UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Tecnura

https://revistas.udistrital.edu.co/index.php/Tecnura/issue/view/1327

DOI: https://doi.org/10.14483/22487638.24865

INVESTIGACIÓN

# Análisis de prácticas para reducir el consumo de energía en las pruebas de software: una revisión sistemática de la literatura

## Analysis of practices to reduce energy consumption in software testing: A Systematic Literature Review

Eduardo López-Chacón [1], Juan Carlos Pérez-Arriaga [2], Ángel J. Sánchez-García [3], Lizbeth Alejandra Hernández-González [4]

**Fecha de Recepción**: 12 de noviembre de 2025 **Fecha de Aceptación**: 12 de marzo de 2026

**Cómo citar:** E. López-Chacón, J.C. Pérez-Arriaga, A.J. Sánchez-García, L.A. Hernández-Gonzáles, «Análisis de prácticas para reducir el consumo de energía en las pruebas de software: una revisión sistemática de la literatura», *Tecnura*, vol. 30, n.° 88, jun. 2026. 102–125. https://doi.org/10.14483/22487638.24865

## Resumen

**Contexto:** El consumo de energía es importante para el desarrollo tecnológico actual, no solo por su impacto económico sino también por sus implicaciones ambientales. En el desarrollo de software, el consumo se intensifica en etapas con mayor demanda de energía, siendo la prueba una de las más notables. Aunque se han establecido nuevas directrices centradas en la sostenibilidad del software, poco se sabe aún sobre las prácticas específicas en la etapa de prueba para disminuir el consumo de energía. Los desarrolladores de software, a menudo, se centran en la funcionalidad y en la prueba de software sin considerar la eficiencia energética.
**Objetivo:** Este estudio analiza las prácticas que minimizan el consumo energético en la etapa de prueba de software, los métodos utilizados para su implementación, e identifica las herramientas de automatización que contribuyen a la reducción del consumo de energía.
**Metodología:** revisión sistemática de la literatura por medio del enfoque de Kitchenham.
**Resultados:** Se analizaron 30 estudios primarios en los que se identificaron prácticas enfocadas en el ahorro de energía durante las pruebas de software. Se destacan la optimización de recursos y el escalado dinámico de voltaje y frecuencia (*DVFS*) que reduce el consumo de energía al ajustar la velocidad de procesamiento en función de la demanda esperada.
**Conclusiones:** Este trabajo proporciona una base de referencia para interesados en el desarrollo de software que busquen estrategias sustentables durante la fase de prueba.

1 Estudiante en Ingeniería de Software por la Universidad Veracruzana. Email: zS20015721@estudiantes.uv.mx

2 Maestro en Ciencias de la Computación y Profesor de Tiempo Completo en la Facultad de Estadística e Informática de la Universidad Veracruzana. Email: juaperez@uv.mx

3 Licenciado en Informática por la Universidad Veracruzana en 2011. En el 2013 obtuvo su grado de Maestro en Inteligencia Artificial y un año más tarde el grado de Especialista en Métodos Estadísticos por dicha Universidad. En 2018 Obtuvo su grado de Doctor en Inteligencia Artificial por el Centro de Investigación en Inteligencia Artificial de la Universidad Veracruzana. Actualmente es profesor de tiempo completo en la Facultad de Estadística e Informática en la Universidad Veracruzana, Licenciatura en Ingeniería de Software, Xalapa, Veracruz, México. Email: angesanchez@uv.mx

4 Licenciada en Informática y Maestra en Ingeniería de Software por la Universidad Veracruzana. Doctora en Ciencias de la Ingeniería por el Instituto Tecnológico de Orizaba, parte del Tecnológico Nacional de México. Hasta la fecha es profesora de tiempo completo en la Universidad Veracruzana en la Licenciatura en Ingeniería de Software campus Xalapa, Veracruz, México. Email: lizhernandez@uv.mx

***Palabras clave:*** Energía verde, Eficiencia energética, Consumo de energía, Pruebas de Software, Ingeniería de software, Pruebas Automatizadas.

## Abstract

**Context:** Energy consumption is important for current technological development, not only due to its economic impact but also because of its environmental implications. In software development, consumption intensifies in stages with higher energy demand, with testing being one of the most notable. Although new guidelines focused on software sustainability have been established, little is yet known about specific practices in the testing stage to reduce energy consumption. Software developers often focus on functionality and software testing without considering energy efficiency.
**Objective:** This study analyzes the practices that minimize energy consumption in the software testing stage, the methods used for their implementation, and identifies automation tools that contribute to reducing energy consumption.
**Methodology:** Systematic literature review using the Kitchenham approach.
**Results:** Thirty primary studies were analyzed, identifying practices focused on energy savings during software testing. Notable among them are resource optimization and Dynamic Voltage and Frequency Scaling (DVFS), which reduces energy consumption by adjusting processing speed based on expected demand.
**Conclusions:** This work provides a baseline reference for stakeholders in software development seeking sustainable strategies during the testing phase.
***Keywords:*** Green energy, Energy efficiency, Energy consumption, Software testing, Software engineering, Automated testing.

## Introducción

En la actualidad, los sistemas de software se han convertido en un elemento clave en entornos cotidianos, educativos y organizacionales. Estos sistemas se diseñan para resolver problemas de diversa índole y facilitar actividades en empresas e instituciones. Durante el proceso de desarrollo se hace un uso intensivo de energía eléctrica debido a la variedad de herramientas empleadas para su construcción, así como durante la etapa de prueba de software, lo que contribuye a las emisiones de CO2 [1].

La ingeniería de software verde puede definirse como la aplicación de principios y prácticas sostenibles durante las etapas del desarrollo de software [2], con el objetivo de reducir su impacto ambiental negativo. Esto implica disminuir tanto el consumo de energía como el uso de materiales a lo largo del proceso de desarrollo.

La eficiencia energética del software es un aspecto fundamental para promover la ingeniería de software verde. El término software ecológico se refiere al software energéticamente eficiente; para lograrlo, es necesario disminuir el consumo de energía desde las primeras etapas de desarrollo de un producto de software y mantener una revisión constante durante todo su ciclo de vida [3].

La prueba de software es un proceso de control de calidad orientado a verificar que los sistemas desarrollados funcionen correctamente y sin complicaciones. Esta etapa es relevante en la industria del

software por la variedad de recursos y conocimientos técnicos que requiere, así como por los elementos que pueden incidir en el resultado final de un proyecto. De acuerdo con Valle [4], completar las etapas del proceso de pruebas de forma manual y repetitiva supone un gran esfuerzo; de hecho, el proceso completo de pruebas puede representar hasta el 40 % del costo total de los proyectos.

Aunque los desarrolladores de software tienen conciencia sobre el uso de energía, este aspecto no siempre se considera formalmente durante el proceso de desarrollo. Esto se debe, en parte, a que la información sobre prácticas para optimizar el consumo energético se encuentra dispersa y a que el esfuerzo de desarrollo suele enfocarse en el rendimiento del hardware y en los aspectos funcionales del software. Sin embargo, la eficiencia energética es un requisito no funcional esencial que requiere mayor atención y comprensión por parte de la comunidad de desarrollo de software [5]. Esta revisión sistemática de la literatura (RSL) analiza y clasifica las prácticas que reducen el consumo de energía en la fase de prueba de software, con el fin de reunir información sobre esta etapa desde una perspectiva sostenible.

## Trabajos relacionados

En los últimos años, diversos estudios han abordado la eficiencia energética desde diferentes perspectivas; Sin embargo, pocos estudios se han centrado explícitamente en las prácticas adoptadas durante la fase de prueba de software.

