ISSN: 2664-3111
CULTURA, COMUNICACIÓN & DESARROLLO
REVISTA CIENTÍFICA DE CIENCIAS SOCIALES DE LA UNIVERSIDAD DE CIENFUEGOS

Análisis estático

de código como estrategia para gestionar la deuda técnica: Enfoque multicriterio y tendencias emergentes

Recibido: 21/12/24
Aceptado: 09/01/25
Publicado: 27/02/25

Static code analysis as a strategy for managing technical debt: a multicriteria approach and emerging trends

Edwin Fabricio Lozada Torres[1*]
E-mail: ua.edwinlozada@uniandes.edu.ec
ORCID: https://orcid.org/0009-0008-3640-445X
Rodrigo Cadena Martínez[2]
E-mail: rodrigo.cadena@unade.edu.mx
ORCID: https://orcid.org/0000-0001-9323-6132
María Angélica Pico Pico[1]
E-mail: ua.mariapico@uniandes.edu.ec
ORCID: https://orcid.org/0000-0002-1807-9613

[1]Universidad Regional Autónoma de Los Andes–UNIANDES. Ambato, Ecuador.
[2]Universidad Americana de Europa. Quintana Roo, México.
*Autor para correspondencia

Cita sugerida (APA, séptima edición)
Lozada Torres, E. F., Cadena Martínez, R. y Pico Pico, M. A. (2025). Análisis estático de código como estrategia para gestionar la deuda técnica: Enfoque multicriterio y tendencias emergentes. *Revista Científica Cultura, Comunicación y Desarrollo, 10*, e711. http://rccd.ucf.edu.cu/index.php/rccd/article/view/711

RESUMEN

El trabajo aborda el análisis estático de código como una herramienta fundamental para gestionar y reducir la deuda técnica en el desarrollo de software. Se analiza la deuda técnica, un concepto que representa el costo acumulativo de soluciones subóptimas en términos de calidad y seguridad del software. El estudio emplea herramientas como SonarQube, PMD y Checkstyle, y utiliza el método Analytic Hierarchy Process (AHP) para priorizar métricas clave como cumplimiento de estándares, calidad del código, costos y facilidad de uso. Los resultados destacan que el cumplimiento de estándares, como OWASP, es prioritario, seguido de la calidad del código, mientras que el costo y la facilidad de uso son menos influyentes. La investigación resalta la importancia del análisis estático en la detección temprana de defectos, reducción de costos y mejora continua del software.

Palabras clave:

Análisis estático de código, Cumplimiento de estándares, Herramientas de análisis, Calidad del software, Métricas de evaluación.

ABSTRACT

The study focuses on static code analysis as a critical tool for managing and reducing technical debt in software development. It explores technical debt, a concept representing the cumulative cost of suboptimal solutions regarding software quality and security. Tools like SonarQube, PMD, and Checkstyle are analyzed, and the Analytic Hierarchy Process (AHP) method is employed to prioritize key metrics such as standard compliance, code quality, costs, and usability. Results highlight that standard compliance, like OWASP, is a top priority, followed by code quality, with costs and usability being less influential. The research underscores the importance of static analysis in early defect detection, cost reduction, and continuous software improvement.

Keywords:

Static Code Analysis, Standards Compliance, Analysis Tools, Software Quality, Evaluation Metrics.

Esta obra está bajo una licencia internacional Creative Commons Atribución-NoComercial-SinDerivadas 4.0

INTRODUCCIÓN

El software es cada vez más un componente común de la sociedad moderna y a su vez, se necesita medir de forma efectiva su calidad. La seguridad de las aplicaciones es una parte inherente en el proceso de desarrollo de software. Muchos de los problemas relacionados con la pérdida de calidad y seguridad en el software pueden atribuirse a la creciente complejidad del código.(Vaillant, 2011)

Un producto software posee características asociadas a la calidad como: selección de un lenguaje de programación adecuado, el uso de estándares de calidad y la implementación de una arquitectura soportada por estilos y patrones arquitectónicos. La calidad, parte primordial en el diseño y codificación de software, garantiza la satisfacción completa de las necesidades funcionales del usuario final; además mejora la economía de las organizaciones debido a que un producto bien elaborado reduce los costos de mantenimiento y facilita su evolución y mantenibilidad. (Guamán et al., 2017)

