Generación de Pruebas Unitarias con LLMs en Entornos Industriales: Desafíos, Evolución y Lecciones Prácticas

Eneko Pizarro[1], **Maider Azanza**[2], **Beatriz Pérez Lamancha**[1]

[1]LKS Next
Goiru, 7
Arrasate-Mondragón, España

[2]Universidad del País Vasco (EHU/UPV)
Facultad de Informática
Donostia-San Sebastian, España

{epizarro,bperez}@lksnext.com, maider.azanza@ehu.eus

Abstract. *Large Language Models (LLMs) show potential for automatically generating unit tests, but their industrial application presents challenges. We present a longitudinal case study on the implementation and evaluation of LLMs for test generation at LKS Next, integrating standard tools such as SonarQube. Our approach reveals findings on the temporal evolution of these technologies in production environments and provides learned lessons. The results offer evidence-based industrial guidance for organizations considering adopting these solutions, highlighting practical integration and maintainability considerations often absent in theoretical studies.*

Resumen. *Los Modelos de Lenguaje de Gran Escala (LLMs) muestran potencial para generar pruebas unitarias automáticamente, pero su aplicación industrial genera desafíos. Presentamos un caso de estudio longitudinal sobre la implementación y evaluación de LLMs para generación de pruebas en la empresa LKS Next, integrando herramientas estándar como SonarQube. Nuestro enfoque revela hallazgos sobre la evolución temporal de estas tecnologías en entornos de producción y proporciona lecciones aprendidas. Los resultados ofrecen una guía industrial basada en evidencia para organizaciones que consideran adoptar estas soluciones, destacando consideraciones prácticas de integración y mantenibilidad a menudo ausentes en estudios teóricos.*

1. Introducción

Las pruebas unitarias continúan siendo un aspecto crítico y a la vez costoso en recursos dentro del desarrollo de software. Las prácticas modernas de desarrollo como DevSecOps [Prates and Pereira 2025] demandan tanto velocidad como fiabilidad en la entrega continua del software. Esto implica contar con un conjunto completo de pruebas unitarias que asegure la calidad del código. Sin embargo, estudios recientes revelan que la escritura de pruebas unitarias es una de las tareas que los desarrolladores disfrutan menos y que preferirían delegar a la inteligencia artificial [Sergeyuk et al. 2025], siendo además un proceso costoso para la organización.

Debido a esto, con la emergencia de la *Inteligencia Artificial Generativa (IAGen)* o, más concretamente *Modelos de Lenguaje de Gran Escala (LLMs por sus siglas en*

inglés), las empresas están explorando su potencial para automatizar la generación de pruebas unitarias y reducir la carga de trabajo de los desarrolladores. Si bien investigaciones recientes han estudiado extensamente las capacidades de generación de pruebas mediante LLMs [Wang et al. 2024b], su adopción industrial genera desafíos que van más allá de los abordados en estudios académicos.

Un reto clave para las empresas que adoptan estas herramientas radica en su rápida evolución: lo que era cierto sobre las capacidades de una herramienta hace seis meses puede no reflejar su rendimiento actual. Esto es particularmente desafiante para organizaciones que implementan *pipelines* de integración/entrega continua *(CI/CD)*, donde la generación de pruebas unitarias debe integrarse perfectamente con los flujos de trabajo existentes y mantener estándares de calidad consistentes, sin estar a expensas de la evolución de las capacidades de los LLM subyacentes.

Este trabajo presenta un marco práctico para la evaluación continua de generadores de pruebas unitarias basados en LLMs en entornos industriales, ilustrado mediante un estudio longitudinal de las capacidades de GitHub Copilot en LKS Next.

Las contribuciones principales de este trabajo son:

1. Un marco práctico de medición aplicado en un entorno industrial para la evaluación continua de LLMs en la generación de pruebas unitarias.
2. Un análisis longitudinal que muestra la evolución temporal de las capacidades de los LLMs en la generación de pruebas en un contexto productivo.
3. Lecciones aprendidas para otras organizaciones que estén considerando la adopción de los LLMs para la automatización de pruebas unitarias.

2. Motivación: el Caso de LKS Next

LKS Next[1], la organización donde se realiza esta investigación, es una empresa de consultoría tecnológica que ofrece soluciones personalizadas utilizando diversas tecnologías y lenguajes de programación. La empresa sigue metodologías ágiles y prácticas DevSecOps [Prates and Pereira 2025] para desarrollar software seguro y de alta calidad que sea mantenible durante todo su ciclo de vida. En este contexto, las pruebas representan un desafío crítico. La empresa debe generar suficientes pruebas para garantizar la calidad de sus proyectos, equilibrando la rigurosidad con la eficiencia en costos y tiempo, para optimizar su salida al mercado.

Las pruebas automatizadas en entornos DevSecOps proporcionan ventajas estratégicas y operativas significativas, garantizando la verificación consistente y repetible de la funcionalidad del software. Este enfoque sistemático minimiza los tiempos de ejecución, mide la cobertura de código y facilita la integración continua, proporcionando retroalimentación inmediata sobre los cambios. La infraestructura DevSecOps utiliza en los proyectos de la empresa, ejecuta automáticamente con cada *commit* una serie de pasos (*pipeline de desarrollo*) que incluyen: compilación, ejecución de pruebas unitarias y de integración, análisis de la calidad y seguridad del código y verificación de la seguridad de sus dependencias. A continuación, se despliega la nueva versión al entorno correspondiente (desarrollo, pruebas, etc.).

[1] https://www.lksnext.com/es/

La irrupción de herramientas de IA, particularmente los LLM, promete soluciones que parecen mejorar mucho la productividad en todas las áreas del desarrollo del software, incluyendo el testing. Las empresas sienten la presión de adoptar herramientas de IA generativa en los desarrollos, pero requieren garantías de sus resultados antes de utilizarlas en entornos productivos. Hasta ahora, el código generado por la IA requería una revisión y modificación significativas para cumplir con los estándares de la organización, pero esto ha ido cambiando a medida que aparecen nuevas herramientas o nuevas versiones de las mismas.