En el trabajo de Asadi. [6] se realizó una revisión sistemática de la literatura sobre Green IT con el objetivo de analizar el estado actual de la investigación en esta área entre los años 2007 y 2016. La revisión identificó 131 estudios primarios enfocados en la sostenibilidad ambiental dentro del ámbito de las tecnologías de la información. Los autores destacan que el impacto ambiental de las tecnologías de la información se origina en todas las etapas de su ciclo de vida: diseño, desarrollo, uso y disposición, lo que genera un consumo de energía y contribuye a las emisiones de $CO_2$, equivalentes al 2 % del total mundial. Uno de los desafíos importantes en desarrollo de software sobre eficiencia energética es la capacidad de medir y reducir el consumo de energía durante la ejecución de la fase de prueba. El trabajo de Jabbarvand *et al.* [7] representa una contribución significativa al proponer técnicas destinadas a minimizar el consumo de energía en aplicaciones móviles. Su investigación se centra en tres enfoques clave: generación de pruebas, evaluación de la adecuación del conjunto de pruebas y minimización del conjunto de pruebas.

Bruce et al. [8] introducen una técnica basada en pruebas dirigidas por búsqueda (SBST), que aplica un algoritmo genético para reducir el consumo de energía del software. Este tipo de enfoque es parte del campo de la ingeniería de software basada en búsquedas y utiliza algoritmos de optimización y heurística para resolver problemas complejos dentro de las pruebas de software, particularmente aquellos

que implican explorar amplios espacios de solución, logrando una reducción del 25 % en el consumo evaluado del programa.

En el ámbito de la ingeniería de software con enfoque sostenible, A.C. Moisés [9) presenta un estudio donde identifico 170 prácticas, de las que 70 están relacionadas con el consumo energía. Aunque el trabajo no se centra exclusivamente en la etapa de prueba, algunas de estas prácticas pueden aplicarse directamente a dicha fase del desarrollo de software. Asimismo, el estudio identifica otras categorías relevantes, como las prácticas de evaluación de la eficiencia energética, centradas en métodos y técnicas de medición del consumo en casos prácticos reales. En particular, para la etapa de prueba, se enfatiza el uso de herramientas y técnicas de monitorización y medición del consumo energético en tiempo real, lo que permite identificar las áreas del proceso que demandan mayor cantidad de recursos y energía, y, en consecuencia, orientar acciones de optimización.

A diferencia de los estudios anteriores, la presente investigación se enfoca específicamente en la identificación, clasificación y análisis de las prácticas que reducen el consumo energético durante la fase de prueba de software. Mientras que las revisiones previas han abordado el tema de la sostenibilidad de manera general o desde perspectivas de hardware, desarrollo o uso, este estudio profundiza en el ámbito de las pruebas, una etapa clave pero poco explorada en términos de eficiencia energética. Además, se incorporan métodos, técnicas y herramientas documentadas en la literatura reciente (2013–2024), estableciendo una base actualizada para tratar el consumo de energía en la etapa de prueba de software.

## Metodología

Este estudio siguió el método propuesto por Kitchenham et al. [10]. Como complemento a la estrategia, se integró el método de búsqueda automatizada Cuasi-Gold Standard, propuesto por Zhang et al. [11], para facilitar la identificación de estudios en el campo de la Ingeniería del Software. Una vez extraídos los datos de los estudios primarios, los hallazgos se organizaron y analizaron utilizando un enfoque de síntesis narrativa, siguiendo el procedimiento descrito por Popay et al. [12].

### A) Planeación

#### 1. Preguntas de investigación

Nuestro estudio se estructuró en torno a las siguientes preguntas principales de investigación, que se presentan en la Tabla 1.

**Tabla 1.** *Preguntas de investigación*

| Pregunta de investigación |
|---|
| RQ1. ¿Cuáles son las prácticas reportadas en la literatura que contribuyen a reducir el consumo de energía en la fase de prueba de software? |
| RQ2. ¿Qué actividades de prueba de software están asociadas con el consumo de energía? |
| RQ3. ¿Qué métodos se utilizan para reducir el consumo de energía en las prácticas de prueba identificadas? |
| RQ4. ¿Qué aportación tienen las herramientas de automatización en la reducción del consumo de energía en actividades de prueba? |

**Fuente:** elaboración propia.

## B) Conducción

### 1. Cadena de búsqueda

Las cadenas de búsqueda se crearon y evaluaron siguiendo el enfoque de Zhang et al. [9]. Como parte de este proceso, se obtuvo una lista inicial de estudios relevantes mediante la exploración manual del motor de búsqueda *IEEE Xplore*. La cadena de búsqueda con mejor rendimiento logra un retorno del 83 %, recuperando un total de 86 estudios, con un esfuerzo del 5,95 %, esta se conforma de la siguiente manera:

("software testing" OR "green practices" OR "green testing" OR "green software engineering") and ("energy minimization" or "energy consumption" or "energy awareness" or "efficient activity") and ("practices" or "methods" or "technique*")

### 2. Motores de búsqueda

En la búsqueda automatizada, se seleccionaron *IEEE Xplore*, *SciencieDirect*, SpringerLink, *ACM Digital Library* fuentes destacadas por Zhang et al. [11], para la búsqueda de estudios.

### 3. Selección de estudios primarios

En esta sección se presentan los criterios utilizados para la selección o exclusión de estudios en las Tabla **2** y **3**.

**Tabla 2.** *Criterios de inclusión*

| Identificador | Criterio |
|---|---|
| CI-1 | Estudios publicados entre 2013 y diciembre de 2024 |
| CI-2 | El artículo de investigación fue publicado en idioma inglés |
| CI-3 | El título muestra indicios de responder al menos 1 pregunta de investigación |
| CI-4 | El resumen da indicios para responder al menos 1 pregunta de investigación. |
| CI -5 | El texto del estudio responde al menos a 1 pregunta de investigación. |

**Fuente:** elaboración propia.

**Tabla 3.** *Criterios de exclusión*

| Identificador | Criterio |
|---|---|
| CE-1 | No se tiene acceso al estudio completo |
| CE-2 | El estudio es referente a la disciplina de la ingeniería de software |
| CE-3 | El estudio no está relacionado con el consumo de energía en la prueba de software |
| CE-4 | El estudio es redundante o duplicado de otros trabajos ya incluidos en la revisión. |
| CE-5 | Es un libro, póster, presentación, resumen o tutorial. |

**Fuente:** elaboración propia.

### 4. Proceso de selección

La selección de los estudios se realizó mediante un sistema de filtrado en fases consecutivas. En cada etapa, se aplicaron criterios progresivamente más restrictivos (de lo general a lo específico), descartando tempranamente los estudios irrelevantes. La Figura 1 resume este proceso.

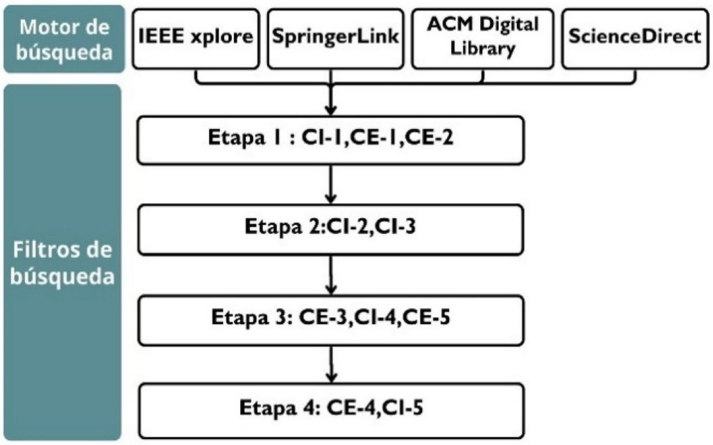


**Figura 1.** Proceso de selección de estudios primarios

**Fuente:** elaboración propia.

### 5. *Snowballing*

Para complementar la búsqueda automatizada, se llevó un proceso iterativo de bola de nieve hacia atrás y hacia adelante en los estudios identificados inicialmente, siguiendo las pautas metodológicas de Wohlin [13]. Todos los nuevos estudios identificados a través de este proceso iterativo se sometieron al mismo protocolo de selección detallado en la Figura 1, lo que garantiza la consistencia metodológica de nuestra revisión.

