan al dataset de Dalpiaz (Dalpiaz, 2018), que contiene requisitos en inglés de dominios estándar. La generalización de estos hallazgos a entornos industriales con requisitos en otros idiomas, o con documentación técnica mixta y des-estructurada, debe realizarse con cautela y requiere validación adicional en estudios de caso empresariales.

Conclusiones y Líneas Futuras

Esta investigación ha completado un ciclo metodológico de *Design Science Research* (DSR) orientado a resolver la tensión entre la flexibilidad de la Inteligencia Artificial Generativa y el rigor exigido por la ingeniería de software. Se partió del problema de relevancia industrial: la dificultad de descubrir límites de microservicios coherentes en escenarios *greenfield* donde la única fuente de información son requisitos textuales ambiguos. Frente a los enfoques manuales no verificables y las alucinaciones estructurales de los LLMs, el artefacto resultante, **ArchiGenMS**, demuestra que es posible sistematizar el diseño arquitectónico mediante un ciclo híbrido de generación evolutiva y verificación formal.

Síntesis de Aportaciones

Desde una perspectiva teórica, este trabajo ha contribuido con una formalización operativa de métricas arquitectónicas clásicas (LCOM, SGM, SGM-SD y Fan-Out). A diferencia de su uso tradicional como indicadores post-mortem, aquí se han redefinido como funciones totales sobre grafos dirigidos en Lean 4, permitiendo su cálculo determinista en presencia de estructuras incompletas. Esta formalización cierra la brecha entre las definiciones conceptuales de la literatura y su implementación computable, habilitando pruebas automáticas de propiedades estructurales como la integridad referencial y la irreflexividad de las llamadas.

En el plano metodológico, la principal innovación reside en la definición de un esquema evolutivo asistido por LLMs que supera las limitaciones de los operadores genéticos tradicionales. Al utilizar el modelo de lenguaje como un motor de variación semántica —guiado por una estrategia de *prompting* con inspiraciones—, el sistema logra explorar el espacio de

diseño de manera inteligente, proponiendo refactorizaciones que respetan la semántica del dominio (como la separación de responsabilidades de lectura/escritura) sin perder la coherencia sintáctica. Esta sinergia entre exploración generativa y restricciones formales reduce significativamente la deriva semántica habitual de los modelos generativos.

La evidencia empírica, obtenida sobre 22 escenarios del dataset de referencia de Dalpiaz, valida la robustez del enfoque. La reducción sostenida del 44% en la función de aptitud y la convergencia hacia arquitecturas con un acoplamiento máximo unitario (Couplingmax ≈ 1.15) confirman estadísticamente que el oráculo formal actúa eficazmente como un gradiente de calidad. Más allá de los números, el análisis cualitativo demuestra que las arquitecturas resultantes no son solo grafos optimizados matemáticamente, sino diseños lógicamente coherentes alineados con los flujos del negocio.

Limitaciones del Estudio

La interpretación de estos resultados debe considerar ciertas limitaciones inherentes al diseño experimental. En primer lugar, la validez de construcción se circunscribe a la calidad estructural estática. Las métricas empleadas (cohesión y acoplamiento) son proxies necesarios pero no suficientes de la calidad del software; aspectos operacionales críticos como la latencia, el throughput y la tolerancia a fallos quedan fuera del alcance de una validación estática. ArchiGenMS valida el "plano.[a] rquitectónico, no el edificio bajo carga.

En segundo lugar, existe una amenaza a la validez externa relacionada con la cobertura de datos. Aunque el dataset utilizado es el estándar en la comunidad de ingeniería de requisitos, se limita a descripciones en inglés y dominios académicos o de gobierno abierto. La extrapolación de los resultados a contextos industriales con documentación técnica mixta (diagramas, especificaciones legadas) o en otros idiomas requiere cautela y validación adicional.

Agenda de Investigación Futura

A partir de los hallazgos y limitaciones expuestas, se proyectan tres líneas de investigación prioritarias para evolucionar el estado del arte en AI4SE:

De la Verificación Estructural a la Simulación Dinámica. La evolución natural de este trabajo es la transición de un pipeline "Generar-Verificar.[a] uno "Generar-Verificar-Simular". Se propone que los candidatos que superen el filtro formal de Lean sean instanciados automáticamente en simuladores de eventos discretos o entornos de orquestación (e.g., Kubernetes). Esto permitiría evaluar atributos de calidad dinámicos (NFRs) y retroalimentar la función de aptitud con datos de ejecución, cerrando el ciclo entre diseño y operación.

Automatización Formal Avanzada. Existe un vasto potencial en profundizar el uso de Lean 4 más allá del cálculo de métricas. Futuras iteraciones podrían demostrar propiedades globales del sistema, como la aciclicidad garantizada del grafo de dependencias o el cumplimiento de políticas de seguridad de flujo de información entre bounded contexts.

La co-evolución de la arquitectura junto con sus pruebas formales es una vía prometedora hacia el desarrollo de software çorrecto por construcción".

Transferencia y Validación Industrial. Finalmente, para consolidar la utilidad práctica del enfoque, es necesario enfrentar la complejidad de los requisitos del mundo real. Esto implica extender el *Prompt Sampler* para procesar entradas multimodales y multilingües, y empaquetar el verificador formal como una herramienta de integración continua (CI) que asista a los arquitectos humanos en la detección temprana de deuda técnica estructural.

En conclusión, esta tesis sostiene que la inteligencia artificial no debe reemplazar el rigor ingenieril, sino potenciarlo. La integración de modelos generativos con métodos formales ofrece un camino pragmático para automatizar el diseño de sistemas complejos, asegurando que la velocidad de la generación no comprometa la solidez de la arquitectura.