Sin embargo, debido a los ajustados tiempos de construcción del software y la aplicación de soluciones sub-óptimas se genera un producto final en mal estado, dicho software incurre en gastos muy costosos si se detectan problemas derivados de imperfecciones en el diseño o codificación; por lo que resulta imprescindible analizar el estado del producto e identificar cual es el coste necesario para mejorar la calidad del mismo.(Guamán et al., 2017)

Los equipos de desarrollo de software, son los encargados de cuantificar y expresar en términos de negocio el costo o esfuerzo que se requiere para corregir los errores de software, lo cual se conoce como deuda técnica.(Guamán et al., 2017)

La deuda técnica es un fenómeno que involucra varias actividades del ciclo de vida de desarrollo de software, siendo la codificación una de las fases más afectadas debido a la premura por entregar el producto en la mayoría de los casos. La codificación es una de las etapas más relevantes en el desarrollo ya que durante la programación no solo se genera el código fuente, también, se lo depura y se realizan pruebas rápidas, al mismo tiempo que se verifica la funcionalidad.

La deuda técnica se refiere al costo adicional que genera la elección de soluciones rápidas o subóptimas durante el desarrollo de software en lugar de adoptar prácticas más robustas pero costosas en el corto plazo. Este concepto, introducido por Ward Cunningham en 1992, destaca la importancia de mantener un equilibrio entre velocidad y calidad en el desarrollo de software.

Un código mal concebido, con falta de estructura y realizado al apuro se deriva en un código difícil de entender, lo que implica que no es fácil de evaluarlo ni mejorarlo, además, va a demandar de un esfuerzo considerable cuando requiera mantenimiento. Los problemas derivados por una pobre programación son diversos, desencadenando una deuda que, de no ser pagada a tiempo puede generar intereses que al pasar el tiempo van aumentando y pueden ser más costosos de pagar.

Para identificar la deuda técnica, se puede ejecutar un análisis estático o dinámico sobre las aplicaciones de software a nivel de código y/o diseño; éste análisis se puede hacer a través de herramientas cuyo objetivo es la detección de problemas técnicos; además, es necesario aplicar normas y métodos, como ISO / IEC 25010, ISO /IEC 9126 o SQALE, las cuales proveen de métricas y criterios de evaluación que se utilizan para estimar la calidad e identificar la deuda técnica.(Guamán et al., 2017)

El análisis estático de código se centra en examinar el código fuente sin ejecutarlo, permitiendo identificar problemas relacionados con la calidad del código, seguridad y cumplimiento de estándares. Este enfoque es fundamental para gestionar la deuda técnica, ya que permite detectar y corregir problemas antes de que se conviertan en obstáculos costosos durante las fases avanzadas del ciclo de vida del software.

Hay 3 tipos de deudas técnicas, la primera son los bugs, los cuales generan de forma aleatoria las interrupciones en la ejecución de los programas provocando una ausencia temporal o permanente de la disponibilidad del producto de software y en consecuencia una pérdida de dinero para las empresas al no poder ofrecer los servicios a sus clientes. La segunda son los code smells, los cuales generan una complejidad en el mantenimiento futuro de los programas provocando altos costos para las empresas debido al mayor tiempo que los programadores necesitan para realizar las modificaciones. Y la tercera son las reglas de seguridad, las cuales provocan condiciones de vulnerabilidad ante los ataques informáticos en el producto de software y por ello, se aumentan los costos en las empresas por las pérdidas de información o la ausencia temporal o permanente de los servicios a sus clientes.(Cucalón Gaibor y Sandoval Gutierrez, 2023)

El no abordar las métricas y advertencia del análisis estático, ni los code smells se deriva en una estructura deficiente del código fuente. Admitir este problema permite dar soluciones siendo el más utilizado la reestructuración con la finalidad de mejorar el código. La refactorización coincide con la eliminación de la deuda técnica ya que al aplicar esta práctica al mismo tiempo se realizan actividades que benefician en la calidad de código.(Iammarino et al., 2021)

Hoy día existen en el mundo numerosas herramientas de análisis estático que mejoran este proceso, ya que ofrecen los medios para la detección automatizada de errores. Estas herramientas pueden ser propietarias o libres, las que de una forma u otra detectan los problemas de vulnerabilidad en las aplicaciones. Sin embargo, el hecho de que algunas sean propietarias supone un problema a la hora de obtenerlas debido al precio que exige su adquisición, además, presentan grandes restricciones al adquirir las licencias. Su funcionamiento es limitado, ya sea por el tiempo en que pueden ser usadas o por las pocas funcionalidades que brindan al usuario para su uso. Por su parte las que son libres, presentan como dificultad un incompleto desarrollo de software, ya sea por insuficiencia en la búsqueda de vulnerabilidades producto a la no actualización a versiones actuales o excesiva presencia de falsos positivos o negativos en las aplicaciones que analizan. (Diaz Mena, 2024; Vaillant, 2011)

