Implementación de prácticas DevSecOps con inteligencia artificial para fortalecer la seguridad y eficiencia en el desarrollo de software en una empresa de fábrica de software

Implementation of DevSecOps practices with artificial intelligence to strengthen security and efficiency in software development in a software factory company

Germán José Gómez Ángeles[1], Fernando Sierra-Liñan[1]

[1] Universidad Tecnológica del Perú. Facultad de ingeniería. Lima, Perú.

DOI: 10.64966/ingeniare.v34.11

Resumen

La creciente necesidad de incorporar la seguridad desde las primeras etapas del desarrollo de software ha impulsado la adopción de prácticas DevSecOps reforzadas con inteligencia artificial (IA). En este contexto, el presente estudio tuvo como objetivo evaluar el efecto de la implementación de estas prácticas sobre la seguridad y la eficiencia del desarrollo de software en una empresa de fábrica de software de Lima durante 2025. Se desarrolló e implementó un pipeline de integración y entrega continua (CI/CD) con controles automáticos de seguridad y calidad, incorporando quality gates mediante SonarQube, escaneo de dependencias con Snyk y asistencia de IA mediante GitHub Copilot, siguiendo la metodología PHVA. Se empleó un diseño preexperimental con medición antes y después en una muestra intencional de 25 despliegues. Se evaluaron la densidad de hallazgos detectados (DH), el tiempo de detección (TD) y el tiempo de corrección (TC). Los resultados evidenciaron un incremento del 198,7 % en la densidad de hallazgos detectados, una reducción del 99,1 % en el tiempo de detección y una disminución del 86,4 % en el tiempo de corrección. Las diferencias fueron estadísticamente significativas en los tres indicadores ($p < 0{,}001$; $r \approx 0{,}87$). Se concluye que la integración de prácticas DevSecOps con IA fortalece la seguridad, mejora la eficiencia del desarrollo y aporta una estrategia de implementación replicable para incrementar la madurez de los procesos de desarrollo seguro.

Palabras clave: DevSecOps, inteligencia artificial, ci/cd, seguridad del software, eficiencia operativa.

Abstract

The growing imperative to integrate security at the earliest stages of software development has led to the adoption of DevSecOps practices augmented by artificial intelligence (AI). The study assessed the impact of implementing these practices on the security and efficiency of software development within a company in Lima during 2025. A continuous integration and continuous delivery (CI/CD) pipeline was designed and implemented with automated security and quality controls. The pipeline incorporated quality gates via SonarQube, dependency scanning with Snyk, and AI assistance through GitHub Copilot, following the PHVA methodology. A pre-experimental design with pre- and post-measurements was used with a selective sample of 25 deployments. The study evaluated the density of detected fin- dings (DF), detection time (DT), and correction time (CT). The results indicated a 198,7% increase in the density of detected findings, a 99,1% reduction in detection time, and an 86,4% decrease in correction time. All differences were statistically significant for all three indicators ($p < 0{,}001$; $r \approx 0{,}87$). Integrating DevSecOps practices with AI strengthens security, improves development efficiency, and provides a replicable implementation strategy to advance the maturity of secure development processes.

Keywords: DevSecOps, artificial intelligence, ci/cd, secure development, software efficiency.

Licencia: CC BY 4.0

Autor de correspondencia: c22334@utp.edu.pe

Recibido: 9 Febrero 2026 | Aceptado: 19 Agosto 2026

INTRODUCCIÓN

En los últimos años, el desarrollo de software se ha convertido en un elemento estratégico para la competitividad de las organizaciones, impulsado por la transformación digital, la adopción de arquitecturas basadas en la nube y la incorporación de metodologías ágiles junto con prácticas DevOps, las cuales buscan acelerar la entrega continua de software mediante una mayor integración entre desarrollo y operaciones [1]. Sin embargo, este incremento en la velocidad de desarrollo también ha ampliado la superficie de exposición a vulnerabilidades de seguridad, defectos de calidad y fallos operativos, generando impactos económicos, técnicos y reputacionales para las organizaciones [2].

Diversas investigaciones han demostrado que una parte importante de las vulnerabilidades de software se origina durante las primeras etapas del ciclo de vida del desarrollo de software (Software Development Life Cycle, SDLC), debido principalmente a prácticas de seguridad insuficientemente integradas, revisiones manuales limitadas y escasa automatización en la identificación temprana de riesgos [3]. En consecuencia, los enfoques tradicionales, donde la seguridad se incorpora únicamente en las fases finales del desarrollo, incrementan significativamente los costos de corrección, retrasan la entrega del software y reducen la capacidad de respuesta ante nuevas amenazas [4].

Esta problemática resulta especialmente crítica en las fábricas de software, donde la ejecución simultánea de múltiples proyectos, los ciclos de entrega acelerados y la necesidad de mantener altos niveles de productividad exigen procesos de desarrollo altamente automatizados y seguros. No obstante, la limitada integración entre los equipos de desarrollo, operaciones y seguridad continúa generando procesos fragmentados, escasa visibilidad de los riesgos y una mayor probabilidad de incidentes en ambientes de producción [5].

Como respuesta a estos desafíos, el enfoque DevSecOps propone incorporar controles y prácticas de seguridad de manera continua e integrada a lo largo de todo el SDLC, promoviendo la automatización de pruebas, el análisis continuo del código y la detección temprana de vulnerabilidades dentro de los pipelines de integración y despliegue continuo. Sin embargo, estudios recientes evidencian que la implementación de DevSecOps aún enfrenta limitaciones relacionadas con la elevada cantidad de alertas generadas por las herramientas de análisis, la presencia de falsos positivos, la limitada capacidad para priorizar riesgos de manera inteligente y la escasa incorporación de mecanismos avanzados de análisis predictivo [6].

En este contexto, la inteligencia artificial (IA), particularmente mediante técnicas de aprendizaje automático y modelos inteligentes de análisis de código, emerge como una tecnología habilitadora para fortalecer las prácticas DevSecOps. Su aplicación permite automatizar la detección de vulnerabilidades, priorizar riesgos según su criticidad, optimizar la ejecución de los pipelines de integración y despliegue continuo, así como apoyar la toma de decisiones durante el proceso de desarrollo, incrementando tanto la seguridad como la eficiencia operativa [7], [8]. A pesar de estos avances, la literatura científica evidencia que aún existe una limitada cantidad de estudios empíricos que documenten la implementación integral de DevSecOps apoyado por inteligencia artificial en entornos reales de fábricas de software, especialmente en organizaciones de tamaño pequeño y mediano, donde las restricciones de recursos dificultan la adopción de estas tecnologías [6], [8].

Aunque investigaciones recientes han explorado la incorporación de inteligencia artificial en entornos DevSecOps, la mayoría se ha centrado en componentes aislados, como la detección automática de vulnerabilidades, la priorización de alertas o el análisis de código mediante modelos de aprendizaje automático [10], [19], [23]. Aún existe una limitada evidencia empírica sobre modelos de integración que articulen de forma conjunta análisis estático, análisis dinámico, automatización del pipeline CI/CD y asistencia mediante modelos de lenguaje en entornos reales de fábricas de software. En este contexto, la presente investigación propone y valida un esquema de integración de prácticas DevSecOps apoyadas por inteligencia artificial que combina estas capacidades para fortalecer simultáneamente la seguridad y la eficiencia del proceso de desarrollo, constituyendo una propuesta metodológica replicable para organizaciones con procesos de integración y despliegue continuo.

En este escenario, la empresa objeto de estudio presenta dificultades relacionadas con la incorporación de prácticas DevSecOps fortalecidas mediante inteligencia artificial, lo que limita la detección temprana de vulnerabilidades, incrementa los tiempos de respuesta frente a riesgos de seguridad y reduce la eficiencia de los procesos de desarrollo. En consecuencia, la presente investigación propone la implementación de prácticas DevSecOps apoyadas en inteligencia artificial con el propósito de fortalecer la seguridad del software, optimizar los procesos de desarrollo y mejorar la eficiencia operativa de una fábrica de software, contribuyendo además a generar evidencia empírica sobre la aplicación de estas tecnologías en un contexto empresarial real.

Luego de revisar la realidad problemática, se estableció las siguientes preguntas de investigación ¿En qué medida la implementación de prácticas DevSecOps reforzadas con IA mejora la seguridad y la eficiencia del desarrollo de software en una fábrica de software?, y de manera específica: ¿En qué medida DevSecOps + IA mejora la oportunidad del aseguramiento de la densidad de hallazgos detectados? ¿En qué medida DevSecOps + IA disminuye el tiempo de detección de vulnerabilidades? ¿En qué medida DevSecOps + IA disminuye el tiempo de corrección de vulnerabilidades?

Por otro lado, la presente investigación se justifica teóricamente porque contribuye al fortalecimiento del conocimiento sobre la integración de prácticas DevSecOps con tecnologías de inteligencia artificial dentro del proceso de desarrollo de software. En un contexto donde las amenazas de ciberseguridad son cada vez más sofisticadas y los ciclos de desarrollo demandan mayor rapidez, resulta necesario estudiar enfoques que permitan incorporar la seguridad de manera continua y automatizada. En este sentido, el estudio permitirá generar evidencia sobre cómo la inteligencia artificial puede complementar las prácticas DevSecOps para mejorar la detección de vulnerabilidades, la gestión de riesgos y la calidad del software desarrollado. Asimismo, desde una perspectiva práctica, la investigación busca atender una necesidad real de las empresas de fábrica de software, las cuales enfrentan el desafío de entregar productos seguros en plazos cada vez más reducidos. Por ello, la implementación de prácticas DevSecOps apoyadas por inteligencia artificial permitirá optimizar los procesos de desarrollo, reducir vulnerabilidades, disminuir tiempos de detección y corrección de fallos, así como fortalecer la seguridad de las aplicaciones. De esta manera, se espera mejorar la eficiencia operativa y la competitividad de la organización, generando beneficios tanto para la empresa como para sus clientes. Finalmente, la investigación se justifica metodológicamente debido a que propone un procedimiento estructurado para evaluar el impacto de la implementación de prácticas DevSecOps con inteligencia artificial mediante indi-

cadores cuantificables de seguridad y eficiencia. En consecuencia, el estudio permitirá obtener resultados medibles y comparables que faciliten la validación de la propuesta, además de proporcionar una metodología replicable para futuras investigaciones y proyectos orientados a la mejora continua de los procesos de desarrollo seguro de software.

Asimismo, el objetivo general de la investigación fue determinar en qué medida la implementación de prácticas DevSecOps apoyadas por inteligencia artificial fortalece la seguridad y la eficiencia del desarrollo de software en una empresa de fábrica de software, evaluadas mediante la densidad de hallazgos detectados, el tiempo de detección de vulnerabilidades y el tiempo de corrección de vulnerabilidades. En concordancia con este propósito, se plantearon como objetivos específicos determinar en qué medida la implementación de estas prácticas mejora la densidad de hallazgos detectados durante el desarrollo de software, disminuye el tiempo de detección de vulnerabilidades y reduce el tiempo de corrección de vulnerabilidades.

El resto del artículo se organiza de la siguiente manera. La sección Background presenta los fundamentos teóricos que sustentan la investigación, mientras que la sección Estado del Arte revisa los estudios más recientes relacionados con la integración de prácticas DevSecOps e inteligencia artificial en el desarrollo seguro de software. Posteriormente, la sección Materiales y Métodos describe el diseño metodológico, las variables e indicadores evaluados, la población y muestra, así como las técnicas de recolección y análisis de datos. La sección Metodología de la Solución expone el proceso de implementación de las prácticas DevSecOps apoyadas por inteligencia artificial en el entorno de estudio. A continuación, la sección Resultados presenta los hallazgos obtenidos a partir de la evaluación experimental, y la sección Discusión analiza estos resultados en contraste con investigaciones previas. Finalmente, la sección Conclusiones resume los principales aportes del estudio, sus limitaciones y las líneas de investigación futura.

BACKGROUND

Fundamentos de DevSecOps y la inteligencia artificial

DevSecOps representa la evolución del paradigma DevOps al incorporar la seguridad como una responsabilidad compartida durante todo el ciclo de vida del desarrollo de software (Software Development Life Cycle, SDLC). Este enfoque integra prácticas de desarrollo, operaciones y ciberseguridad mediante la automatización de controles, pruebas y monitoreo continuo, permitiendo identificar vulnerabilidades desde las primeras etapas del desarrollo bajo el principio de Shift-Left Security. La implementación de pipelines de integración y entrega continua (Continuous Integration/Continuous Delivery, CI/CD) facilita la ejecución automática de actividades como compilación, validación, pruebas y despliegue, favoreciendo la entrega continua de software sin comprometer la seguridad ni la calidad [1], [2], [5], [6], [21], [22].

La incorporación de inteligencia artificial (IA) ha ampliado las capacidades de DevSecOps al automatizar procesos como el análisis de vulnerabilidades, la clasificación de riesgos, la detección de anomalías y la priorización de acciones correctivas. Mediante técnicas de aprendizaje automático (Machine Learning, ML), los modelos inteligentes analizan información proveniente del código fuente, dependencias, registros de ejecución y resultados de pruebas para apoyar la detección

temprana de amenazas y optimizar la toma de decisiones durante el desarrollo. Estas capacidades complementan los mecanismos tradicionales de seguridad, fortaleciendo la automatización y la eficiencia de los pipelines CI/CD [7], [10] - [13], [19], [23], [24], [28].

Un avance reciente en este ámbito corresponde a los modelos de lenguaje de gran tamaño (Large Language Models, LLMs), los cuales incorporan capacidades de comprensión y generación de código mediante procesamiento del lenguaje natural. En ingeniería de software, los LLMs apoyan actividades como generación y revisión de código, explicación de vulnerabilidades, documentación, refactorización y recomendación de acciones de remediación. Herramientas como GitHub Copilot ejemplifican esta evolución al asistir a los desarrolladores durante la programación; sin embargo, estudios recientes evidencian riesgos asociados con la generación de código inseguro, alucinaciones, ataques de prompt injection y problemas de privacidad. Por ello, la literatura recomienda utilizar los LLMs como mecanismos de apoyo, complementados por procesos de validación automatizada y supervisión humana [8], [9], [15] - [18], [20], [23].

Desde una perspectiva arquitectónica, un pipeline DevSecOps seguro integra herramientas especializadas de análisis estático de aplicaciones (SAST), análisis de composición de software (SCA), controles de quality gates y mecanismos de automatización dentro del flujo CI/CD para verificar continuamente la calidad y seguridad del software antes de su despliegue. La incorporación de IA y LLMs fortalece este ecosistema al proporcionar capacidades de análisis contextual y asistencia inteligente, mientras que herramientas de validación objetiva garantizan el cumplimiento de criterios técnicos y de seguridad. En conjunto, esta arquitectura híbrida combina automatización, controles de seguridad y asistencia basada en inteligencia artificial para construir procesos de desarrollo más seguros, eficientes y resilientes [22], [23], [25] - [28].

ESTADO DEL ARTE

La aceleración de los ciclos de desarrollo de software y el incremento de las amenazas de ciberseguridad han impulsado la evolución de DevOps hacia DevSecOps, integrando controles de seguridad desde las primeras etapas del ciclo de vida del desarrollo de software (SDLC). Este enfoque incorpora la automatización de pruebas, el monitoreo continuo y la colaboración entre desarrollo, operaciones y seguridad con el propósito de reducir vulnerabilidades sin afectar la velocidad de entrega. La literatura coincide en que DevSecOps constituye una estrategia efectiva para mejorar simultáneamente la seguridad y la eficiencia del desarrollo de software [6], [21], [22].

Las revisiones recientes muestran un elevado consenso respecto a los componentes fundamentales de DevSecOps. Entre las principales dimensiones identificadas se encuentran las prácticas organizacionales, la automatización de controles de seguridad, las métricas de desempeño y la cultura colaborativa. No obstante, pese a la consolidación conceptual del enfoque, persisten desafíos relacionados con la integración efectiva de herramientas de seguridad dentro de pipelines de integración y entrega continua (CI/CD), la automatización de controles y la escalabilidad de las soluciones en entornos industriales [6], [21], [22]. Estos resultados evidencian que la madurez de DevSecOps continúa siendo heterogénea entre organizaciones.

En este contexto, la inteligencia artificial (IA) ha emergido como un habilitador para fortalecer los procesos DevSecOps. La evidencia científica demuestra que la IA permite automatizar actividades

como el análisis de vulnerabilidades, la clasificación de riesgos, la detección de anomalías, el monitoreo continuo y la respuesta a incidentes, incrementando la capacidad predictiva de los mecanismos de seguridad y reduciendo la carga operativa asociada a la revisión manual [23], [28].