A esto se suma la proliferación de herramientas de IA generativa, que implica decidir qué herramienta es la más adecuada. El modelo de pago por token introduce costes variables que escalan con el uso, por lo que las organizaciones deben equilibrar productividad, los gastos en IA y la confiabilidad en la solución resultante. Es por ello que las empresas requieren mecanismos para validar y revisar el código generado por IA. En el caso del presente trabajo, el objetivo es la generación de pruebas unitarias, ya que juegan un papel crucial en los procesos DevSecOps.

Establecer un marco de referencia para poder comparar los LLMs y sus versiones en el tiempo resulta crucial para las empresas. Este marco de referencia debe tener en cuenta las técnicas de generación de casos de prueba, tanto estructurales como basadas en conocimiento de expertos, midiendo la cobertura de las pruebas generadas, el grado de parametrización y el grado de aislamiento de las pruebas unitarias mediante técnicas de dobles de pruebas (*mocks*).

Para abordar estos desafíos en un contexto industrial real, desarrollamos y aplicamos un marco sistemático de evaluación que presentamos a continuación.

3. Estructura del Marco: Métricas y Categorías de Evaluación

El acelerado desarrollo de las capacidades de los LLMs requiere un marco sólido, reproducible y longitudinal para evaluar la calidad de las pruebas generadas. Nuestro enfoque responde a los desafíos específicos de las pruebas generadas por inteligencia artificial a través de un sistema de evaluación que prioriza la reproducibilidad, la integración con herramientas industriales estándar y facilita la evaluación continua a lo largo del tiempo.

Para analizar la capacidad de los LLMs en la generación de pruebas unitarias, hemos desarrollado un conjunto de métricas organizadas en tres categorías: (1) métricas de calidad de código, (2) métricas estructurales y (3) métricas basadas en el conocimiento de expertos. Estas métricas se integran mediante un modelo de ponderación que permite calcular una valoración cuantitativa para cada LLM, facilitando la comparación objetiva entre diferentes modelos y sus versiones a lo largo del tiempo.

3.1. Métricas de Calidad de Código

Las métricas de calidad del código se centran en dos aspectos: la compilación y el análisis estático de las pruebas **IAGen**. La compilación representa el nivel más básico de evaluación y, aunque seguramente se elimine del marco de evaluación en el futuro dada la mejora en las pruebas IAGen (véase el apartado *Resultados*), resulta relevante ya que nos permite apreciar el progreso que los distintos LLM han tenido a lo largo de la toma de las mediciones. En nuestras pruebas iniciales observamos que los LLM introducían frecuen-

Tabla 1. Métricas de calidad de código

Métrica	Definición	Medición
Incidencias de compilación	Mide problemas en el uso de bibliotecas externas y declaraciones de importación, uso incorrecto de elementos específicos del proyecto (clases, métodos, constructores), problemas de corrección sintáctica y estructural básica del código de prueba generado y cantidad de errores de compilación encontrados en las pruebas IAGen.	Cantidad de errores de compilación encontrados en las pruebas IAGen
Incidencias de análisis estático de código	Mide incidencias encontradas con la herramienta Sonarqube, siguiendo el perfil Sonar-Way [Campbell and Papapetrou 2013]. Calcula la mantenibilidad, confiabilidad y seguridad del código de las pruebas IAGen.	Cantidad de incidencias encontradas en el análisis de código estático de las pruebas IAGen

temente fallos de compilación en las clases de las pruebas, evitando poder tomar métricas de ejecución.

Por otro lado, es importante evaluar la calidad del código generado, para lo cual utilizamos la herramienta de análisis estático de código Sonarqube[2], que proporciona reglas específicas para analizar la mantenibilidad, confiabilidad y seguridad del código generado. Herramientas como Sonarqube son especialmente importantes en entornos industriales, ya que permiten hacer un seguimiento de la calidad del código previo a la puesta en producción como parte integral de los desarrollos DevSecOps.

La Tabla 1 describe las métricas *incidencias de compilación* e *incidencias de análisis estático de código* y cómo se miden. Consideramos que esta categoría de métricas es objetiva, ya que los resultados serán los mismos, sin depender de un juicio experto que los valide. Además, consideramos que las métricas en esta categoría penalizan al resultado final del LLM siendo evaluado, ya que a mayor cantidad de errores de compilación o mayor cantidad de incidencias de calidad de código, peores son las pruebas IAGen.

3.2. Métricas Estructurales

Las métricas estructurales evalúan el grado en que las pruebas IAGen ejercitan el código bajo prueba. Estas métricas miden aspectos de cobertura del código siguiendo distintas técnicas. Además, hemos incorporado una métrica para determinar el grado de aislamiento en pruebas unitarias que alcanzan las pruebas IAGen.

La Tabla 2 muestra las cuatro métricas que utilizamos en este apartado. Las tres primeras tienen que ver con la cobertura de código, partiendo de la más simple, la *cobertura de líneas*, que indica las líneas de código que han sido ejecutadas con las pruebas IAGen. Este criterio es necesario pero no suficiente, por lo que además hemos incluido la *cobertura de condiciones*, donde se evalúa que cada condición haya sido evaluada como verdadera y falsa. Un criterio aún más restrictivo es el de *cobertura de condición/decisión*, donde se pide que todas las decisiones dentro de una condición sean probadas al ejecutar las pruebas [H. Washizaki 2024].