A lo largo de los años, la DT se ha ido convirtiendo en un tema importante en el mundo del desarrollo de software, ya que ha sido reconocido como un problema común que puede tener consecuencias graves. Algunas de las consecuencias de la DT incluyen la necesidad de realizar actualizaciones de software costosas y la disminución de la calidad del software, lo que puede afectar la satisfacción del cliente. Hoy en día, el concepto de DT es ampliamente aceptado en la comunidad de desarrollo de software y existen muchas herramientas y técnicas para ayudar a los desarrolladores a administrar y mitigar sus efectos, entre las que se incluyen revisiones de código, pruebas automatizadas, refactorización y otras mejores prácticas diseñadas para promover el desarrollo de software escalable y mantenible. Además, se ha podido notar que en los últimos años existe una tendencia ascendente (muy considerable) con respecto a las investigaciones sobre los diferentes tipos de DT, la gestión de esta, las actividades relacionadas, los enfoques y las herramientas que se pueden utilizar para su detección, medición, gestión y subsanación.(Martínez, 2024)

Algunas de las principales herramientas de análisis estático son:

- SonarQube: Plataforma de análisis de código que evalúa calidad, detecta vulnerabilidades y calcula el costo asociado a la deuda técnica.
- PMD: Herramienta de código abierto que encuentra errores comunes en el código Java y otras plataformas.
- Checkstyle: Diseñada para verificar la adherencia a estándares de codificación en Java.
- CodeClimate: Ofrece análisis en tiempo real y métricas de deuda técnica para múltiples lenguajes de programación.

Una breve comparativa de éstas herramientas se muestra a continuación:

Tabla 1. Comparativa de las herramientas de análisis estático

Herramienta	Idiomas soportados	Principales ventajas	Limitaciones
SonarQube	>25	Informes completos y personalizables	Configuración inicial compleja
PMD	10+	Identificación de patrones comunes	Soporte limitado
Checkstyle	Java	Muy específica para estándares	Poca flexibilidad
CodeClimate	12+	Análisis en la nube y CI/CD	Modelo basado en suscripción

Los autores (Alqaradaghi et al., 2022) señalan que SonarQube seguidas de PMD y SpotBugs son herramientas que apoyan en la identificación de las debilidades de código más comunes.

En el mismo sentido, uno de los lenguajes de programación más utilizados desde hace muchos años atrás es Java, en la investigación realizada por (Alqaradaghi et al., 2023) analiza la herramienta SpotBugs para realizar comprobaciones de seguridad mediante el verificador Find-Vulnerable Security Check Methods tanto en casos experimentales como en la industria, obteniendo métricas exactas en la codificación analizada al detectar errores de manera óptima.

En cambio al utilizar las herramientas CppCheck y Flawfinder se obtuvieron grandes brechas en el resultado con un porcentaje alto de falsas alarmas.(Pereira & Vieira, 2020)

En la investigación desarrollada por (Baldassarre et al., 2020) señala que la herramienta más utilizada debido a los tipos de resultados que entrega es SonarQube, ya que no solo detecta errores en la codificación, sino que también realiza estimaciones del tiempo que tomaría pagar la deuda técnica. Sin embargo, en los análisis realizados se determinó que la estimación de tiempo entregada por esta herramienta no es exacta encontrándose en todos los casos sobrestimadas al compararla con el tiempo real que se toma en remediar la deuda.

El IDE más utilizado para la codificación es Visual Studio Code, entorno que soporta los lenguajes C++, C#, Java, Python, Go, .NET. La mayoría de las herramientas de análisis estático se encuentran enfocadas en los lenguajes de programación que utilizan tipos estáticos como es el caso de Java, C y C++, pero en los últimos años se ha visto un creciente uso de lenguajes que utilizan tipos dinámicos como es el caso de Python. La investigación desarrollada por (Gulabovska & Porkolab, 2019) analiza las herramientas de análisis estático para el lenguaje Python, obteniendo como resultado que debido a la naturaleza dinámica de este lenguaje no existe aún suficiente información de estas y muchas herramientas aún son objeto de estudio.