No obstante, la literatura presenta diferentes enfoques para incorporar IA en DevSecOps. Algunos estudios concentran su aplicación en tareas específicas del pipeline CI/CD, como el análisis inteligente de código, el escaneo automatizado de dependencias y la detección temprana de vulnerabilidades [11], [13], mientras que otros proponen arquitecturas adaptativas orientadas al monitoreo continuo y la detección de amenazas en tiempo real [12], [19]. Esta diversidad refleja que la investigación continúa enfocándose en optimizar componentes particulares del pipeline más que en proponer modelos integrales de implementación y evaluación aplicables a entornos empresariales.

Paralelamente, la incorporación de los modelos de lenguaje de gran tamaño (Large Language Models, LLMs) representa una de las principales líneas de evolución de la ingeniería de software. Diversas revisiones sistemáticas evidencian que estos modelos pueden asistir en la generación y revisión de código, la explicación de vulnerabilidades, la priorización de riesgos y la recomendación de acciones de remediación, ampliando significativamente las capacidades de automatización dentro de los procesos DevSecOps [8], [20], [23]. Asimismo, la evidencia empírica indica que herramientas basadas en LLMs, como GitHub Copilot, pueden incrementar la productividad de los desarrolladores cuando se utilizan como apoyo durante el desarrollo de software [18].

Sin embargo, la incorporación de LLMs también introduce nuevos desafíos relacionados con la seguridad y la confiabilidad del software generado. Diversos estudios advierten riesgos asociados con la generación de código inseguro, alucinaciones, ataques de prompt injection, vulnerabilidades de tipo backdoor y problemas de privacidad, lo que limita su adopción sin mecanismos adicionales de validación [9], [15] – [17]. En consecuencia, la literatura converge en recomendar arquitecturas híbridas que combinen las capacidades de razonamiento de los LLMs con herramientas tradicionales de seguridad, como el análisis estático de código (SAST), el análisis dinámico (DAST), el análisis de composición de software (SCA) y controles automáticos integrados en el pipeline CI/CD. Asimismo, se enfatiza la necesidad de incorporar principios de gobernanza, explicabilidad y supervisión humana para garantizar un uso seguro de estas tecnologías [15] – [17], [23], [25].

En síntesis, la evidencia científica demuestra que la integración de DevSecOps con inteligencia artificial fortalece la seguridad y mejora la eficiencia del desarrollo de software. Sin embargo, la literatura presenta tres limitaciones relevantes. En primer lugar, predominan revisiones sistemáticas, estudios conceptuales y propuestas metodológicas. En segundo lugar, son escasos los trabajos que integran de manera conjunta herramientas tradicionales de seguridad, inteligencia artificial y LLMs dentro de un mismo pipeline CI/CD. Finalmente, existe limitada evidencia empírica obtenida en fábricas de software mediante indicadores cuantitativos que permitan evaluar simultáneamente el impacto sobre la seguridad y la eficiencia del proceso de desarrollo. Estas brechas justifican el presente estudio, orientado a evaluar el efecto de la implementación de prácticas DevSecOps fortalecidas con inteligencia artificial en un entorno industrial real, utilizando métricas objetivas de seguridad y eficiencia que contribuyan a generar evidencia empírica y una estrategia de implementación replicable.

MATERIALES Y MÉTODOS

DevSecOps incorpora la seguridad de forma transversal en todas las etapas del ciclo de vida del software mediante automatización, integración continua, colaboración interdisciplinaria y detección temprana de vulnerabilidades. Su adopción práctica se encuentra respaldada por marcos recientes que estructuran las prácticas, herramientas, métricas y desafíos asociados con su implementación [18]. Asimismo, la integración de inteligencia artificial y, particularmente, de Large Language Models (LLMs) fortalece este enfoque al automatizar el análisis de código, apoyar la priorización inteligente de vulnerabilidades, asistir en la generación de remediaciones y proporcionar métricas objetivas para evaluar el impacto de las prácticas DevSecOps sobre la seguridad y la eficiencia del proceso de desarrollo [19], [20].

Se aplicó la metodología PHVA (Planificar, Hacer, Verificar y Actuar) para implementar y evaluar prácticas DevSecOps fortalecidas con inteligencia artificial. Como parte de la intervención, se diseñó un entorno funcional que integró herramientas de asistencia a la codificación basadas en inteligencia artificial, análisis estático de código (SAST) y análisis de composición de software (SCA) dentro de un pipeline de integración y despliegue continuo (CI/CD) en una empresa de fábrica de software ubicada en Lima, Perú.

La investigación fue de tipo aplicada, con enfoque cuantitativo y diseño preexperimental con mediciones de pretest y postest. La población estuvo conformada por los 62 despliegues de software ejecutados en la rama de desarrollo del proyecto Capa de Servicios del ERP de la empresa Software Enterprise Services, realizado durante el período comprendido entre enero y diciembre de 2025.

La muestra estuvo constituida por 25 despliegues de software, seleccionados mediante un muestreo no probabilístico intencional, aplicando como criterio de inclusión aquellos despliegues que registraron el mayor número de líneas de código modificadas (Lines of Code Modified, LOC) durante el período de estudio. Para ello, los 62 despliegues fueron ordenados de forma descendente según el volumen de código modificado y se seleccionaron los primeros 25 registros (Tabla 1).

Este criterio se adoptó debido a que los despliegues con mayor volumen de cambios presentan una mayor complejidad técnica y un mayor riesgo de incorporar defectos funcionales y vulnerabilidades de seguridad. En consecuencia, estos escenarios permiten evaluar con mayor sensibilidad el impacto de la implementación de prácticas DevSecOps apoyadas por inteligencia artificial sobre los indicadores de seguridad y eficiencia, al concentrar el análisis en despliegues donde los efectos de la intervención resultan más evidentes. Asimismo, el criterio de selección fue definido previamente al análisis de los datos, garantizando un procedimiento objetivo, reproducible y consistente con el propósito de la investigación.

Tabla 1. Población y muestra de la investigación.

Descripción	Población	Muestra
Despliegues de software ejecutados en el período de estudio	62	25

La unidad de análisis estuvo constituida por cada despliegue de software ejecutado a través del pipeline de integración y despliegue continuo (CI/CD), entendido como el proceso completo de integración, validación y liberación de una versión del software. En cada despliegue se evaluó la variable "Eficiencia y seguridad en el desarrollo de software", medida mediante tres dimensiones técnicas y cuantificables: densidad de hallazgos detectados (DH), tiempo de detección de vulnerabilidades (TD) y tiempo de corrección de vulnerabilidades (TC). Estos indicadores fueron registrados antes y después de la implementación de las prácticas DevSecOps fortalecidas con inteligencia artificial utilizando fichas de observación diseñadas específicamente para la investigación, cuya operacionalización se presenta en la Tabla 2.

Tabla 2. Relación de indicadores con dimensiones de la variable 2.

Dimensión	Indicador	Técnica	Instrumento
Calidad del desarrollo	Densidad de hallazgos detectados por cada 500 líneas de código.	Observación	Ficha de observación
Eficiencia operativa	Tiempo de detección de vulnerabilidades (TD).	Observación	Ficha de observación
Eficiencia en respuesta	Tiempo de corrección de vulnerabilidades (TC).	Observación	Ficha de observación

En consecuencia, los instrumentos propuestos fueron sometidos a un proceso de validación mediante juicio de expertos, contando con la participación de tres especialistas de la Facultad de Ingeniería. Los expertos designados evaluaron la claridad, objetividad, actualidad, organización, suficiencia, intencionalidad, metodología, relevancia, coherencia y consistencia de los instrumentos, emitiendo su respectivo dictamen técnico, garantizando de esta manera la validez y rigurosidad de los instrumentos utilizados en la investigación. La Tabla 3 presenta los indicadores utilizados en la prueba de validez, aprobada por los expertos. En ella se obtuvo la siguiente calificación para el material grabado: Deficiente (0-20%), Regular (21-40%), Buena (41-60%), Muy buena (61-80%) y Excelente (81-100%).

Tabla 3. Indicadores de validez.

Criterio	Indicador	Puntaje		
		Experto 1	Experto 2	Experto 3
Claridad	Está formulado con un lenguaje apropiado.	85%	85%	90%
Objetividad	Se expresa de manera coherente y lógica.	85%	80%	85%
Actualidad	Es adecuado para los avances tecnológicos.	90%	85%	90%
Organización	Existe una organización lógica de variables e indicadores.	85%	80%	80%
Suficiencia	Hay coherencia entre indicadores y dimensiones.	85%	90%	80%
Intencionalidad	Es adecuado para los valores y aspectos relacionados con el tema.	80%	85%	85%
Consistencia	Se considera que todos los ítems utilizados en este instrumento son pertinentes y específicos para el campo de investigación.	80%	80%	80%

Tabla 3. Indicadores de validez. (continuación)

Criterio	Indicador	Puntaje		
		Experto 1	Experto 2	Experto 3
Coherencia	Se considera que la estructura de este instrumento es adecuada para el tipo de usuario al que está dirigido.	80%	85%	90%
Metodología	La estrategia responde al propósito de la investigación.	90%	85%	85%
Relevancia	Es adecuado para abordar el tema de investigación.	90%	90%	90%

La Tabla 4 presenta el OWASP Top 10 del año 2021, una clasificación ampliamente reconocida que identifica las principales categorías de vulnerabilidades en aplicaciones web. En el presente estudio, estas categorías fueron operacionalizadas como criterios de referencia para la detección, clasificación y seguimiento de vulnerabilidades durante la implementación de prácticas DevSecOps con inteligencia artificial. Los hallazgos obtenidos a partir de herramientas automatizadas de análisis de código y seguridad permitieron identificar defectos asociados a categorías como Broken Access Control, Injection, Security Misconfiguration y Cryptographic Failures, entre otras.

La información recopilada fue utilizada para evaluar las dimensiones de la variable dependiente. En la dimensión calidad del desarrollo, las vulnerabilidades detectadas fueron consideradas dentro del cálculo de la densidad de hallazgos detectados por cada 500 líneas de código, permitiendo medir la calidad del software generado. En la dimensión eficiencia operativa, las categorías OWASP sirvieron como referencia para determinar el tiempo de detección de vulnerabilidades (TD), calculado desde la introducción del defecto hasta su identificación mediante los mecanismos de seguridad implementados. Finalmente, en la dimensión eficiencia en respuesta, dichas vulnerabilidades permitieron medir el tiempo de corrección de vulnerabilidades (TC), definido como el intervalo transcurrido entre la detección del hallazgo y la aplicación de la solución correspondiente. De esta manera, el OWASP Top 10 constituyó un marco de evaluación que permitió cuantificar el impacto de la implementación de DevSecOps con inteligencia artificial sobre la seguridad y eficiencia del proceso de desarrollo de software.

Tabla 4. Vulnerabilidades más importantes.

Código	Categoría OWASP Top 10:2021	Descripción breve
A01	Broken Access Control	Fallos en las reglas de autorización que permiten el acceso no autorizado.
A02	Cryptographic Failures	Uso incorrecto o ausencia de criptografía, provocando exposición de datos.
A03	Injection	Inyección de código malicioso (SQL, OS, LDAP, etc.) en la entrada de datos.
A04	Insecure Design	Diseño del sistema sin enfoque en seguridad desde etapas tempranas.
A05	Security Misconfiguration	Configuración insegura del sistema o de sus componentes.
A06	Vulnerable and Outdated Components	Uso de librerías, frameworks o módulos con vulnerabilidades conocidas.
A07	Identification and Authentication Failures	Fallas en la autenticación e identificación de usuarios.
A08	Software and Data Integrity Failures	Confianza indebida en software o datos sin verificar su integridad.

Tabla 4. Vulnerabilidades más importantes. (continuación)

Código	Categoría OWASP Top 10:2021	Descripción breve
A09	Security Logging and Monitoring Failures	Falta de registros y monitoreo que impide detectar actividades maliciosas.
A10	Server-Side Request Forgery (SSRF)	El servidor realiza peticiones no autorizadas hacia recursos internos o externos.

METODOLOGÍA DE LA SOLUCIÓN

La implementación de las prácticas DevSecOps fortalecidas con inteligencia artificial se desarrolló mediante la metodología PHVA (Planificar, Hacer, Verificar, Actuar), utilizada como marco de mejora continua para estructurar la intervención y evaluar sistemáticamente su impacto sobre la seguridad y la eficiencia del proceso de desarrollo de software. La aplicación de este ciclo permitió planificar la estrategia de implementación, ejecutar las acciones definidas, verificar los resultados obtenidos y establecer acciones de mejora para su consolidación y escalabilidad.

La metodología se aplicó a través de las siguientes fases:

Planificar (Plan): Se realizó el diagnóstico del proceso de desarrollo de software de la empresa con el propósito de identificar brechas en materia de seguridad, automatización y eficiencia. Asimismo, se definieron los requerimientos de la solución, se seleccionaron las herramientas de apoyo basadas en inteligencia artificial y se diseñó la estrategia de implementación de las prácticas DevSecOps dentro del pipeline de integración y despliegue continuo (CI/CD).

Hacer (Do): Se implementó el entorno DevSecOps mediante la integración de herramientas de asistencia a la codificación basada en inteligencia artificial, análisis estático de código (SonarQube), análisis de composición de software (Snyk) y automatización del pipeline CI/CD utilizando GitHub Actions y GitHub Copilot. Durante esta fase se configuraron los controles de seguridad automatizados y se ejecutaron los despliegues correspondientes al período de evaluación.

Verificar (Check): Se evaluó el impacto de la implementación mediante la comparación de los indicadores definidos en la investigación: densidad de hallazgos detectados (DH), tiempo de detección de vulnerabilidades (TD) y tiempo de corrección de vulnerabilidades (TC). Los resultados obtenidos en las mediciones de pretest y postest permitieron determinar el efecto de las prácticas DevSecOps fortalecidas con inteligencia artificial sobre la seguridad y la eficiencia del proceso de desarrollo.

Actuar (Act): Con base en los resultados obtenidos, se realizaron ajustes al pipeline DevSecOps, se establecieron recomendaciones para optimizar la integración de las herramientas implementadas y se elaboró una hoja de ruta para favorecer la adopción progresiva del modelo en futuros proyectos de la organización y su posible replicación en otras empresas de desarrollo de software.

La aplicación de la metodología PHVA permitió estructurar la implementación de manera iterativa, asegurando un proceso de mejora continua basado en evidencia, la evaluación objetiva de los resultados y la consolidación de un modelo DevSecOps apoyado por inteligencia artificial orientado al fortalecimiento de la seguridad y la eficiencia del desarrollo de software (Figura 1).

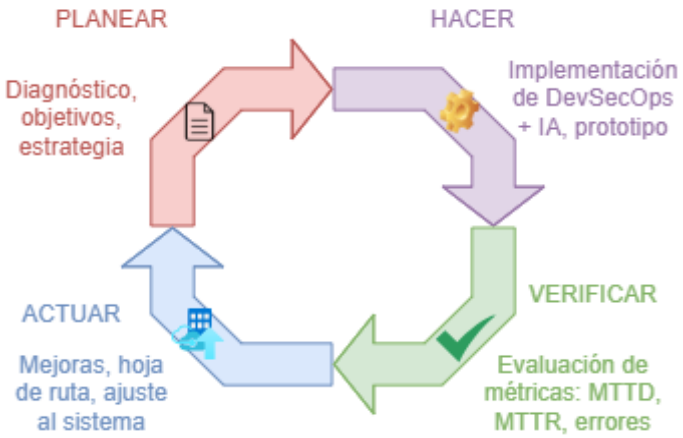


Figura 1. Metodología PHVA.

Analizar la situación actual

En esta fase se realizó el diagnóstico del proceso de desarrollo de software de la empresa con el propósito de identificar oportunidades de mejora relacionadas con la seguridad, la automatización y la eficiencia del ciclo de desarrollo. Para ello, se analizaron los repositorios de código fuente, la configuración de los pipelines de integración y despliegue continuo (CI/CD) y las prácticas de desarrollo empleadas en el proyecto “Capa de Servicios del ERP”. Asimismo, se llevaron a cabo entrevistas semiestructuradas con los integrantes de los equipos de desarrollo y operaciones para complementar la caracterización del proceso.

El diagnóstico permitió identificar brechas asociadas a la incorporación de controles de seguridad, el nivel de automatización de los pipelines CI/CD y la ausencia de mecanismos de análisis proactivo para la detección temprana de vulnerabilidades. Del mismo modo, se evidenció la necesidad de fortalecer la integración entre los equipos de desarrollo, operaciones y seguridad mediante un flujo de trabajo unificado basado en prácticas DevSecOps y herramientas apoyadas por inteligencia artificial.