[2] https://www.sonarsource.com/products/sonarqube

Tabla 2. Métricas estructurales

Métrica	Definición	Medición
Cobertura de líneas	Porcentaje de líneas de código que se han ejecutado con las pruebas IAGen [ISO 2021]	(Cantidad de líneas ejecutadas por los test IAGen / Total de líneas ejecutables) × 100
Cobertura de condiciones	Pocentaje de condiciones que se han ejecutado con las pruebas IAGen. Prueba que cada condición del código sea ejecutada al menos una vez [ISO 2021]	(Cantidad de condiciones evaluadas por las pruebas IAGen/ Total de condiciones) × 100
Cobertura de condicion/decisión	Pocentaje de condiciones y decisiones que se han ejecutado con las pruebas IAGen. Prueba cada decisión dentro de una condición. [ISO 2021]	(Cantidad de condiciones y decisiones ejecutadas por las pruebas IAGen / Total de condiciones y decisiones) × 100
Aislamiento de las pruebas	Evalúa cuan bien las pruebas IAGen crean objetos simulados (mocks, stubs, etc.) que al ejecutarse cuenten con todo lo que necesita de su entorno. [Freeman et al. 2004]	(Cantidad de pruebas IAGen aisladas / Cantidad de pruebas aisladas que se esperaban) × 100

Para la evaluación de estas métricas de cobertura, existen herramientas que facilitan su cómputo automático. En nuestro caso, para el lenguaje Java, hemos empleado Jacoco [3], que proporciona reportes detallados sobre la cobertura alcanzada por las pruebas IAGen en cada una de estas dimensiones.

Además de la cobertura, es esencial considerar el aislamiento en las pruebas unitarias. Por definición, estas se centran en probar clases individuales y garantizar su correcto funcionamiento, controlando todos los aspectos del contexto en el cual la clase bajo prueba es ejecutada. Para lograr este aislamiento, se reemplazan los colaboradores reales por dobles de pruebas (”test doubles”), implementaciones simuladas que separan el código bajo prueba de dependencias externas como capas de servicios, sistemas y recursos [Kaczanowski 2013].

En este contexto, hemos definido la métrica *Aislamiento de las pruebas* para evaluar si las pruebas IAGen manejan apropiadamente estas dependencias, aislando el código de distracciones introducidas por servicios web, sistemas de archivos, bases de datos o software de terceros, mientras validan que el código aislado funciona según lo esperado. Esta métrica también tiene carácter objetivo, puesto que las pruebas unitarias sin un aislamiento adecuado no podrán ejecutarse correctamente en entornos de integración continua, donde frecuentemente no se dispone del entorno completo de ejecución.

3.3. Métricas Basadas en el Conocimiento de Expertos

Estas métricas se fundamentan en los requisitos y especificaciones del sistema, utilizando técnicas de caja negra. Su objetivo principal es determinar en qué medida las pruebas IAGen consideran los aspectos funcionales del código bajo prueba. A diferencia de las categorías anteriores, que se basan en criterios objetivos, la categoría de *métricas basadas en el conocimiento de expertos* incorpora elementos subjetivos que requieren validación especializada.

[3]https://github.com/jacoco/jacoco

Tabla 3. Métricas basadas en el conocimiento de expertos

Métrica	Definición	Medición
Cobertura de Partición de Equivalencia	Una clase de equivalencia es un subconjunto de los datos que se consideran iguales por el sistema bajo prueba [ISO 2021]. Evalúa qué porcentaje de la clases de equivalencia identificadas por expertos han sido cubiertas por las pruebas IAGen	(Cantidad de particiones de equivalencia identificadas por las pruebas IAGen / Total de clases de equivalencia identificadas por expertos)x100
Cobertura de Valores Límites	El análisis de valor límite se centra en los límites de las clases de equivalencia. Se prueban valores por debajo y por encima de dichos límites [ISO 2021]. Evalúa qué porcentaje de los valores límites identificados por expertos han sido cubiertos por las pruebas IAGen	(Cantidad de valores límites encontrados por las pruebas IAGen / Total de valores límites identificados por expertos) × 100
Parametrización de las pruebas	La parametrización de las pruebas unitarias y de integración permite reutilizar pruebas con distintos conjuntos de datos [Tillmann and Schulte 2005]. Evalúa que porcentaje de parametrizaciones han sido realizadas correctamente por las pruebas IAGen	(Cantidad de prueba IAGen que se han parametrizado correctamente / Cantidad de pruebas que se esperaba parametrizar) × 100
Cobertura de Pruebas de Referencia	Porcentaje de pruebas unitarias generadas por IAGen que replican pruebas manuales de expertos, las cuales cumplen con los estándares mínimos de calidad de la empresa. Evalúa la capacidad del LLM para igualar el enfoque y rigor de desarrolladores en un entorno industrial.	(Cantidad de pruebas replicadas por la IA Gen / Cantidad de pruebas manuales generadas por expertos) x 100

Para establecer una base de comparación o *ground truth*, hemos desarrollado pruebas manuales de referencia por cada caso analizado, avaladas por expertos de la empresa. Este enfoque nos permite contrastar las pruebas generadas por IAGen con aquellas que se obtendrían mediante un desarrollo manual en el mismo entorno empresarial. Si bien estas métricas incluyen juicios subjetivos utilizando técnicas de caja negra—que pueden considerar distintos valores de prueba según el criterio del tester—ofrecen una perspectiva valiosa sobre la calidad práctica de las pruebas automatizadas.

La Tabla 3 describe las métricas que utilizamos en este apartado. La técnica de *Partición de Equivalencia* divide los datos de entrada en clases que el sistema bajo prueba procesa de la misma manera. Complementariamente, la técnica de *Análisis de Valores Límites* se centra en examinar los bordes de estas clases de equivalencia, evaluando valores cercanos a los límites donde suelen ocurrir más errores [ISO 2021]. La efectividad de ambas técnicas depende fundamentalmente de la correcta identificación de las clases de equivalencia, tarea que requiere el juicio experto de profesionales en pruebas.

Particularmente importante en entornos industriales resulta la *Parametrización de las pruebas*, que permite reutilizar las pruebas con diferentes conjuntos de datos para verificar su funcionamiento bajo diversas condiciones. Las pruebas unitarias parametrizadas mejoran la reutilización, mantenibilidad y documentación del proceso de pruebas [Tillmann and Schulte 2005], aspectos cruciales para la sostenibilidad a largo plazo de los juegos de pruebas.