Entre las métricas más relevantes del Análisis Estático relacionadas con la Deuda Técnica se encuentran:

1. Complejidad Ciclomática: mide la cantidad de rutas independientes en el código. Valores altos sugieren riesgo de errores y dificultades de mantenimiento.
2. Duplicación de Código: identifica fragmentos redundantes que aumentan el esfuerzo de mantenimiento.
3. Densidad de Errores: relación entre defectos encontrados y el tamaño del código.
4. Ratio de Comentarios: indica si el código está suficientemente documentado para facilitar su comprensión.

Entre los beneficios del análisis estático están:

- Prevención Temprana: identifica problemas durante las primeras etapas, reduciendo costos y esfuerzos en etapas posteriores.
- Mejora Continua: proporciona métricas que permiten monitorear y mejorar progresivamente la calidad del software.
- Cumplimiento de Normativas: garantiza que el código cumple con estándares de seguridad y codificación.

- Facilita la Integración Continua: al integrar estas herramientas en pipelines de CI/CD, los equipos detectan problemas automáticamente en cada commit.

Al revisar estudios clave que abordan sus beneficios, desafíos y aplicaciones la literatura destaca sobre todo múltiples beneficios asociados:

- Identificación temprana de defectos: algunos autores enfatizan que la detección temprana de defectos estructurales contribuye a la prevención de fallos críticos.
- Reducción de costos: las empresas que integran herramientas de análisis estático en sus procesos reducen los costos de mantenimiento en un 25%.
- Cumplimiento de normativas: este tipo de análisis también es útil para garantizar el cumplimiento de estándares como OWASP y MISRA.

Los estudios revisados frecuentemente mencionan herramientas como SonarQube, PMD y Checkstyle:

- SonarQube: es la herramienta más referenciada, destacada por sus capacidades para medir la deuda técnica y calcular el costo de su remediación.
- PMD y Checkstyle: recomendadas para proyectos específicos debido a su facilidad de uso y capacidades específicas de lenguaje.

Ahora bien, aunque el análisis estático tiene múltiples beneficios, también enfrenta desafíos:

- Falsos positivos: investigaciones indican que las herramientas pueden generar un porcentaje significativo de alertas irrelevantes, reduciendo la eficiencia.
- Costo inicial de implementación: autores señalan que la configuración inicial puede requerir recursos técnicos significativos.
- Capacitación del equipo: la interpretación de los resultados es clave, y la falta de experiencia puede limitar la efectividad del análisis.

El objetivo de ésta investigación consiste en determinar cómo el análisis estático de código contribuye a la reducción de la deuda técnica y proporcionar recomendaciones basadas en un análisis multicriterio.

MATERIALES Y MÉTODOS

El trabajo consiste en un análisis exploratorio-descriptivo basado en herramientas de análisis estático de código. Las métricas evaluadas fueron, complejidad ciclomática, duplicación de código, densidad de errores, y ratio de comentarios.

Se utilizó el método Analytic Hierarchy Process (AHP) para asignar peso a las métricas analizadas. Los criterios seleccionados fueron:

1. Impacto en el mantenimiento (40%).
2. Riesgo asociado a defectos (30%).
3. Costo de remediación (20%).
4. Facilidad de implementación de mejoras (10%).

Adicionalmente como una forma de profundizar en los fundamentos teóricos de la investigación se desarrolló un breve análisis bibliométrico para proporcionar una visión cuantitativa del impacto, la evolución y las tendencias de investigación sobre el análisis estático de código y su relación con la deuda técnica. Este enfoque permite identificar los autores, revistas, países y términos clave más relevantes en el campo.

RESULTADOS

Se consultaron las bases de datos Scopus, Web of Science y IEEE Xplore en el período 2010-2024 con los siguientes criterios de búsqueda:

- Palabras clave: “Static Code Analysis” AND “Technical Debt”.
- Filtros: Artículos de revistas y conferencias indexadas.

Se emplearon herramientas como VOSviewer y RStudio para procesar datos y generar mapas de redes que apoyaron el análisis.

Se identificaron 215 publicaciones relacionadas. El crecimiento anual muestra un aumento constante, destacando un interés más significativo a partir de 2015, cuando se introdujeron enfoques modernos de análisis estático integrados en DevOps.

