DEPÓSITO LEGAL ZU2020000153
ISSN 0041-8811
E-ISSN 2665-0428

Revista de la Universidad del Zulia

Fundada en 1947
por el Dr. Jesús Enrique Lossada

Ciencias Exactas, Naturales y de la Salud

Año 17 N° 49
Mayo - Agosto 2026
Tercera Época
Maracaibo-Venezuela

79
ANIVERSARIO

Mantenibilidad sostenible: Modelo de dominio para integrar Green IT en el mantenimiento de software

Ana Karina Fernandes*
María Pérez **

RESUMEN

La gestión del software enfrenta una desconexión importante: los estándares de calidad vigentes no cuantifican el costo energético de la deuda técnica durante la fase de mantenimiento. El objetivo de esta investigación es presentar un Modelo de Dominio que integre los principios de Green IT dentro de los procesos de mantenimiento definidos por la norma ISO/IEC 14764 y las características de calidad de la ISO/IEC 25010. A nivel metodológico, el estudio documenta los resultados de la etapa de Acción del primer ciclo de una Investigación Acción. Tras un diagnóstico que reveló vacíos en la literatura, se procedió a la construcción del modelo conceptual como mecanismo de intervención. Los resultados presentan un modelo conceptual en UML, que permitieron identificar la Deuda Técnica Ambiental como el eslabón causal entre la baja mantenibilidad y el desperdicio energético. Se estructura una relación donde las fases de mantenimiento no solo corrigen fallos, sino que son auditadas por un actor gestor de energía, mediante indicadores físicos. Se concluye que la construcción de este artefacto teórico es el primer paso para instrumentalizar la sostenibilidad, sometiéndose ahora a evaluación para definir los objetivos de implementación en ciclos futuros.

PALABRAS CLAVE: Mantenimiento de software, Ingeniería de software, Eficiencia energética, Gestión de recursos, Desarrollo sostenible.

*Profesora. Universidad Católica Andrés Bello, Caracas, Venezuela. ORCID: https://orcid.org/0009-0007-7422-5682. E-mail: fernnde@ucab.edu.ve
**Profesora. Universidad Católica Andrés Bello, Caracas, Venezuela. ORCID: https://orcid.org/0009-0000-9288-5763. E-mail: mperezpu@ucab.edu.ve

Recibido:09/03/2026 Aceptado: 30/04/2026

Sustainable Maintainability: Domain Model for Integrating Green IT into Software Maintenance Phase

ABSTRACT

Software management faces a significant disconnect: current quality standards do not quantify the energy cost of technical debt during the maintenance phase. The objective of this research is to present a Domain Model that integrates Green IT principles into the maintenance processes defined by the ISO/IEC 14764 standard and the quality characteristics of ISO/IEC 25010. Methodologically, the study documents the results of the Action stage of the first cycle of an Action Research project. Following a diagnosis that revealed gaps in the literature, a conceptual model was constructed as an intervention mechanism. The results present a conceptual model in UML, which allowed for the identification of Environmental Technical Debt as the causal link between low maintainability and energy waste. A relationship is structured wherein maintenance phases not only correct faults but are also audited by an energy manager actor using physical indicators. It is concluded that the construction of this theoretical artifact is the first step toward operationalizing sustainability, which is now undergoing evaluation to define implementation objectives for future cycles.

KEYWORDS: Software maintenance, Software engineering, Energy efficiency, Resources management, Sustainable development.

Introducción

Según (Checkland, P., 2000) para abordar la creciente complejidad de los requerimientos tecnológicos, la Ingeniería de Software se establece como la disciplina fundamental que aplica principios de ingeniería al diseño, desarrollo, prueba, evaluación y mantenimiento de software. La ingeniería de software moderna, además, incorpora un enfoque en la sustentabilidad, asegurando que el desarrollo minimice el impacto ambiental y promueva la eficiencia energética.