	Categoría	Métrica Final para valorar el LLM	Peso	Fórmula para evaluar cada ejemplo del LLM	Valoración del LLM
Métricas Objetivas	Métricas de calidad de código	Total de Incidencias de compilación (IC)*	-20%	IC para el LLM = ∑(IC para cada ejemplo del LLM)	Valoración IC para el LLM = -(IC para el LLM / MAXIMO(IC para todos los LLMs))x20%
		Total de Incidencia de Análisis de Código (IAC)*	-5%	IAC para el LLM = ∑(IAC para cada ejemplo del LLM)	Valoración IAC para el LLM = -(IAC para el LLM / MAXIMO(IAC para todos los LLMs))x5%
	Métricas Estructurales	Cobertura de Líneas (CL)	50%	CL para el LLM = ∑(CL para cada ejemplo del LLM) / Total de ejemplos	Valoración de Métricas Estructurales para el LLM = PROMEDIO(CL para el LLM, CC para el LLM, CCD para el LLM, AP para el LLM)x50%
		Cobertura de Condiciones (CC)		CC para el LLM = ∑(CC para cada ejemplo del LLM) / Total de ejemplos	
		Cobertura de Condicion/ Decisión (CCD)		CCD para el LLM = ∑(CCD para cada ejemplo del LLM) / Total de ejemplos	
		Aislamiento de las Pruebas (AP)		AP para el LLM = ∑(AP para cada ejemplo del LLM) / Total de ejemplos	
Métricas subjetivas (Juicio de expertos)	Métricas basadas en el conocimiento de expertos	Cobertura de Partición de Equivalencia (CPE)	50%	CPE para el LLM = ∑(CPE para cada ejemplo del LLM) / Total de ejemplos	Valoración de Métricas basadas en la Especificación para el LLM = PROMEDIO(CPE para el LLM, CVL para el LLM, PP para el LLM, CPR para el LLM)x50%
		Cobertura de Valores Límites (CVL)		CVL para el LLM = ∑(CVL para cada ejemplo del LLM) / Total de ejemplos	
		Prametrización de las pruebas (PP)		PP para el LLM = ∑(PP para cada ejemplo del LLM) / Total de ejemplos	
		Cobertura de Pruebas de Referencia (CPR)		CPR para el LLM = ∑(CPR para cada ejemplo del LLM) / Total de ejemplos	

Figura 1. Modelo de ponderación para valorar cada LLM

Finalmente, la *Cobertura de Pruebas de Referencia* establece una comparación directa entre las pruebas generadas automáticamente y aquellas desarrolladas manualmente por los expertos, las cuales cumplen con los estándares mínimos de calidad establecidos por la empresa. Esta métrica evalúa la capacidad del LLM para replicar patrones, escenarios y enfoques considerados fundamentales por desarrolladores experimentados en un entorno industrial. Además, permite identificar discrepancias entre los enfoques automatizados y manuales [Kracht et al. 2014], señalando posibles áreas de mejora en la generación de pruebas mediante LLM.

3.4. Modelo de Ponderación de Métricas de Prueba

Para evaluar de manera integral la calidad de las pruebas generadas por LLMs, hemos desarrollado un modelo de ponderación que combina las métricas previamente descritas en una valoración cuantitativa consolidada. Este modelo equilibra factores objetivos y subjetivos, permitiendo comparaciones sistemáticas entre diferentes modelos LLM y sus evoluciones temporales.

El modelo estructura las métricas en las tres categorías fundamentales ya presentadas:

- *Métricas de calidad de código*: Evalúan aspectos técnicos básicos como la compilabilidad y la calidad estructural del código generado.
- *Métricas estructurales*: Miden la efectividad de las pruebas para ejercitar el código bajo análisis y su capacidad de aislamiento.
- *Métricas basadas en el conocimiento de expertos*: Valoran aspectos funcionales y de diseño que requieren juicio especializado.

Para reflejar tanto criterios objetivos como subjetivos, asignamos una ponderación equitativa del 50 % a las métricas objetivas (combinando calidad de código y estructurales) y 50 % a las métricas subjetivas (basadas en conocimiento experto). Esta distribución equilibrada reconoce la importancia complementaria de ambas dimensiones en la evaluación de pruebas unitarias en entornos industriales.

Un aspecto distintivo de nuestro modelo es el tratamiento de las *Métricas de calidad de código* como factores de penalización. Dado que los errores de compilación imposibilitan la ejecución de pruebas, la métrica *incidencias de compilación* reduce un 20 % del puntaje total. Similarmente, la métrica *incidencias del análisis estático de código* penaliza un 5 % adicional, reflejando su impacto en la mantenibilidad a largo plazo.

La Figura 1 muestra el modelo completo con las ponderaciones asignadas a cada métrica y las fórmulas utilizadas para los cálculos. Para las métricas de penalización (IC e IAC), se realiza la sumatoria de incidencias detectadas en todos los ejemplos evaluados. Para las demás métricas, se calcula la puntuación promedio de los valores obtenidos en los ejemplos.

Es importante destacar que los pesos específicos reflejan las prioridades de LKS Next, y el marco está diseñado para ser adaptable. Otras empresas pueden ajustar estas ponderaciones según sus propios criterios de calidad y objetivos estratégicos, manteniendo la estructura general del modelo como base para evaluaciones comparativas coherentes.

4. Aplicación Práctica: Procedimiento y Herramientas

Para abordar los desafíos de evaluación de pruebas unitarias generadas por LLMs en un contexto industrial real, hemos definido una metodología sistemática que permite tanto la evaluación objetiva como la mejora progresiva de los resultados. Nuestro enfoque se caracteriza por su naturaleza iterativa, su integración con herramientas industriales estándar y su capacidad para adaptarse a la rápida evolución de las capacidades de los LLMs. A continuación, detallamos este proceso, comenzando por la estructura general del flujo de trabajo de evaluación.