### 6. Síntesis

Para la síntesis de datos, se optó el método de síntesis narrativa propuesto por Popay et al. [12], un enfoque flexible que puede complementar otras metodologías de síntesis. Este proceso de síntesis, así como referencias de los estudios primarios se encuentra en ExtracciónDeDatos.xlsx (https://

docs.google.com/spreadsheets/d/1Fe47vuB7aWOBeZa1bq9xCoE4715d-OCN/edit?gid=1828509478#-gid=1828509478) de EP1 (Estudio primario 1) hasta EP30.

## Resultados

Después de implementar la estrategia de búsqueda mediante la cadena definida en las fuentes seleccionadas, se obtuvieron 16 estudios iniciales. Este conjunto inicial sirvió de base para aplicar la interacción de bola de nieve hacia adelante y hacia atrás, utilizando los mismos criterios de selección establecidos. Como resultado, se incorporaron 14 estudios adicionales, que pasaron por las mismas etapas de selección, como se muestra en detalle en la Figura 2b. Así, se conformó una selección final de 30 estudios primarios que constituyen la base de esta investigación. Los resultados de la búsqueda automatizada se presentan en detalle en la Figura 2, y la distribución de estudios por año, en la Figura 3.

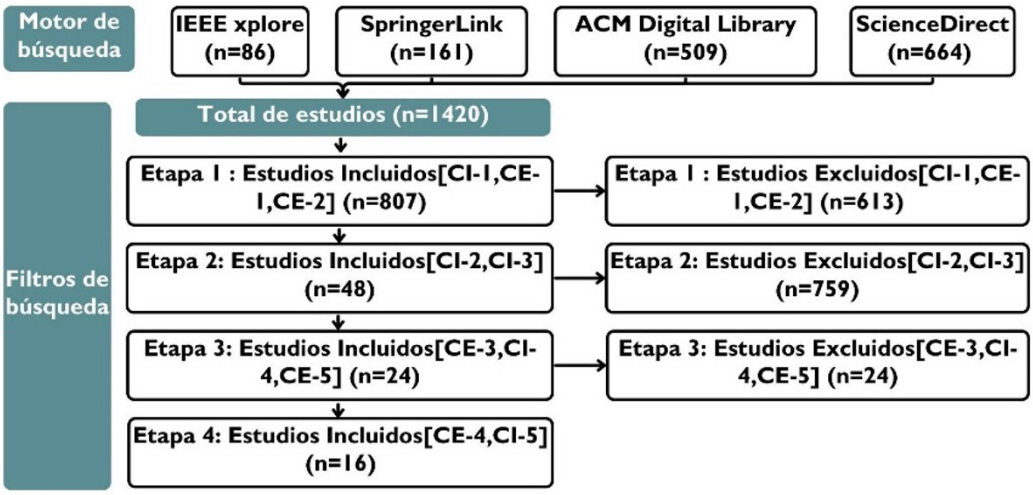


**Figura 2.** Resultados de selección de estudios primarios por etapa

**Fuente:** elaboración propia.

El proceso de selección de estudios mediante la técnica de *snowballing*, aplicada en ambas direcciones (*backward y forward*) sobre los resultados obtenidos en *Google Scholar*. A partir de un total de 946 estudios identificados inicialmente (433 provenientes del backward y 513 de la primera iteración), se aplicaron los criterios de inclusión y exclusión en cuatro etapas sucesivas. En la Etapa 1, se incluyeron 459 estudios que cumplían con los criterios iniciales (CI-1 y CE-1, CE-2), mientras que 487 fueron descartados. En la Etapa 2, tras aplicar los criterios CI-2 y CI-3, se mantuvieron 107 estudios, excluyéndose 352. Posteriormente, en la Etapa 3, al aplicar los criterios CE-3, CI-4 y CE-5, se seleccionaron 53 estudios adicionales. Finalmente, en la Etapa 4, se validaron 14 estudios que cumplían con los criterios CE-4 y CI-5, conformando el conjunto final derivado del snowballing. Este proceso, basado en fuentes provenientes de Google Scholar, permitió agregar más fuentes de literatura y garantizar la relevancia y calidad de los estudios primarios incluidos en la revisión.

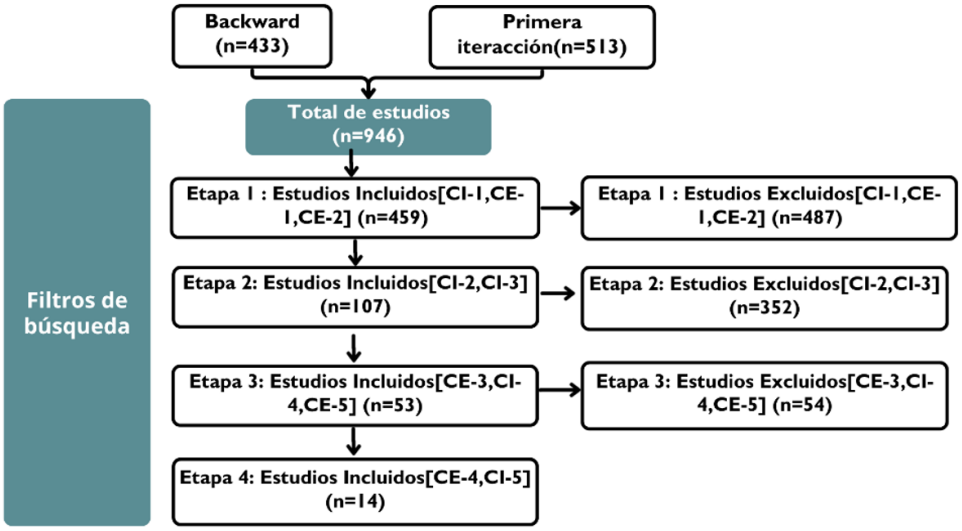


**Figura 2b.** Resultados de selección de snowballing de estudios primarios por etapa

**Fuente:** elaboración propia.

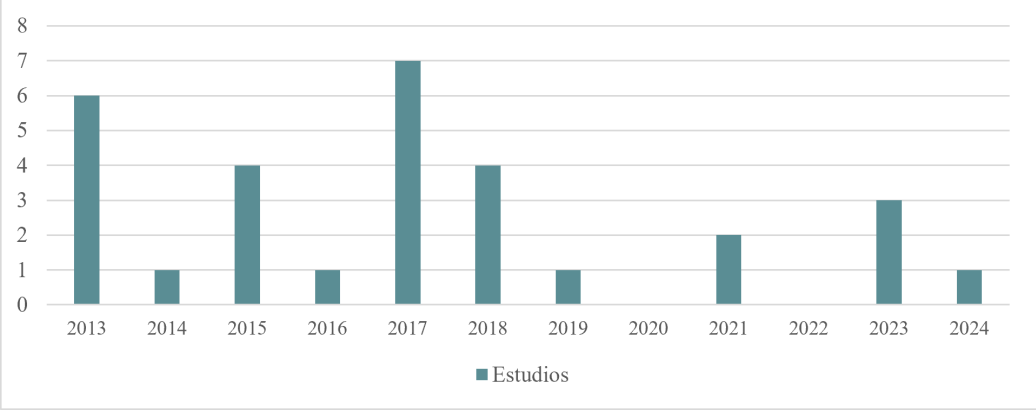


**Figura 3.** Distribución de estudios por año de publicación

**Fuente:** elaboración propia.

**PI1. ¿Qué prácticas reportadas en la literatura contribuyen a reducir el consumo de energía en la fase de prueba de software?**

Durante esta investigación, se analizaron diferentes enfoques, técnicas y métodos para identificar prácticas relacionadas a reducir el consumo de energía en pruebas de software. Estas prácticas se presentan en estudios que buscan hacer que el proceso de prueba sea más eficiente energéticamente, como la reducción de casos de prueba, el uso de algoritmos de optimización, herramientas de estimación de energía y técnicas de monitoreo de energía. La Figura 4 muestra las prácticas identificadas en los artículos.