Para Gómez y Rodríguez (2023) la sustentabilidad se entiende como un enfoque integral a largo plazo que busca la protección y conservación de los recursos, equilibrando factores sociales, económicos y ecológicos. En el desarrollo de software, esto implica el desarrollo y mantenimiento de sistemas de software que optimicen el uso de recursos y puedan minimizar la huella de carbono a lo largo de todo el ciclo de vida. Esto se alinea con la disciplina de la Green

IT (Tecnologías de la Información Verdes), que se enfoca en la producción y operación de TI con un impacto ambiental mínimo, y la Green Software Engineering, que promueve la eficiencia energética desde el diseño hasta el desmantelamiento.

Dentro del ciclo de vida, la fase de mantenimiento es indispensable, ya que el software requiere actualización continua para corregir errores (mantenimiento correctivo), mejorar el rendimiento (mantenimiento perfectivo), adaptarse a nuevos entornos (mantenimiento adaptativo) y, fundamentalmente, prevenir futuros fallos (mantenimiento preventivo, incluyendo refactorización). La mantenibilidad y la longevidad del software son consideradas métricas de sostenibilidad clave, ya que reducen la necesidad de reescrituras completas y conservan recursos a largo plazo tal como lo manifiesta (Kapoor, S., 2024).

Nos gusta pensar que el software vive en la nube y no pesa, pero esa visión tiene sus riesgos. Nos hemos acostumbrado a pensar en el código fuente como una entidad lógica que vive en la nube, olvidando que esa nube está hecha de servidores, cables y sistemas de enfriamiento que operan 24/7. Durante décadas, la prioridad de la industria fue clara: entregar rápido y que funcione. La eficiencia energética se consideraba un lujo o, peor aún, un problema ajeno al desarrollador.

Esa mentalidad ha generado una deuda técnica silenciosa. Cada línea de código redundante o mal optimizada exige un peaje físico en vatios. El problema se agrava aún más durante la fase de mantenimiento, y es allí donde el software pasa la mayor parte de su vida útil. Aquí es donde nos topamos con un muro metodológico, los estándares que usamos para trabajar no hablan el idioma de la sostenibilidad. Si un ingeniero quiere reducir la huella de carbono de su proyecto hoy, se encuentra con herramientas desconectadas de su flujo de trabajo habitual. La norma ISO/IEC 25010 le permite evaluar la eficiencia de desempeño, sí, pero no integra nativamente la traducción de esa eficiencia en impacto ambiental. Falta un puente que una la calidad del código con la factura energética.

Esta investigación en progreso busca construir ese puente. No se trata de crear una nueva metodología desde cero, sino de reorganizar el tablero. Proponemos como primer avance un Modelo de Dominio que integra los principios de Green IT dentro de los procesos de mantenimiento ya existentes, para comprender mejor este complejo planteamiento.

1. Contexto de la investigación

Existe la ilusión de que el software es un bien intangible, exento de las leyes físicas. Ahora bien, incidentes recientes como el fallo global de CrowdStrike en 2024 o la caída de AT&T nos recordaron una verdad incómoda: el código no vive en el aire, sino en hardware que falla, se calienta y consume energía. El software ha dejado de ser una simple herramienta de soporte operativo, y se ha convertido en la infraestructura crítica de la civilización moderna (Amazon Web Services, n.d.).

Pero esta dependencia tiene un precio oculto. Cuando un sistema crítico falla o es ineficiente, no solo genera pérdidas millonarias (estimadas en 5 mil millones de dólares en el caso CrowdStrike); también desperdicia recursos computacionales masivos tratando de recuperarse. Por lo tanto, garantizar que el software sea mantenible, no es solo una cuestión económica, sino de responsabilidad física y ambiental (Bamiduro, 2024).

La necesidad de gestionar la complejidad creciente y las altas expectativas de los usuarios exige un enfoque sistemático y disciplinado. Si bien Winters et al. (2020) definen la Ingeniería de Software mediante la aplicación de principios de diseño y prueba, sostenemos que esta definición debe evolucionar. Hoy en día, la disciplina exige incorporar la eficiencia energética como un requisito no funcional tan crítico como la seguridad o la fiabilidad.

En este marco, la sustentabilidad se ha consolidado como un pilar fundamental. La cual se entiende como un enfoque integral que busca la protección y conservación de los recursos mediante la integración de factores sociales, económicos y ecológicos en una perspectiva de largo plazo (Macías et al., 2006). Por consiguiente, la ingeniería de software sostenible (Green Software Engineering) busca minimizar el impacto ambiental del software, optimizando el uso de recursos y promoviendo la eficiencia energética en cada etapa del desarrollo.