4.1. Proceso y Entorno Iterativo de Evaluación

El proceso inicia con una fase de preparación donde configuramos el entorno de desarrollo, establecemos las dependencias y herramientas de compilación e implementamos la infraestructura de recopilación de datos para el seguimiento de métricas.

Un componente crucial de esta fase preparatoria es el desarrollo de pruebas de referencia por parte de expertos. Estas pruebas sirven como estándar de comparación (*ground truth*) y se crean siguiendo las mejores prácticas de la organización, como se ha detallado en secciones anteriores.

Una vez completada la preparación, iniciamos un ciclo iterativo estructurado que consta de cuatro pasos principales:

1. **Generación de Pruebas:** Utilizamos el LLM con el prompt actual (inicial o refinado) para generar pruebas unitarias para el código objetivo.
2. **Análisis Multidimensional:** Evaluamos las pruebas generadas mediante todas las métricas definidas en nuestro marco, contrastándolas directamente con las pruebas de referencia para las métricas basadas en conocimiento experto.
3. **Identificación de Deficiencias:** Determinamos qué aspectos específicos de las pruebas generadas difieren de las pruebas de referencia, como la falta de cobertura en ciertas condiciones, la ausencia de particiones de equivalencia relevantes, o deficiencias en el aislamiento.

4. **Refinamiento del Prompt:** Modificamos el prompt para abordar explícitamente las deficiencias identificadas. Por ejemplo, si las pruebas de referencia utilizan dobles de prueba para aislar componentes externos mientras las pruebas IAGen no lo hacen, refinamos el prompt para enfatizar esta técnica.

Este ciclo se repite hasta que: (1) las pruebas generadas se aproximan suficientemente a la calidad de las pruebas de referencia según nuestro modelo de ponderación, o (2) se alcanza un punto donde refinamientos adicionales no producen mejoras significativas.

Las pruebas de referencia cumplen así una doble función: como parámetro de evaluación contra el cual se mide el rendimiento de las pruebas generadas, y como guía para el refinamiento específico de los prompts. Esta estrategia nos permite abordar sistemáticamente las brechas de calidad y mejorar progresivamente la capacidad de los LLMs para generar pruebas que sean aplicables en entornos industriales.

4.2. Selección de Ejemplos a Probar y Pruebas de Referencia

Para garantizar una evaluación robusta y significativa, hemos aplicado criterios específicos en la selección de los ejemplos de prueba. Primero, seleccionamos código que no contara con pruebas preexistentes, ya que nuestras evaluaciones exploratorias iniciales revelaron un fenómeno de ”fuga de datos”(*data leakage*) donde los LLMs, al estar entrenados con repositorios públicos, reproducían pruebas existentes en lugar de generar nuevas [López et al. 2025, Wu et al. 2024]. Esto habría distorsionado nuestra evaluación, ofreciendo una visión artificialmente optimista de sus capacidades reales.

Adicionalmente, dado que nuestro estudio se centra en la aplicación industrial práctica, priorizamos funciones y clases representativas de escenarios reales de desarrollo corporativo. Estos ejemplos debían reflejar patrones comunes y desafíos habituales encontrados en proyectos empresariales, asegurando así la relevancia de nuestros hallazgos.

Considerando estos requisitos y las limitaciones prácticas asociadas a la evaluación experta manual, seleccionamos siete funciones que representan diversos desafíos de prueba frecuentes en entornos industriales:

- **Ensamblador de Alquileres** (`assemble`): Método complejo que involucra múltiples objetos de dominio y requiere simulación (mocking) de bases de datos.
- **Comprobador de Números Primos** (`isPrime`): Función con alta complejidad ciclomática y manejo de excepciones.
- **Gestión de Usuarios** (`addUser`): Manejo de condiciones nulas (null) y simulación de acceso a bases de datos.
- **Calculadora de Bonificaciones** (`getBonus`): Lógica condicional compleja con dependencias externas.
- **Validador de IPV4** (`isIPV4Valid`): Requiere pruebas de integración mediante funciones auxiliares.
- **Número Estrobogramático** (`isStrobogrammic`): Manipulación de estructuras de datos.
- **Detector de Palíndromos** (`palindrome`): Integración de múltiples funciones.

Para cada una de estas funciones, expertos en pruebas de la empresa desarrollaron pruebas de referencia siguiendo las mejores prácticas y estándares de calidad establecidos.

Estas pruebas de referencia constituyeron nuestro *ground truth* contra el cual evaluamos las pruebas generadas por los LLMs.

4.3. Metodología de Ingeniería del Prompt

Nuestro enfoque de ingeniería de prompts se fundamenta en una estrategia de Prompt Chaining [Wu et al. 2022], diseñada para guiar a los LLMs en la generación de pruebas unitarias que cumplan con los estándares de calidad industrial. Esta técnica divide el proceso de generación en componentes secuenciales que abordan diferentes aspectos de las pruebas, permitiendo un control más preciso sobre los resultados. El desarrollo del prompt siguió un proceso iterativo alineado con nuestro marco de evaluación, refinándolo progresivamente según los hallazgos cuantitativos de cada iteración. Nuestro análisis reveló que el uso del inglés mejoraba significativamente los resultados en comparación con el español, lo que es consistente con otros estudios sobre el impacto del idioma en la generación de código mediante LLMs [Jiang et al. 2024, Wang et al. 2024a]. El prompt final optimizado, disponible en nuestro paquete de replicación, se estructura en cuatro componentes principales:

