PRISMA ODS
REVISTA MULTIDISCIPLINARIA
SOBRE DESARROLLO SOSTENIBLE
ISSN: 3072-8452

DEVSECOPS Y HERRAMIENTAS PARA EL DESARROLLO SEGURO DE SOFTWARE: UNA REVISIÓN BIBLIOGRÁFICA

DEVSECOPS AND TOOLS FOR SECURE SOFTWARE DEVELOPMENT: A LITERATURE REVIEW

AUTOR/AS

LYDIA CUEVAS BRACAMONTES
TECNOLÓGICO NACIONAL DE MÉXICO / INSTITUTO TECNOLÓGICO DE IGUALA
MÉXICO

SILVIA VALLE BAHENA
TECNOLÓGICO NACIONAL DE MÉXICO / INSTITUTO TECNOLÓGICO DE IGUALA
MÉXICO

SINDYA YADIRA CASTILLO ORTIZ
TECNOLÓGICO NACIONAL DE MÉXICO / INSTITUTO TECNOLÓGICO DE IGUALA
MÉXICO

ENRIQUE MENA SALGADO
TECNOLÓGICO NACIONAL DE MÉXICO / INSTITUTO TECNOLÓGICO DE IGUALA
MÉXICO

MARÍA DEL CARMEN URIÓSTEGUI PERALTA
TECNOLÓGICO NACIONAL DE MÉXICO / INSTITUTO TECNOLÓGICO DE IGUALA
MÉXICO

doi 10.65011

Volumen 5 - Número 5
2026

PRISMA ODS
REVISTA MULTIDISCIPLINARIA SOBRE DESARROLLO SOSTENIBLE
Redilat Red de Investigadores Latinoamericanos
LATINDES INSTITUTO LATINOAMERICANO EN DESARROLLO SOSTENIBLE

DevSecOps y herramientas para el desarrollo seguro de software: una revisión bibliográfica

DevSecOps and Tools for Secure Software Development: A Literature Review

Lydia Cuevas Bracamontes
lydia.cuevas@iguala.tecnm.mx
https://orcid.org/0000-0003-1624-7377
Tecnológico Nacional de México / Instituto Tecnológico de Iguala
Iguala, Guerrero – México

Sindya Yadira Castillo Ortiz[1]
sindya.castillo@iguala.tecnm.mx
https://orcid.org/0009-0007-6065-6764
Tecnológico Nacional de México / Instituto Tecnológico de Iguala
Iguala, Guerrero – México

Silvia Valle Bahena
silvia.valle@iguala.tecnm.mx
https://orcid.org/0000-0001-5587-7641
Tecnológico Nacional de México / Instituto Tecnológico de Iguala
Iguala, Guerrero – México

Enrique Mena Salgado
enrique.mena@iguala.tecnm.mx
https://orcid.org/0000-0002-8862-7355
Tecnológico Nacional de México / Instituto Tecnológico de Iguala
Iguala, Guerrero – México

María del Carmen Urióstegui Peralta
carmen.uriostegui@iguala.tecnm.mx
https://orcid.org/0009-0007-6661-9403
Tecnológico Nacional de México / Instituto Tecnológico de Iguala
Iguala, Guerrero – México

Artículo recibido: 06/10/2026
Aceptado para publicación: 06/11/2026
Conflictos de Intereses: Ninguno que declarar

[1] Autora de correspondencia.

RESUMEN

En la actualidad el desarrollo de software rara vez se lleva a cabo mediante el uso de una única herramienta. Por el contrario, un mismo proyecto puede involucrar diversas plataformas y recursos como un tablero de tareas, un repositorio con control de versiones, un motor de integración continua, un escáner de dependencias y finalmente, un entorno de despliegue en la nube, todo antes de que el usuario final vea una sola línea de código funcionando. La presente revisión bibliográfica analiza la gestión e integración de herramientas orientadas al desarrollo seguro de software, con énfasis en la incorporación de prácticas de seguridad desde las primeras etapas de ciclo de vida del desarrollo, en lugar de considerarlas como una actividad exclusiva de las fases previas a la puesta en producción. Para ello se examinaron veinte fuentes académicas y especializadas publicadas entre 2021 y 2025, abordando aspectos relacionados con DevSecOps, control de versiones, automatización, pruebas de seguridad y fundamentos de ciberseguridad. Los resultados de la revisión evidencian que la mayoría de los resultados analizados coinciden en que la incorporación de mecanismos de seguridad automatizados en los flujos de integración y entrega continua (CI/CD) favorece una detección más de vulnerabilidades y reduce la dependencia de revisiones realizadas en etapas avanzadas del ciclo de desarrollo. Asimismo, se identificó una presencia recurrente de herramientas como SonarQube, CodeQL, Snyk, Trivy y OWASP ZAP, las cuales son utilizadas para identificar vulnerabilidades, analizar el código y evaluar la seguridad de las aplicaciones sin afectar significativamente la continuidad de los procesos de desarrollo y despliegue. Como aporte de esta revisión, se propone una matriz de análisis que relaciona las herramientas estudiadas con su función técnica y el ámbito de ciberseguridad que atienden. Esta matriz busca servir como recurso de referencia para estudiantes, desarrolladores y equipos de trabajo interesados en seleccionar e integrar tecnologías de seguridad de acuerdo con las necesidades específicas de sus procesos de desarrollo de software.

Palabras clave: DevSecOps, control de versiones, automatización, ciberseguridad, desarrollo seguro de software

ABSTRACT

Today, software development is rarely carried out with a single tool. Instead, a single project may involve a range of platforms and resources, such as a task board, a version-controlled repository, a continuous integration engine, a dependency scanner, and finally a cloud deployment environment, all before the end user sees a single line of working code. This literature review examines the management and integration of tools for secure software development, with an emphasis on incorporating security practices from the earliest stages of the development life cycle rather than treating them as an activity confined to the phases preceding production release. To this end, twenty academic and specialized sources published between 2021 and 2025 were examined, covering topics related to DevSecOps, version control, automation, security testing, and cybersecurity fundamentals. The findings of the review show that most of the studies analyzed agree that incorporating automated security mechanisms into continuous integration and continuous delivery (CI/CD) pipelines enables earlier detection of vulnerabilities and reduces reliance on reviews conducted at advanced stages of the development cycle. Furthermore, tools such as SonarQube, CodeQL, Snyk, Trivy, and OWASP ZAP were found to appear recurrently; these are used to identify vulnerabilities, analyze code, and assess application security without significantly disrupting the continuity of development and deployment processes. As a contribution of this review, an analysis matrix is proposed that maps the tools studied to their technical function and the cybersecurity domain they address. This matrix is intended to serve as a reference resource for students, developers, and teams interested in selecting and integrating security technologies according to the specific needs of their software development processes.