Este compromiso se materializa en la adopción de Green IT (Tecnologías de la Información Verdes), que se enfoca en la minimización del impacto ambiental de los productos y la infraestructura tecnológica, y Green IS (Sistemas de Información Verdes), que amplía la óptica a los procesos organizacionales para mejorar el comportamiento medioambiental.

La característica de Eficiencia de Desempeño y su sub característica de Utilización de Recursos es la relación de la ISO/IEC 25010 con Green IT (que busca reducir el impacto

ambiental y la huella de carbono de las TI)(ISO/IEC 25010). Esta característica mide el rendimiento del software en relación con la cantidad y tipo de recursos utilizados (CPU, memoria, energía, etc.) al llevar a cabo su función; su sub característica clave es Utilización de recursos, es decir, el grado en que el software utiliza los recursos de manera que no exceda lo especificado es directamente proporcional a la eficiencia energética. Un software que optimiza el uso de la CPU y la memoria requiere menos potencia de cómputo del hardware subyacente.

Por otro lado, el objetivo del Green IT es la minimización del consumo energético. Un software diseñado con una alta eficiencia de desempeño (específicamente, baja utilización de recursos) contribuye directamente a reducir el consumo de energía en servidores, centros de datos y dispositivos de usuario final, cumpliendo así con los principios de la informática verde. En este sentido, los requisitos de eficiencia de desempeño en el desarrollo de software actúan como requisitos de software verde, asegurando que el producto final sea sostenible desde el punto de vista del consumo de recursos (Calero y Piattini, 2015).

Si bien la literatura clásica define el desarrollo de software como un continuo lineal (Pressman y Maxim, 2020), la perspectiva de sostenibilidad exige replantear la fase de mantenimiento, no como una etapa final, sino como el ciclo dominante del consumo de recursos. Prácticas obsoletas como el sobrediseño o la mentalidad de codificar primero y corregir después generan una deuda técnica que, inevitablemente, se traduce en ineficiencia energética a largo plazo, básicamente la fase de mantenimiento se convierte en el garante de la longevidad del software (Bamiduro, 2024). Al adoptar la clasificación estándar de la ISO/IEC 14764 (ISO/IEC/IEEE, 2022), destacamos especialmente el mantenimiento preventivo y la refactorización como estrategias clave de Green IT. Estas actividades no solo corrigen la estructura del código, sino que actúan directamente sobre la eficiencia operativa, reduciendo el desperdicio computacional y la necesidad de reescrituras completas que dispararía la huella de carbono del proyecto.

Técnicamente, existe una simbiosis entre la calidad del producto y su huella operativa. La característica de mantenibilidad ISO/IEC 25010 actúa como un habilitador de la sostenibilidad, un software diseñado con alta modularidad y analizabilidad reduce drásticamente el esfuerzo y los recursos de hardware necesarios para ejecutar cambios futuros. Por el contrario, un sistema

con baja mantenibilidad obliga a realizar intervenciones costosas y energéticamente intensivas, ya sea para corregir fallos o para adaptar el sistema a nuevos entornos.

Sin embargo, el diseño del software basado en los criterios de la ISO/IEC 25010, aunque necesario, es insuficiente si no se gestiona bajo una óptica ambiental. Si bien el estándar ISO/IEC/IEEE 14764 proporciona un marco robusto al clasificar el mantenimiento en correctivo, adaptativo, perfectivo y preventivo, su enfoque es predominantemente funcional y operativo. El estándar detalla *cómo* gestionar el cambio, pero carece de métricas explícitas para evaluar el impacto ambiental de dichos cambios. Esta desconexión normativa justifica la necesidad del Modelo de Dominio que proponemos y requiere revisar las definiciones tradicionales bajo esta nueva óptica.