**Figura 4.** Frecuencia de Prácticas

**Fuente:** elaboración propia.

### A. Uso de técnicas de evaluación de energía

El consumo de energía en las pruebas de software ha sido abordado desde varias perspectivas. Se han propuesto el uso de minería de datos y aprendizaje automático para construir modelos predictivos de consumo energético, mientras que otros han explorado las pruebas metamórficas como una alternativa para evaluar la relación entre entradas y salidas sin necesidad de un valor de salida esperado. Sin embargo, este enfoque enfrenta desafíos como la definición de relaciones metamórficas y la automatización del proceso. Otra técnica utilizada es la creación de perfiles de energía de software, que permite estimar el consumo basándose en datos de aplicaciones similares, aunque su precisión puede verse afectada por la influencia del entorno de ejecución [14, 3, 15]. Uno de los ejemplos para aplicar este tipo de práctica, es el enfoque que integra los niveles tradicionales de prueba con la medición del consumo energético, aprovechando casos de uso existentes para evaluar tanto la funcionalidad como el impacto energético. La Estimación del consumo energético mediante el tamaño, complejidad y dependencias del código (EP1). El estimador de energía desarrollado busca reducir el impacto de las características del hardware y permite realizar pruebas mediante monitoreo de hardware y técnicas de perfilado.

### B. Uso de técnicas de ahorro de energía

Para este tipo de práctica a diferencia de utilizar técnicas de evaluación, el ahorro energético es durante todo el proceso de prueba de software de recursos físicos que están involucrados. Esto puede lograrse mediante estrategias que optimicen el uso de los recursos computacionales, una de las técnicas en este ámbito es el Escalado Dinámico de Voltaje y Frecuencia (DVFS), que reduce el consumo de energía al ajustar la velocidad de procesamiento de acuerdo con la demanda esperada [16]. Esta práctica resulta

útil en pruebas de regresión, donde los casos de prueba se ejecutan repetidamente en diferentes versiones del software, ofreciendo la oportunidad de aplicar enfoques adaptativos sin afectar el rendimiento. Un estudio piloto con programas de escala media demostró que la aplicación de DVFS puede generar un ahorro energético promedio del 34,2%, evidenciando su eficacia en entornos de prueba. En dicho estudio se compararon diversos algoritmos de asignación de frecuencia, entre los cuales PAST (P) ajusta la frecuencia del CPU en función de su utilización (aumentándola por encima del 70% y reduciéndola por debajo del 50%), mientras que Online (O) emplea aprendizaje en línea para seleccionar la frecuencia óptima.

### C. Evaluación de energía en técnicas de prueba

La evaluación energética en las técnicas de prueba representa una práctica que busca integrar criterios de sostenibilidad en los procesos de prueba que comúnmente se ocupan. Asi como también existe una gran variedad de técnicas de prueba, el consumo de energía puede variar significativamente según factores como el tipo de prueba, la configuración del entorno y la complejidad del software evaluado. El estudio informa que el uso de herramientas como Green-JEXJ permite la medición simultánea de la efectividad de las pruebas y su impacto energético [17]. Esta herramienta integra JEXJ, utilizado para calcular la cobertura de sucursales en programas Java, con JouleMeter, que permite monitorizar el consumo energético. La evaluación energética se realiza durante la ejecución del caso de prueba, lo que permite la identificación de pruebas costosas en términos de consumo. Además, se incorporan componentes como JEXNCT, que mejora la cobertura a través de transformaciones de código, y JCUTE, una herramienta de ejecución simbólica dinámica que supera las limitaciones comunes en los motores concolic.

### D. Uso de técnicas minimización en prueba

El uso de técnicas de minimización en pruebas se presenta como una práctica esencial para optimizar recursos y mejorar la calidad del producto. Entre los elementos más notables de estas técnicas se encuentra la priorización de casos de prueba [18], que permite reducir los casos de prueba sin poner en riesgo la detección de fallos. Uno de estos ejemplos es la optimización de pruebas de regresión, una estrategia clave para reducir el esfuerzo de prueba, especialmente a través de la priorización de casos de prueba. Este enfoque busca ejecutar los casos en un orden que maximice la detección temprana de fallas de regresión, utilizando criterios de cobertura y algoritmos de búsqueda para definir dicho orden [19].

Otro elemento relevante es el uso de algoritmos genéticos basados en hipervolumen (HGA) para priorizar casos de prueba considerando múltiples criterios [20]. Los resultados de este enfoque sugieren que HGA es más eficiente y rentable que otros métodos, ya que mantiene una alta cobertura y reduce el consumo energético asociado a las pruebas. La minimización de suites de prueba basada en el consumo energético (EP3) aborda la minimización de suites orientadas a energía mediante la priorización

de casos de prueba que cubren puntos críticos de consumo energético. Por su parte, la minimización de suites de prueba para aplicaciones móviles (EP10) responde a la necesidad de gestionar eficientemente grandes suites diseñadas para evaluar el consumo energético de aplicaciones móviles. Se introducen métricas como eCoverage, que mide la cobertura de segmentos de código con alto consumo energético por cada caso de prueba y permite priorizar los casos más relevantes. Los resultados experimentales indican una reducción promedio del 84 % en el tamaño de las suites de prueba, manteniendo su efectividad en la detección de errores energéticos.

La optimización de suites de prueba basada en energía (EP11) utiliza programación lineal entera (ILP) para optimizar suites de prueba, asegurando que los casos seleccionados cumplan criterios específicos de minimización energética. Esta técnica logra una reducción de hasta el 95 % en el consumo energético de las pruebas, manteniendo la cobertura de requisitos. El método propuesto optimiza el consumo de energía en suites de prueba, en las que la selección de casos se formula como un problema de minimización con restricciones. Cada caso de prueba se representa con una variable binaria (bi), y el objetivo es minimizar la energía total mientras se cumplen criterios como la cobertura de código, expresados mediante restricciones lineales. La técnica se integra fácilmente en flujos existentes y es compatible con diversos criterios de prueba.

Otro ejemplo es la comparación del consumo energético de suites de prueba minimizadas mediante EDTSO (Energy Directed Test Suite Optimizer) frente a un método tradicional basado en tamaño, a partir de 70 000 problemas de minimización (10 000 por aplicación) con requisitos de cobertura aleatorios. Los resultados mostraron que, en el 92 % de los casos, EDTSO produjo suites con menor consumo energético que el enfoque tradicional, sin ningún caso de mayor consumo, con ahorros promedio del 3,8 % al 17,9 % y mejoras máximas cercanas al 100 % en ejecuciones individuales. Estos hallazgos evidencian que EDTSO es consistentemente más eficiente y permite reducciones significativas en el consumo de energía sin comprometer la cobertura de prueba.

### E. Uso de Herramientas de Evaluación de Energía

Para implementar la ingeniería de software ecológica y sostenible, los equipos de desarrollo pueden apoyarse en herramientas como la Integración Continua (CI), que permite minimizar el esfuerzo de integración y obtener retroalimentación constante sobre la calidad del software (EP8). Al integrar la medición de eficiencia energética en este proceso, los desarrolladores pueden monitorear el consumo de energía a lo largo del ciclo de desarrollo. Se utilizan métodos como medidores de potencia de hardware y modelos de estimación de consumo energético basados en el uso de CPU para medir el impacto de las pruebas. Herramientas como *TestNG* y servidores CI como Jenkins pueden extender sus informes para incluir métricas de eficiencia energética, lo que permite a los desarrolladores identificar y corregir problemas de consumo energético.