Keywords: DevSecOps, version control, automation, cybersecurity, secure software development

INTRODUCCIÓN

El término de "seguridad del software" se creía que se hablaba sobre una auditoría que se realizaba cuando se concluía algún proyecto, esto se pensaba hace años atrás, cuando el tiempo se consumía y resultaba costoso emplear tiempo en la corrección de errores de diseño. Esta idea está quedando atrás. Actualmente la forma de trabajar de los equipos de desarrollo actuales, son en periodos más cortos de tiempo, además de integraciones de software, despliegue continuo y automatización de pruebas, lo que hace que queda poco tiempo para revisión de seguridad, misma que se realiza al finalizar el proyecto (Rajapakse et al., 2021). Debido a esto inicia DevSecOps, como una forma de reflexionar el proceso: se pretende no dejar la ciberseguridad a lo último, sino incluirla como una capa adicional, la propuesta es que ciberseguridad vaya adherido entre todos los procesos del desarrollo de software, hasta culminarlo (Chittibala, 2023; Prates y Pereira, 2024).

Es casi normal observar que el desarrollo de software tanto en la industria como en lo académico se llega a emplear un sin número de herramientas para planear, codificar, probar, integrar, desplegar y monitorear sin seguir algunas medidas de seguridad que las entrelace. Como resultado a esto se deja a exposición una derivación de vulnerabilidades, credenciales filtradas en repositorios públicos, dependencias desactualizadas, configuraciones incorrectas en la nube y accesos que no son auditables. Bautista Ramos y Yoo (2025) escriben en su libro que la fisura entre el manejo de DevOps y de emplearlo de forma segura, sigue siendo amplia en muchas empresas, a pesar de que las herramientas ya están disponibles.

Existen algunos organismos que proponen prácticas donde pretenden aplicarse sin que les interese la metodología de desarrollo que se emplee, por mencionar algunos organismos como: el NIST, con su Secure Software Development Framework (SSDF), y OWASP, con el modelo SAMM (Software Assurance Maturity Model). Realmente proponen una serie de funciones: como proteger el código, verificar la integridad de lo que se construye, gestionar vulnerabilidades, entre otras más. Su propuesta no se refiere a imponer alguna herramienta. El cuestionamiento que impulsa este artículo se refiere a cómo llevar a cabo en la práctica diaria: en qué etapa del ciclo de vida entran, qué aspectos de seguridad atienden y qué herramientas están empleando.

No es posible dejar puertas abiertas al ignorar la seguridad, no se deben minimizar estos descuidos y sobre todo para los estudiantes de Sistemas Computacionales, lo cual se considera importante, ya que este artículo cuenta con aspectos que van más allá de lo técnico.

Iniciarse en la programación ya no es suficiente, si al mismo tiempo, no se aprende a gestionar el entorno de herramientas que auxilian a la programación como: gestores de secretos, escáneres, pipelines, repositorios, contenedores, etc.

Por lo anterior escrito, cabe hacer mención que no se pretende evaluar productos comerciales, ni tampoco hacer recomendaciones de herramientas, lo que si se desea mostrar es un panorama ordenado, teniendo como base literatura científica actual, sobre la pregunta ¿cómo se está gestionando esta intersección entre desarrollo de software y ciberseguridad básica?

De acuerdo con la revisión bibliográfica, tenemos como objetivo general, analizar las herramientas y prácticas que se llevan a cabo para la gestión del desarrollo seguro de software, teniendo como consideración aspectos de ciberseguridad durante las fases, desde la planeación, codificación, control de versiones, pruebas, integración, despliegue y mantenimiento. De forma más explícita, se requiere identificar las principales herramientas que se utilizan en cada etapa, que en cada una de ellas se mencione y se realice los aspectos de ciberseguridad, analizar el enfoque DevSecOps como modelo de integración y sugerir una matriz de clasificación que se pueda ocupar como referencia para quienes deben seleccionar entre distintas opciones tecnológicas

METODOLOGÍA

Este artículo se desarrolló mediante una revisión bibliográfica narrativa, incorporando algunos elementos propios de la revisión sistemática con el propósito de dar mayor orden y claridad al proceso de búsqueda y selección de las fuentes. No se considera una revisión sistemática exhaustiva en sentido estricto, ya que no se siguió un protocolo PRISMA completo; sin embargo, sí se establecieron y documentaron criterios para determinar qué fuentes serían incluidas y cuáles quedarían fuera del análisis.

La búsqueda documental se realizó en bases de datos académicas reconocidas en las áreas de ingeniería de software y ciberseguridad, entre ellas IEEE Xplore, ACM Digital Library, ScienceDirect, SpringerLink, MDPI y Scopus. Estas fuentes se complementaron con búsquedas en Google Scholar y con consultas directas a la documentación oficial de NIST y OWASP. El periodo de búsqueda comprendió publicaciones entre 2020 y 2026. Se dio prioridad a los estudios publicados durante los últimos tres años, aunque también se conservaron algunas referencias anteriores cuando aportaban conceptos o fundamentos que continúan siendo relevantes para el tema.

Las palabras clave utilizadas, tanto en español como en inglés, incluyeron: desarrollo seguro de software / secure software development, herramientas de desarrollo / software development tools, ciberseguridad en software / software cybersecurity, DevSecOps, integración continua / continuous integration, despliegue continuo / continuous deployment, análisis estático de código / static application security testing, análisis de dependencias / software composition analysis, seguridad en el ciclo de vida del software / secure SDLC y gestión de vulnerabilidades / vulnerability management.

Como criterios de inclusión se consideraron publicaciones correspondientes al periodo 2020-2026 y relacionadas con las áreas de ingeniería de software, ciberseguridad, DevOps o DevSecOps. También se contemplaron artículos científicos, revisiones, normas técnicas, guías oficiales y documentación técnica reconocida, publicados en español o inglés. Como criterio principal, las fuentes debían relacionar de manera explícita las herramientas de desarrollo con algún aspecto de la seguridad del software. Se excluyeron blogs sin respaldo académico o técnico, documentos claramente desactualizados —excepto aquellos considerados referentes conceptuales—, publicaciones enfocadas únicamente en programación sin relación con la gestión o seguridad, estudios duplicados o con información redundante respecto de otras fuentes seleccionadas y documentos que no hicieran referencia a herramientas utilizadas en proyectos reales de desarrollo.