El mantenimiento correctivo es la actividad más reactiva y se enfoca en resolver fallas después de que el sistema ha sido entregado. Su objetivo es reparar errores, fallos y defectos (*bugs*) descubiertos por los usuarios o el equipo de operaciones que impiden que el software cumpla sus requisitos funcionales o no funcionales. Una alta incidencia de este tipo de mantenimiento indica una baja fiabilidad del software, caracteristica propuesta por ISO 25010, para productos de software de alta calidad

El mantenimiento adaptativo es necesario para mantener la operatividad y la relevancia del software en un entorno tecnológico y regulatorio cambiante. Su objetivo es modificar el software para que pueda adaptarse a cambios en su entorno operativo externo; cuando el software tiene una alta portabilidad, el costo y la dificultad de este mantenimiento se reducen.

El mantenimiento perfectivo se enfoca en mejorar la calidad interna del software que no es inmediatamente visible para el usuario, haciéndolo más eficiente y fácil de gestionar. Su objetivo es mejorar la calidad interna y rendimiento del software sin añadir nuevas funcionalidades significativas. Con este tipo de mantenimiento se busca mejorar la mantenibilidad (analizabilidad, modularidad) y la eficiencia de desempeño (comportamiento temporal y utilización de recursos), lo que indirectamente reduce los costos de futuros mantenimientos.

Según la norma (ISO/IEC/IEEE 14764:2022) este tipo de mantenimiento consiste en la modificación del producto de software después de su entrega para detectar y corregir fallas

latentes antes de que se conviertan en fallas efectivas. A diferencia de los enfoques reactivos, el mantenimiento preventivo adopta una postura proactiva, buscando mejorar la fiabilidad y la mantenibilidad futura del sistema. Ahora bien, la sostenibilidad digital, esta categoría es crítica. Incluye técnicas como la refactorización verde y la detección de patrones de degradación de software que, si bien no interrumpen la operatividad inmediata, generan un consumo incremental de recursos de hardware. Al intervenir proactivamente código ineficiente o librerías obsoletas, se extiende la vida útil del software y se evita la deuda técnica ambiental, alineándose con los principios de eficiencia energética.

Desde una perspectiva operativa, existe una dependencia funcional clara entre ambos estándares: la ISO/IEC 14764 dicta el proceso (qué hacer) y la ISO/IEC 25010 establece el estándar (con qué calidad). Sin embargo, aquí radica el problema que abordamos: es perfectamente posible cumplir con ambos estándares y aun así generar software ecológicamente ineficiente. Un equipo puede ejecutar un mantenimiento perfectivo impecable (según la 14764) que mejore la mantenibilidad del código (según la 25010), pero que, inadvertidamente, dispare el consumo de recursos del servidor. Esta especie de ceguera ambiental, en la normativa actual es el vacío que nuestro Modelo de Dominio busca llenar, inyectando la variable de sostenibilidad justo en la intersección de estos dos estándares. Esta falta de conexión, que vincula de manera sistemática los principios de Green IT con la mantenibilidad (dimensión de la sostenibilidad técnica) y que defina un conjunto de métricas accionables, justifica la necesidad de esta investigación. La industria del software necesita desarrollar herramientas de medición precisas para promover una mayor adopción de prácticas de desarrollo sostenible.

Para abordar la sostenibilidad desde la ingeniería, es necesario armonizar dos visiones tradicionalmente separadas. Por un lado, la familia de normas ISO 25000 (SQuaRE) las cuales establecen *qué* debemos medir en términos de calidad interna del software. Por otro lado, la Ingeniería de Software Verde, introduce el criterio de eficiencia energética como una prioridad de diseño. No obstante, la carencia principal detectada en la literatura, es la falta de un proceso estandarizado que una estos mundos. Mientras que la ISO 14764 describe los procesos de mantenimiento (correctivo, adaptativo, etc.), no explicita cómo estas actividades deben gestionar la huella de carbono. La presente investigación utiliza el concepto de Deuda Técnica

como el eslabón perdido: el código difícil de mantener no solo es costoso financieramente, sino también energéticamente ineficiente.