1. **Calidad de código y análisis estático:** Instrucciones específicas para evitar incidencias detectables por SonarQube, incluyendo directivas como: *"When creating the test class, do not generate any line of code that could create any Issue, nor Bug nor Code Smell."*
2. **Cobertura estructural:** Requisitos explícitos para maximizar la cobertura, por ejemplo: *"You must use the White Box method using Condition/Decision coverage, so use the truth tables to cover all the cases of the composed condition statements"* y *"I need a 100 % of line, condition and condition/decision coverage."*
3. **Técnicas de caja negra:** Instrucciones para implementar partición de equivalencias y análisis de valores límite: *"You must do the Black Box testing using equivalence partitioning and boundary value analysis. If you find redundancy with any of the other methods, remove the redundant tests."*
4. **Buenas prácticas de ingeniería de pruebas:** Directrices sobre estructura, nomenclatura y técnicas de refactorización: *"Use proper naming conventions for the tests and structure them correctly. If there is repeated code you must use @BeforeAll, @BeforeEach @AfterAll and @AfterEach statements."*

Este enfoque estructurado nos permitió mejorar progresivamente la calidad de las pruebas generadas, abordando sistemáticamente las debilidades identificadas en cada iteración del proceso evaluativo.

4.4. Infraestructura y Herramientas

Nuestro marco de evaluación emplea herramientas estándar de la industria para garantizar la robustez y reproducibilidad del análisis. Para la ejecución y parametrización de pruebas utilizamos JUnit, mientras que el análisis estático se realiza mediante Sonarqube Cloud y Sonarqube IDE, herramientas que proporcionan las métricas de calidad del código. La cobertura se analiza con JaCoCo, que ofrece informes detallados sobre líneas, condiciones y decisiones ejecutadas. Todo el proceso se integra en un flujo de trabajo automatizado utilizando GitHub y GitHub Actions como plataformas de control de versiones y CI/CD, respectivamente. La gestión de dependencias se centraliza mediante Maven, asegurando

así la reproducibilidad de las compilaciones y la coherencia en las versiones de las herramientas a lo largo de todas las evaluaciones. Esta infraestructura, además de facilitar la evaluación sistemática, refleja el entorno real de desarrollo industrial, aumentando la validez externa de nuestros resultados.

5. Resultados

Nuestro análisis abarca la evolución de GitHub Copilot entre marzo y diciembre de 2024, ofreciendo una perspectiva longitudinal sobre cómo esta herramienta de generación automatizada ha mejorado sus capacidades para crear pruebas unitarias en entornos industriales.

	Categoría	*Nombre de Métrica*	Peso	ChatGPT-4-1º Vez (Marzo 2024)	ChatGPT-4 - Iterativo (Mayo 2024)	o1-Mini (Dic. 2024)	GPT-o (Dic. 2024)	o1-Preview (Dic. 2024)
Métricas Objetivas	Métricas de Calidad de Código	Total de Incidencias de Compilación (IC)*	-20%	31	3	7	2	0
		Total de Incidencia de Análisis de Código (IAC)*	-5%	45	18	10	29	15
	Métricas Estructurales	Cobertura de Líneas (CL)	50%	39,14%	65,57%	28,57%	70,14%	98,00%
		Cobertura de Condiciones (CC)		39,14%	65,57%	28,57%	71,29%	95,71%
		Cobertura de Condicion/Decisión (CCD)		36,57%	61,57%	28,57%	68,29%	95,71%
		Aislamineto de las Pruebas (AP)		85,71%	100,00%	100,00%	100,00%	100,00%
Métricas Subjetivas (Juicio de Expertos)	Métricas Basadas en el Conocimiento de Expertos	Cobertura de Partición de Equivalencia (CPE)	50%	71,19%	75,00%	85,12%	79,88%	84,52%
		Cobertura de Valores Límite (CVL)		69,39%	71,77%	83,57%	78,20%	81,53%
		Parametrización de las Pruebas (PP)		12,70%	38,89%	88,10%	83,81%	91,84%
		Cobertura de Pruebas de Referencia (CPR)		65,77%	75,36%	81,57%	66,23%	97,94%
	Evaluación del Peso Total			**27,45%**	**65,28%**	**59,88%**	**72,72%**	**91,49%**

*Errores de compilacion sobre el valor maximo de las iteraciones

Figura 2. Resumen de los resultados obtenidos (marzo 2024 - diciembre 2024)

La Figura 2 presenta un resumen completo de los resultados obtenidos. Es importante destacar que GitHub Copilot ha ido evolucionando en el periodo analizado, incorporando diferentes modelos LLM subyacentes. En nuestra primera evaluación (marzo 2024), Copilot utilizaba ChatGPT-4 como modelo base. En evaluaciones posteriores, Copilot incorporó nuevos modelos como GPT-4-turbo (mayo 2024), o1-Mini (octubre 2024), GPT-o y o1-Preview (diciembre 2024), lo que nos ha permitido observar cómo la herramienta ha mejorado sus capacidades a medida que integra modelos más avanzados.

Respecto a las métricas de calidad de código, observamos una mejora notable y consistente. El total de incidencias de compilación muestra una clara tendencia descendente, comenzando con 31 incidencias en la primera evaluación con ChatGPT-4 (marzo 2024) y reduciéndose a 0 con o1-Preview (diciembre 2024). De manera similar, las incidencias de análisis de código disminuyeron desde 45 hasta 15 en la configuración más reciente, evidenciando una mejora sustancial en la calidad estructural del código generado.

En el ámbito de las métricas estructurales, los avances son igualmente significativos. La cobertura de líneas aumentó desde un modesto 39,14 % inicial hasta un notable 98 % con o1-Preview. La cobertura de condiciones y la cobertura de condición/decisión siguieron una progresión similar, alcanzando ambas un 95,71 % en la configuración más reciente. Particularmente destacable es el aislamiento de las pruebas, que alcanzó y mantuvo un rendimiento óptimo del 100 % en las tres últimas evaluaciones.