Después de aplicar estos criterios, se seleccionaron veinte fuentes que constituyen la base documental del artículo. Entre ellas se encuentran revisiones sistemáticas sobre DevSecOps (Bautista Ramos y Yoo, 2025; Dasanayake et al., 2025; Rajapakse et al., 2021), estudios sobre herramientas específicas para el análisis estático (Dalaq et al., 2025), trabajos relacionados con la aplicación de inteligencia artificial a la seguridad durante el desarrollo (Abdiukov, 2024; Fu et al., 2024; Khan et al., 2025), así como literatura enfocada en los aspectos organizacionales y educativos del tema (Akbar et al., 2024; Mittal et al., 2025; Naidoo y Möller, 2022).

El análisis de las fuentes seleccionadas se organizó en diferentes categorías: gestión del proyecto, control de versiones, desarrollo colaborativo, automatización CI/CD, seguridad del código, seguridad de dependencias, gestión de secretos, contenedores e infraestructura, monitoreo y respuesta, y cumplimiento y madurez. Estas categorías sirvieron posteriormente como base para estructurar las secciones que conforman el artículo.

DESARROLLO

Marco conceptual

Antes de abordar las herramientas y tecnologías utilizadas en este ámbito, es conveniente precisar algunos conceptos que, aunque son ampliamente utilizados en la industria del software, pueden presentar diferencias de interpretación dependiendo del autor o del contexto en el que se empleen.

El desarrollo de software puede entenderse como el conjunto de actividades técnicas y organizativas que permiten diseñar, construir, probar, implementar y mantener una aplicación o sistema. Estas actividades forman parte de un proceso que, aunque tradicionalmente se ha representado de manera secuencial, en la práctica suele ser iterativo y requiere una constante retroalimentación entre sus diferentes etapas. Este proceso se organiza alrededor de lo que se conoce como ciclo de vida del desarrollo de software (SDLC, por sus siglas en inglés), el cual contempla actividades como la planeación, el análisis de requerimientos, el diseño, la codificación, las pruebas, el despliegue y el mantenimiento. Cuando las prácticas y controles de seguridad se incorporan de manera explícita a lo largo de estas etapas, en lugar de considerarse únicamente al final del proceso, se habla de un Secure SDLC o ciclo de vida seguro del desarrollo de software (Shashank y Venkata, 2025).

Dentro de esta evolución de las prácticas de desarrollo surge DevOps, un enfoque que combina aspectos culturales, organizativos y técnicos con el propósito de mejorar la colaboración entre los equipos de desarrollo (Dev) y operaciones (Ops). Su finalidad no se limita a acelerar la entrega de software, sino también a facilitar una comunicación más estrecha, una mayor automatización y una retroalimentación continua durante el ciclo de vida de las aplicaciones. Para ello, DevOps se apoya en prácticas como la integración continua, la entrega o despliegue automatizado y el monitoreo de los sistemas.

A partir de esta misma filosofía surge DevSecOps, que incorpora la seguridad como una responsabilidad compartida entre los diferentes participantes del proceso de desarrollo. En lugar de considerar la seguridad como una actividad exclusiva de un área especializada que revisa el software al finalizar su construcción, DevSecOps propone integrarla desde las primeras etapas y mantenerla presente durante todo el ciclo de vida (Naidoo y Möller, 2022). En términos prácticos, esto implica incorporar controles de seguridad dentro de los procesos automatizados de desarrollo, por ejemplo, mediante análisis del código fuente, revisión de dependencias, detección de vulnerabilidades y diferentes tipos de pruebas de seguridad. De

esta manera, los problemas pueden identificarse y atenderse de manera temprana, antes de que lleguen a ambientes de producción.

La integración continua (continuous integration, CI) es un elemento fundamental de estos enfoques. Consiste en incorporar con frecuencia los cambios de los desarrolladores a un repositorio compartido, donde cada cambio puede validarse automáticamente mediante procesos de compilación y pruebas, con el fin de detectar errores o conflictos lo antes posible. Así se obtiene una retroalimentación rápida sobre el estado del software. A la vez, disminuye el riesgo de que los problemas se acumulen hasta etapas posteriores del desarrollo.

Estrechamente ligada a la integración continua está la entrega continua (continuous delivery). En ella, el software que ha superado las validaciones establecidas se mantiene en condiciones de liberarse a producción. El despliegue continuo (continuous deployment) va un paso más allá: los cambios que cumplen los criterios definidos se despliegan automáticamente en el entorno de producción. Estas prácticas suelen integrarse en lo que se conoce como un pipeline de CI/CD. En un entorno DevSecOps, sin embargo, el pipeline no se limita a ellas. También incorpora controles de seguridad que actúan como puntos de validación (security gates), capaces de detener el proceso cuando se identifica, por ejemplo, una vulnerabilidad de gravedad alta o una dependencia que representa un riesgo conocido (Fu et al., 2024).

Otro componente importante es el control de versiones, entendido como el conjunto de mecanismos que permiten registrar, comparar y administrar los cambios realizados en el código fuente a lo largo del tiempo. Su función más visible es conservar un historial de modificaciones, pero su utilidad va más allá. Facilita la colaboración entre desarrolladores, permite recuperar versiones anteriores y deja constancia de quién realizó cada cambio y cuándo. Desde la perspectiva de la seguridad, hace posible además implementar prácticas como la revisión de código entre pares, la protección de las ramas principales y la restricción de cambios no autorizados. Todo ello contribuye a fortalecer la trazabilidad y el control sobre el código que forma parte del producto de software.

En este contexto, la ciberseguridad básica no debe entenderse como un componente aislado de la ingeniería de software, sino como un conjunto de principios y controles que han de considerarse durante el desarrollo y la operación de cualquier sistema. Entre ellos se encuentran las propiedades fundamentales de la información: confidencialidad, integridad y disponibilidad.

Se encuentran también los mecanismos de autenticación, que permiten verificar la identidad de los usuarios, y los de autorización, que permiten controlar su acceso a los recursos. A estos elementos se suma la gestión de vulnerabilidades, entendida como un proceso sistemático para identificar, analizar, priorizar y corregir aquellas debilidades que puedan ser explotadas por un atacante (Ahsan et al., 2022; Khan et al., 2025).

Es importante considerar que la gestión de vulnerabilidades no debe limitarse al código desarrollado internamente, dado que los sistemas actuales suelen depender de bibliotecas, paquetes, frameworks y otros componentes desarrollados por terceros, por lo que una vulnerabilidad presente en cualquiera de estas dependencias puede convertirse también en un riesgo para la aplicación. Por esta razón, el análisis y monitoreo de las dependencias forman parte importante de una estrategia de seguridad dentro de DevSecOps, especialmente considerando que los componentes de terceros pueden constituir un vector de riesgo que no siempre recibe la atención necesaria en los proyectos de software (Dalaq et al., 2025).