Diversos estudios han explorado formas de medir el consumo energético del software (SEC), comparando el consumo de energía entre diferentes versiones de productos y proporcionando información relevante para los desarrolladores. Sin embargo, se ha identificado un conocimiento limitado sobre la eficiencia energética y la falta de prácticas claras para reducir el SEC, además de incertidumbre sobre cómo el software consume energía. En este contexto, se ha desarrollado un estudio de casos múltiples que propone un tablero de energía para monitorear el consumo energético durante el desarrollo del software [21]. Para medir el consumo energético del software (SEC), se utilizó Microsoft Joulemeter (JM), un enfoque basado en software que estima el consumo total de energía del sistema en tiempo de ejecución, con mediciones cada segundo. El tablero muestra el consumo de recursos con base en un punto de referencia, en los resultados se evidenció una reducción del 2,15% en el consumo energético de un nuevo lanzamiento en comparación con el anterior, representado visualmente mediante colores que indican la magnitud del cambio: verde para disminuciones y rojo para incrementos. Este enfoque, aplicado a múltiples lanzamientos, permite evaluar la evolución del consumo energético a lo largo del tiempo.

### F. Correción de Code Smells

Los olores de código en Android podrían afectar el consumo de energía, se presentan dos herramientas: Adoctor, que detecta 15 olores de código específicos de Android mediante análisis estático y fue evaluado en 18 aplicaciones. Los resultados muestran que los "code smells" aumentan significativamente el consumo de energía, especialmente aquellos métodos que contienen múltiples olores de código. Cuatro olores de código con mayor impacto en el consumo energético fueron identificados: *"Leaking Thread", "Member Ignoring Method*", "*Slow Loop*" e "*Internal Setter*". La refactorización de estos olores juega un papel crucial en la mejora de la eficiencia energética [22]. PETRA es una herramienta que extrae perfiles de energía de aplicaciones Android, y se evaluó con 54 aplicaciones móviles utilizando un conjunto de datos público, comparando sus mediciones de energía con datos reales de consumo. La corrección de olores de código en pruebas de software es una estrategia efectiva para mejorar el consumo de energía durante la fase de prueba. Misu et al. [23] demuestra que los *test smells*, es decir, malas prácticas o estructuras ineficientes en el código de prueba, incrementan significativamente el consumo. Esta situación se agrava cuando un test case acumula múltiples instancias de smells, ya que la energía requerida crece proporcionalmente.

Entre los *test smells* que el autor asoció con un alto consumo energético destacan *Assertion Roulette* (AR), *Lazy Test* (LT) y *Eager Test* (ET). Estos olores no solo dificultan la mantenibilidad del código, sino que también generan sobrecarga computacional innecesaria durante la ejecución de pruebas, lo cual se traduce directamente en mayor gasto de energía. La eliminación mediante reescritura o reestructuración del código permite reducir este consumo, las evidencias de pruebas refactorizadas presentaron reducciones significativas en el consumo energético, con una diferencia promedio de hasta

20 joules por caso de prueba. Este resultado valida que la corrección de test smells no solo mejora la calidad del código, sino que también contribuye a la sostenibilidad del proceso de prueba.

**PI2. ¿Qué actividades de prueba de software están asociadas con el consumo de energía?**

Gestión de datos y entornos de prueba [24], es una actividad crítica que puede influir en el consumo de energía. El conocimiento sobre la organización, sus procesos, el negocio, el dominio del producto y las preferencias de los colegas es esencial para lograr los objetivos del proyecto. En este contexto, los desafíos relacionados con la gestión del conocimiento pueden explicarse por la falta de recursos adecuados, como conjuntos de datos y entornos de prueba mal aprovisionados, además de una estructura organizativa que no asigna recursos dedicados a las actividades de prueba.

En la Figura 5, se puede apreciar actividades con mayor consumo reportado en los artículos.

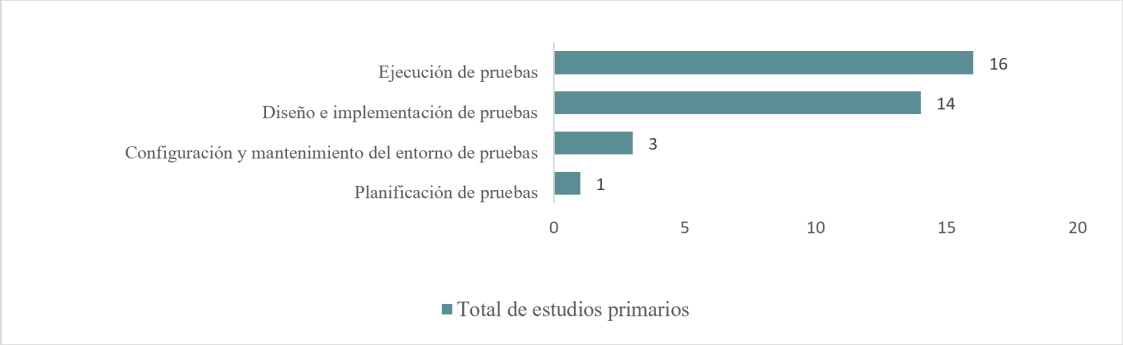


**Figura 5.** Actividades con mayor consumo de energía

**Fuente:** elaboración propia.

Las actividades de prueba de software asociadas con el consumo de energía incluyen la ejecución del conjunto de pruebas. Se ha observado que el consumo de energía de una prueba de funcionamiento es menor en comparación con una construcción completa, lo que representa entre el 65 % y el 73 % del consumo total de energía de una construcción. Sin embargo, existe en el consumo de energía una diferencia entre la construcción del software y la ejecución de pruebas, dependiendo de la plataforma utilizada [18].

La estimación del consumo de energía en las pruebas de software depende no solo de la ejecución directa de la prueba, sino también de factores estructurales del software, como las dependencias del método [14]. Desde esta perspectiva, la dependencia energética se refiere a la energía consumida por fragmentos de código ejecutados en el contexto del método bajo prueba; es decir, incluye el consumo de cualquier método invocado directamente. Esta información se puede derivar del gráfico de llamadas del programa, que proporciona una visión más completa del impacto energético de una prueba individual.

**PI3. ¿Qué métodos se utilizan para reducir el consumo de energía en las prácticas de prueba identificadas?**

Los métodos representan enfoques técnicos y estratégicos que buscan optimizar tanto la ejecución de pruebas como el uso de recursos computacionales. En este sentido, la Tabla 4 presenta de manera organizada los principales métodos encontrados dentro de los estudios analizados.

**Tabla 4.** *Métodos para reducir el consumo de energía*

| ID | Práctica | Método o herramienta | Estudio |
|---|---|---|---|
| P1 | Uso de técnicas de evaluación de energía | Energy profiling<br>Random decision forests for profiling energy usage<br>Estimación de consumo energético mediante el tamaño, complejidad y dependencias del código | Ep1, ep4, ep13, ep28 |
| P2 | Uso de técnicas de ahorro de energía | Dynamic voltage and frequency scaling | Ep2 |
| P3 | Evaluación de energía en técnicas de prueba | Green-jexj | Ep6 |
| P4 | Uso de técnicas minimización en prueba | Test-suite minimization<br>Hypervolume genetic algorithm for test case prioritization<br>Energy-aware test-suite minimization problem can be represented as an ip model<br>Energy-directed test suite optimizer (edtso)<br>Hypervolume based search for test case prioritization<br>Gtcm<br>Generalized tree reduction algorithm (gtr)<br>Delta<br>Multi-criteria test case prioritization | Ep3, ep7, ep11, ep12 ep16, ep17. Ep20, ep24, ep25, ep26, ep29, ep30 |
| P5 | Uso de herramientas de evaluación energética | Mats tool<br>Ct-3 powermeter,<br>Microsft joulemeter<br>Green-j3 model<br>Greenabce<br>Petra | Ep5, ep9, ep4, ep14, ep15, ep18, ep21, ep23 ,ep10 |
| P6 | Corrección de code smells | Adoctor<br>Refactorización | Ep7, ep27 |

**Fuente:** elaboración propia

La práctica de evaluación energética se centra en analizar cómo el software consume energía durante su ejecución. Una de las técnicas más empleadas es *Energy Profiling*, que permite identificar las partes del código o los procesos que más recursos consumen. Esta técnica recopila datos sobre el uso del CPU, la memoria y el tiempo de ejecución, relacionándolos con el consumo energético total (EP4). Entre los métodos utilizados para el perfilado energético destaca el uso de *Random Decision Forests*, un enfoque basado en aprendizaje automático que permite predecir y clasificar patrones de consumo energético a partir de mediciones previas [25]. Complementariamente, el método de estimación de consumo energético mediante el tamaño, la complejidad y las dependencias del código, que consiste en calcular el gasto energético esperado de un programa en función de métricas estáticas del software, como el número de

líneas de código, la cantidad de llamadas entre módulos y la complejidad ciclomática (EP1). En conjunto, estos enfoques apoyan la detección temprana de ineficiencias y permiten aplicar optimizaciones dirigidas a reducir el consumo energético en la fase de prueba.