2. Materiales y métodos

A diferencia de las ciencias básicas que describen fenómenos existentes, este estudio adopta un enfoque constructivo propio de la ingeniería: no buscamos solo observar el mantenimiento de software, sino intervenir en él creando un artefacto nuevo. Por ello, seleccionamos la Investigación-Acción (I-A) como método rector. Este enfoque permite el desarrollo de la investigación a partir de hipótesis teóricas, al tiempo que facilita una participación activa de los profesionales o practicantes en el proceso general de construcción de conocimiento. Dada la existencia de diversas interpretaciones de la I-A en la literatura (Baskerville, 1999), para la orientación metodológica de la presente investigación se ha adoptado la vista de I-A como un proceso continuo y cíclico de cinco etapas:

Diagnóstico (DI) que es la identificación de oportunidades de mejora en el grupo u organizaciones bajo estudio. Planificación de la Acción (PA) que es el desarrollo conjunto de cursos de acción alternativos para alcanzar la mejora y el desarrollo de conocimiento. Luego la toma de acción (TA) que es la selección e implementación del curso de acción definido. Posteriormente la evaluación (E) que es el análisis riguroso de los resultados obtenidos tras la acción implementada y finalmente la especificación del aprendizaje (EA) que no es más que la valoración de los resultados y generación de conocimiento en forma de modelos teóricos que describen la situación estudiada.

Este carácter cíclico permite que el proceso de la I-A continúe independientemente del éxito inicial de la acción tomada, para profundizar en el conocimiento de la organización o validar la solidez de los marcos teóricos relevantes. Como resultado de estos ciclos, se generan aprendizajes tanto para las organizaciones involucradas como elementos teóricos significativos para la academia. Esta naturaleza cíclica es la se utiliza en la investigación en progreso, el presente artículo sistematiza el primer ciclo completo de dicha metodología, reportando la evolución desde la identificación del problema hasta la generación del artefacto de solución. En el diagnóstico se realizó una revisión crítica de la literatura y los estándares ISO/IEC 14764 y 25010. Este análisis reveló la existencia de un vacío instrumental, es decir, la falta de conexión

entre los procesos de mantenimiento y las métricas de sostenibilidad. Dada la complejidad del problema detectado en el diagnóstico, se determinó la necesidad de desarrollar un Modelo de Dominio. El objetivo de esta etapa fue diseñar una herramienta conceptual que permitiera ponerle nombre a los vacíos encontrados y estructurar las relaciones entre calidad de software y consumo de energía, pasando así a la acción realizando un modelado activo de la solución utilizando UML. Durante la construcción del modelo, se identificaron y formalizaron nuevos conceptos, específicamente el de Deuda Técnica Ambiental, el cual surgió como el hallazgo central de la acción. El modelo resultante no es solo una representación, sino el producto de la intervención teórica sobre el problema. Actualmente nos encontramos en la fase final del primer ciclo que se basa en someter el modelo propuesto al escrutinio de la comunidad académica para validar su consistencia lógica y teórica. Los resultados de esta evaluación alimentarán la fase de aprendizaje, la cual definirá los objetivos específicos para los futuros ciclos.

3. Resultados - Modelo de Dominio

El Modelo de Dominio (ver figura 1) propuesto opera como el artefacto central de la Fase de Diagnóstico (DI) de esta investigación, no se limita a una representación estática de conceptos, sino que propone una estructura integrada donde los procesos de ingeniería, los estándares de calidad y la infraestructura física convergen para operativizar la sostenibilidad. A diferencia de los enfoques tradicionales, este modelo articula el cómo y el con qué, se logra un mantenimiento de software eco eficiente. En primera instancia, el modelo establece que el Ciclo de Vida del Software (SDLC) no actúa de forma aislada; su ejecución se rige por marcos de trabajo y metodologías que encuentran su fundamento en los estándares de Ingeniería de Software. Específicamente, se adopta la ISO/IEC/IEEE 14764 para estructurar los procesos de mantenimiento y la ISO 25010 para los modelos de calidad. La operatividad de esta estructura se manifiesta en las Fases de Ejecución, las cuales tienen la responsabilidad de realizar las Etapas del ciclo de vida. Es en este punto donde la teoría se vuelve práctica: cada fase emplea un conjunto de Herramientas especializadas para producir artefactos específicos, los cuales ahora incorporan criterios de sostenibilidad desde su concepción. La conexión con la calidad del producto se logra mediante la integración de la ISO 25010. El modelo identifica la Mantenibilidad como el eje

central dentro de la etapa de mantenimiento, vinculando directamente con la Eficiencia de Desempeño. Aquí se introduce un concepto crítico: la Deuda Técnica Ambiental. Se propone que un software con baja mantenibilidad acumula ineficiencias que degradan el rendimiento, incrementando el esfuerzo computacional y, por ende, el impacto ecológico. Al mitigar esta deuda en la fase de mantenimiento, el software recupera su eficiencia.