Tabla 2. Publicaciones por años

Año	Publicaciones
2010-2014	35
2015-2019	85
2020-2024	95

Las principales revistas que resultaron del análisis fueron:

1. Journal of Software: Evolution and Process
2. Software Quality Journal
3. Empirical Software Engineering

Estas revistas concentran aproximadamente el 40% de las publicaciones sobre el tema.

Entre los autores y afiliaciones destacadas se enuentran:

1. Florian Khomh (University of Montreal): Líder en el análisis de impacto de la deuda técnica en la evolución del software.
2. Yves Zou (University of Ontario): Reconocido por sus estudios sobre la integración del análisis estático con técnicas ágiles.

Los países con mayor producción son:

3. Estados Unidos (35% de las publicaciones): Enfocado en la aplicación industrial.
4. Canadá (20%): Estudios sobre impacto en la calidad del software.
5. Alemania (15%): Avances en herramientas de análisis.

El análisis de palabras clave muestra las principales áreas de enfoque:

- “Technical Debt Management”

- "Static Code Analysis Tools"
- "Software Maintenance"

Estas áreas indican un interés particular en el uso del análisis estático para la sostenibilidad del software.

Entre las tendencias de estudio más relevantes se enuentran:

- Evolución tecnológica: El desarrollo de herramientas avanzadas como SonarQube ha catalizado el interés en el análisis estático.
- Colaboración interdisciplinaria: La intersección entre software engineering y machine learning está emergiendo como un campo prometedor.
- Adopción industrial: Empresas tecnológicas como Microsoft y Google han influido en la adopción de estas herramientas, destacando su impacto práctico.

Las tendencias actuales indican un enfoque hacia la personalización de herramientas y la integración con IA para mejorar la precisión y relevancia del análisis.

El análisis multicriterio desarrollado muestra los siguientes resultados sobre las métricas evaluadas.

El problema que aborda el análisis es determinar cuáles criterios y métricas son más relevantes para evaluar y gestionar la deuda técnica utilizando herramientas de análisis estático.

La jerarquía se organiza en tres niveles principales:

Nivel 1: Objetivo General

- Priorizar criterios para gestionar la deuda técnica.

Nivel 2: Criterios Principales

- Costo: Incluye los recursos necesarios para implementar soluciones.
- Impacto en la calidad del código: Mejora en aspectos como mantenibilidad, legibilidad y modularidad.
- Cumplimiento de estándares: Alineación con normativas y buenas prácticas de la industria.
- Facilidad de uso: Grado de usabilidad de las herramientas de análisis.

Nivel 3: Subcriterios

- Dentro de Costo: tiempo de implementación, costo financiero.
- Dentro de Impacto en la calidad del código: reducción de defectos, disminución de deuda técnica.
- Dentro de Cumplimiento de estándares: cumplimiento OWASP, cumplimiento MISRA.
- Dentro de Facilidad de uso: curva de aprendizaje, personalización.

Los criterios se comparan entre sí según su importancia relativa en una escala de 1 a 9 (donde 1 es igual importancia y 9 es máxima preferencia de un criterio sobre otro).

Tabla 3. Criterios evaluados.

Criterios	Costo	Impacto en la Calidad	Cumplimiento	Facilidad de Uso
Criterios	1	1/3	1/5	1/2
Impacto en la Calidad	3	1	1/2	1
Cumplimiento	5	2	1	3
Facilidad de Uso	2	1	1/3	1

Cálculo de Pesos:

1. Normalización de la matriz: Cada elemento se divide por el total de su columna.
2. Cálculo del peso promedio: Se calcula el promedio de cada fila, obteniendo los pesos de los criterios.

Pesos obtenidos:

- Costo: 15%
- Impacto en la calidad: 35%
- Cumplimiento de estándares: 40%
- Facilidad de uso: 10%

Verificación de Consistencia:

El índice de consistencia (IC) se calcula para garantizar que las comparaciones sean coherentes. Si el IC es menor a 0.10, se considera que la matriz es consistente.

Fórmula del IC según la ecuación 1:

$$IC = \frac{\lambda_{max} - n}{n - 1} \quad (1)$$

Donde λmax es el mayor valor propio de la matriz, y n es el número de criterios.

Aplicación a Subcriterios:

El proceso se repite para cada conjunto de subcriterios, obteniendo pesos locales que luego se combinan con los pesos globales de los criterios principales.

Resultados Finales:

La suma ponderada de los criterios y subcriterios produce un ranking de importancia para la toma de decisiones.