Técnicas en ahorro energía en la fase de prueba de software como optimizar el uso de los recursos, el ajuste dinámico de la frecuencia del procesador en función de la carga de trabajo. Una técnica destacada en este ámbito es el Escalado Dinámico de Voltaje y Frecuencia (DVFS), permite reducir el consumo de energía ajustando la velocidad de procesamiento según la demanda prevista. En particular, en las pruebas de regresión, donde los casos de prueba suelen ejecutarse repetidamente en diferentes versiones del software, se pueden aplicar enfoques basados en DVFS para minimizar la disipación energética sin afectar significativamente el rendimiento. Los algoritmos de asignación de frecuencia comparados en el estudio incluyen enfoques para pruebas y pruebas de regresión. PAST (P) ajusta la frecuencia dinámicamente según la utilización del CPU, aumentando si supera el 70% y disminuyendo si cae por debajo del 50%. Online (O) emplea estadísticas de ejecución y aprendizaje en línea para seleccionar la frecuencia más adecuada. EClass-Target (ET) crea un perfil de frecuencia basado en intervalos de ejecución, aunque introduce sobrecarga en las pruebas de regresión. Case Majority (CM) selecciona la frecuencia más eficiente para la mayoría de los casos de prueba sin necesidad de perfilado. Case Optimal (CO) optimiza aún más esta estrategia utilizando la frecuencia ideal previamente determinada para cada caso de prueba, representando el mejor rendimiento en técnicas no intrusivas. Las técnicas CO, CM y ET demostraron una reducción del consumo de energía en la prueba de software, siendo CO la mejor, logrando ahorros de hasta un 57.9 % en comparación con PAST y más del 30 % frente a la técnica aleatoria de referencia.

La evaluación de la energía mediante herramientas como JouleMeter, JPCT y JCUTE dentro del framework Green-ABCE. Este enfoque busca mejorar la cobertura de ramas en las pruebas concólicas mientras se monitorea y calcula el consumo de energía asociado [26, 27, 28]. Aunque la mejora en la cobertura de ramas conlleva un aumento en el consumo energético, el uso de JouleMeter permite medir y registrar el consumo de energía de forma precisa, lo que permite optimizar el proceso. Además, el empleo de técnicas como la transformación de programas con JPCT y la generación automática de casos de prueba con JCUTE contribuye a la eficiencia de las pruebas y la reducción del consumo energético, a pesar de que la generación de casos de prueba puede volverse más lenta debido al tiempo adicional requerido para la transformación del programa. Estas técnicas mostraron menor variabilidad en los consumos, lo que su eficiencia es consistente en diferentes casos de prueba.

Los métodos y técnicas de minimización en la fase de prueba y la reducción de entradas en los casos de prueba [29] y la reducción de casos de prueba con prioridad, se realizan a través del empleo de una técnica de priorización mediante una cola de prioridad que organiza los nodos por medio de criterios, como el orden de recorrido de los padres y nodos [15, 30, 31]. Esto permite procesar los nodos de

manera más eficiente, minimizando el tiempo de ejecución y el consumo de energía asociado con el procesamiento individual de cada nodo. Este enfoque de reducción basada en prioridades y agrupamiento eficiente contribuye a mejorar la eficiencia energética del proceso de prueba al disminuir los tiempos de cómputo sin comprometer la efectividad de las pruebas.

Por su parte, el marco de *GTCM* (*Green Test Case Minimization*) se centra en reducir el número de casos de prueba necesarios para cubrir los criterios MC/DC, utilizando una serie de herramientas integradas [32]. GTCM utiliza herramientas como jCUTE para generar casos de prueba y calcular la cobertura de ramas, mientras que Joulemeter mide el consumo de energía asociado con la ejecución de estos casos de prueba, como resultado presenta una disminución en el consumo de energía, ya que se reduce el número de casos de prueba como el consumo de energía del sistema.

El hipervolumen mide la calidad de un conjunto de soluciones como el tamaño total del espacio objetivo dominado por una o más soluciones, lo cual permite equilibrar la cobertura de declaraciones y minimizar el costo de ejecución de una suite de pruebas [33]. Este enfoque tiene como objetivo maximizar la cobertura y minimizar el costo de ejecución de los casos de prueba, lo que indirectamente contribuye a la reducción del consumo de energía durante las pruebas al optimizar el uso de recursos computacionales. La minimización de suites de prueba para aplicaciones móviles [34] esta técnica aborda la necesidad de gestionar eficientemente grandes suites de prueba diseñadas para evaluar el consumo energético de aplicaciones móviles.

El enfoque OPDD (Optimized Parallel Delta Debugging) optimiza la reducción de casos de prueba al evitar repeticiones innecesarias, logrando reducir significativamente el número de pruebas realizadas. La optimización ayudo a reducir el tiempo de respuesta de los casos de prueba entre un 1,3% y un 43%, dependiendo de los casos [15], y en algunos casos, el tiempo de respuesta se redujo hasta en un 74%. En conjunto, estos enfoques muestran el potencial de combinar técnicas de cobertura, minimización y optimización para lograr pruebas de software más eficientes y energéticamente sostenibles.

La priorización de casos de prueba utilizando el proceso *Analytic Hierarchy Process* (AHP) y su extensión con lógica difusa (FAHP) permite manejar la incertidumbre y mejorar la precisión en la toma de decisiones [35]. Este enfoque se basa en la identificación de criterios relevantes, la determinación de alternativas, la medición del impacto de los criterios en cada alternativa, la fuzificación de datos lingüísticos y la aplicación de AHP como técnica de apoyo a la toma de decisiones (DSS). Entre los criterios utilizados para la priorización se encuentran la eficiencia en costos y tiempo, la cobertura de requisitos, la probabilidad de detección de fallos, la conclusividad del veredicto y un indicador de desviación de los requisitos. La minimización del conjunto de pruebas utilizando criterios múltiples, como lo implementa la herramienta Nemo*: Multi-Criteria Test-Suite Minimization with Integer Nonlinear Programming* [36]. Este enfoque selecciona el mejor subconjunto del conjunto original de pruebas considerando criterios de

restricción, como mantener la cobertura de sentencias original, y criterios de optimización, como maximizar la detección de fallos.

La estimación del consumo energético en pruebas de software puede abordarse mediante distintas herramientas, entre ellas los métodos basados en modelos estiman el consumo de energía mediante funciones matemáticas, pero también requieren calibración para obtener estimaciones confiables. Los métodos basados en software, como PETRA, presentan un costo más bajo asi como una mejor capacidad para estimar rápidamente el consumo de energía a nivel de método. PETRA mostró ser eficaz, alcanzando una precisión dentro del 5 % en comparación con herramientas de hardware, y permitiendo identificar métodos específicos en aplicaciones Android que consumen más energía. [37, 38].