Finalmente, el modelo cierra la brecha entre el software y el hardware a través del subdominio de Green IT. La Eficiencia de Desempeño del software es monitoreada activamente por un Gestor de Energía, quien supervisa el comportamiento del Hardware y registra el Consumo de Energía real. Estos datos no quedan en el aire; alimentan directamente los Indicadores de Gestión, permitiendo que las Métricas de Evaluación dejen de ser abstractas y se conviertan en valores cuantificables de ahorro energético y reducción de huella de carbono. Esta trazabilidad completa es lo que permite validar el cumplimiento de los principios verdes en el mantenimiento de software.

Para la formalización y representación del Modelo de Dominio, se ha seleccionado el Lenguaje Unificado de Modelado (UML) que constituye un estándar ampliamente aceptado en Ingeniería de Software que facilita la especificación, visualización, construcción y documentación de los artefactos de un sistema (OMG, 2017). Se utiliza el Diagrama de Clases de UML para modelar la estructura estática del dominio, esta notación garantiza el rigor técnico y la claridad en la comunicación del modelo propuesto, asegurando que los elementos conceptuales y sus interconexiones sean comprendidos de manera inequívoca.

4. Discusión de los aportes del Modelo

El análisis del Modelo de Dominio propuesto, revela que la sostenibilidad del software no puede gestionarse como un atributo aislado, sino como una consecuencia directa de la calidad del código. A diferencia de los enfoques tradicionales de Johansson (2017) o Tate (2021), que sugieren prácticas verdes generales, nuestra propuesta instrumentaliza estas prácticas dentro del flujo de trabajo estándar. Al conectar semánticamente las fases de ejecución con las etapas del ciclo de vida, se demuestra que el mantenimiento verde no requiere reinventar la ingeniería de software, sino añadir una capa de auditoría física a los procesos existentes.