Tabla 4. Ranking de importancia

Métrica	Peso Global (%)
Cumplimiento de OWASP	25
Reducción de defectos	20
Costo financiero	15
Curva de aprendizaje	10

DISCUSIÓN

Los resultados obtenidos del análisis multicriterio utilizando el método AHP ofrecen una visión jerarquizada de los factores más relevantes en la evaluación y gestión de la deuda técnica a través del análisis estático de código. En este apartado, se analizan los resultados en el contexto de su aplicabilidad práctica, implicaciones y limitaciones, ofreciendo una perspectiva crítica sobre su utilidad en proyectos de software.

Importancia del Cumplimiento de Estándares

El criterio de cumplimiento de estándares obtuvo el mayor peso global (40%), reflejando la prioridad de alinear los procesos de desarrollo con normativas internacionales como OWASP y MISRA. Este resultado es consistente con estudios previos que destacan que las empresas que implementan estándares de la industria experimentan una reducción en defectos de seguridad y mayor confianza en sus sistemas.

Herramientas como SonarQube y Checkstyle, que ofrecen integración con estándares, se posicionan como opciones preferidas para organizaciones que priorizan el cumplimiento normativo.

La implementación de estas normativas puede aumentar la carga inicial de configuración, especialmente en proyectos heredados con grandes volúmenes de código.

Impacto en la Calidad del Código

El impacto en la calidad del código (35%) fue identificado como el segundo criterio más relevante. Esto evidencia el valor que los desarrolladores y equipos de software otorgan a la mantenibilidad, legibilidad y modularidad del código, aspectos directamente relacionados con la sostenibilidad del software a largo plazo.

Varias investigaciones destacan que la calidad del código es fundamental para reducir la deuda técnica acumulada, ya que permite adaptaciones más rápidas a cambios futuros.

Las herramientas de análisis estático no siempre capturan métricas de calidad más abstractas, como la cohesión o el acoplamiento, lo que sugiere una oportunidad para integrar enfoques híbridos con análisis dinámico.

Costo como Criterio Secundario

Aunque el costo (15%) tiene una influencia menor en comparación con los criterios anteriores, sigue siendo un factor crítico, especialmente para pequeñas y medianas empresas. El análisis indica que los costos financieros y el tiempo de implementación pueden ser barreras para adoptar herramientas avanzadas.

En algunos trabajos revisados se observó que los altos costos iniciales asociados con herramientas de análisis estático pueden desincentivar su adopción, a pesar de los beneficios a largo plazo.

La promoción de herramientas de código abierto y la integración temprana en el ciclo de vida del desarrollo pueden mitigar este desafío.

Facilidad de Uso y Curva de Aprendizaje

Con un peso global de 10%, la facilidad de uso fue el criterio menos priorizado, pero su relevancia no debe subestimarse. Las herramientas con interfaces complejas o requerimientos técnicos avanzados pueden reducir la eficacia del análisis al dificultar su adopción por parte de equipos menos experimentados.

El tiempo necesario para capacitar a los desarrolladores y adaptar las herramientas al entorno específico puede ser significativo, especialmente en equipos con alta rotación.

Es importante desarrollar interfaces más intuitivas y ofrecer recursos de capacitación accesibles para maximizar el impacto del análisis estático.

Interpretación de los Subcriterios

Dentro de los subcriterios, el cumplimiento con OWASP (25%) y la reducción de defectos (20%) destacaron como las métricas más relevantes, reflejando un equilibrio entre la seguridad del software y la mejora de la calidad interna.

La creciente preocupación por la seguridad en aplicaciones modernas refuerza la importancia del cumplimiento con estándares como OWASP. Este enfoque también es compatible con la evolución hacia sistemas más seguros y robustos.

Aunque la reducción de defectos es prioritaria, los falsos positivos generados por herramientas de análisis estático pueden dificultar la focalización de los esfuerzos.

Consistencia de los Resultados

El índice de consistencia calculado en el análisis AHP fue satisfactorio (IC<0.10IC<0.10), lo que valida la coherencia de las comparaciones realizadas. Esto refuerza la confiabilidad de los pesos asignados y la jerarquización de los criterios.

Relevancia para la toma de decisiones

Los resultados del análisis multicriterio son de gran importancia para priorizar métricas y criterios en función de las necesidades específicas de cada proyecto. Esto es especialmente valioso para organizaciones que buscan equilibrar calidad, costo y seguridad en sus sistemas de software.