La Tabla 5 presenta un resumen de las principales brechas identificadas durante el diagnóstico. Estas fueron clasificadas en cuatro categorías: Seguridad, Automatización, Colaboración y Formación, con el propósito de establecer una base objetiva para el diseño del plan de implementación y priorizar las acciones de mejora contempladas en la presente investigación.

Tabla 5. Brechas detectadas en el diagnóstico inicial.

Categoría	Brecha identificada
Seguridad	No existen herramientas para escaneo automatizado de vulnerabilidades.
Automatización	No existe un proceso de automatización de despliegues.
Colaboración	Limitada comunicación entre los equipos.
Formación	Bajo conocimiento sobre DevSecOps e inteligencia artificial aplicada.

Los resultados del diagnóstico evidencian que la ausencia de mecanismos automatizados de escaneo y de controles de seguridad integrados en el pipeline de integración y despliegue

continuo (CI/CD) incrementa el riesgo de que vulnerabilidades sean incorporadas al software sin ser detectadas oportunamente. Asimismo, la limitada integración entre los equipos de desarrollo, operaciones y seguridad dificulta la comunicación y la priorización de las acciones de remediación, afectando la capacidad de respuesta ante riesgos de seguridad. Finalmente, el reducido nivel de conocimiento sobre prácticas DevSecOps apoyadas por inteligencia artificial pone de manifiesto la necesidad de fortalecer las competencias técnicas del personal mediante procesos de capacitación, con el fin de facilitar una adopción efectiva de las herramientas propuestas.

Selección de herramientas tecnológicas

En base a estas evidencias, se procedió a definir los objetivos específicos de esta fase, los cuales consistieron en seleccionar herramientas alineadas al enfoque DevSecOps que incorporaran capacidades de automatización e inteligencia artificial para fortalecer tanto la calidad como la seguridad del desarrollo de software. Las herramientas seleccionadas fueron GitHub Copilot, SonarQube y Snyk, debido a su complementariedad dentro del ciclo de desarrollo.

GitHub Copilot fue incorporado como un asistente de codificación basado en inteligencia artificial cuya función principal consistió en apoyar al desarrollador durante la etapa de construcción del software mediante la generación de sugerencias de código, recomendaciones de buenas prácticas y propuestas de implementación orientadas a reducir errores de programación. Su contribución se evaluó de manera indirecta a través de indicadores relacionados con la calidad del desarrollo, específicamente la disminución de la densidad de hallazgos detectados por cada 500 líneas de código, comparando los resultados obtenidos antes y después de su implementación.

Por su parte, SonarQube cumplió el rol de analizar estáticamente el código fuente para identificar defectos, vulnerabilidades y problemas de mantenibilidad en etapas tempranas del desarrollo, permitiendo medir el tiempo de detección de vulnerabilidades (TD). Asimismo, Snyk fue utilizado para el análisis de dependencias y bibliotecas externas, identificando vulnerabilidades asociadas a componentes de terceros y contribuyendo a la reducción del tiempo de corrección de vulnerabilidades (TC).

A partir de las brechas identificadas, la Tabla 6 presenta el conjunto mínimo de herramientas seleccionadas para el prototipo DevSecOps + IA. Cada una desempeñó una función específica dentro del flujo de desarrollo y generó evidencias medibles para evaluar el impacto de la propuesta. Mientras GitHub Copilot actuó de forma preventiva durante la creación del código, SonarQube y Snyk funcionaron como mecanismos de verificación y control orientados a la detección y gestión de vulnerabilidades. Esta diferenciación permitió analizar la contribución de cada herramienta sobre los indicadores definidos en la investigación.

Tabla 6. Herramientas seleccionadas para el prototipo.

Herramienta	Indicador asociado	Funcionalidad principal	Rol dentro del modelo DevSecOps + IA
GitHub Copilot	Densidad de hallazgos detectados por cada 500 líneas de código.	Asistente inteligente para codificación.	Sugerencias seguras, aumento de productividad.
SonarQube	Tiempo de detección de vulnerabilidades (TD).	Análisis estático de código fuente.	Identificación temprana de errores y vulnerabilidades.

Tabla 6. Herramientas seleccionadas para el prototipo. (continuación)

Herramienta	Indicador asociado	Funcionalidad principal	Rol dentro del modelo DevSecOps + IA
Snyk	Tiempo de corrección de vulnerabilidades (TC).	Escaneo de dependencias y librerías externas.	Detección de vulnerabilidades en paquetes utilizados.

El stack tecnológico seleccionado fue definido en función de las brechas identificadas en la Tabla 6, con el propósito de fortalecer los procesos de seguridad, automatización y colaboración dentro del pipeline CI/CD. La integración de estas herramientas no solo contribuye a mitigar los problemas diagnosticados, sino que también proporciona métricas objetivas y trazables que permiten evaluar el impacto de la implementación de prácticas DevSecOps apoyadas por inteligencia artificial sobre los indicadores de seguridad y eficiencia del desarrollo de software.

Definir métricas de éxitos

Además, se definieron los indicadores clave de evaluación que permitieron medir los cambios producidos tras la implementación del modelo. Estos indicadores fueron la densidad de hallazgos detectados (DH), el tiempo de detección de vulnerabilidades (TD) y el tiempo de corrección de vulnerabilidades (TC). En conjunto, estos indicadores se alinearon con los objetivos de mejorar la calidad del código, anticipar riesgos de seguridad y optimizar la respuesta ante incidentes.

Para medir el efecto del prototipo se definieron indicadores operativos y comparables en el tiempo. La densidad de hallazgos detectados (DH) se definió como la cantidad de defectos de código identificados mediante análisis estático, incluyendo errores de programación, fallos de mantenibilidad y vulnerabilidades reportadas por SonarQube, normalizados por cada 500 líneas de código analizadas. Por su parte, el tiempo de detección (TD) y el tiempo de corrección (TC) se utilizaron para evaluar la eficiencia del proceso de gestión de vulnerabilidades (Tabla 7).

Glosario: NTHD = número total de hallazgos detectados mediante herramientas automatizadas de análisis de código y seguridad (incluye vulnerabilidades, defectos de código, code smells y otros problemas de mantenibilidad); TLC = total de líneas de código analizadas. Las unidades se consignan por indicador para asegurar la comparabilidad entre iteraciones.

Tabla 7. Indicadores de evaluación definidos.

Indicador	Descripción	Fórmula	Unidad
Densidad de hallazgos detectados (DH)	número de hallazgos identificados por las herramientas automatizadas de análisis de código y seguridad, incluyendo vulnerabilidades, defectos de código (bugs), code smells y otros problemas de mantenibilidad, por cada 500 líneas de código analizadas (LOC).	DH = NTHD / (TLC / 500)	Hallazgos detectados/500 LOC
Tiempo de detección (TD)	Tiempo promedio en detectar vulnerabilidades.	TD = Σ tiempos detección / Nº vulnerabilidades	Horas
Tiempo de corrección (TC)	Tiempo promedio en corregir vulnerabilidades.	TC = Σ tiempos corrección / Nº vulnerabilidades	Horas

En consecuencia, una disminución de la densidad de hallazgos detectados (DH), así como de los tiempos de detección (TD) y corrección (TC), evidenciará que el modelo DevSecOps + IA contribuye a mejorar la calidad del software y la eficiencia de los procesos de seguridad durante el ciclo

de desarrollo.

Plan de capacitación para el equipo

Como parte de la implementación de las prácticas DevSecOps fortalecidas con inteligencia artificial, se diseñó y ejecutó un plan de capacitación dirigido al equipo de desarrollo y operaciones, con el propósito de fortalecer las competencias técnicas necesarias para la adopción de las herramientas y prácticas incorporadas en la solución propuesta. La capacitación estuvo orientada a proporcionar conocimientos sobre el desarrollo seguro, la automatización de controles de seguridad y el uso de herramientas basadas en inteligencia artificial dentro del pipeline de integración y despliegue continuo (CI/CD).

El plan de capacitación se estructuró en cinco módulos que abordaron los fundamentos de DevSecOps, el funcionamiento del pipeline CI/CD, el uso de herramientas de análisis de seguridad, la integración de GitHub Copilot como asistente de codificación y la aplicación de criterios de calidad mediante Quality Gates. Asimismo, cada módulo definió objetivos de aprendizaje, contenidos, duración y mecanismos para evidenciar su ejecución, tal como se presenta en la Tabla 8.

Tabla 8. Plan de capacitación para la implementación de DevSecOps con inteligencia artificial.

Módulo	Objetivo	Contenido principal	Duración	Modalidad / Evidencia
1. Fundamentos de DevSecOps	Comprender los principios de DevSecOps y su integración en el ciclo de vida del desarrollo de software.	Cultura DevSecOps, Shift-Left Security, CI/CD y automatización de seguridad.	2 h	Taller teórico-práctico/ Registro de asistencia.
2. Flujo de trabajo CI/CD	Comprender el funcionamiento del pipeline de integración y despliegue continuo implementado en la organización.	Gestión de ramas, pull requests, integración continua y automatización del pipeline.	2 h	Taller práctico/ Desarrollo de ejercicios.
3. Herramientas de seguridad	Aplicar herramientas de análisis automatizado de seguridad durante el desarrollo.	SonarQube (SAST), Snyk (SCA), interpretación de resultados y tiempos de ejecución de escaneos.	3 h	Laboratorio/ Reportes generados por el pipeline
4. IA aplicada al desarrollo seguro	Utilizar herramientas de inteligencia artificial como apoyo a la codificación y revisión del código fuente.	GitHub Copilot, asistencia en codificación, revisión de pull requests y generación de recomendaciones.	2 h	Laboratorio/ Casos prácticos desarrollados
5. Calidad y buenas prácticas	Aplicar criterios de calidad y seguridad antes del despliegue del software.	Quality Gates, gestión de vulnerabilidades y buenas prácticas de desarrollo seguro.	2 h	Taller práctico/ Validación de escenarios de prueba

La ejecución del plan de capacitación permitió homogeneizar los conocimientos del equipo respecto a las prácticas DevSecOps y al uso de las herramientas implementadas, favoreciendo su integración dentro del proceso de desarrollo. Asimismo, contribuyó a fortalecer las competencias necesarias para la automatización de controles de seguridad, la interpretación de los resultados generados por las herramientas de análisis y la adopción de una cultura de mejora continua orientada al desarrollo seguro del software.

Fase 2: Hacer

En la Figura 2 se muestra el diagrama de la arquitectura DevSecOps con inteligencia artificial. En esta fase se presentan un resumen de las evidencias de la implementación del modelo propuesto.

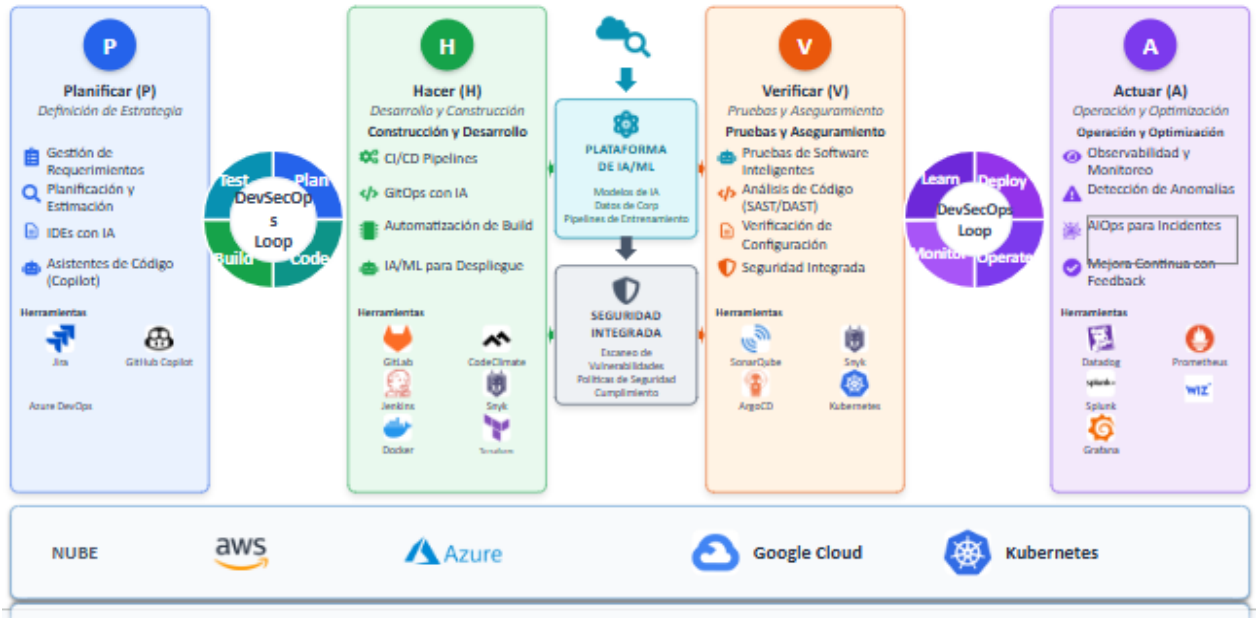


Figura 2. Diagrama de arquitectura DevSecOps más inteligencia artificial, automatización, seguridad inteligente y mejora continua.

Implementación de pipelines DevSecOps fortalecido con Inteligencia artificial

En esta fase se ejecutó la propuesta diseñada durante la etapa de planificación mediante la implementación de un pipeline de integración y despliegue continuo (CI/CD) fortalecido con prácticas DevSecOps e inteligencia artificial. La solución se implementó sobre el proyecto “Capa de Servicios del ERP” de la empresa Software Enterprise Services, integrando herramientas de asistencia a la codificación, análisis automatizado de seguridad y validación continua con el propósito de fortalecer la seguridad y la eficiencia del proceso de desarrollo.

Para la ejecución de la solución se definió un plan de acción que comprendió la configuración del entorno CI/CD, la integración de herramientas de análisis de seguridad e inteligencia artificial, la capacitación del equipo de desarrollo y el registro de los indicadores utilizados para evaluar el impacto de la intervención. La Tabla 9 resume las actividades desarrolladas, las herramientas empleadas y los resultados esperados durante la implementación.

Tabla 9. Plan de acción para la implementación del pipeline DevSecOps con inteligencia artificial.

Actividad	Herramienta(s)	Descripción	Resultado esperado	Dimensión evaluada
Configuración del entorno CI/CD	GitHub Actions	Se implementó un pipeline automatizado que integró las etapas de compilación, pruebas y validación del código fuente.	Automatización del proceso de integración y despliegue continuo.	**Tiempo de detección de vulnerabilidades (TD).**
Análisis estático de código (SAST)	SonarQube	Se configuró SonarQube para ejecutar análisis estático del código en cada *pull request*, verificando calidad, mantenibilidad y vulnerabilidades potenciales mediante *Quality Gates*.	Detección temprana de vulnerabilidades y validación de la calidad del código.	**Densidad de hallazgos detectados (DH) Tiempo de detección de vulnerabilidades (TD).**
Análisis de composición de software (SCA)	Snyk	Se configuró el análisis automático de dependencias para identificar vulnerabilidades conocidas y riesgos asociados a bibliotecas de terceros antes del despliegue.	Identificación temprana y reducción de vulnerabilidades en dependencias.	**Densidad de hallazgos detectados (DH) Tiempo de detección de vulnerabilidades (TD).**
Asistencia inteligente a la codificación	GitHub Copilot	Se habilitó GitHub Copilot para asistir a los desarrolladores durante la codificación y revisión de *pull requests*, proporcionando sugerencias de código y recomendaciones de mejora.	Incremento de la productividad y apoyo a la generación de código más seguro.	**Tiempo de detección de vulnerabilidades (TD). Tiempo de corrección de vulnerabilidades (TC).**
Capacitación y acompañamiento	Materiales de capacitación, GitHub Copilot, SonarQube y Snyk	Se desarrollaron sesiones de capacitación y acompañamiento técnico para fortalecer las competencias del equipo en el uso de las herramientas implementadas y en la adopción de prácticas DevSecOps.	Fortalecimiento de las competencias técnicas y adopción efectiva de las herramientas.	**Densidad de hallazgos detectados (DH) Tiempo de detección de vulnerabilidades (TD). Tiempo de corrección de vulnerabilidades (TC).**
Registro y evaluación de indicadores	Fichas de observación, registros del pipeline y reportes de herramientas	Se registraron los indicadores definidos para la investigación antes y después de la implementación, permitiendo evaluar objetivamente el impacto de la intervención.	Obtención de evidencia cuantitativa para el análisis estadístico de la investigación.	**DH, TD y TC.**

Como parte de la implementación, se configuró el pipeline CI/CD para automatizar las actividades de integración, compilación, análisis de seguridad y validación del código fuente en cada pull request. Esta configuración permitió incorporar controles de seguridad desde las primeras

etapas del ciclo de desarrollo (Shift-Left Security), favoreciendo la detección temprana de vulnerabilidades y la aplicación de criterios objetivos para aprobar la integración de cambios. La configuración general del pipeline implementado se presenta en la Figura 3.