En conjunto, estos conceptos permiten comprender que DevSecOps no representa únicamente la incorporación de algunas herramientas de seguridad dentro de un pipeline de desarrollo. Se trata de un cambio en la manera de concebir el proceso de construcción y operación del software, en el que desarrollo, operaciones y seguridad participan de forma coordinada y en el que la identificación y atención de riesgos se busca realizar de manera temprana, continua y automatizada.

RESULTADOS

Herramientas para la gestión del desarrollo de software

La gestión de la seguridad durante el desarrollo de software requiere la integración de diferentes herramientas capaces de apoyar las actividades de planificación, desarrollo control de cambios, automatización, análisis de vulnerabilidades y monitoreo. Desde la perspectiva de DevSecOps, estas herramientas no deben considerarse elementos independientes, sino componentes de un ecosistema tecnológico orientado a incorporar controles de seguridad de manera continua a lo largo del ciclo de vida del software. En este sentido, la selección y configuración adecuada de las herramientas puede contribuir tanto la trazabilidad del proyecto como la identificación temprana de riesgos y vulnerabilidades.

Durante la etapa de planeación y gestión del proyecto, plataformas como Jira, Trello, Azure Boards y Git Projects facilitan la organización de actividades, asignación de

responsabilidades, seguimiento de incidencias y documentación de decisiones. Si bien estas herramientas no constituyen mecanismos de seguridad en sentido estricto, su utilización favorece la trazabilidad de las actividades y permite asignar responsables a las tareas relacionadas con la gestión de riesgos. Esta característica resulta particularmente relevante en organizaciones que buscan incorporar la seguridad no únicamente como una actividad posterior al desarrollo, más bien como una responsabilidad transversal. En este sentido, Akbar et al. (2024) señalan que la consolidación de las prácticas de seguridad requiere condiciones organizacionales que permitan mantenerlas de manera sistemática y sostenible a lo largo del tiempo.

Es importante considerar, además, el control de versiones como otro componente fundamental para la gestión segura del desarrollo. Git y las plataformas que lo integran (entre ellas GitHub, GitLab y Bitbucket) permiten conservar un historial detallado de modificaciones, identificar quién realizó cada cambio y recuperar versiones anteriores del código. De forma adicional, estas plataformas proporcionan mecanismos de protección de ramas, revisión de código y aprobación de cambios mediante Pull Requests o Merge Requests, según la plataforma. Dichos mecanismos establecen puntos de control previos a la incorporación de modificaciones en las ramas principales y, por tanto, reducen la posibilidad de que errores de programación o cambios no autorizados se integren directamente al producto. De acuerdo con Prates y Pereira (2024), las prácticas de revisión colaborativa constituyen mecanismos relevantes para fortalecer la calidad y la seguridad del código dentro de los procesos de desarrollo.

En relación con los entornos de desarrollo colaborativo, herramientas como Visual Studio Code, GitHub Codespaces y Gitpod ofrecen alternativas para disponer de espacios de trabajo estandarizados y en determinados escenarios, aislados del entorno local del desarrollador. La utilización de estos ambientes puede facilitar la reproducción de configuraciones y reducir diferencias entre los entornos de los integrantes de un equipo. Sin embargo, el empleo de entornos administrados o basados en la nube no elimina por sí mismo los riesgos de seguridad. La protección de credenciales, tokens de acceso, extensiones y configuraciones del entorno continúa siendo necesaria para evitar que un componente comprometido pueda afectar otros recursos del proyecto.

Uno de los elementos centrales de la automatización corresponde a los procesos de integración continua y entrega o despliegue continuo (CI/CD). Herramientas como Jenkins,

GitHub Actions, GitLab CI/CD y Azure Pipelines permiten automatizar actividades que anteriormente dependían de intervenciones manuales, entre ellas la compilación, ejecución de pruebas, generación de artefactos y despliegue de aplicaciones. La incorporación de controles de seguridad dentro de estos procesos constituye uno de los principios fundamentales de DevSecOps, debido a que permite trasladar actividades de verificación hacia etapas tempranas y repetirlas de manera sistemática ante cada modificación relevante del software. De esta manera, un pipeline de CI/CD puede incorporar validaciones de seguridad y establecer criterios que impidan avanzar hacia etapas posteriores cuando se detectan vulnerabilidades o incumplimientos previamente definidos (Fu et al., 2024; Sinan et al., 2025).

Dentro de los mecanismos de evaluación automatizada destaca el análisis estático de seguridad de aplicaciones (SAST. Static Application Security Testing). Herramientas como SonarQube, CodeQL y Semgrep realizan un análisis estático del código fuente, esto es, lo examinan sin necesidad de ejecutar la aplicación, en busca de patrones asociados con errores de programación, prácticas inseguras y posibles vulnerabilidades. La incorporación de este tipo de análisis en los pipelines permite generar retroalimentación temprana para los desarrolladores y disminuir el costo que implica corregir defectos descubiertos en fases avanzadas del desarrollo. Si bien estos beneficios son relevantes, la efectividad de dichas soluciones depende de su capacidad para distinguir los problemas significativos de aquellos resultados que no representan vulnerabilidades reales. En este sentido, Dalaq et al. (2025), en su revisión sobre herramientas de pruebas estáticas de seguridad de aplicaciones (Static Application Security Testing, SAST), identifican los falsos positivos como uno de los principales desafíos para su adopción efectiva. La razón es que un volumen elevado de alertas irrelevantes puede provocar fatiga por alertas en los desarrolladores y disminuir su atención hacia los problemas de seguridad.

Otro aspecto relevante es la gestión de dependencias y componentes de terceros. Las aplicaciones modernas dependen de numerosas bibliotecas, paquetes y frameworks desarrollados externamente; por tanto, una vulnerabilidad presente en alguno de estos componentes puede afectar de manera indirecta al sistema que los incorpora. En este contexto, las herramientas de análisis de composición de software (Software Composition Analysis, SCA), como Dependabot, Snyk y OWASP Dependency-Check, permiten identificar dependencias con vulnerabilidades conocidas, determinar las versiones afectadas y

facilitar su actualización. Este tipo de análisis cobra especial importancia en proyectos que utilizan gestores de paquetes y ecosistemas con un elevado número de componentes externos.