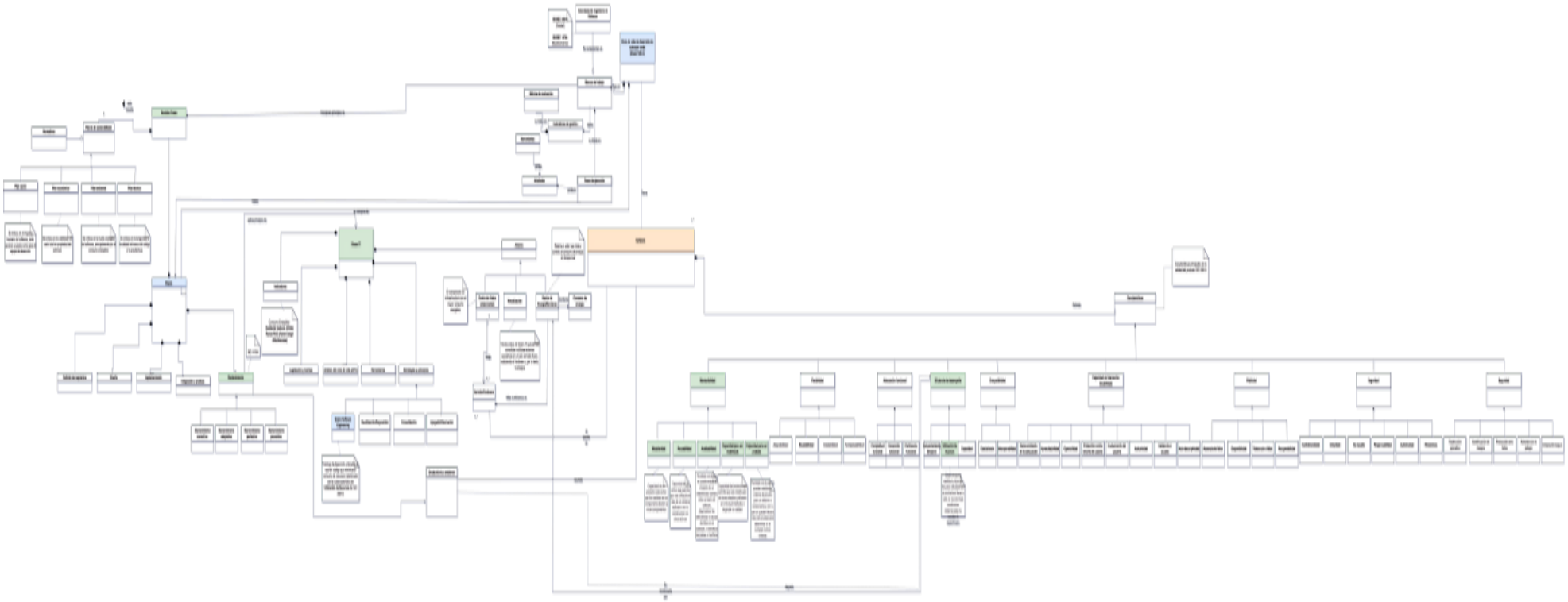

Figura 1. Modelo de Dominio para la Integración de Principios Green IT en la Fase de Mantenimiento de Software

Un hallazgo de esta modelación es la insuficiencia de la norma ISO/IEC 25010 cuando se aplica por sí sola. Si bien esta norma define la Eficiencia de Desempeño, nuestros resultados indican que, sin el actor Gestor de Energía y sin métricas de consumo de hardware, dicha eficiencia se queda en un plano teórico. El modelo presenta el vacío operativo al transformar la Mantenibilidad en un predictor de consumo, un software difícil de mantener (baja calidad interna) se traduce inevitablemente en una mayor carga computacional.

Aquí radica el aporte diferenciador de la investigación, la conceptualización de la Deuda Técnica Ambiental. Mientras que la literatura clásica define la deuda técnica como un costo financiero futuro (retrabajo), este estudio la reinterpreta como un costo energético inmediato. Esto implica que postergar la refactorización o mantener código obsoleto (prácticas comunes en la industria) tiene un impacto ecológico medible. Esta visión contrasta con la gestión reactiva tradicional, sugiriendo que la prevención de defectos como lo indica la ISO/IEC 14764 es, de hecho, la estrategia de ahorro energético más efectiva a largo plazo.

Finalmente, la inclusión explícita de los Indicadores de Gestión alimentados por el hardware resuelve la desconexión histórica entre los equipos de Desarrollo (Dev) y Operaciones (Ops). El modelo sugiere que para lograr una verdadera Ingeniería de Software Verde, los datos de consumo eléctrico deben ser visibles para quien escribe el código, cerrando así el ciclo de retroalimentación que hoy se encuentra roto en la mayoría de las organizaciones.

Conclusiones

La ingeniería de software ha operado tradicionalmente bajo la premisa de que la eficiencia del código es un problema de rendimiento, no de sostenibilidad. Al contrastar los estándares ISO/IEC 14764 e ISO/IEC 25010, se evidencia una brecha instrumental, disponemos de normas para gestionar procesos y evaluar calidad, pero carecemos de los mecanismos formales para auditar el costo energético de nuestras decisiones técnicas.

La principal contribución de este trabajo radica en la formalización de la Deuda Técnica Ambiental dentro del Modelo de Dominio propuesto. Lejos de ser un concepto abstracto, este modelo presenta que la mala calidad del software (código complejo, duplicado o difícil de mantener) tiene una traducción física directa en el consumo de recursos de infraestructura. Por

tanto, integrar principios de Green IT en la fase de mantenimiento no es solo una cuestión ética, sino una estrategia operativa para optimizar el ciclo de vida del producto.

Es importante destacar que el modelo propuesto no automatiza decisiones, sino que instrumentaliza la visibilidad del consumo energético. Al establecer relaciones semánticas claras entre los artefactos de software y los indicadores físicos, se ha transformado una preocupación ecológica abstracta en un artefacto.