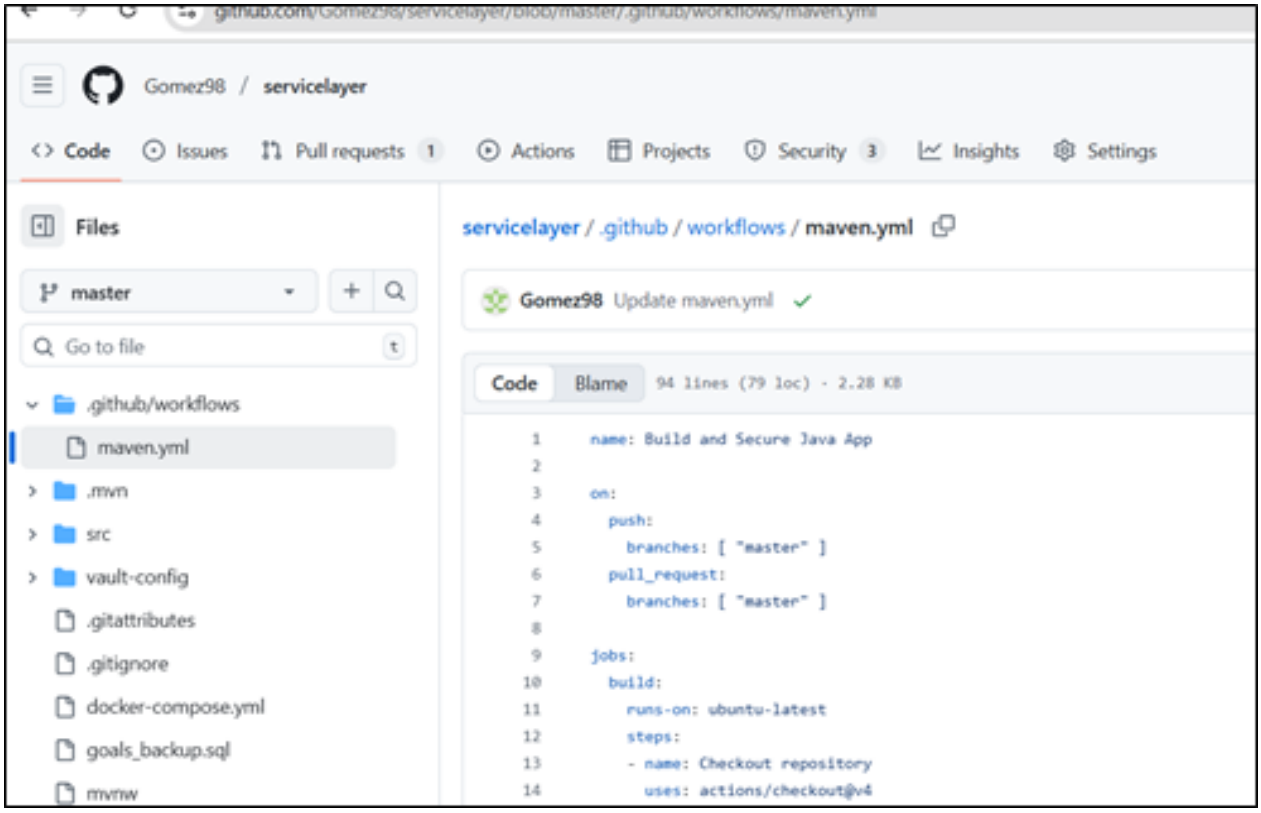


Figura 3. Configuración del pipeline CI/CD implementado.

La configuración integra de forma automatizada las etapas de compilación, ejecución de pruebas, análisis estático de código mediante SonarQube, análisis de dependencias con Snyk y validación de Quality Gates antes de autorizar la integración y el despliegue del software. Esta arquitectura permitió establecer controles continuos de seguridad y calidad, reduciendo la intervención manual y fortaleciendo la trazabilidad del proceso de desarrollo.

De manera complementaria, se habilitó GitHub Copilot como herramienta de asistencia basada en inteligencia artificial para apoyar las actividades de codificación y revisión del código fuente. Su incorporación facilitó la generación de sugerencias de código y recomendaciones durante el desarrollo, complementando los mecanismos automatizados de validación implementados en el pipeline DevSecOps. La activación de esta herramienta en el repositorio del proyecto se presenta en la Figura 4.

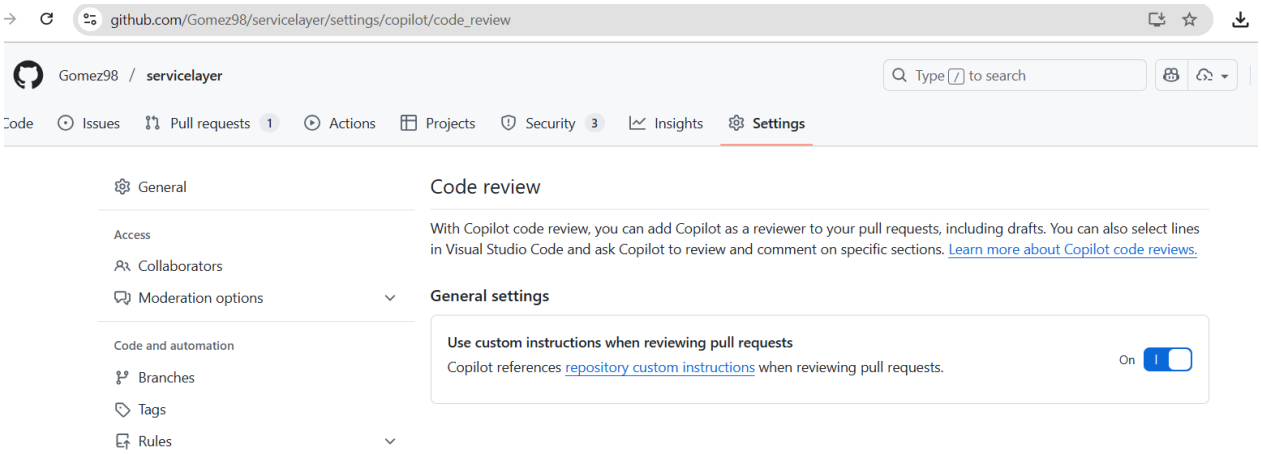


Figura 4. Integración de GitHub Copilot.

La habilitación de GitHub Copilot requirió la asignación de una licencia institucional y su configuración en el entorno de desarrollo utilizado por el equipo. Una vez integrada, la herramienta proporcionó asistencia durante la escritura y revisión del código; sin embargo, todas las sugerencias generadas fueron validadas mediante los análisis automatizados de SonarQube y Snyk, así como por el proceso de revisión de pull requests, garantizando que únicamente el código que cumplía los criterios de calidad y seguridad establecidos fuera incorporado al repositorio principal.

Ejecución de la capacitación del equipo

Como parte de la implementación de la solución, se ejecutó el plan de capacitación dirigido al equipo de desarrollo y operaciones con el propósito de fortalecer las competencias necesarias para la adopción de las prácticas DevSecOps apoyadas por inteligencia artificial. La capacitación estuvo orientada a estandarizar el uso de las herramientas incorporadas en el pipeline CI/CD y promover una comprensión integral del flujo de trabajo propuesto. La Figura 5 presenta parte del material utilizado durante las sesiones de capacitación impartidas al personal involucrado en el proyecto

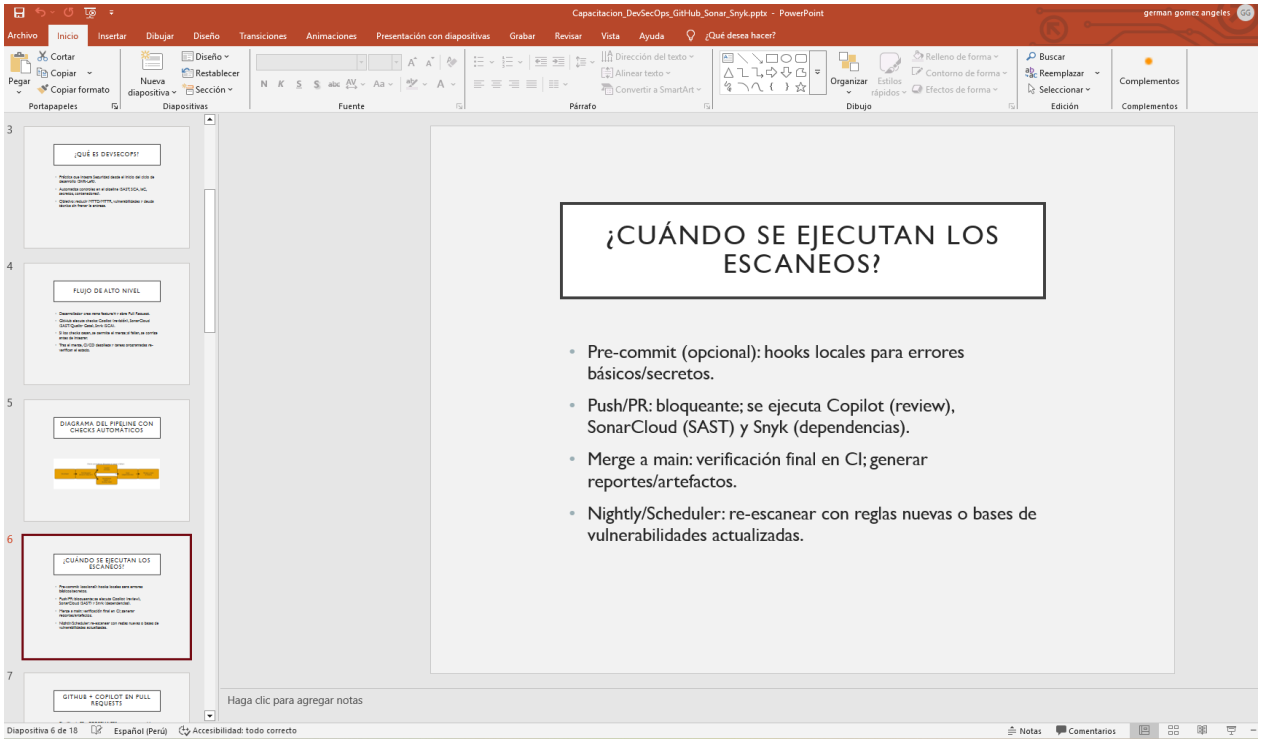


Figura 5. Material de capacitación sobre la implementación de DevSecOps con inteligencia artificial.

Durante las sesiones se abordó la arquitectura del pipeline implementado para el proyecto "ServiceLayer", describiendo las etapas de integración, análisis y validación del código fuente. Asimismo, se explicó el funcionamiento de las herramientas incorporadas, incluyendo GitHub Copilot como asistente de codificación, SonarQube para el análisis estático del código (SAST) y Snyk para el análisis de composición de software (SCA). También se presentó la ubicación e interpretación de los reportes de vulnerabilidades generados por cada herramienta, los criterios de validación mediante Quality Gates y las principales buenas prácticas para la aplicación de DevSecOps fortalecidas con inteligencia artificial.

La ejecución de la capacitación permitió homogeneizar los conocimientos del equipo sobre el funcionamiento del pipeline y el uso de las herramientas implementadas, facilitando su adopción dentro del proceso de desarrollo y contribuyendo a una aplicación consistente de las prácticas de desarrollo seguro establecidas en la investigación.

Registro de datos para las métricas de éxito

Para la recolección de la información se utilizaron fichas de observación diseñadas específicamente para la investigación, las cuales permitieron registrar los indicadores definidos para evaluar la variable "Eficiencia y seguridad en el desarrollo de software". En total, se documentaron 25 despliegues de software correspondientes a las mediciones realizadas antes (pretest) y después (postest) de la implementación de las prácticas DevSecOps fortalecidas con inteligencia artificial.

En la fase de pretest, los datos fueron obtenidos a partir de los registros históricos del proceso de desarrollo y de los reportes generados por las herramientas utilizadas por el equipo antes de la intervención. En la fase de postest, la información se recopiló automáticamente a partir de los reportes generados por el pipeline CI/CD, incluyendo los resultados del análisis estático de código mediante SonarQube/SonarCloud, el análisis de composición de software realizado con Snyk y los registros de ejecución del pipeline. Estos insumos permitieron medir objetivamente la densidad de hallazgos detectados (DH), el tiempo de detección de vulnerabilidades (TD) y el tiempo de corrección de vulnerabilidades (TC). Si bien GitHub Copilot formó parte de la intervención como herramienta de asistencia a la codificación, sus resultados no fueron utilizados como fuente directa para la medición de los indicadores. La Figura 6 y Figura 7 muestran ejemplos de las fichas de observación empleadas durante las etapas de pretest y postest, respectivamente.

FICHA DE OBSERVACIÓN: DENSIDAD DE HALLAZGOS DETECTADOS

(PRE-TEST)

Investigador(a)	German José Gomez Angeles	
Lugar de investigación	Software Enterprise Services	
Proceso a observar	Densidad de hallazgos	$DH = \frac{NTEE}{TLC}$
Numero de Despliegues	**Despliegues**	
	NTEE (Hallazgos detectados)	**TLC (Líneas de código)**	**DH (Densidad de hallazgos)**
Despliegue 1	7	4171	0.84
Despliegue 2	4	3919	0.51
Despliegue 3	13	4735	1.37
Despliegue 4	11	1130	4.87
Despliegue 5	8	2685	1.49
Despliegue 6	13	4380	1.48
Despliegue 7	5	1769	1.41
Despliegue 8	7	3391	1.03
Despliegue 9	10	2515	1.99
Despliegue 10	3	4485	0.33
Despliegue 11	7	3853	0.91
Despliegue 12	11	3433	1.6
Despliegue 13	11	2215	2.48

Figura 6. Ficha de observación utilizada en la fase de pretest.

FICHA DE OBSERVACIÓN: DENSIDAD DE HALLAZGOS DETECTADOS

(POST-TEST)

Investigador(a)	German José Gomez Angeles		
Lugar de investigación	Software Enterprise Services		
Proceso a observar	Densidad de Hallazgos		$DH = \frac{NTEE}{TLC}$
Numero de Despliegue	Despliegues		
	NTEE (Hallazgos detectados)	TLC (Líneas de código)	DH (Densidad de hallazgos)
Despliegue 1	17	4171	2,04
Despliegue 2	14	3919	1,79
Despliegue 3	36	4735	3,80
Despliegue 4	30	1130	13,27
Despliegue 5	22	2685	4,10
Despliegue 6	36	4380	4,11
Despliegue 7	14	1769	3,96
Despliegue 8	27	3391	3,98
Despliegue 9	31	2515	6,16
Despliegue 10	14	4485	1,56
Despliegue 11	22	3853	2,85
Despliegue 12	33	3433	4,81
Despliegue 13	32	2215	7,22
Despliegue 14	24	1955	6,14
Despliegue 15	15	3324	2,26

Figura 7. Ficha de observación utilizada en la fase de postest.

Fase 3: Verificar

Estrategia de verificación

En esta fase se evaluó el impacto de la implementación de las prácticas DevSecOps fortalecidas con inteligencia artificial sobre la seguridad y la eficiencia del proceso de desarrollo de software. Para ello, se compararon los resultados obtenidos antes (pretest) y después (postest) de la intervención, utilizando los indicadores definidos en la investigación y registrados mediante las fichas de observación.

La estrategia de verificación se estructuró de la siguiente manera:

i. Unidad de análisis: cada despliegue de software ejecutado a través del pipeline de integración y despliegue continuo (CI/CD) correspondiente al proyecto “Capa de Servicios del ERP”.

ii. Datos recopilados: registros históricos del proceso de desarrollo (pretest) y reportes generados por SonarQube/SonarCloud, Snyk y el pipeline CI/CD (postest), incluyendo el número de líneas de código modificadas (LOC), la densidad de hallazgos detectados (DH), el tiempo de detección de vulnerabilidades (TD) y el tiempo de corrección de vulnerabilidades (TC).