Referencias

Al-Debagy, O., & Martinek, P. (2020). A metrics framework for evaluating microservices architecture designs. *Journal of Web Engineering*, *19*(3–4), 341-370.

Bajaj, D., Bharti, U., Gupta, I., Gupta, P., & Yadav, A. (2024). GTMicro—Microservice identification approach based on deep NLP transformer model for greenfield developments. *International Journal of Information Technology*, *16*(5), 2751-2761.

Bajaj, D., Goel, A., & Gupta, S. C. (2022). GreenMicro: identifying microservices from use cases in greenfield development. *IEEE Access*, *10*, 67008-67018.

Battaglia, N., García, A. N., & Congiusti, A. (2024). Descubrimiento de Microservicios en Metodologías Ágiles: un mapeo sistemático de la literatura. *XXX Congreso Argentino de Ciencias de la Computación (CACIC)(La Plata, 7 al 11 de octubre de 2024)*.

Clarke, E. M., & Wing, J. M. (1996). Formal methods: State of the art and future directions. *ACM Computing Surveys (CSUR)*, *28*(4), 626-643.

Clarke, E., Grumberg, O., & Peled, D. A. (1999). Model checking the mit press. *Cambridge, Massachusetts, London, UK*, *988*.

Dalpiaz, F. (2018, julio). *Requirements data sets (user stories)*. Mendeley Data. https://doi.org/10.17632/7zbk8zsd8y.1

Esposito, M., Li, X., Moreschini, S., Ahmad, N., Cerny, T., Vaidhyanathan, K., Lenarduzzi, V., & Taibi, D. (2025). Generative AI for Software Architecture. Applications, Trends, Challenges, and Future Directions. *arXiv preprint arXiv:2503.13310*.

Gross, J. L., Yellen, J., & Anderson, M. (2018). *Graph theory and its applications*. Chapman; Hall/CRC.

Harman, M., Mansouri, S. A., & Zhang, Y. (2012). Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)*, *45*(1), 1-61.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS quarterly*, 75-105.

Kalia, A. K., Xiao, J., Krishna, R., Sinha, S., Vukovic, M., & Banerjee, D. (2021). Mono2micro: a practical and effective tool for decomposing monolithic java applications to microservices. *Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, 1214-1224.

Mitchell, B. S., & Mancoridis, S. (2008). On the evaluation of the bunch search-based software modularization algorithm. *Soft Computing*, *12*(1), 77-93.

Moura, L. d., & Ullrich, S. (2021). The lean 4 theorem prover and programming language. *International Conference on Automated Deduction*, 625-635.

Narváez, D. (2025). ArchiGenMS: Reproducible Package [Accedido: 22 jul. 2025].

Narváez, D., Battaglia, N., Fernández, A., & Rossi, G. (2025a). Designing microservices using ai: A systematic literature review. Software, 4(1), 6.

Narváez, D., Battaglia, N., Fernández, A., & Rossi, G. (2025b). Descubrimiento automático de microservicios mediante modelos generativos y verificación formal. XXXI Congreso Argentino de Ciencias de la Computación (CACIC) (Viedma, 6 al 10 de octubre de 2025).

Narváez, D., Rossi, G. H., & Battaglia, N. (2024). Aplicación de inteligencia artificial en el diseño de microservicios. XXX Congreso Argentino de Ciencias de la Computación (CACIC)(La Plata, 7 al 11 de octubre de 2024).

Neri, D., Soldani, J., Zimmermann, O., & Brogi, A. (2020). Design principles, architectural smells and refactorings for microservices: a multivocal review. SICS Software-Intensive Cyber-Physical Systems, 35(1), 3-15.

Newman, S. (2021). Building microservices: designing fine-grained systems. .^{O}'Reilly Media, Inc."

Pérez, G., Mostaccio, C., & Antonelli, L. (2025). Análisis comparativo de arquitecturas de NLP para detectar similitudes entre escenarios en español. *Workshop on Requirements Engineering (WER)*.

Stojanovic, T., & Lazarevi´c, S. D. (2023). The application of ChatGPT for identification of microservices. E-business technologies conference proceedings, 3(1), 99-105.

Taibi, D., Lenarduzzi, V., & Pahl, C. (2017). Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. *IEEE Cloud Computing*, *4*(5), 22-32.

Taibi, D., Lenarduzzi, V., Pahl, C., & Janes, A. (2017). Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages. Proceedings of the XP2017 Scientific Workshops, 1-5.

Taibi, D., & Systä, K. (2019). A Decomposition and Metric-Based Evaluation Framework for Microservices. https://arxiv.org/abs/1908.08513

Ünlü, H., Kennouche, D. E., Soylu, G. K., & Demirörs, O. (2024). Microservice-based projects in agile world: A structured interview. Information and Software Technology, 165, 107334.

Velepucha, V., & Flores, P. (2023). A survey on microservices architecture: Principles, patterns and migration challenges. IEEE access, 11, 88339-88358.

Vera-Rivera, F. H., Cuadros, E. G. P., Perez, B., Astudillo, H., & Gaona, C. (2023). SEM-GROMI—a semantic grouping algorithm to identifying microservices using semantic similarity of user stories. PeerJ Computer Science, 9, e1380.

Zhong, C., Li, S., Huang, H., Liu, X., Chen, Z., Zhang, Y., & Zhang, H. (2024). Domaindriven design for microservices: An evidence-based investigation. IEEE Transactions on Software Engineering, 50(6), 1425-1449.