Las métricas basadas en conocimiento de expertos también reflejan mejoras sustanciales. La cobertura de partición de equivalencia mejoró desde un 71,19 % hasta al-

canzar un 84,52 % con o1-Preview. Especialmente notable es el progreso en la parametrización de las pruebas, que evolucionó desde un modesto 12,70 % hasta un sobresaliente 91,84 %. La cobertura de valores límite y la cobertura de pruebas de referencia también experimentaron mejoras significativas, alcanzando 81,53 % y 97,94 % respectivamente en la configuración más reciente.

La evaluación del peso total, calculada según nuestro modelo de ponderación, refleja esta tendencia positiva, evolucionando desde un 27,45 % inicial hasta un 91,49 % con o1-Preview. Esta progresión evidencia cómo GitHub Copilot ha mejorado consistentemente su capacidad para generar pruebas de calidad industrial a medida que integra modelos LLM más avanzados..

6. Lecciones Aprendidas

Este trabajo nos ha permitido extraer lecciones aprendidas que son susceptibles de ser generalizadas a otras organizaciones. En primer lugar, el marco ha tenido un impacto directo en la toma de decisiones de la organización. En marzo de 2024, cuando GitHub Copilot generaba pruebas que apenas compilaban y requerían considerable intervención manual, la inversión en esta tecnología probablemente hubiera aumentado el esfuerzo de pruebas en lugar de reducirlo. Sin embargo, las mejoras observadas en diciembre de 2024 (ver sección anterior) han llevado a la organización a revaluar su postura, ya que los últimos resultados sugieren potenciales ganancias en productividad con su uso.

Además, ha quedado de manifiesto la necesidad de realizar una evaluación longitudinal continua. La rápida evolución de las capacidades de los LLMs significa que las evaluaciones se desactualizan rápidamente. Lo que era cierto sobre hace seis meses puede no reflejar el rendimiento actual. Las organizaciones necesitan mantener procesos de evaluación continuos en lugar de basarse en evaluaciones puntuales.

En tercer lugar, se reafirma la necesidad de la supervisión por expertos. A pesar de las altas puntuaciones alcanzadas en nuestras métricas, ésta sigue siendo esencial. Las métricas basadas en conocimiento experto muestran que, si bien los LLMs han mejorado en la aplicación de técnicas de prueba, aún no igualan la experiencia humana en decisiones de diseño de pruebas. Cabe recalcar que los ejemplos utilizados son básicos, pero por la tendencia que vemos, dichos ejemplos deberán ser escalados a ejemplos más complejos.

Estos hallazgos sugieren que, aunque las herramientas de generación de pruebas basadas en LLMs son cada vez más viables para uso industrial, su adopción exitosa requiere tanto marcos de evaluación sistemáticos como expectativas realistas. Las empresas deberían ver estas herramientas como ayudas para aumentar la productividad del desarrollador, no como reemplazos del conocimiento experto.

7. Limitaciones y Amenazas a la Validez

Nuestro estudio presenta ciertas limitaciones que deben considerarse al interpretar los resultados:

Validez de Constructo: Aunque nuestro marco es exhaustivo, podría no captar todos los aspectos relevantes del valor real de las pruebas generadas por LLMs. Para mitigar esto, combinamos métricas automatizadas con evaluaciones expertas validadas por profesionales de QA experimentados.

Validez Interna: La mejora observada podría atribuirse tanto a la evolución natural de los LLMs como al refinamiento de nuestras estrategias de prompt. Hemos documentado rigurosamente los cambios metodológicos y validado mediante múltiples ejecuciones para aislar estos factores.

Validez Externa: Los resultados provienen principalmente de una organización y tipos específicos de proyectos. El marco se diseñó para ser adaptable a diferentes contextos organizacionales, centrándose en herramientas y prácticas estándar de la industria.

Fiabilidad: Para abordar la subjetividad inherente a las evaluaciones expertas, establecimos criterios de evaluación claros y documentados, manteniendo registros detallados de todos los procedimientos y criterios de decisión.

A pesar de estas limitaciones, consideramos que nuestras estrategias de mitigación proporcionan un nivel razonable de confianza en los hallazgos presentados.

8. Trabajo Relacionado

Desde la emergencia de los LLMs en el desarrollo de software, la investigación sobre su aplicación en pruebas ha avanzado rápidamente. Wang et al. [Wang et al. 2024b] analizaron 102 artículos sobre LLMs en pruebas de software, identificando su uso principal en generación de pruebas unitarias, oráculos y entradas para pruebas del sistema, señalando también limitaciones en cobertura y confiabilidad. Complementando este análisis, nuestro trabajo aporta evidencia empírica longitudinal sobre cómo estas limitaciones han evolucionado en un entorno industrial real.

En cuanto a frameworks específicos, Liu et al. [Liu et al. 2024] presentaron AutoTestGPT con prompts estructurados y refinamiento iterativo, reduciendo el tiempo de generación en más del 70 % comparado con métodos manuales. Chen et al. [Chen et al. 2024] desarrollaron ChatUniTest incorporando mecanismos de generación-validación-reparación. Mientras estos enfoques se centran en la optimización técnica del proceso de generación, nuestro marco evalúa además la integración con herramientas estándar de la industria (SonarQube, JaCoCo) y considera métricas basadas en conocimiento experto, aspectos críticos para la adopción empresarial.

En evaluaciones empíricas, Siddiq et al. [Siddiq et al. 2024] analizaron varios LLMs usando benchmarks HumanEval y EvoSuite SF110, revelando desafíos en cobertura y compilabilidad. Schäfer et al. [Schäfer et al. 2024] evaluaron sistemáticamente la generación de pruebas unitarias con LLMs en JavaScript. Estos estudios ofrecen evaluaciones puntuales valiosas, pero carecen de la perspectiva longitudinal de nuestro trabajo, que permite observar cómo las capacidades de los LLMs han evolucionado a lo largo de nueve meses en aplicaciones industriales reales.