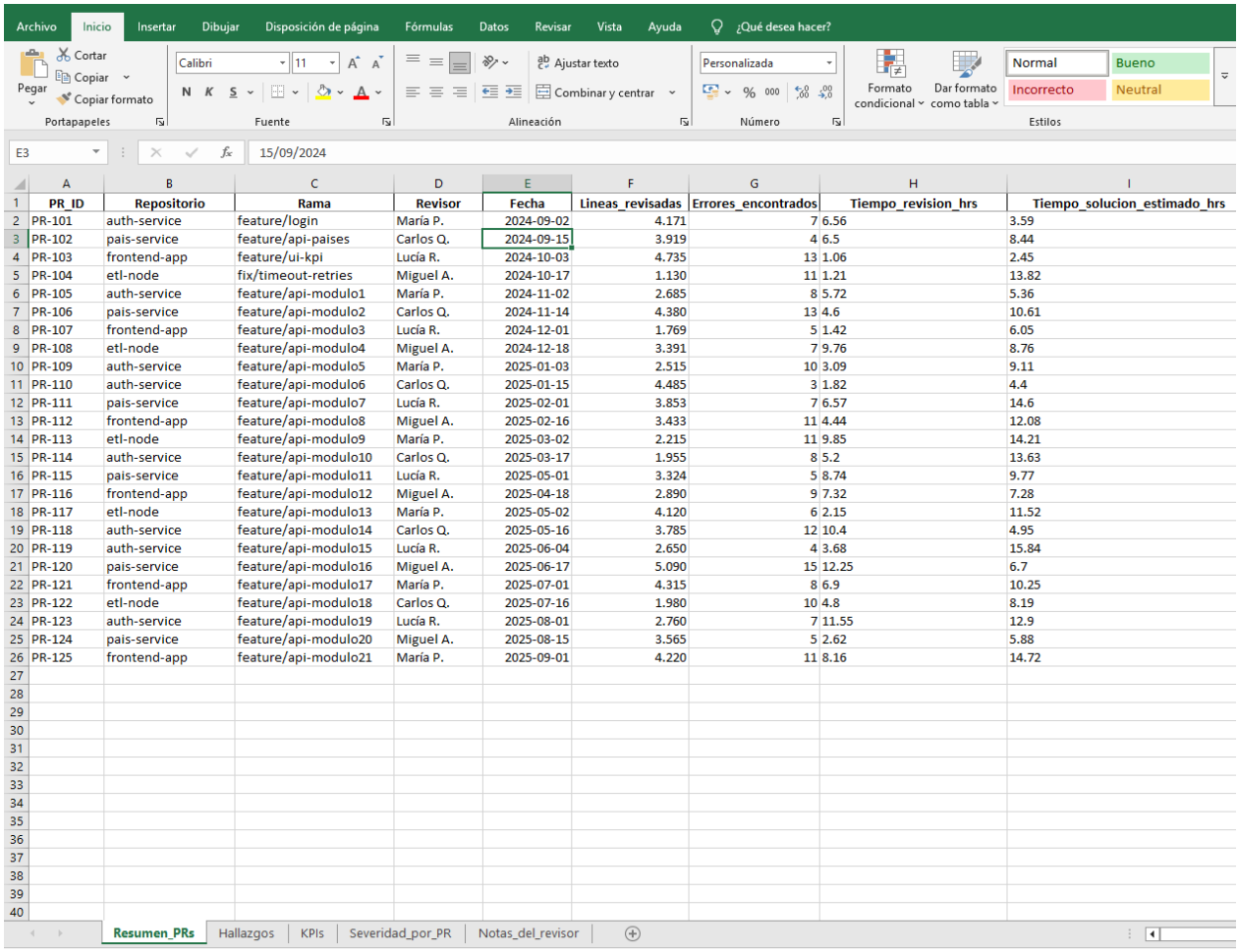
iii. Indicadores evaluados: densidad de hallazgos detectados (DH), tiempo de detección de vulnerabilidades (TD) y tiempo de corrección de vulnerabilidades (TC), definidos como indicadores de la variable “Eficiencia y seguridad en el desarrollo de software”.

iv. Estrategia de comparación: los resultados obtenidos mediante el proceso de desarrollo previo a la intervención fueron comparados con los obtenidos tras la implementación del pipeline DevSecOps fortalecido con inteligencia artificial, con el propósito de determinar el efecto de la intervención sobre los indicadores evaluados.

Análisis de resultados

La fase de verificación tuvo como propósito evaluar objetivamente el efecto de la implementación de las prácticas DevSecOps fortalecidas con inteligencia artificial sobre el proceso de desarrollo de software. Para ello, se compararon los resultados obtenidos antes (pretest) y después (postest) de la intervención utilizando la misma muestra de 25 despliegues de software, con base en los indicadores definidos en la investigación: densidad de hallazgos detectados (DH), tiempo de detección de vulnerabilidades (TD) y tiempo de corrección de vulnerabilidades (TC).

Como línea base, se analizaron los registros correspondientes al proceso de revisión previo a la implementación de la solución. La Figura 8 presenta un ejemplo del registro utilizado para documentar la revisión manual de los despliegues, incluyendo información relacionada con las líneas de código modificadas, el número de hallazgos identificados, el tiempo empleado en la revisión y el tiempo estimado para la corrección de las vulnerabilidades.

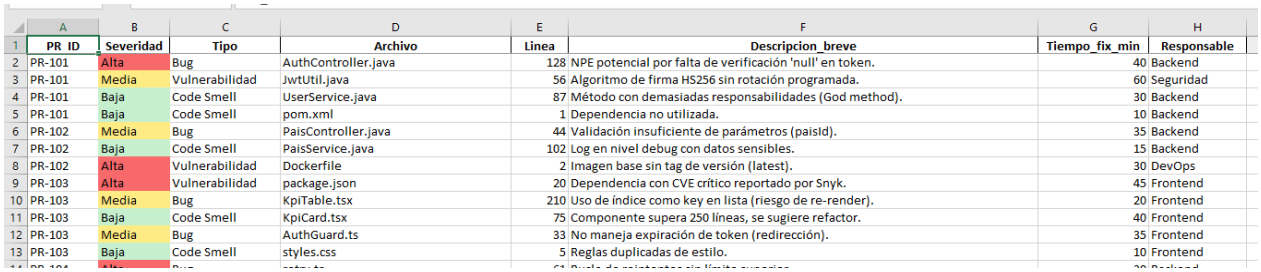


PR_ID	Repositorio	Rama	Revisor	Fecha	Lineas_revisadas	Errores_encontrados	Tiempo_revision_hrs	Tiempo_solucion_estimado_hrs
PR-101	auth-service	feature/login	María P.	2024-09-02	4.171	7	6.56	3.59
PR-102	pais-service	feature/api-paises	Carlos Q.	2024-09-15	3.919	4	6.5	8.44
PR-103	frontend-app	feature/ui-kpi	Lucía R.	2024-10-03	4.735	13	1.06	2.45
PR-104	etl-node	fix/timeout-retries	Miguel A.	2024-10-17	1.130	11	1.21	13.82
PR-105	auth-service	feature/api-modulo1	María P.	2024-11-02	2.685	8	5.72	5.36
PR-106	pais-service	feature/api-modulo2	Carlos Q.	2024-11-14	4.380	13	4.6	10.61
PR-107	frontend-app	feature/api-modulo3	Lucía R.	2024-12-01	1.769	5	1.42	6.05
PR-108	etl-node	feature/api-modulo4	Miguel A.	2024-12-18	3.391	7	9.76	8.76
PR-109	auth-service	feature/api-modulo5	María P.	2025-01-03	2.515	10	3.09	9.11
PR-110	auth-service	feature/api-modulo6	Carlos Q.	2025-01-15	4.485	3	1.82	4.4
PR-111	pais-service	feature/api-modulo7	Lucía R.	2025-02-01	3.853	7	6.57	14.6
PR-112	frontend-app	feature/api-modulo8	Miguel A.	2025-02-16	3.433	11	4.44	12.08
PR-113	etl-node	feature/api-modulo9	María P.	2025-03-02	2.215	11	9.85	14.21
PR-114	auth-service	feature/api-modulo10	Carlos Q.	2025-03-17	1.955	8	5.2	13.63
PR-115	pais-service	feature/api-modulo11	Lucía R.	2025-05-01	3.324	5	8.74	9.77
PR-116	frontend-app	feature/api-modulo12	Miguel A.	2025-04-18	2.890	9	7.32	7.28
PR-117	etl-node	feature/api-modulo13	María P.	2025-05-02	4.120	6	2.15	11.52
PR-118	auth-service	feature/api-modulo14	Carlos Q.	2025-05-16	3.785	12	10.4	4.95
PR-119	auth-service	feature/api-modulo15	Lucía R.	2025-06-04	2.650	4	3.68	15.84
PR-120	pais-service	feature/api-modulo16	Miguel A.	2025-06-17	5.090	15	12.25	6.7
PR-121	frontend-app	feature/api-modulo17	María P.	2025-07-01	4.315	8	6.9	10.25
PR-122	etl-node	feature/api-modulo18	Carlos Q.	2025-07-16	1.980	10	4.8	8.19
PR-123	auth-service	feature/api-modulo19	Lucía R.	2025-08-01	2.760	7	11.55	12.9
PR-124	pais-service	feature/api-modulo20	Miguel A.	2025-08-15	3.565	5	2.62	5.88
PR-125	frontend-app	feature/api-modulo21	María P.	2025-09-01	4.220	11	8.16	14.72

Figura 8. Registro de revisión manual de los despliegues.

Estos registros constituyeron la referencia inicial para comparar el desempeño alcanzado tras la implementación del pipeline DevSecOps con inteligencia artificial.

La Figura 9 muestra el detalle de los hallazgos identificados durante la revisión manual. Cada registro incluye información sobre el nivel de severidad, el archivo afectado, el tipo de vulnerabilidad, la línea de código comprometida y el responsable de su corrección. Esta información permitió caracterizar el estado inicial del proceso de desarrollo y establecer la línea base para el análisis comparativo.



PR_ID	Severidad	Tipo	Archivo	Linea	Descripcion_breve	Tiempo_fix_min	Responsable
PR-101	Alta	Bug	AuthController.java	128	NPE potencial por falta de verificación 'null' en token.	40	Backend
PR-101	Media	Vulnerabilidad	JwtUtil.java	56	Algoritmo de firma HS256 sin rotación programada.	60	Seguridad
PR-101	Baja	Code Smell	UserService.java	87	Método con demasiadas responsabilidades (God method).	30	Backend
PR-101	Baja	Code Smell	pom.xml	1	Dependencia no utilizada.	10	Backend
PR-102	Media	Bug	PaisController.java	44	Validación insuficiente de parámetros (paisId).	35	Backend
PR-102	Baja	Code Smell	PaisService.java	102	Log en nivel debug con datos sensibles.	15	Backend
PR-102	Alta	Vulnerabilidad	Dockerfile	2	Imagen base sin tag de versión (latest).	30	DevOps
PR-103	Alta	Vulnerabilidad	package.json	20	Dependencia con CVE crítico reportado por Snyk.	45	Frontend
PR-103	Media	Bug	KpiTable.tsx	210	Uso de índice como key en lista (riesgo de re-render).	20	Frontend
PR-103	Baja	Code Smell	KpiCard.tsx	75	Componente supera 250 líneas, se sugiere refactor.	40	Frontend
PR-103	Media	Bug	AuthGuard.ts	33	No maneja expiración de token (redirección).	35	Frontend
PR-103	Baja	Code Smell	styles.css	5	Reglas duplicadas de estilo.	10	Frontend

Figura 9. Detalle de hallazgos identificados durante la revisión manual.

Como parte de la intervención, se incorporó GitHub Copilot como herramienta de asistencia basada en inteligencia artificial para apoyar el proceso de codificación y revisión del código fuente. La Figura 10 presenta un ejemplo de las recomendaciones generadas por la herramienta durante un despliegue. Estas sugerencias facilitaron la corrección de problemas identificados por las herramientas de análisis automatizado y apoyaron al desarrollador en la aplicación de buenas prácticas de programación, sin reemplazar el proceso de validación realizado mediante los controles de seguridad del pipeline.

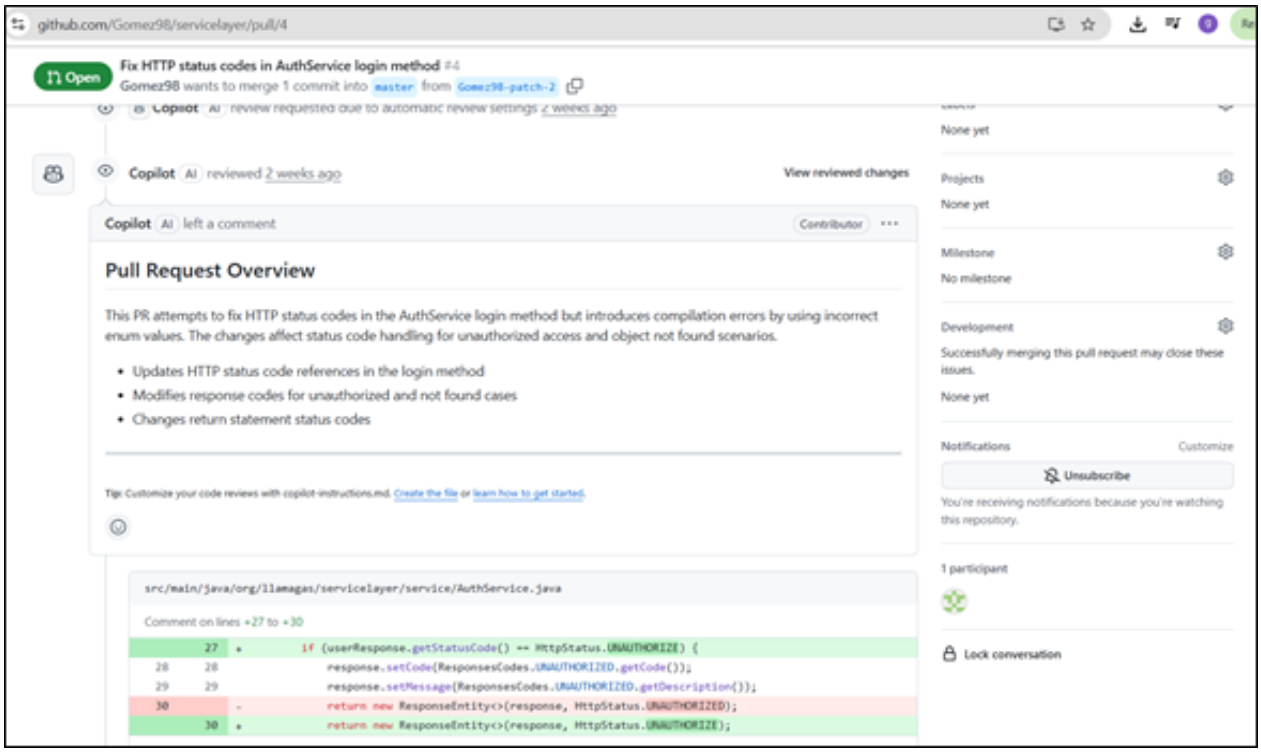


Figura 10. Ejemplo de asistencia proporcionada por GitHub Copilot durante un despliegue.

Posteriormente, el código fue sometido al proceso de validación automatizada mediante SonarQube Cloud, cuyos resultados se presentan en la Figura 11. El reporte generado incluye la clasificación de los hallazgos según su severidad, tipo de vulnerabilidad, ubicación dentro del código fuente y tiempo estimado de remediación, además de indicadores de calidad y mantenibilidad utilizados para la validación mediante Quality Gates. Estos reportes constituyeron una de las principales fuentes de información para el cálculo de la densidad de hallazgos detectados (DH) y del tiempo de detección de vulnerabilidades (TD).