La seguridad también debe extenderse hacia los contenedores y la infraestructura como código. En arquitecturas que utilizan Docker, herramientas como Trivy y Clair pueden emplearse para analizar imágenes de contenedores e identificar vulnerabilidades conocidas antes de su utilización en ambientes productivos. De forma complementaria, tecnologías de infraestructura como código, entre ellas Terraform y Ansible, permiten automatizar el aprovisionamiento y configuración de recursos, pero también pueden introducir riesgos cuando contienen configuraciones inseguras. Herramientas como Checkov permiten analizar estas definiciones y detectar determinadas configuraciones que podrían representar problemas de seguridad antes de que los recursos sean desplegados. Vakhula et al. (2023) destacan la importancia de incorporar mecanismos de seguridad en las distintas capas de los procesos automatizados, particularmente cuando las organizaciones utilizan infraestructura dinámica y servicios basados en la nube.

Las pruebas dinámicas de seguridad complementan los mecanismos de análisis estáticos al evaluar el comportamiento de una aplicación durante su ejecución. En el ámbito de las aplicaciones web, OWASP ZAP y Burp Suite Community pueden utilizarse para identificar determinadas vulnerabilidades mediante la interacción controlada con la aplicación desplegada. Estas herramientas permiten evaluar aspectos relacionados con entradas de usuario, mecanismos de autenticación, gestión de sesiones y configuraciones de seguridad. Su incorporación al proceso de desarrollo permite complementar los resultados obtenidos mediante SAST y SCA, debido a que cada técnica analiza diferentes características del sistema y, por tanto, puede identificar categorías de problemas que no necesariamente son detectadas por los demás mecanismos.

Finalmente, al hablar de monitoreo y observabilidad, nos encontramos con actividades necesarias para mantener una visión del comportamiento del software una vez que este se encuentra en operación. Soluciones como Prometheus, Grafana y el conjunto de herramientas conocido como ELK Stack (Elasticsearch, Logstash y Kibana) permiten recopilar, visualizar y analizar las métricas y los registros (logs) generados por las aplicaciones y la infraestructura. Si bien estas tecnologías no sustituyen a las herramientas especializadas en análisis de vulnerabilidades, proporcionan información que puede utilizarse para identificar comportamientos anómalos, errores recurrentes o eventos que requieren investigación. Desde

la perspectiva de DevSecOps, esta información puede retroalimentar las etapas de desarrollo y operación y, por tanto, contribuir a un ciclo de mejora continua, esto es, un ciclo en el que los hallazgos obtenidos durante la operación se consideren en futuras modificaciones del software.

En conjunto, las herramientas analizadas muestran que la seguridad del software no depende de una única solución tecnológica, sino de la articulación de diferentes mecanismos a lo largo del ciclo de vida del desarrollo. Cada uno de ellos aporta algo distinto: la planeación, trazabilidad y asignación de responsabilidades; el control de versiones, la revisión y protección del código; los procesos de CI/CD, la automatización de controles; el análisis SAST y SCA, la identificación temprana de vulnerabilidades; las pruebas dinámicas, una evaluación complementaria del comportamiento de las aplicaciones; y las herramientas de monitoreo, información sobre el sistema durante la operación. Por tanto, la adopción de un enfoque DevSecOps implica no solamente incorporar herramientas de seguridad, sino establecer una integración coherente entre ellas. Esa integración debe ir acompañada de procedimientos, políticas y responsabilidades que permitan transformar los resultados técnicos en acciones concretas de prevención y mejora continua.

Ciberseguridad en el ciclo de vida del software

El tema de "ciberseguridad básica" en el desarrollo de software, no quiere decir que significa un catálogo de un sin número de controles avanzados. La mayoría de las ocasiones se trata solo de algunas propiedades que se deben de agregar en cada fase del ciclo de vida, y que la bibliografía revisada intenta de forma recurrente empleando distintos nombres: autenticación, autorización, integridad, confidencialidad, disponibilidad y gestión de vulnerabilidades (Patel, 2025).

La seguridad inicia por tomar decisiones ajenas al código (en la fase de planeación y diseño): en la cual se tiene acceso al repositorio, qué información va a utilizar el sistema y qué nivel de sensibilidad tienen. Pasar por alto esto en cada etapa, es como contar con parches demasiado caros. Freitas et al. (2025) documentan cómo el cumplimiento de regulaciones como GDPR, HIPAA o la LGPD brasileña obliga a que ciertas decisiones de privacidad y protección de datos se realicen desde el diseño, sin pensar en los cambios después.

Mientras se programa, los aspectos de seguridad básica consisten en evitar quehaceres llamados como inseguros: manejar errores sin exponer información sensible en los mensajes, validar entradas de usuarios, no construir consultas SQL concatenando texto directamente y,

sobre todo el cuidar especialmente el uso de la IA generativa en el momento de programar. Kiashemshaki et al. (2025) menciona que los modelos de lenguaje pueden sugerir código funcionalmente bien hecho, pero con fallos de seguridad, debido a que fueron hechos y practicados con muchas entradas de información que no siguen las buenas prácticas.

Hay una capa poco mencionada, el control de versiones: proteger ramas principales, revisar código entre pares antes de unir los módulos y evitar que credenciales o claves API terminen escritas directamente en el código y suban al repositorio.

Con respecto a esto, hay un tema el cual consiste en uno de los descuidos más comunes, pero a la vez más sencilla de prevenir con herramientas como Vault, GitHub Secrets o AWS Secrets Manager que permiten inyectar credenciales en tiempo de ejecución sin que queden almacenadas en texto plano en ningún archivo del proyecto.

De acuerdo con este último tema referente a la protección de secretos, donde se considera uno de los descuidos más comunes, pero al mismo tiempo, más sencillos de prevenir utilizando herramientas como AWS Secrets Manager o GitHub Secrets y Vault, las cuales hacen lo posible de inyectar credenciales en tiempo de ejecución sin que queden almacenadas en texto plano en ningún archivo del proyecto.

En la etapa de pruebas, la ciberseguridad básica trata de adelantarse comprobando que las funciones requeridas solicitadas funcionen: se necesita probar además qué pasa cuando alguien intenta usar el sistema de forma maliciosa. Para eso son las pruebas de seguridad estáticas y dinámicas, además que es uno de los puntos donde la IA, toma sus funciones más frecuentemente, ayudando a ordenar por prioridad qué vulnerabilidades es necesario atender primero de acuerdo con la gravedad del problema (Abdiukov, 2024).

En la fase de integración y despliegue, quien lleva los papeles principales son la disponibilidad y la integridad: si configuramos mal un pipeline puede desplegar una versión con vulnerabilidad crítica sin que nadie se dé cuenta, o puede ser que un atacante logre penetrar código malicioso en el proceso de construcción propio.

Por último, en la fase del mantenimiento y monitoreo, se soporta la seguridad mediante registro de eventos, alertas y, sin lugar a duda, con el buen trabajo de mantener copias de seguridad actualizadas de código, configuraciones y bases de datos, acciones que se escuchan básicas pero que continúa siendo, de acuerdo con Ahsan et al. (2022), una de las actividades más efectivas frente a incidentes de ransomware.