La principal contribución de nuestro trabajo respecto a la literatura existente radica en tres aspectos diferenciadores: (1) un marco práctico para evaluación continua en entornos industriales reales con métricas ponderadas adaptables a diferentes contextos organizacionales, (2) evidencia empírica sobre la evolución temporal de las capacidades de LLM en generación de pruebas, y (3) lecciones prácticas derivadas de la implementación real en una empresa de desarrollo de software. Esta combinación ofrece orientación concreta para organizaciones que consideran adoptar estas tecnologías.

9. Conclusiones y Trabajo Futuro

Este estudio presenta un marco sistemático para la evaluación continua de las capacidades de generación de pruebas unitarias mediante LLMs en entornos industriales. Los resultados obtenidos durante el período de marzo 2024 a diciembre 2024 demuestran una mejora significativa en las capacidades de los LLMs, con modelos recientes como o1-Preview alcanzando valoraciones superiores al 90 % en nuestras métricas ponderadas.

Las principales contribuciones incluyen: (1) la validación de un marco de evaluación que integra métricas objetivas y subjetivas, permitiendo una evaluación holística de las pruebas generadas, (2) la demostración empírica de la evolución de las capacidades de generación de pruebas de los LLMs en un entorno industrial y (3) la identificación de lecciones aprendidas relevantes para organizaciones que buscan adoptar estas tecnologías. Sin embargo, nuestros hallazgos también sugieren que, si bien las herramientas basadas en LLMs han alcanzado un nivel de madurez prometedor, aún deben considerarse como complementos al conocimiento experto más que como reemplazos. Las métricas basadas en el conocimiento de expertos indican que los LLMs, aunque han mejorado significativamente en la aplicación de técnicas de prueba, todavía no igualan la experiencia humana en aspectos críticos del diseño de pruebas.

El trabajo futuro se centrará en la evaluación de casos más complejos, incluyendo pruebas combinatorias, y en la realización de una nueva replicación del estudio en seis meses para continuar monitoreando la evolución de estas tecnologías. Además, planeamos expandir el marco para incluir evaluaciones de eficiencia en términos de tiempo y recursos necesarios para la generación y mantenimiento de pruebas.

Disponibilidad de Artefactos

Tanto las funciones como los resultados están disponibles en https://doi.org/10.5281/zenodo.15076786 para facilitar la reproducibilidad de este estudio.

Agradecimientos

Este trabajo ha sido financiado por MCIN/AEI/10.13039/501100011033, el *European Union NextGeneration EU/PRTR* con la referencia PID2021-125438OB-I00, Universidad del País Vasco dentro del programa *Universidad-Empresa-Sociedad* (US24/10).

Referencias

Campbell, A. and Papapetrou, P. (2013). *SonarQube in action*. Manning Publications Co.

Chen, Y., Hu, Z., Zhi, C., Han, J., Deng, S., and Yin, J. (2024). Chatunitest: A framework for llm-based test generation. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE)*. ACM.

Freeman, S., Mackinnon, T., Pryce, N., and Walnes, J. (2004). jMock: supporting responsibility-based design with mock objects. In *Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA*. ACM.

H. Washizaki, e. (2024). *Guide to the Software Engineering Body of Knowledge (SWEBOK Guide)*. IEEE Computer Society.

ISO (2021). IEEE/ISO/IEC international standard - software and systems engineering–software testing–part 4: Test techniques. *ISO/IEC/IEEE 29119-4:2021(E)*.

Jiang, W., Gao, X., Zhai, J., Ma, S., Zhang, X., and Shen, C. (2024). From effectiveness to efficiency: Comparative evaluation of code generated by lcgms for bilingual programming questions.

Kaczanowski, T. (2013). *Practical Unit Testing with JUnit and Mockito*. Tomasz Kaczanowski, POL.

Kracht, J. S., Petrovic, J. Z., and Walcott-Justice, K. R. (2014). Empirically evaluating the quality of automatically generated and manually written test suites. *14th International Conference on Quality Software*, pages 256–265.

Liu, H., Liu, L., Yue, C., Wang, Y., and Deng, B. (2024). Autotestgpt: A system for the automated generation of software test cases based on chatgpt. *Journal of Software*, 19(4).

López, J. A. H., Chen, B., Saad, M., Sharma, T., and Varró, D. (2025). On inter-dataset code duplication and data leakage in large language models. *IEEE Transactions on Software Engineering*, 51(1).

Prates, L. and Pereira, R. (2025). Devsecops practices and tools. *International Journal of Information Security*, 24(1):1–25.

Schäfer, M., Nadi, S., Eghbali, A., and Tip, F. (2024). An empirical evaluation of using large language models for automated unit test generation. *IEEE Transactions on Software Engineering*, 50(1):85–105.

Sergeyuk, A., Golubev, Y., Bryksin, T., and Ahmed, I. (2025). Using ai-based coding assistants in practice: State of affairs, perceptions, and ways forward. *Information and Software Technology*, 178.

Siddiq, M., Da Silva, J., Tanvir, R., Ulfat, N., Al Rifat, F., and Carvalho, V. (2024). Using large language models to generate junit tests: An empirical study. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. ACM.

Tillmann, N. and Schulte, W. (2005). Parameterized unit tests. In *Proceedings of the 10th European Software Engineering Conference*. ACM.

Wang, C., Li, Z., Gao, C., Wang, W., Peng, T., Huang, H., Deng, Y., Wang, S., and Lyu, M. (2024a). Exploring multi-lingual bias of large code models in code generation.

Wang, J., Huang, Y., Chen, C., Liu, Z., Wang, S., and Wang, Q. (2024b). Software testing with large language models: Survey, landscape, and vision. *IEEE Transactions on Software Engineering*, 50(4):911–936.

Wu, T., Terry, M., and Cai, C. (2022). Ai chains: Transparent and controllable human-ai interaction by chaining large language model prompts. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. ACM.

Wu, Y., Li, Z., Zhang, J. M., and Liu, Y. (2024). Condefects: A complementary dataset to address the data leakage concern for llm-based fault localization and program repair. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE)*. ACM.