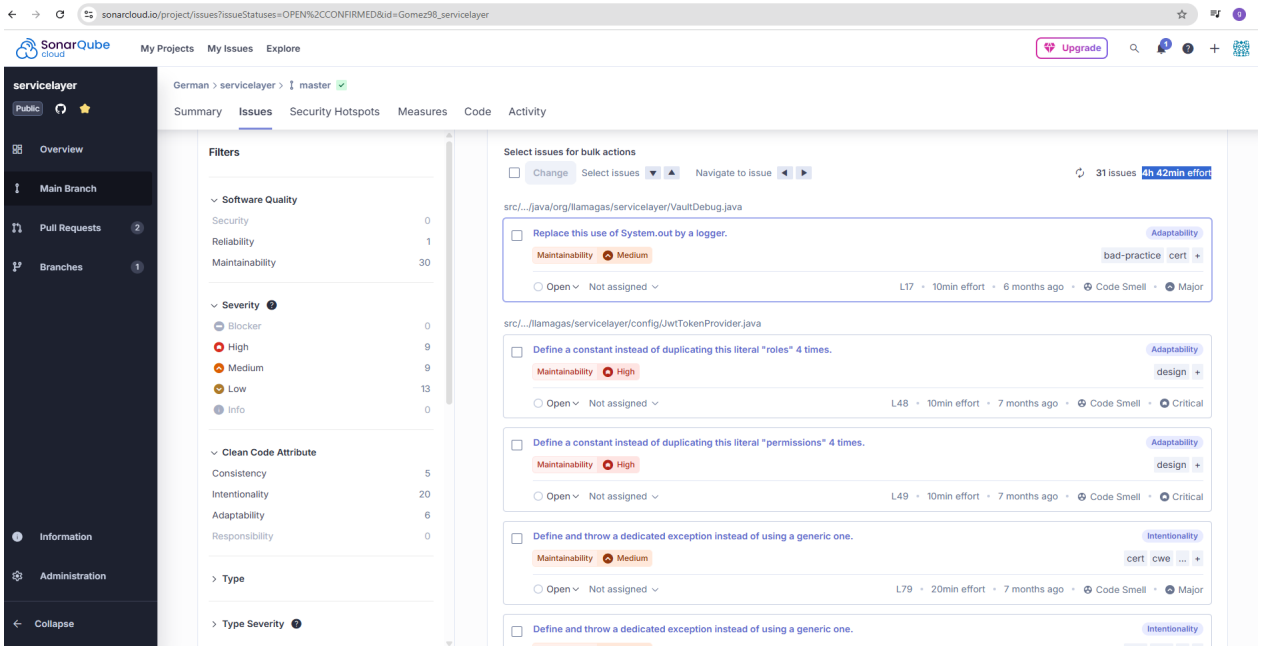


Figura 11. Reporte de análisis generado por SonarQube Cloud.

En conjunto, la evidencia presentada demuestra el cambio del proceso de revisión manual hacia un esquema automatizado basado en prácticas DevSecOps fortalecidas con inteligencia artificial. La integración de GitHub Copilot, SonarQube Cloud y Snyk permitió disponer de información objetiva y en tiempo real sobre la calidad y seguridad del software, proporcionando la base para el análisis estadístico de los indicadores evaluados y la determinación del impacto de la intervención.

A partir de los registros obtenidos mediante las herramientas implementadas y las fichas de observación, se consolidó la base de datos correspondiente a los 25 despliegues analizados. Sobre esta información se calcularon los indicadores DH, TD y TC, cuyos resultados descriptivos y comparativos se presentan en la siguiente sección resultados.

Evaluación de cumplimiento de objetivos

Los resultados obtenidos evidencian el cumplimiento de los objetivos planteados en la investigación, al demostrar que la implementación de prácticas DevSecOps fortalecidas con inteligencia artificial contribuyó a mejorar la seguridad y la eficiencia del proceso de desarrollo de software. La comparación entre las mediciones de pretest y postest permitió identificar mejoras en los indicadores evaluados, reflejadas en una mayor capacidad para detectar vulnerabilidades durante las etapas tempranas del ciclo de desarrollo, así como en la reducción de los tiempos requeridos para su identificación y remediación.

La integración de GitHub Copilot, SonarQube y Snyk dentro del pipeline de integración y despliegue continuo (CI/CD) permitió automatizar actividades de análisis y validación del código fuente, fortaleciendo los mecanismos de control de calidad y seguridad durante el proceso de desarrollo. Asimismo, los reportes generados por estas herramientas proporcionaron información objetiva y trazable para el cálculo de la densidad de hallazgos detectados (DH), el tiempo de

detección de vulnerabilidades (TD) y el tiempo de corrección de vulnerabilidades (TC), facilitando la evaluación cuantitativa del impacto de la intervención.

La Tabla 10 presenta la evaluación del cumplimiento de los objetivos específicos de la investigación, relacionando cada objetivo con los indicadores utilizados y los resultados alcanzados.

Tabla 10. Evaluación del cumplimiento de los objetivos de la investigación.

Objetivo específico	Indicador de evaluación	Resultado esperado	Resultado obtenido	Cumplimiento
Implementar un pipeline CI/CD basado en prácticas DevSecOps fortalecidas con inteligencia artificial.	Implementación funcional del pipeline.	Pipeline DevSecOps operativo con integración de GitHub Copilot, SonarQube y Snyk.	Se implementó y validó un pipeline CI/CD con integración de herramientas de inteligencia artificial y seguridad.	Si
Incrementar la detección temprana de vulnerabilidades durante el desarrollo de software.	Densidad de hallazgos detectados (DH).	Mayor identificación de vulnerabilidades antes del despliegue.	Se incrementó la cantidad de hallazgos detectados durante las etapas tempranas del desarrollo mediante análisis automatizados.	Si
Reducir el tiempo de detección de vulnerabilidades.	Tiempo de detección de vulnerabilidades (TD).	Disminución del tiempo requerido para identificar vulnerabilidades.	Se redujo el tiempo necesario para detectar vulnerabilidades gracias a la automatización del pipeline y los análisis continuos.	Si
Reducir el tiempo de corrección de vulnerabilidades.	Tiempo de corrección de vulnerabilidades (TC).	Disminución del tiempo empleado para corregir vulnerabilidades.	La asistencia proporcionada por GitHub Copilot y la retroalimentación automática de las herramientas permitieron reducir el tiempo de remediación.	Si

En conjunto, los resultados obtenidos respaldan la efectividad de la solución propuesta y demuestran que la incorporación de prácticas DevSecOps apoyadas por inteligencia artificial constituye una estrategia viable para fortalecer la seguridad y la eficiencia del desarrollo de software. La automatización de los procesos de análisis, la integración continua de controles de seguridad y el apoyo de herramientas basadas en inteligencia artificial favorecieron la detección temprana de vulnerabilidades, optimizaron los tiempos de respuesta y contribuyeron a mejorar la calidad del software desarrollado.

Fase 4: Actuar

Hoja de ruta para la institucionalización de DevSecOps fortalecido con inteligencia artificial

Con base en los resultados obtenidos durante la fase de verificación, se estableció una hoja de ruta orientada a consolidar e institucionalizar las prácticas DevSecOps fortalecidas con inteligencia artificial dentro del proceso de desarrollo de software de la organización. El propósito de esta fase fue garantizar la sostenibilidad de la solución implementada, promoviendo la mejora continua y facilitando su escalabilidad hacia otros proyectos de la empresa.

Como primera acción, se formalizaron los controles de seguridad y calidad que demostraron mayor efectividad durante la intervención. Entre ellos se incluyó la adopción obligatoria de Quality Gates en SonarQube como requisito previo para la aprobación de pull requests, la ejecución automática de análisis de vulnerabilidades mediante Snyk durante los procesos de integración continua y en ejecuciones programadas, así como la incorporación de verificaciones previas al commit para fortalecer la detección temprana de vulnerabilidades.

De manera complementaria, se diseñó una hoja de ruta para la evolución del modelo DevSecOps, contemplando la incorporación progresiva de nuevas capacidades de seguridad y automatización. Las principales iniciativas propuestas se presentan en la Figura 12.

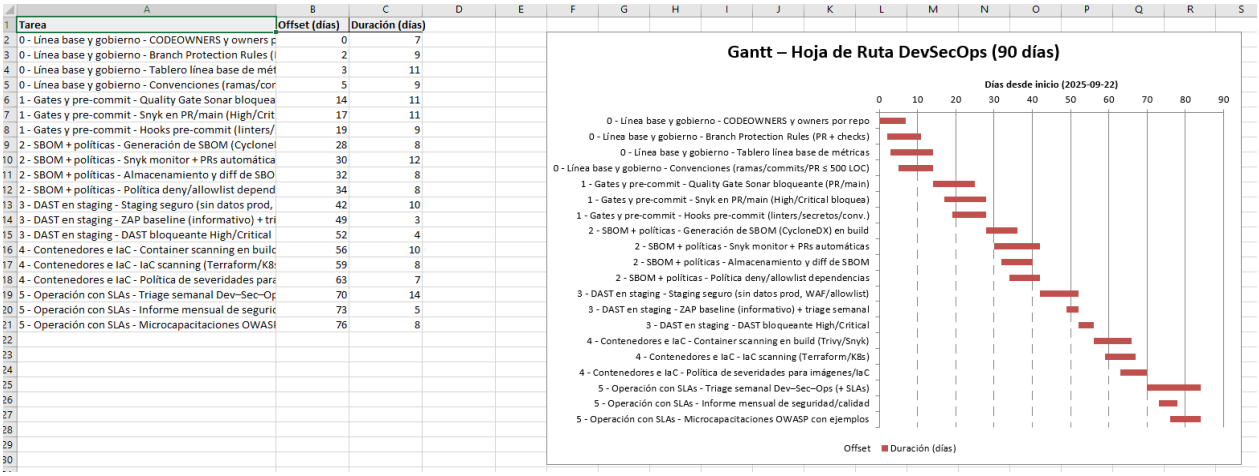


Figura 12. Hoja de ruta para la evolución del modelo DevSecOps fortalecido con inteligencia artificial.

La hoja de ruta considera la incorporación de análisis dinámico de seguridad de aplicaciones (DAST) en el entorno de staging, la generación y gestión del Software Bill of Materials (SBOM) para fortalecer la trazabilidad de componentes de software, la automatización de políticas de actualización de dependencias y el fortalecimiento de los mecanismos de monitoreo continuo del pipeline CI/CD. Estas acciones permitirán ampliar progresivamente el alcance de la estrategia de seguridad implementada y mejorar la capacidad de respuesta frente a nuevas amenazas.

Asimismo, se establecieron acuerdos operativos entre los equipos de desarrollo, seguridad y operaciones con el propósito de fortalecer la gobernanza del proceso DevSecOps. Entre las acciones definidas se incluyeron reuniones periódicas para la revisión y priorización de vulnerabilidades según su nivel de criticidad e impacto sobre el negocio, el seguimiento de indicadores de desempeño y la definición de metas orientadas a la mejora continua del proceso de desarrollo.

Finalmente, para garantizar la sostenibilidad de la solución implementada, se definieron acciones de mejora orientadas a la calibración de las reglas de análisis de seguridad, la optimización del proceso de revisión de pull requests, la estandarización de criterios para la gestión de excepciones y el fortalecimiento continuo de las competencias del equipo mediante programas de capacitación sobre desarrollo seguro y vulnerabilidades del OWASP Top 10. Estas acciones consolidan el ciclo PHVA, al transformar los resultados obtenidos durante la intervención en prácticas organizacionales permanentes, favoreciendo la mejora continua, la replicabilidad del modelo y su adopción

progresiva en otros proyectos de desarrollo de software.

RESULTADOS

Confiabilidad de los instrumentos

Con el propósito de garantizar la consistencia del instrumento de recolección de datos, se aplicó un procedimiento de confiabilidad test-retest, comparando las mediciones obtenidas en dos momentos del estudio. Los resultados presentados en la Tabla 11 evidencian coeficientes superiores al umbral de 0,70, lo que indica una adecuada estabilidad temporal del instrumento.

En este sentido, se concluye que la ficha de observación utilizada en la investigación presenta niveles aceptables de confiabilidad, permitiendo asegurar la consistencia de las mediciones de los indicadores DH, TD y TC durante las fases de pretest y postest.

Tabla 11. Confiabilidad.

Instrumento	Prueba	Indicador	Resultado
Ficha de observación pre y pos	Test y retest	Densidad de hallazgos detectados (DH)	0,994
Ficha de observación pre y pos	Test y retest	Tiempo de detección (TD)	0,822
Ficha de observación pre y pos	Test y retest	Tiempo de corrección (TC)	0,939

Análisis descriptivo

A continuación, se presenta los estadísticos descriptivos obtenidos para los indicadores evaluados antes (pretest) y después (postest) de la implementación de las prácticas DevSecOps fortalecidas con inteligencia artificial.

En relación con la densidad de hallazgos detectados (DH), el promedio aumentó de 1,95 a 5,79 hallazgos por despliegue, lo que evidencia una mayor capacidad del proceso para identificar vulnerabilidades durante las etapas tempranas del desarrollo (Figura 13).

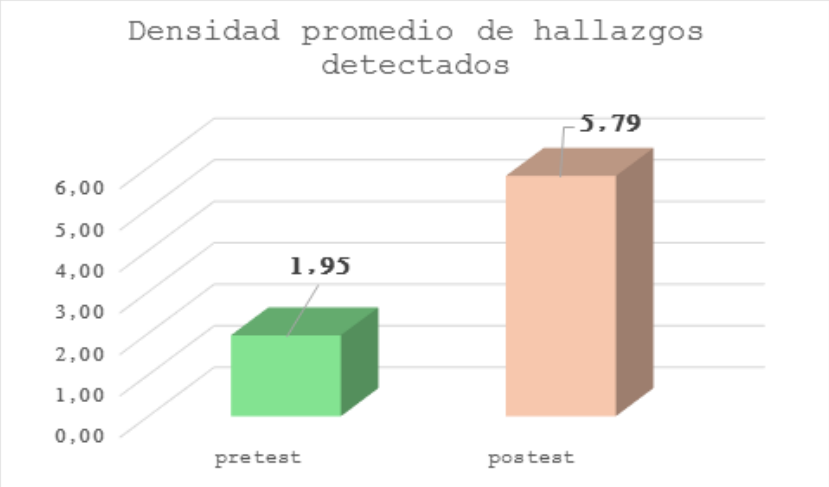


Figura 13. Densidad promedio de hallazgos detectados (DH).

Respecto al tiempo de detección de vulnerabilidades (TD), el promedio disminuyó de 1,47 horas a 0,014 horas, indicando una reducción sustancial del tiempo requerido para detectar vulnerabilidades mediante el pipeline automatizado (Figura 14).

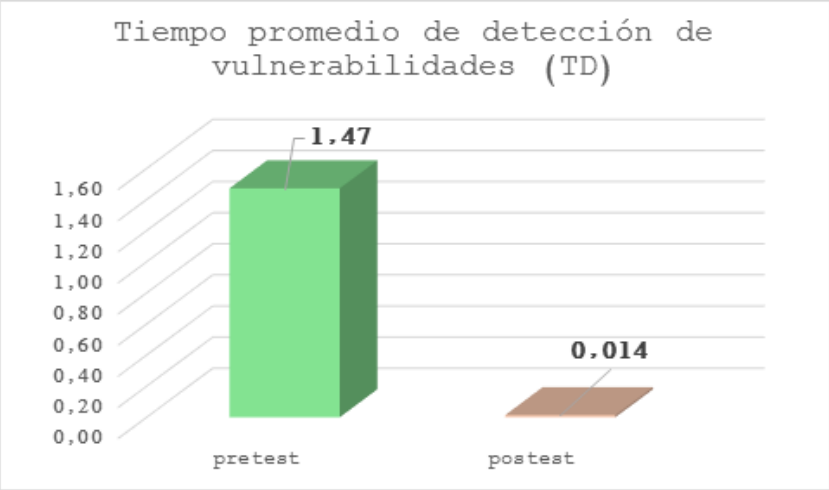


Figura 14. Tiempo promedio de detección de vulnerabilidades (TD).

Asimismo, el tiempo promedio de corrección de vulnerabilidades (TC) presentó una reducción importante, pasando de 3,31 horas en el pretest a 0,41 horas en el postest, lo que refleja una mejora significativa en la eficiencia del proceso de remediación (Figura 15).

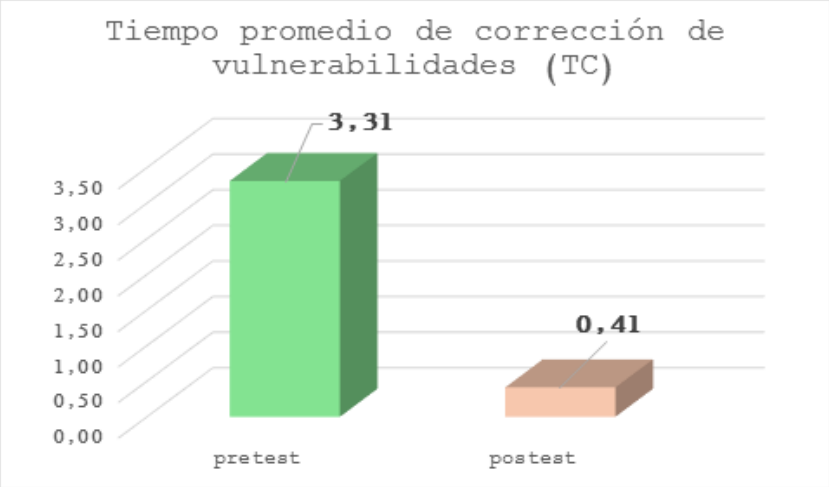


Figura 15. Tiempo promedio de corrección de vulnerabilidades (TC).