DevSecOps como enfoque de integración

La tesis central de la literatura revisada puede resumirse en una idea que constituye, en esencia, el núcleo de DevSecOps: la seguridad funciona mejor cuando deja de ser un evento aislado y se convierte en un proceso continuo.

El modelo tradicional de seguridad «al final», en el que un equipo distinto del de desarrollo audita el software poco antes de su liberación, tiene un problema estructural. Para cuando se detecta un fallo, este ya está incrustado en decisiones de arquitectura que difícilmente pueden revertirse sin retrasar el proyecto. Frente a ello, DevSecOps propone desplazar los controles de seguridad hacia las primeras fases del ciclo de desarrollo, lo que en la literatura se conoce como shift left. También propone integrarlos en el mismo flujo de trabajo que ya emplean los desarrolladores, en lugar de crear uno paralelo (Chittibala, 2023; Naidoo y Möller, 2022).

En la práctica, cada cambio de código que ingresa a un pipeline de CI/CD se somete no solo a pruebas funcionales, sino también a un conjunto de verificaciones de seguridad automatizadas. Estas incluyen el análisis estático (SAST) del código nuevo, el análisis de dependencias (SCA), el escaneo de la imagen de contenedor cuando corresponde y la detección de secretos expuestos.

Si alguna de estas verificaciones falla de forma crítica, el pipeline se detiene antes de llegar a producción. De forma adicional, Fu et al. (2024) describen cómo la inteligencia artificial empieza a incorporarse a este esquema, no solo para detectar vulnerabilidades, sino para priorizarlas e incluso sugerir correcciones automáticas, si bien advierten que esto todavía está lejos de reemplazar el criterio humano en la toma de decisiones complejas.

Sin embargo, no todo en DevSecOps es automatización. Varios autores insisten en que se trata, en el fondo, de un cambio más cultural que tecnológico. Naidoo y Möller (2022) lo abordan desde una perspectiva sociotécnica y argumentan que la resistencia a adoptarlo rara vez proviene de la falta de herramientas, que hoy son abundantes y en muchos casos gratuitas o de código abierto, sino de estructuras organizacionales en las que desarrollo, operaciones y seguridad continúan funcionando como equipos separados, con incentivos distintos, cuando no directamente en conflicto. En la misma línea, Akbar et al. (2024) proponen pensar en una "madurez de las personas" para DevSecOps, esto es, no basta con instalar las herramientas correctas si el equipo no desarrolla también las competencias y la cultura de responsabilidad compartida que el modelo exige.

Por su parte, Rajapakse et al. (2021), en una de las revisiones sistemáticas más citadas sobre el tema, identifican retos recurrentes en la adopción de DevSecOps: la dificultad para automatizar prácticas que tradicionalmente eran manuales, como las pruebas de penetración; la resistencia cultural ya mencionada; la falta de formación específica en seguridad entre los desarrolladores, y el riesgo de que la automatización genere una falsa sensación de seguridad cuando las herramientas no se configuran ni se actualizan de manera adecuada. Nagasundari et al. (2025) complementan este panorama con una revisión centrada en los modelos de amenazas dentro de DevSecOps y señalan que buena parte de los incidentes documentados no se debe a la ausencia de herramientas, sino a configuraciones por defecto que nadie revisó.

Otro aspecto que comienza a ganar espacio en la literatura es la formación. Mittal et al. (2025) documentan una experiencia de integración curricular de DevSecOps en programas universitarios, en la que se emplean pipelines reales como parte de la enseñanza, y argumentan que esperar a que los estudiantes aprendan estas prácticas hasta su primer empleo resulta demasiado tarde. Esto se vincula directamente con la justificación de esta revisión, dado que, si DevSecOps se está convirtiendo en un estándar de facto en la industria, es pertinente que la formación en Sistemas Computacionales lo considere parte del currículo básico, y no un tema optativo o avanzado.

Es importante matizar que DevSecOps no es una solución mágica ni un conjunto fijo de herramientas. Dasanayake et al. (2025), en su revisión sobre software de trading, muestran que su implementación varía considerablemente según el dominio de la aplicación, el nivel de regulación al que está sujeta y la madurez tecnológica del equipo. Lo que sí parece constante entre los estudios revisados es la dirección del cambio: hacia una seguridad continua, automatizada y compartida por todo el equipo, más que hacia una seguridad centralizada y tardía.

Matriz comparativa de herramientas y prácticas seguras

A partir de las categorías identificadas durante la revisión, se construyó la matriz de la Tabla 1, en la que se vincula cada etapa del ciclo de vida del software con las herramientas más mencionadas en la literatura y con el aspecto de ciberseguridad que cada una atiende principalmente. Si bien existen decenas de herramientas comerciales y de código abierto que cumplen funciones similares, la matriz no pretende ser un catálogo exhaustivo, sino una síntesis orientada a quienes necesitan un punto de partida para la selección de tecnologías, ya sea en un proyecto académico o en un proyecto real de menor escala.

Tabla 1. Herramientas por etapa del ciclo de vida y control de seguridad asociado

ETAPA	***HERRAMIENTAS***	***ASPECTO DE CIBERSEGURIDAD ASOCIADO***
Planeación	Jira, Trello, Azure Boards, GitHub Projects	Gestión de riesgos, responsables y trazabilidad de incidencias.
Control de versiones	Git, GitHub, GitLab, Bitbucket	Control de accesos, revisión de código y protección de ramas.
Desarrollo	Visual Studio Code, GitHub Codespaces, Gitpod	Extensiones seguras, control de credenciales, ambientes aislados.
Revisión de código	GitHub Pull Requests, GitLab Merge Requests	Revisión colaborativa y detección temprana de errores.
CI/CD	Jenkins, GitHub Actions, GitLab CI/CD, Azure Pipelines	Automatización de pruebas, despliegues controlados y validaciones.
Análisis estático	SonarQube, CodeQL, Semgrep	Detección de vulnerabilidades en el código fuente (SAST).
Dependencias	Dependabot, Snyk, OWASP Dependency-Check	Detección de librerías vulnerables y paquetes inseguros (SCA).
Contenedores	Docker, Trivy, Clair	Escaneo de imágenes y reducción de configuraciones inseguras.
Infraestructura como código	Terraform, Ansible, Checkov	Revisión de configuraciones antes del despliegue.
Monitoreo	Prometheus, Grafana, ELK Stack	Registro de eventos, alertas y observabilidad.
Pruebas de seguridad	OWASP ZAP, Burp Suite Community	Detección de vulnerabilidades en aplicaciones web (DAST).
Gestión de secretos	Vault, GitHub Secrets, AWS Secrets Manager	Protección de tokens, claves API y credenciales.