El modelo Green-J3 propone una metodología para medir el consumo energético en pruebas de software basadas en cobertura de condiciones modificadas/decisiones (MC/DC) usando pruebas concólicas [25, 26]. Este enfoque consta de cinco módulos principales: JPCT, JCUTE, JCA, JouleMeter y Energy Calculator. La integración de estos módulos permite analizar el tiempo de ejecución, la energía consumida y el rendimiento del programa bajo prueba. El modelo ha demostrado una mejora en la cobertura de ramas en un 7.45 % y una reducción significativa del consumo energético, evidenciando su eficancia como herramienta para optimizar pruebas de software desde una perspectiva sostenible. El *framework* Green-ABCE utiliza herramientas como JouleMeter, JPCT y JCUTE, este enfoque busca mejorar la cobertura de ramas en las pruebas concolicas mientras se monitorea y calcula el consumo de energía asociado (EP6). Aunque la mejora en la cobertura de ramas conlleva un aumento en el consumo energético, el uso de JouleMeter permite medir y registrar el consumo de energía de forma precisa, lo que permite optimizar el proceso.

**PI4. ¿Qué aportación tienen las herramientas de automatización en la reducción del consumo de energía en actividades de prueba?**

Análisis de mutaciones de energía y generación automatizada de pruebas [22]. La herramienta MATS utiliza el análisis de mutaciones de energía y la generación automatizada de pruebas para evaluar la efectividad de los casos de prueba en la identificación de anomalías energéticas. El proceso comienza con la creación de un modelo abstracto del sistema bajo prueba (SUT) y una consulta para guiar la generación de casos de prueba. Este enfoque combina simulación, mutación controlada y criterios para optimizar las pruebas en sistemas basados en energía.

Aunque los *testers* cuentan con herramientas para verificar la funcionalidad y la lógica de las aplicaciones, no existen herramientas específicas para realizar pruebas desde una perspectiva de ingeniería de software verde y sostenible [39]. Por lo tanto, se destaca la importancia de contar con herramientas diseñadas específicamente para evaluar la eficiencia energética de las aplicaciones, lo que podría implicar la

integración de criterios de sostenibilidad en las fases de prueba. Además, se menciona que el aumento en la demanda de un alto rendimiento y nuevos modelos de uso continuará impulsando la necesidad de mejorar la eficiencia energética, especialmente en plataformas móviles y de escritorio.

Las herramientas de automatización contribuyen a reducir el consumo de energía en las pruebas de software optimizando la ejecución de casos de prueba y mejorando la cobertura del código. GreenJEXJ, que integra JEXJ para la cobertura de sucursales en Java con JouleMeter para medir el consumo de energía, lo que permite cuantificar y minimizar el consumo de energía de las pruebas (EP6). La adición de *Java Exclusive-NOR Code Transformer* (JEXNCT) aumentó la cobertura de la sucursal en un 7,45 % en promedio y redujo el consumo de energía en aproximadamente 75.945,1 julios.

## Discusión

Esta investigación identificó las prácticas reportadas en la literatura. Las prácticas que tuvieron mayor representación fueron aquellas orientadas a la reducción del número de casos de prueba, sin comprometer la cobertura ni la detección de fallos, algunas como Test-Suite Minimization, EDTSO, GTCM, GTR, DELTA y enfoques multicriterio, que abarcaron la mayor cantidad de estudios analizados.

Estas prácticas se han reportado en dos contextos principales: entornos controlados y proyectos industriales, cada uno con características y desafíos distintivos. En entornos controlados, predominan los enfoques basados en herramientas automatizadas y algoritmos de optimización ILP, DVFS que se emplean en los estudios [16, 35], donde las métricas de eficiencia energética se validan en condiciones ideales. Estos estudios suelen enfatizar la reducción teórica del consumo (hasta un 95 % en suites de prueba optimizadas) pero con limitaciones en escalabilidad o integración con pipelines industriales (EP11; EP2). Por otro lado, en los proyectos industriales, las prácticas reportadas se centran en la integración con CI/CD (Jenkins), monitoreo continuo (tableros de energía) y refactorización de code smells (Leaking Thread), con ahorros más modestos pero aplicables a contextos reales (EP7).

La literatura reporta técnicas para reducir el consumo energético en pruebas de software, destacando enfoques como el Escalado Dinámico de Voltaje y Frecuencia (DVFS) con evidencia de ahorros del 34.2 % al ajustar dinámicamente la frecuencia del procesador según la carga de trabajo, particularmente efectivo en pruebas de regresión [16]. Los algoritmos de optimización como ILP (Programación Lineal Entera) y genéticos mostraron mayor versatilidad, logrando reducciones de hasta el 95 % en suites de prueba al priorizar casos críticos (EP11; EP7). Para aplicaciones móviles, herramientas como PETRA y Adoctor permitieron identificar y corregir olores de código (e.g., *Leaking Thread*) que incrementan hasta un 300 % el consumo, validando que la refactorización guiada por métricas energéticas es clave en este dominio (EP7). Sin embargo, técnicas emergentes como pruebas metamórficas o modelos predictivos

con ML enfrentan desafíos de generalización, evidenciando que las soluciones deben adaptarse al contexto específico [22, 40].

El análisis reveló que la ejecución de pruebas consume entre el 20 % y 88 % de la energía de una compilación completa, variando según la plataforma y frecuencia de ejecución [27]. Proyectos con alta actividad como Apache Flink registraron consumos anuales de 23,746 kWh (3,218 compilaciones), mientras que otros con menor actividad como LinkedIn Cruise-control mostraron solo 0.612 kWh (68 compilaciones), demostrando que la frecuencia de integración continua impacta significativamente la huella energética. Además, la preparación de entornos y datos de prueba surgió como una actividad crítica, donde la automatización de configuraciones y la gestión eficiente de recursos redujeron hasta un 15% el consumo en proyectos distribuidos (EP8).

El framework Green-J3 integró herramientas como JCUTE y JouleMeter para medir consumo durante pruebas concolicas, logrando un 7.45% más de cobertura con reducción energética (EP14). Técnicas como *Pardis Hybrid* (EP16) y OPDD (EP26) optimizaron la reducción de casos de prueba mediante agrupamiento, disminuyendo hasta un 74% el tiempo de ejecución. Destacan especialmente los enfoques basados en ILP, que resolvieron problemas de minimización energética en menos de 1 segundo, manteniendo la cobertura de código [41). No obstante, métodos avanzados como el análisis de mutaciones energéticas (MATS) requieren mayor validación industrial, pues, aunque detectan eficientemente anomalías, su configuración es compleja (EP5).

Las herramientas automatizadas demostraron ser esenciales para escalar las prácticas de eficiencia energética. Green-JEXJ redujo 75,945 Joules al integrar mediciones energéticas en pruebas de cobertura (EP6), mientras que tableros de energía en CI/CD (Jenkins) permitieron monitorear reducciones del 2.15% entre versiones de software (EP9). PETRA destacó por su precisión del 5% frente al hardware especializado, siendo clave para perfilar aplicaciones móviles (EP7). Sin embargo, persiste una brecha en las herramientas para pruebas sostenibles, ya que el 83% de las soluciones analizadas son académicas y pocas se integran con ecosistemas industriales como Selenium o TestNG (EP23).

## Amenazas a la validez

Esta investigación se enfocó en identificar y analizar las prácticas, técnicas y herramientas orientadas a la reducción del consumo energético durante la etapa de prueba de software, sin pretender constituir una guía exhaustiva de aplicación. Su alcance se limita a la recopilación y síntesis de hallazgos documentados en la literatura académica, con el propósito de comprender los beneficios y desafíos asociados a las prácticas sostenibles en pruebas de software.

Entre las principales limitaciones, se reconoce que si bien el estudio se apoya en bases teóricas del área de prueba, no profundiza completamente en todos los niveles y tipos de prueba existentes, debido a las restricciones de tiempo y recursos disponibles. Limitó la inclusión de un mayor número de prácticas y técnicas reportadas en la literatura, enfocando el análisis en aquellas con evidencia y relevancia en la etapa de prueba.

Respecto a las amenazas este estudio, el acceso limitado a bibliotecas digitales y bases de datos de publicaciones. En consecuencia, se excluyeron estudios potencialmente relevantes, lo que puede haber afectado la profundidad del análisis. Esta limitación podría haber llevado a la omisión de ideas valiosas y contribuciones recientes que podrían haber enriquecido los resultados de esta revisión. Por otro lado, la fiabilidad de la selección del estudio radica en la subjetividad del investigador encargado de realizarlo manualmente.