Análisis inferencial

Prueba de normalidad

Antes de realizar el análisis inferencial, se evaluó el supuesto de normalidad mediante la prueba de Shapiro–Wilk, recomendada para muestras pequeñas ($n < 50$) debido a su mayor potencia estadística frente a otras pruebas de normalidad. Los resultados evidenciaron valores de significancia inferiores a 0,05 para los tres indicadores analizados (DH: $W = 0{,}896$, $p = 0{,}015$; TD: $W = 0{,}898$, $p = 0{,}017$; TC: $W = 0{,}645$, $p < 0{,}001$), lo que indica que las distribuciones difieren significativamente de una distribución normal. En consecuencia, se rechazó el supuesto de normalidad y se optó por emplear la prueba no paramétrica de rangos con signo de Wilcoxon para comparar las mediciones obtenidas en el pretest y el postest (Tabla 12).

Tabla 12. Prueba de Shapiro-Wilk.

Indicador	Estadístico W	Shapiro-Wilk p
DH	0,896	0,015
TD	0,898	0,017
TC	0,645	<0,001

Contrastación de Hipótesis

Tabla 13. Prueba de Wilcoxon.

Indicador	Z	p	r	Magnitud
DH	-4,37	<0,001	0,87	Grande
TD	-4,37	<0,001	0,87	Grande
TC	-4,37	<0,001	0,87	Grande

Además del nivel de significancia estadística, se calculó el tamaño del efecto (r) para cuantificar la magnitud de las diferencias observadas entre el pretest y el postest. El tamaño del efecto fue obtenido mediante la expresión $r = |\mathbb{Z}|/\sqrt{N}$, donde Z corresponde al estadístico de la prueba de

Wilcoxon y N al número de observaciones pareadas. Los tres indicadores presentaron un tamaño del efecto de r = 0,87, considerado grande de acuerdo con los criterios propuestos por [29], lo que indica que la implementación de prácticas DevSecOps fortalecidas con inteligencia artificial produjo cambios sustanciales tanto en la detección como en la corrección de vulnerabilidades. Asimismo, para determinar si existían diferencias estadísticamente significativas entre las mediciones de pretest y postest se aplicó la prueba de rangos con signo de Wilcoxon para muestras relacionadas (Tabla 13).

Los resultados evidenciaron diferencias estadísticamente significativas en los tres indicadores evaluados ($p<0,001$). En consecuencia, se rechazó la hipótesis nula y se aceptó la hipótesis de investigación, concluyendo que la implementación de prácticas DevSecOps fortalecidas con inteligencia artificial produjo mejoras significativas en la densidad de hallazgos detectados, así como reducciones significativas en el tiempo de detección y el tiempo de corrección de vulnerabilidades.

Medianas e IQR

Tabla 14. Medianas e IQR.

Indicador	Pretest Mediana (IQR)	Postest Mediana (IQR)
DH	1,51 (1,37–2,54)	4,53 (3,96–7,43)
TD	1,30 (1,08–1,88)	0,012 (0,010–0,015)
TC	2,43 (2,05–3,03)	0,33 (0,292–0,39)

Debido a que las variables no presentaron distribución normal, además de los promedios se reportaron medidas robustas de tendencia central y dispersión. La mediana de la densidad de hallazgos detectados aumentó de 1,51 (IQR: 1,37–2,54) a 4,53 (IQR: 3,96–7,43). Asimismo, la mediana del tiempo de detección disminuyó de 1,30 horas (IQR: 1,08–1,88) a 0,012 horas (IQR: 0,010–0,015), mientras que el tiempo de corrección disminuyó de 2,43 horas (IQR: 2,05–3,03) a 0,33 horas (IQR: 0,292–0,39), Tabla 14.

Boxplots

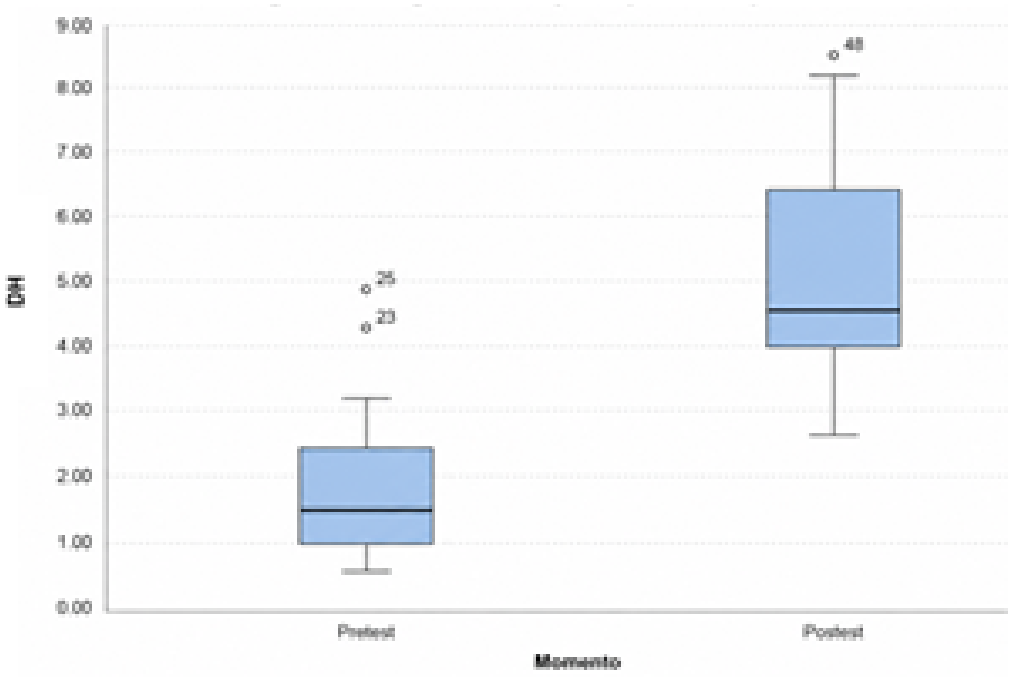


Figura 16. Diagrama de cajas simple de DH por momento.

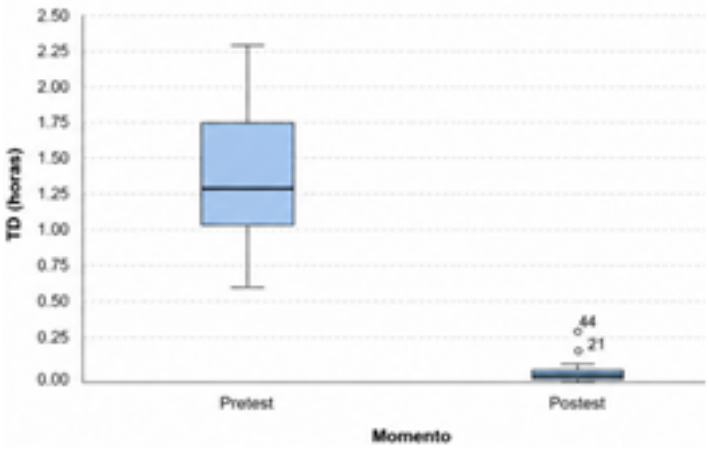


Figura 17. Diagrama de cajas simple de TD por momento.

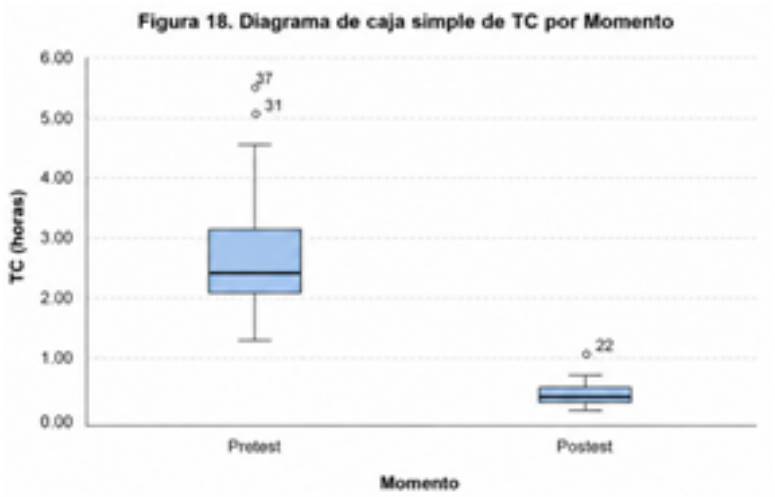


Figura 18. Diagrama de cajas simple de TC por momento.

La Figura 16, Figura 17 y Figura 18 muestran los diagramas de caja correspondientes a los tres indicadores evaluados antes y después de la implementación de las prácticas DevSecOps fortalecidas con inteligencia artificial. En conjunto, los boxplots evidencian un cambio sustancial en la distribución de los datos entre el pretest y el postest. La densidad de hallazgos detectados (DH) presenta un desplazamiento de la mediana hacia valores superiores en el postest, indicando una mayor capacidad para identificar vulnerabilidades durante el proceso de desarrollo. En contraste, el tiempo de detección de vulnerabilidades (TD) y el tiempo de corrección (TC) muestran un desplazamiento de sus medianas hacia valores considerablemente menores, reflejando una reducción importante en los tiempos requeridos para detectar y remediar vulnerabilidades. Asimismo, la menor dispersión observada en los boxplots del postest para TD y TC sugiere un proceso más estable y consistente tras la implementación de la estrategia DevSecOps con inteligencia artificial. Aunque se observan algunos valores atípicos en ambos momentos de evaluación, estos no alteran la tendencia general de mejora evidenciada en los tres indicadores, respaldando visualmente los resultados obtenidos mediante la prueba de rangos con signo de Wilcoxon.

DISCUSIÓN

Interpretación general de los resultados

Los resultados obtenidos demuestran que la implementación de prácticas DevSecOps fortalecidas con inteligencia artificial mejoró significativamente los indicadores de seguridad y eficiencia del proceso de desarrollo de software. El incremento observado en la densidad de hallazgos detectados no representa un deterioro de la calidad del software, sino una mayor capacidad del pipeline CI/CD para identificar de manera temprana los hallazgos reportados por las herramientas automatizadas de análisis, incluyendo vulnerabilidades de seguridad, defectos de código (bugs), code smells y otros problemas de mantenibilidad. En este contexto, el indicador refleja la efectividad de los mecanismos de detección incorporados durante el proceso de integración continua y no un incremento en la cantidad de problemas introducidos en el software.

Comparación con estudios anteriores

Los resultados obtenidos son consistentes con la revisión multivocal de [21], quienes identificaron que la automatización de controles de seguridad, la adopción del enfoque shift-left y la integración continua constituyen las principales dimensiones que favorecen el éxito de DevSecOps. Del mismo modo, [22] concluyeron que la integración temprana de controles de seguridad dentro del ciclo de desarrollo reduce la exposición a vulnerabilidades y mejora la colaboración entre los equipos de desarrollo, operaciones y seguridad.

En relación con la incorporación de inteligencia artificial, los resultados coinciden con lo reportado por [23], quienes sostienen que los modelos de IA permiten automatizar tareas de análisis de código, priorización de vulnerabilidades y asistencia en la toma de decisiones dentro del pipeline DevSecOps. De forma similar, [24] señalan que la aplicación de técnicas de aprendizaje automático contribuye a optimizar los procesos de pruebas de software y acelerar la identificación de defectos durante el desarrollo.

Explicar el incremento de DH

El incremento de la densidad de hallazgos detectados debe interpretarse como un indicador de mejora en la capacidad de inspección del proceso y no como un incremento real de vulnerabilidades o defectos del software. La integración de herramientas de análisis estático (SAST), análisis de composición de software (SCA) y mecanismos automatizados de validación permitió detectar un mayor número de vulnerabilidades, defectos de código, code smells y otros hallazgos de calidad que anteriormente podían pasar inadvertidos durante las revisiones manuales o los controles tradicionales. Esta interpretación coincide con las recomendaciones de [25], quienes evidenciaron que la combinación de diferentes técnicas de análisis incrementa significativamente la cobertura de detección respecto al uso aislado de herramientas SAST.

Implicaciones prácticas de los hallazgos

Los resultados obtenidos evidencian que la incorporación de prácticas DevSecOps fortalecidas con inteligencia artificial genera beneficios relevantes para la gestión de la seguridad y la eficiencia en el desarrollo de software. En primer lugar, la detección temprana de vulnerabilidades durante la evaluación de los pull requests reduce la probabilidad de que código inseguro sea integrado a las ramas principales del repositorio, disminuyendo la exposición del software a riesgos de seguridad y favoreciendo la aplicación del enfoque shift-left dentro del ciclo de vida del desarrollo de software (SDLC) [21], [22].

Asimismo, la reducción observada en el tiempo de detección (TD) y en el tiempo de corrección (TC) demuestra que la automatización de los procesos de análisis de seguridad contribuye a optimizar el flujo de desarrollo, reduciendo retrabajos y acelerando la entrega de versiones más seguras del software. La integración de herramientas como SonarQube, Snyk y GitHub Copilot permitió proporcionar retroalimentación inmediata a los desarrolladores, favoreciendo la corrección temprana de vulnerabilidades antes de su propagación hacia etapas posteriores del proceso [23], [21].

Desde la perspectiva organizacional, la implementación de políticas de calidad basadas en Quality Gates, criterios estandarizados de severidad, excepciones controladas y acuerdos de niveles de

servicio (SLA) para la remediación de vulnerabilidades fortalece la gobernanza del proceso DevSecOps. Estas prácticas favorecen la trazabilidad de las decisiones de seguridad, promueven una gestión más eficiente de los riesgos y facilitan la integración efectiva entre los equipos de desarrollo, operaciones y seguridad [21], [22].

Por otra parte, la inteligencia artificial actúa como un mecanismo de apoyo para la toma de decisiones dentro del pipeline CI/CD, proporcionando capacidades de análisis automatizado, priorización de vulnerabilidades y asistencia durante la generación y refactorización del código. En este sentido, la IA no reemplaza la revisión realizada por los desarrolladores, sino que complementa el proceso mediante recomendaciones basadas en datos que permiten incrementar la productividad y mejorar la calidad del software desarrollado [25], [23].

Finalmente, los resultados obtenidos respaldan la conveniencia de evolucionar progresivamente el modelo implementado mediante la incorporación de capacidades complementarias orientadas a fortalecer la seguridad del proceso de desarrollo. Entre estas se encuentran la generación y gestión del Software Bill of Materials (SBOM) para mejorar la trazabilidad de los componentes de software y el análisis de la infraestructura como código (IaC) para identificar vulnerabilidades de configuración desde las etapas iniciales del ciclo de desarrollo. Estas prácticas amplían la cobertura de los controles de seguridad y contribuyen a fortalecer la gestión de riesgos en entornos DevSecOps [26], [27].

Limitaciones del estudio

La presente investigación empleó un diseño preexperimental con medición antes y después de la implementación de prácticas DevSecOps apoyadas por inteligencia artificial. Si bien este diseño permite evaluar los cambios observados en los indicadores de seguridad y eficiencia del proceso de desarrollo de software, presenta limitaciones relacionadas con la validez interna debido a la ausencia de un grupo de control, lo que restringe la capacidad para atribuir de manera exclusiva los efectos observados a la intervención implementada.

Entre las principales amenazas a la validez interna se identifican las siguientes:

Variables de confusión. Durante el período de implementación pudieron presentarse cambios organizacionales, tecnológicos o metodológicos ajenos a la propuesta, tales como actualizaciones de herramientas de desarrollo, mejoras en la infraestructura tecnológica o modificaciones en las políticas de gestión de proyectos, que potencialmente influyeron en los indicadores evaluados.

Efecto de aprendizaje del equipo. El desempeño del equipo de desarrollo pudo mejorar como consecuencia de la experiencia adquirida, la práctica continua y la familiarización progresiva con las herramientas y procesos utilizados, independientemente de la implementación de las prácticas DevSecOps apoyadas por inteligencia artificial.

Madurez incremental del proceso. La evolución natural de los procesos de desarrollo y de integración continua dentro de la organización pudo generar mejoras graduales en la calidad del software, la detección de vulnerabilidades y la eficiencia operativa, aun en ausencia de la intervención propuesta.

Sesgos temporales o históricos. Durante el período de estudio pudieron ocurrir eventos externos que afectaran los resultados, como variaciones en la carga de trabajo, cambios en la complejidad de los despliegues, modificaciones en los requerimientos funcionales o ajustes en la planificación del proyecto.