Fuente: Elaboración propia a partir de la revisión bibliográfica.

Un patrón que salta a la vista al construir esta matriz es que la mayoría de las etapas cuentan tanto con una opción de código abierto como con una alternativa comercial equivalente, lo que reduce la barrera de entrada para proyectos académicos con presupuesto limitado. También se observa que los controles de seguridad más automatizables —análisis estático, escaneo de dependencias, escaneo de imágenes— se concentran en las etapas intermedias del ciclo de vida (codificación, integración y despliegue), mientras que la planeación y el mantenimiento dependen todavía en buena medida de la disciplina organizacional más que de una herramienta específica.

DISCUSIÓN

A continuación, se examinan algunos hallazgos de patrones, que fueron encontrados en las secciones anteriores:

En primer lugar, es pertinente señalar que existe consenso sobre la necesidad de automatizar la seguridad incluida en el pipeline de CI/CD, aunque no persiste el consenso sobre cómo realizarlo, donde herramientas como SonarQube, CodeQL o Snyk aparecen de forma continua como soluciones efectivas, debiendo tomar en cuenta que gran parte de su efectividad, es relativa al nivel de configuración de forma personalizada y la atención por parte del equipo a las alertas que se generan. Mientras Dalaq et al. (2025), señalan el problema referente a los falsos positivos, donde se generan demasiadas alertas no importantes. Esto cobra importancia porque el grupo puede optar por ignorarlas generando que no se cumpla el propósito mismo de la herramienta. Por tanto, la selección de una herramienta no debería basarse solo en su capacidad técnica de detección, sino también en qué tan bien se integra al flujo de trabajo real de cada equipo.

El segundo patrón tiene que ver con el costo, entendido en un sentido amplio que va más allá del precio de la licencia. Muchas de las herramientas revisadas, como Git, Docker, OWASP ZAP, Trivy y Semgrep, son gratuitas y de código abierto, lo que reduce la barrera económica de entrada. Eso no elimina, sin embargo, otros costos: el tiempo de configuración, la curva de aprendizaje, el mantenimiento continuo y, sobre todo, la formación del equipo. Akbar et al. (2024) y Mittal et al. (2025) apuntan en esa misma dirección: si no hay personas capacitadas para interpretar los resultados de las herramientas y actuar en consecuencia, la inversión tecnológica rinde muy poco.

El tercer punto aparece en varias de las fuentes revisadas, aunque quizá recibe menos atención en la literatura técnica. Se trata de la tensión entre velocidad y seguridad, que DevSecOps promete resolver sin lograrlo siempre del todo. Rajapakse et al. (2021) documentan que, en la práctica, algunos equipos desactivan o relajan los controles de seguridad cuando los perciben como un obstáculo para cumplir los plazos de entrega. Surge entonces una pregunta incómoda: ¿hasta qué punto puede la automatización resolver el problema por sí sola, si la decisión de ignorar una alerta sigue siendo humana? Nagasundari et al. (2025) ofrecen una respuesta parcial. Según estos autores, buena parte de los incidentes documentados no proviene de vulnerabilidades sofisticadas, sino de configuraciones predeterminadas o de controles existentes que nadie ajustó correctamente. Una vez más, esto apunta a un problema de gestión más que de disponibilidad tecnológica.

La irrupción de la inteligencia artificial merece una mención aparte. Fu et al. (2024), Abdiukov (2024) y Khan et al. (2025) coinciden en que la IA empieza a desempeñar un papel

relevante en la seguridad del ciclo de desarrollo. Ese papel abarca la generación automatizada de pruebas de seguridad, la priorización de vulnerabilidades según su severidad real e incluso la propuesta de correcciones de código. Kiashemshaki et al. (2025) advierten, en cambio, del riesgo opuesto: la IA generativa empleada para escribir código sin supervisión suficiente puede introducir precisamente las vulnerabilidades que las herramientas de seguridad tendrán que detectar después. En el desarrollo seguro, por tanto, la IA parece funcionar mejor como un componente más del pipeline, sujeto a los mismos controles que cualquier otro, que como sustituto de la revisión humana o de las prácticas ya consolidadas.

Por último, es importante reconocer las limitaciones de esta revisión. Al tratarse de una revisión narrativa, y no de una revisión sistemática exhaustiva con protocolo formal, la selección de fuentes conlleva cierto grado de subjetividad, que se procuró mitigar mediante criterios explícitos de inclusión y exclusión. A ello se suma la rápida evolución del campo de DevSecOps. Varias de las herramientas mencionadas han publicado actualizaciones importantes tan solo en los últimos dos años, de modo que algunos hallazgos podrían quedar desactualizados en poco tiempo. Esto refuerza la necesidad de realizar revisiones periódicas, más que un estudio único y definitivo.

CONCLUSIÓN

Como resultado de la presente revisión, se sugiere que no pueden interpretarse como asuntos independientes la ciberseguridad y las herramientas de desarrollo, dado que están muy relacionadas, y la seguridad debe integrarse en cada parte de los flujos de desarrollo, desde el control de versiones, así como la integración continua y el despliegue continuo, evitando el error común de integrarla por separado como elemento al final del proyecto.

Como una valoración de los objetivos iniciales, se ha cubierto la identificación de las herramientas empleadas en cada etapa del ciclo de vida del desarrollo de software, donde se inicia con la planeación a partir del uso de Jira o Trello y herramientas de seguridad como SonarQube, Snyk, Trivy u OWASP ZAP. Asimismo, se han cubierto aspectos de ciberseguridad que es deseable incluir en cada fase, como los principios mismos: confidencialidad, integridad, disponibilidad, autenticación, autorización y la gestión adecuada de vulnerabilidades.

Por último, se analizó DevSecOps, que, de acuerdo con la mayoría de las fuentes consultadas, es el enfoque que mejor articula estos elementos en un proceso ágil y automatizado.

La matriz comparativa que propone este artículo pretende ser, ante todo, un punto de partida práctico. Funciona como un mapa que relaciona etapa, herramienta y control de seguridad, y está pensada para estudiantes de Sistemas Computacionales que, al organizar un proyecto, deben decidir con criterio qué tecnologías incorporar desde el diseño, y no como un parche posterior.

Con todo, la revisión también deja claro que ninguna herramienta resuelve por sí sola la seguridad en el desarrollo de software. Si bien las fuentes analizadas coinciden en que la tecnología es una condición necesaria, también coinciden en que no es suficiente, por lo que se requieren además tres elementos: una formación técnica específica, una cultura organizacional que asuma la seguridad como una responsabilidad compartida y procesos de gestión que garanticen la atención de las alertas que generan estas herramientas, en lugar de que estas se conviertan en ruido que el equipo aprende a ignorar.