De esta manera, se cierra la fase constructiva del primer ciclo de Investigación Acción. El modelo resultante actúa ahora como una base teórica estabilizada, indispensable para avanzar hacia las siguientes etapas de la investigación. Su valor no reside en ofrecer una métrica final inmediata, sino en proporcionar la arquitectura lógica necesaria para que, en futuros ciclos experimentales, se puedan recolectar datos y validar correlaciones reales entre la calidad del código y la eficiencia energética.

Referencias

Amazon Web Services. (n.d.). Sustainability. Recuperado de https://aws.amazon.com/es/sustainability/.

Bamiduro, M. (2024). Sustainable software development lifecycle: a view on practices, stakeholders and organizational barriers. Recuperado de https://lutpub.lut.fi/bitstream/handle/10024/168073/masterthesis_bamiduro_mercy.pdf?sequence=1&isAllowed=y

Baskerville, R. y Pries-Heje, J. (1999) Grounded Action Research: A Method For Understanding IT in Practice. *Accounting, Management and Information Technologies* 9 pp. 1-23.

Calero, C., & Piattini, M. (2015). *Green in Software Engineering*. Springer.

Checkland, P. (2000) Systems Thinking, systems practice. Includes a 30- year retrospective. USA: John Wiley& Sons.

CIO. (2024, July 15). 8 major IT disasters of 2024. Retrieved from https://www.cio.com/article/3624552/8-major-it-disasters-of-2024.html.

Gómez, M. del P., & Rodríguez, M. del P. (2023). La sostenibilidad como principio rector en la formación del ingeniero de sistemas. *Revista Científica General José María Córdova*, 21(40), 375–396. https://portal.amelica.org/ameli/journal/368/3685191005/html/

ISO/IEC/IEEE. (2022). ISO/IEC/IEEE International Standard - Software engineering, Software life cycle processes - Maintenance ISO/IEC/IEEE 14764:2022.

ISO25000.com. (s.f.). *ISO/IEC 25010*. https://iso25000.com/index.php/normas-iso-25000/iso-25010

Johansson, B. (2017). Green Software Engineering: Principles and Practices. Springer.

Kapoor, S. (2024). Green Software Quality: A Comprehensive Framework for Sustainable Metrics in Software Development. Recuperado de https://www.ijcttjournal.org/2024/Volume-72%20Issue-10/IJCTT-V72I10P118.pdf

Macias, H., Téllez, O., Dávila, P., & Casas, A. (2006). Los estudios de sustentabilidad. *Revista Ciencias*. Número 81, pp. 20-31. https://www.redalyc.org/pdf/644/64408104.pdf

Marcos, E. (2003). Investigación en Ingeniería de Software vs Desarrollo de Software. Grupo KYBELE. ACTAS. Métodos de Investigación y Fundamentos filosóficos e Ingeniería del Software y Sistemas de Información. Universidad Rey Juan Carlos España.

OMG. (2017). OMG Unified Modeling Language (OMG UML), Version 2.5.1. Object Management Group, Inc. Recuperado de https://www.omg.org/spec/UML/2.5.1

Pressman, R. S., & Maxim, B. R. (2020). Software Engineering: A Practitioner's Approach (9th ed.). McGraw-Hill Education.

Sommerville, I. (2011). Ingeniería de Software. Novena edición. Addison-Wesley.

Tate, K. (2021). Sustainable Software Engineering: Principles and Practices. O'Reilly Media.

Winters, T., Manshreck, T., & Wright, H. (2020). Software Engineering at Google: Lessons Learned from Programming Over Time. O'Reilly Media.

Conflicto de interés

Los autores de este manuscrito declaran no tener ningún conflicto de interés.

Declaración ética

Los autores declaran que el proceso de investigación que dio lugar al presente manuscrito se desarrolló siguiendo criterios éticos, por lo que fueron empleadas en forma racional y profesional las herramientas tecnológicas asociadas a la generación del conocimiento.

Copyright

Licencia Creative Commons

REVISTA DE LA UNIVERSIDAD DEL ZULIA, Fundada el 31 de mayo de 1947

UNIVERSIDAD DEL ZULIA, Fundada el 11 de septiembre de 1891