Instrumentación. La incorporación o actualización de herramientas automatizadas para el análisis estático de código, el análisis de dependencias o la detección de vulnerabilidades pudo incrementar la capacidad de identificación de hallazgos, generando diferencias entre las mediciones del pretest y el postest que no dependieran exclusivamente de la intervención.

Con el propósito de reducir parcialmente estas amenazas, el estudio se desarrolló sobre un mismo proyecto de software, utilizando el mismo pipeline de integración y despliegue continuo (CI/CD), los mismos indicadores de evaluación, criterios homogéneos de medición y un período continuo de observación. Asimismo, la muestra fue seleccionada mediante un criterio técnico previamente definido, considerando los despliegues con mayor volumen de líneas de código modificadas (LOC), lo que permitió mantener uniformidad en las condiciones de evaluación. No obstante, estas medidas no eliminan completamente las amenazas propias de un diseño preexperimental.

En consecuencia, los resultados deben interpretarse como evidencia de una asociación entre la implementación de prácticas DevSecOps apoyadas por inteligencia artificial y las mejoras observadas en los indicadores de seguridad y eficiencia. Sin embargo, debido a la ausencia de un grupo de control, no es posible establecer una relación causal definitiva ni atribuir de manera exclusiva todos los cambios observados a la intervención implementada.

Como línea futura de investigación, se recomienda desarrollar estudios con diseños cuasiexperimentales o experimentales que incorporen grupos de control, asignación de tratamientos y períodos de seguimiento más amplios, con el propósito de fortalecer la inferencia causal y validar con mayor rigor el impacto de la implementación de prácticas DevSecOps apoyadas por inteligencia artificial.

Otros alcances

En conjunto, la evidencia obtenida demuestra que la implementación de prácticas DevSecOps fortalecidas con inteligencia artificial constituye una estrategia viable para incrementar simultáneamente la seguridad y la eficiencia del desarrollo de software. El aumento de la densidad de hallazgos detectados refleja una mayor capacidad para identificar vulnerabilidades en etapas tempranas del ciclo de desarrollo, mientras que la disminución del tiempo de detección y del tiempo de corrección evidencia una mejora significativa en la eficiencia operativa del proceso.

Estos resultados son consistentes con la literatura reciente, que reconoce la integración temprana de controles de seguridad, la automatización del pipeline CI/CD y el uso de inteligencia artificial como factores clave para la madurez de DevSecOps y el fortalecimiento del desarrollo seguro de software [10], [21], [22], [28]. Asimismo, los hallazgos respaldan la conveniencia de continuar evolucionando el modelo mediante la incorporación de capacidades complementarias, como la gestión de Software Bill of Materials (SBOM), el análisis de infraestructura como código (IaC) y otras prácticas orientadas a fortalecer la seguridad de la cadena de suministro del software [26], [27].

A partir de los resultados obtenidos, se plantean diversas recomendaciones orientadas a fortalecer la adopción de prácticas DevSecOps apoyadas por inteligencia artificial y ampliar el alcance de futuras investigaciones.

En primer lugar, se recomienda consolidar la estandarización organizacional mediante la definición de políticas institucionales que establezcan Quality Gates obligatorios, criterios mínimos de severidad para la integración de cambios y acuerdos de nivel de servicio (SLA) para la remediación de vulnerabilidades, incorporando estos lineamientos dentro de la definición de Done de los equipos de desarrollo.

En segundo lugar, resulta conveniente ampliar la cobertura de los controles de seguridad incorporando mecanismos complementarios, tales como análisis dinámico de seguridad (DAST) en entornos de staging, análisis de infraestructura como código (IaC), generación y validación de Software Bill of Materials (SBOM), detección continua de credenciales expuestas y estrategias automatizadas para la actualización y gestión de dependencias. Estas prácticas contribuirán a fortalecer la seguridad de la cadena de suministro del software y a incrementar la madurez del modelo DevSecOps implementado.

Asimismo, se recomienda implementar mecanismos de monitoreo continuo mediante tableros de control que integren indicadores clave de desempeño, como la densidad de hallazgos detectados (DH), el tiempo de detección de vulnerabilidades (TD), el tiempo de corrección (TC), el porcentaje de pull requests aprobados en el primer intento y la evolución de la deuda técnica. El seguimiento periódico de estos indicadores facilitará la toma de decisiones basada en evidencia y la mejora continua de los procesos de desarrollo.

Desde la perspectiva del desarrollo de capacidades, se sugiere establecer programas permanentes de capacitación orientados a fortalecer las competencias del equipo en DevSecOps, inteligencia artificial aplicada al desarrollo seguro y buenas prácticas de ciberseguridad, complementados con ejercicios prácticos basados en vulnerabilidades reales identificadas durante el desarrollo de los proyectos.

Finalmente, futuras investigaciones podrían validar el modelo propuesto en diferentes organizaciones y contextos tecnológicos, incorporando muestras de mayor tamaño y diseños cuasiexperimentales o experimentales que permitan incrementar la validez externa de los resultados. Asimismo, sería pertinente evaluar indicadores adicionales, como la reducción de defectos en producción, el impacto económico asociado a la disminución del retrabajo, la calidad del código generado por asistentes basados en Large Language Models (LLMs) y la incorporación de nuevas capacidades de seguridad impulsadas por inteligencia artificial. Estas líneas de investigación contribuirán a consolidar un modelo de adopción gradual, escalable y sostenible de DevSecOps con inteligencia artificial en empresas dedicadas al desarrollo de software.

CONCLUSIONES

La implementación de prácticas DevSecOps fortalecidas con inteligencia artificial demostró ser una estrategia efectiva para mejorar la seguridad y la eficiencia del desarrollo de software en la empresa objeto de estudio. La integración de herramientas de análisis estático de código, análisis

de composición de software, políticas de Quality Gates y asistencia basada en inteligencia artificial permitió automatizar controles de seguridad, fortalecer el proceso de integración continua y optimizar la detección y remediación de vulnerabilidades durante las etapas tempranas del ciclo de vida del software. En consecuencia, se alcanzó satisfactoriamente el objetivo general de la investigación.

Con respecto al primer objetivo específico, se determinó que la implementación de prácticas DevSecOps con inteligencia artificial incrementó significativamente la densidad de hallazgos detectados. El promedio pasó de 1,95 a 5,79 hallazgos por despliegue, lo que representa un incremento del 196,9 %. Este resultado evidencia una mayor capacidad del proceso para identificar vulnerabilidades y problemas de calidad antes de la integración del código, fortaleciendo el aseguramiento de la seguridad sin afectar la continuidad del desarrollo.

En relación con el segundo objetivo específico, se comprobó una reducción significativa en el tiempo de detección de vulnerabilidades. El tiempo promedio disminuyó de 1,47 horas a 0,014 horas, equivalente a una reducción del 99,05 %, demostrando que la incorporación de controles automatizados dentro del pipeline CI/CD permitió detectar vulnerabilidades de manera prácticamente inmediata y reducir considerablemente la exposición del software a riesgos de seguridad.

Respecto al tercer objetivo específico, se evidenció una mejora significativa en el proceso de remediación de vulnerabilidades. El tiempo promedio de corrección pasó de 3,31 horas a 0,41 horas, lo que representa una reducción del 87,61 %. Este resultado demuestra que la combinación de herramientas de análisis automatizado y asistencia basada en inteligencia artificial favoreció la priorización y resolución temprana de vulnerabilidades, reduciendo el retrabajo e incrementando la eficiencia operativa del equipo de desarrollo.

Asimismo, el análisis inferencial confirmó que las diferencias observadas entre las mediciones previas y posteriores a la implementación fueron estadísticamente significativas para los tres indicadores evaluados ($p < 0,001$), con tamaños del efecto elevados, lo que evidencia que las mejoras alcanzadas poseen no solo significancia estadística, sino también relevancia práctica dentro del contexto de una fábrica de software.

Finalmente, los resultados permiten concluir que la integración de prácticas DevSecOps con inteligencia artificial constituye una alternativa viable para fortalecer la madurez de los procesos de desarrollo seguro. La automatización de controles, el uso de asistentes inteligentes para apoyar la codificación y la incorporación de mecanismos de validación continua contribuyen a mejorar la calidad del software, reducir los tiempos de respuesta ante vulnerabilidades y fortalecer la competitividad de las organizaciones dedicadas al desarrollo de software.

El modelo propuesto constituye una evidencia empírica de que la integración de prácticas DevSecOps con herramientas de inteligencia artificial puede generar mejoras cuantificables en los procesos de aseguramiento de la seguridad y eficiencia del desarrollo de software, aportando una referencia para futuras implementaciones en empresas de fábrica de software.

FINANCIAMIENTO

Los autores declaran no haber recibido financiamiento para este trabajo.

REFERENCIAS

[1] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*, Boston, MA, USA: Addison-Wesley Professional, 2015. [↑]

[2] G. McGraw, *Software Security: Building Security In*, Boston, MA, USA: Addison-Wesley Professional, 2006. [↑]

[3] N. M. Mohammed, M. Niazi, M. Alshayeb, and S. Mahmood, "Exploring software security approaches in software development lifecycle: A systematic mapping study," Computer Standards & Interfaces, vol. 50, pp. 107–115, 2017, https://doi.org/10.1016/j.csi.2016.10.001 [↑]

[4] B. W. Boehm, *Software Engineering Economics*, Englewood Cliffs, NJ, USA: Prentice-Hall, 1981 [↑]

[5] R. N. Rajapakse, M. Zahedi, M. A. Babar, and H. Shen, "Challenges and solutions when adopting DevSecOps: A systematic review," *Information and Software Technology*, vol. 141, Art. no. 106700, 2022, https://doi.org/10.1016/j.infsof.2021.106700 [↑]

[6] H. Myrbakken and R. Colomo-Palacios, "DevSecOps: A multivocal literature review," in *Software Process Improvement and Capability Determination, Communications. Computer and Information Science*, A. Mas, A. Mesquida, R. O'Connor, T. Rout, and A. Dorling, Eds., vol. 770, Cham, Switzerland: Springer, 2017, pp. 17–29, https://doi.org/10.1007/978-3-319-67383-7_2 [↑]

[7] Z. Kotti, R. Galanopoulou, and D. Spinellis, "Machine learning for software engineering: a tertiary study," *ACM Computing Surveys*, vol. 55, no. 12, Art. no. 256, 2023, https://doi.org/10.1145/3572905 [↑]

[8] L. B. Germano, R. R. Goldschmidt, R. C. Noya, and J. C. Duarte, "A systematic review on detection, repair, and explanation of vulnerabilities in source code using large language models," *IEEE Access*, vol. 13, pp. 192263–192293, 2025, https://doi.org/10.1109/ACCESS.2025.3631363 [↑]

[9] H. Alqahtani and G. Kumar, "Large language models for cybersecurity intelligence: a systematic review of emerging threats, defensive capabilities, and security evaluation frameworks," *Computers, Materials & Continua*, vol. 87, no. 3, 2026, https://doi.org/10.32604/cmc.2026.077367 [↑]

[10] A. Mittal, "AI-enabled DevSecOps for secure cloud deployments," in *Revolutionizing the Cloud: Generative AI, Security, and Sustainability*, B. Pandey and A. Patel, Eds. Cham, Switzerland: Springer, 2026, pp. 323–347, https://doi.org/10.1007/978-3-032-07479-9_16 [↑]

[11] N. Alugunuri, "AI for continuous security in DevOps (DevSecOps): integrating machine learning into CI/CD pipelines," *International Journal of Intelligent Systems and Applications in Engineering,* vol. 12, no. 23s, pp. 3048–3058, 2024. [Online]. Available: https://ijisae.org/index.php/IJISAE/article/view/7603 [↑]

[12] G. P. Rusum, "Adaptive DevSecOps: integrating ai-driven threat detection in continuous delivery pipelines," *American International Journal of Computer Science and Technology*, vol. 7, no. 5, pp.

13–27, 2025, https://doi.org/10.63282/3117-5481/AIJCST-V7I5P102 [↑]

[13] C. S. Kummarapurugu, "A framework for real-time AI-driven secure code analysis integrated with DevSecOps in cloud-native CI/CD pipelines," *International Journal of Scientific Research in Engineering and Management*, vol. 6, no. 12, pp. 1–10, 2022, https://doi.org/10.55041/IJSREM17273 [↑]

[14] R. Ahuja, "The role of artificial intelligence and machine learning in DevSecOps: Leveraging predictive analytics, automated threat detection, and anomaly identification for secure software delivery pipelines," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 11, no. 2, pp. 381–389, 2023. [Online]. Available: https://ijritcc.org/index.php/ijritcc/article/view/11946 [↑]

[15] B. C. Das, M. H. Amini, and Y. Wu, "Security and privacy challenges of large language models: A survey," *ACM Computing Surveys*, vol. 57, no. 6, Art. no. 152, 2025, https://doi.org/10.1145/3712001 [↑]

[16] Y. Qu, S. Huang, and P. Nie, "A review of backdoor attacks and defenses in code large language models: Implications for security measures," *Information and Software Technology*, vol. 182, Art. no. 107707, 2025, https://doi.org/10.1016/j.infsof.2025.107707 [↑]

[17] N. O. Jaffal, M. Alkhanafseh, and D. Mohaisen, "Large language models in cybersecurity: a survey of applications, vulnerabilities, and defense techniques," *AI*, vol. 6, no. 9, Art. no. 216, 2025, https://doi.org/10.3390/ai6090216 [↑]

[18] S. Imai, "Is GitHub copilot a substitute for human pair-programming? an empirical study," in *2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, Pittsburgh, PA, USA, 2022, pp. 319–321, https://doi.org/10.1145/3510454.3522684 [↑]

[19] S. Yulianto and G. N. Cac Ngo, "Enhancing DevSecOps pipelines with ai-driven threat detection and response," in *2024 International Conference on ICT for Smart Society (ICISS)*, Bandung, Indonesia, 2024, pp. 1-8, https://doi.org/10.1109/ICISS62896.2024.10751269 [↑]

[20] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: a systematic literature review," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 8, Art. no. 220, pp. 1–79, 2024, https://doi.org/10.1145/3695988 [↑]

[21] X. Zhao, T. Clear, and R. Lal, "Identifying the primary dimensions of DevSecOps: A multi-vocal literature review," *Journal of Systems and Software*, vol. 214, Art. no. 112063, 2024, https://doi.org/10.1016/j.jss.2024.112063 [↑]

[22] M. Sinan, M. Shahin, and I. Gondal, "Integrating security controls in devsecops: challenges, solutions, and future research directions," *Journal of Software: Evolution and Process*, vol. 37, no. 6, Art. no. e70029, 2025, https://doi.org/10.1002/smr.70029 [↑]

[23] M. C. Fu, J. Pasuksmit, and C. Tantithamthavorn, "AI for DevSecOps: A landscape and future opportunities," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 4, Art. no.115, pp. 1–61, 2025, https://doi.org/10.1145/3712190 [↑]

[24] S. Ajorloo, A. Jamarani, M. Kashfi, M. H. Kashani, and M. A. B. Bafghi, "A systematic review of machine learning methods in software testing," *Applied Soft Computing*, vol. 162, Art. no. 111805, 2024, https://doi.org/10.1016/j.asoc.2024.111805 [↑]

[25] K. Li, S. Chen, L. Fan, R. Feng, H. Liu, C. Liu, Y. Liu, and Y. Chen, "Comparison and evaluation on static application security testing (SAST) Tools for Java," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2023, pp. 921–933, https://doi.org/10.1145/3611643.3616262 [↑]

[26] T. Stalnaker, N. Wintersgill, O. Chaparro, M. Di Penta, D. German, and D. Poshyvanyk, "BOMs away! inside the minds of stakeholders: A comprehensive study of bills of materials for software systems," in *Proceedings of the International Conference on Software Engineering*, 2024, Art. no. 44, https://doi.org/10.1145/3597503.3623347 [↑]

[27] A. Rahman, C. Parnin, and L. Williams, "The Seven Sins: Security smells in Infrastructure as Code scripts," in *Proceedings of the International Conference on Software Engineering* (ICSE), Montreal, QC, Canada, 2019, https://doi.org/10.1109/ICSE.2019.00033 [↑]

[28] M. Bedoya, S. Palacios, D. Díaz-López, E. Laverde, S. Suárez, and P. Nespoli, "Enhancing DevSecOps practice with Large Language Models and Security Chaos Engineering," *International Journal of Information Security*, vol. 23, pp. 3765–3788, 2024, https://doi.org/10.1007/s10207-024-00909-w [↑]

[29] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*, 2nd ed. Hillsdale, NJ, USA: Lawrence Erlbaum Associates, 1988. [↑]