Como línea de trabajo futura, convendría complementar esta revisión narrativa con un estudio de caso aplicado, por ejemplo, la implementación de un pipeline DevSecOps básico en un proyecto académico real. Así podría contrastarse lo que reporta la literatura con lo que ocurre al integrar estas herramientas en un contexto universitario con recursos limitados. También sería pertinente dar seguimiento al creciente papel de la inteligencia artificial en la automatización de la seguridad. A juzgar por el volumen de publicaciones recientes, es un área que apenas empieza a consolidarse.

REFERENCIAS

Abdiukov, T. (2024). Automated security testing in DevSecOps pipelines: Integrating AI-based vulnerability discovery and compliance validation. World Journal of Advanced Research and Reviews. https://doi.org/10.30574/wjarr.2024.22.1.1083

Ahsan, M., Nygard, K. E., Gomes, R., Chowdhury, M., Rifat, N., & Connolly, J. F. (2022). Cybersecurity Threats and Their Mitigation Approaches Using Machine Learning - A Review. Journal of Cybersecurity and Privacy, 2, 527-555. https://doi.org/10.3390/jcp2030027

Akbar, M., Rafi, S., Hyrynsalmi, S., & Khan, A. (2024). Towards People Maturity for Secure Development and Operations: A vision. Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering. https://doi.org/10.1145/3661167.3661238

Bautista Ramos, R. C., & Yoo, S. G. (2025). Cybersecurity in DevOps Environments: A Systematic Literature Review. IEEE Access, 13, 191959-191979. https://doi.org/10.1109/access.2025.3582892

Chittibala, D. R. (2023). DevSecOps: Integrating Security into the DevOps Pipeline. International Journal of Science and Research (IJSR). https://doi.org/10.21275/sr24304171058

Dalaq, D., Daya, K. F., Dalaq, A., Arefin, M., & Niazi, M. (2025). A Systematic Literature Review on Static Application Security Testing (SAST) Tools: Evaluation, Benchmarks, Challenges, and Future Directions. Proceedings of the 2025 29th International Conference on Evaluation and Assessment in Software Engineering Companion. https://doi.org/10.1145/3727967.3756838

Dasanayake, S., Senanayake, J., & Wijayanayake, W. (2025). Devsecops for Continuous Security in Trading Software Application Development: a Systematic Literature Review. Journal of Desk Research Review and Analysis. https://doi.org/10.4038/jdrra.v2i2.52

Freitas, D. D., Oliveira, A. A. D., Moreno, E. D., & Silva, G. J. F. D. (2025). DevSecOps Practices for GDPR, HIPAA or LGPD Compliance in Software Development: A Systematic Review. 145-153. https://doi.org/10.5753/sbsi.2025.246400

Fu, M., Pasuksmit, J., & Tantithamthavorn, C. (2024). AI for DevSecOps: A Landscape and Future Opportunities. ACM Transactions on Software Engineering and Methodology, 34, 1-61. https://doi.org/10.1145/3712190

Khan, H. U., Khan, R. A., Alwageed, H., Almagrabi, A. O., Ayouni, S., & Maddeh, M. (2025). AI-driven cybersecurity framework for software development based on the ANN-ISM paradigm. Scientific Reports, 15. https://doi.org/10.1038/s41598-025-97204-y

Kiashemshaki, K., Torkamani, M. J., & Mahmoudi, N. (2025). Secure coding for web applications: Frameworks, challenges, and the role of LLMs. ArXiv, abs/2507.22223. https://doi.org/10.48550/arxiv.2507.22223

Mittal, A., Sudarsan, S., & Sekar, A. (2025). Curricular Integration of DevSecOps: A Real-World Pipeline Approach to Secure Software Engineering Education. Proceedings of

the 26th ACM Annual Conference on Cybersecurity & Information Technology Education. https://doi.org/10.1145/3769694.3771132

Nagasundari, S., Manja, P., Mathur, P., & Honnavalli, P. B. (2025). Extensive Review of Threat Models for DevSecOps. IEEE Access, 13, 45252-45271. https://doi.org/10.1109/access.2025.3547932

Naidoo, R., & Möller, N. (2022). Building Software Applications Securely with DevSecOps: A Socio-Technical Perspective. European Conference on Cyber Warfare and Security. https://doi.org/10.34190/eccws.21.1.295

Patel, J. S. (2025). The growing importance of cybersecurity in society: A quality engineering perspective. World Journal of Advanced Research and Reviews. https://doi.org/10.30574/wjarr.2025.26.1.1457

Prates, L., & Pereira, R. (2024). DevSecOps practices and tools. International Journal of Information Security, 24. https://doi.org/10.1007/s10207-024-00914-z

Rajapakse, R., Zahedi, M., Babar, M., & Shen, H. (2021). Challenges and solutions when adopting DevSecOps: A systematic review. Information and Software Technology. https://doi.org/10.1016/j.infsof.2021.106700

Shashank, S., & Venkata, G. (2025). Secure software development: Integrating encryption protocols from design to deployment. International Journal of Applied Mathematics. https://doi.org/10.12732/ijam.v38i2s.714

Sinan, M., Shahin, M., & Gondal, I. (2025). Integrating Security Controls in DevSecOps: Challenges, Solutions, and Future Research Directions. Journal of Software: Evolution and Process, 37. https://doi.org/10.1002/smr.70029

Vakhula, O., Opirskyy, I., & Mykhaylova, O. (2023). Research on Security Challenges in Cloud Environments and Solutions based on the "Security-as-Code" Approach. 55-69. https://doi.org/10.55056/ceur-ws.org/vol-3550/paper5.pdf

© Los autores. Este artículo se publica en Prisma ODS bajo la Licencia Creative Commons Atribución 4.0 Internacional (CC BY 4.0). Esto permite el uso, distribución y reproducción en cualquier medio, incluidos fines comerciales, siempre que se otorgue la atribución adecuada a los autores y a la fuente original.



: https://doi.org/10.65011/prismaods.v5.i5.405

Cómo citar este artículo (APA 7ª edición):

Cuevas Bracamontes, L., Castillo Ortiz , S. Y., Valle Bahena, S., Mena Salgado, E., & Urióstegui Peralta, M. del C. (2026). DevSecOps y herramientas para el desarrollo seguro de software: una revisión bibliográfica. Prisma ODS: Revista Multidisciplinaria Sobre Desarrollo Sostenible, *5*(5), 1091-1113. https://doi.org/10.65011/prismaods.v5.i5.405