CONCLUSIONES

El análisis estático se confirmó como una herramienta clave para gestionar la deuda técnica mediante la detección temprana de defectos, la mejora continua del código y el cumplimiento normativo. Además, los resultados refuerzan la utilidad del método Analytic Hierarchy Process (AHP) para jerarquizar criterios y subcriterios en la evaluación de métricas, lo que aporta una visión estructurada para la toma de decisiones en proyectos de software.

El análisis también destacó oportunidades significativas, como la integración de enfoques híbridos que combinen análisis estático con técnicas dinámicas o el uso de inteligencia artificial para reducir la incidencia de falsos positivos y aumentar la precisión. Estas tendencias, junto con el desarrollo de herramientas personalizables y orientadas al

aprendizaje automático, prometen avanzar en la efectividad del análisis estático y su adopción industrial.

El análisis bibliométrico reveló un crecimiento constante en la investigación sobre deuda técnica y análisis estático desde 2015, impulsado por la integración de herramientas avanzadas en entornos DevOps. Asimismo, la adopción industrial por parte de empresas tecnológicas como Microsoft y Google demuestra su relevancia práctica, con énfasis en la personalización y la integración en pipelines de CI/CD para maximizar su utilidad.

REFERENCIAS BIBLIOGRÁFICAS

Alqaradaghi, M., Morse, G. y Kozsik, T. (2022). Detecting security vulnerabilities with static analysis – A case study. ***Pollack Periodica***, ***17***(2), 1-7. https://doi.org/https://doi.org/10.1556/606.2021.00454.

Alqaradaghi, M., Nazir, M., & Kozsik, T. (2023). Design and Implement an Accurate Automated Static Analysis Checker to Detect Insecure Use of SecurityManager. ***Computers***, ***12***, 247. https://doi.org/10.3390/computers12120247.

Baldassarre, M. T., Lenarduzzi, V., Romano, S., & Saarimäki, N. (2020). On the diffuseness of technical debt items and accuracy of remediation time when using SonarQube. ***Information and Software Technology***, ***128***, 106377. https://doi.org/https://doi.org/10.1016/j.infsof.2020.106377.

Cucalón Gaibor, J. F. y Sandoval Gutierrez, J. (2023). Deudas técnicas en el sistema transaccional. ***ReCIBE***(2). https://recibe.cucei.udg.mx/index.php/ReCIBE/article/view/293/215.

Diaz Mena, D. (2024). ***Una métrica para medir deuda técnica basada en el análisis de las más usadas. Caso de estudio del repositorio Square.*** Universidad Nacional de Colombia]. Medellín, Colombia. https://repositorio.unal.edu.co/bitstream/handle/unal/86250/1077453954.2024.pdf?sequence=2&isAllowed=y.

Guamán, D., Quezada-Sarmiento, P. A., Barba-Guaman, L. y Enciso, L. (2017). Use of SQALE and tools for analysis and identification of code technical debt through static analysis. 2017 12th Iberian Conference on Information Systems and Technologies (CISTI).

Gulabovska, H. y Porkolab, Z. (2019). ***Survey on Static Analysis Tools of Python Programs*** 8thWorkshop of Software Quality, Analysis, Monitoring, Improvement, and Applications, https://ceur-ws.org/Vol-2508/paper-gul.pdf

Iammarino, M., Zampetti, F., Aversano, L. y Di Penta, M. (2021). An empirical study on the co-occurrence between refactoring actions and Self-Admitted Technical Debt removal. ***Journal of Systems and Software***, ***178***, 110976. https://doi.org/https://doi.org/10.1016/j.jss.2021.110976.

Martínez González, A. C. (2024). ***Priorización de la deuda técnica alineada con el negocio: un caso de estudio industrial*** Universidad de Valladolid. https://uvadoc.uva.es/bitstream/handle/10324/71530/TFM-G1997.pdf?sequence=1&isAllowed=y.

Pereira, J. D. A. y Vieira, M. (2020, 7-10 Sept. 2020). On the Use of Open-Source C/C++ Static Analysis Tools in Large Projects. 2020 16th European Dependable Computing Conference (EDCC),

Vaillant, Y. M. (2011). ***Título: "Plataforma para el Análisis Estático de Vulnerabilidades del Código"*** Universidad de las Ciencias Informáticas]. https://repositorio.uci.cu/bitstream/ident/TD_04062_11/1/TD_04062_11.pdf.