Finalmente, la lengua materna del investigador es el español, no el inglés, lo que puede haber influido en la interpretación de los textos, la extracción de datos y la síntesis de la información. Sin embargo, este sesgo se mitigó mediante la supervisión parcial de los coautores de la investigación durante el desarrollo de la síntesis narrativa. En conjunto, estas consideraciones permiten establecer análisis transparente y riguroso, delimitando claramente los alcances, limitaciones y posibles fuentes de sesgo, al tiempo que fortalecen la credibilidad y consistencia de la revisión sistemática realizada.

## Conclusiones

En conclusión, esta investigación resalta la importancia de utilizar prácticas que contribuyen a minimizar el consumo energético durante la fase de pruebas de software. La revisión permitió identificar enfoques, entre los que destacan las técnicas de minimización de pruebas, los métodos de evaluación y estimación del consumo energético, el uso de herramientas especializadas, la optimización de recursos mediante ajustes de voltaje y frecuencia, y la corrección de code smells. Esta revisión siguió una metodología establecida con fines académicos y de investigación, evaluando estudios primarios publicados entre 2013 y 2024. Asimismo, se documentaron las actividades específicas del proceso de estas prácticas.

En cuanto a los métodos utilizados, la investigación evidenció que los enfoques de reducción y priorización de casos de prueba son los más ampliamente explorados. Estos incluyen técnicas como *Test-Suite Minimization*, *Energy-Directed Test Suite Optimizer* (EDTSO), *Green Test Case Minimization* (GTCM) y *Generalized Tree Reduction* (GTR). Estas técnicas buscan mantener la cobertura y la detección de fallos con el mínimo número de ejecuciones, logrando reducciones de energía de entre 30% y 58% en los estudios revisados.

La optimización de recursos en la ejecución de pruebas se logra mediante métodos como el *Dynamic Voltage and Frequency Scaling* (DVFS), que ajusta dinámicamente la frecuencia y el voltaje del procesador según la carga de trabajo. Este enfoque demostró ahorros energéticos promedio del 34,2 %, alcanzando en algunos casos reducciones de hasta 57,9 % al aplicar algoritmos como *Case Optimal* (CO) *y Case Majority* (CM) en entornos de integración continua. Estos métodos confirman que es posible disminuir el consumo de energía sin comprometer el rendimiento del sistema ni la cobertura de las pruebas.

Respecto a las técnicas de evaluación energética, se identificaron métodos de perfilado como *Energy Profiling y Random Decision Forests for Profiling Energy Usage*, así como herramientas como PETRA, Softwatt, Green-J3, MATS, GreenABCE y Microsoft JouleMeter, que permiten estimar y monitorear el consumo energético con una precisión del 5 % en comparación con dispositivos de medición por hardware. En particular, el modelo Green-J3 integró módulos de cobertura (JPCT, JCUTE, JCA) con medición energética (JouleMeter y Energy Calculator), logrando una mejora del 7,45 % en cobertura y una reducción significativa en el consumo energético durante las pruebas basadas en cobertura MC/DC.

Por otra parte, la refactorización de *code smells* mediante herramientas como ADOCTOR también se consolidó como una práctica efectiva para reducir el consumo de energía, ya que mejora la estructura del código, reduce operaciones redundantes y optimiza la eficiencia en pruebas de regresión y mantenimiento. No obstante, los hallazgos también revelan desafíos importantes. Entre ellos, la falta de estandarización en las métricas de medición energética donde persiste una brecha de conciencia y formación en torno a la ingeniería de software verde, lo que limita la adopción de prácticas sostenibles en entornos productivos.

En síntesis, esta revisión demuestra que es posible mejorar la eficiencia energética de la fase de prueba mediante la combinación de técnicas de reducción, optimización y medición, apoyadas por herramientas y algoritmos especializados. Este trabajo ofrece una visión general de las prácticas existentes y sirve como referencia para investigadores y profesionales que buscan tratar la eficiencia energética en las pruebas de software.

## Referencias

[1] H. Münzel, "Towards an ethical foundation of green software engineering," in *Proc. 10th IEEE Int. Conf. Global Softw. Eng. Workshops (ICGSEW)*, Ciudad Real, Spain, 2015, pp. 23–26. https://doi.org/10.1109/ICGSEW.2015.17 ↑

[2] S. Murugesan, "Harnessing green IT: Principles and practices," in *Green IT: Technologies and Applications*. Berlin, Germany: Springer, 2012, pp. 3–26. https://doi.org/10.1109/MITP.2008.10 ↑

[3] E. Jagroep *et al.*, "Awakening awareness on energy consumption in software engineering," in *Proc. 39th IEEE/ACM Int. Conf. Softw. Eng.: Softw. Eng. Soc. Track (ICSE-SEIS)*, Buenos Aires, Argentina, 2017, pp. 76–85. https://doi.org/10.1109/ICSE-SEIS.2017.10 ↑

[4] K. J. Valle-Gómez, P. Delgado-Pérez, I. Medina-Bulo, and J. Magallanes-Fernández, "Software testing: Cost reduction in Industry 4.0," in *Proc. 14th IEEE/ACM Int. Workshop Autom. Softw. Test (AST)*, Montreal, QC, Canada, 2019, pp. 69–70. https://doi.org/10.1109/AST.2019.00018 ↑

[4] I. Manotas *et al.*, "An empirical study of practitioners' perspectives on green software engineering," in *Proc. 38th IEEE/ACM Int. Conf. Softw. Eng. (ICSE)*, Austin, TX, USA, 2016, pp. 237–248. https://doi.org/10.1145/2884781.2884810 ↑

[5] S. Asadi, A. R. C. Hussin, and H. M. Dahlan, "Organizational research in the field of green IT: A systematic literature review from 2007 to 2016," *Telemat. Inform.*, vol. 34, no. 7, pp. 1191–1249, Nov. 2017. https://doi.org/10.1016/j.tele.2017.05.009 ↑

[6] R. Jabbarvand and S. Malek, "Advancing energy testing of mobile applications," in *Proc. 39th IEEE/ACM Int. Conf. Softw. Eng. Companion (ICSE-C)*, Buenos Aires, Argentina, 2017, pp. 491–492. https://doi.org/10.1109/ICSE-C.2017.45 ↑

[7] B. R. Bruce, J. Petke, and M. Harman, "Reducing energy consumption using genetic improvement," in *Proc. 2015 Annu. Conf. Genetic Evol. Comput. (GECCO '15)*, New York, NY, USA: ACM, 2015, pp. 1327–1334. https://doi.org/10.1145/2739480.2754752 ↑

[8] A. C. Moises, A. Malucelli, and S. Reinehr, "Prácticas de consumo energético para la ingeniería de software sostenible," in *Proc. 9th Int. Green Sustain. Comput. Conf. (IGSC)*, Pittsburgh, PA, USA, 2018, pp. 1–6. https://doi.org/10.1109/IGCC.2018.8752151 ↑

[9] B. A. Kitchenham, D. Budgen, and P. Brereton, *Evidence-Based Software Engineering and Systematic Reviews*, vol. 4. Boca Raton, FL, USA: CRC Press, 2015. https://doi.org/10.5555/2994449 ↑

[10] H. Zhang, M. A. Babar, and P. Tell, "Identifying relevant studies in software engineering," *Inf. Softw. Technol.*, vol. 53, no. 6, pp. 625–637, Jun. 2011. https://doi.org/10.1016/j.infsof.2010.12.010 ↑

[11] J. Popay *et al.*, "Guidance on the conduct of narrative synthesis in systematic reviews: A product from the ESRC Methods Programme," ESRC Methods Programme, Technical Report, 2006. https://doi.org/10.13140/2.1.1018.4643 ↑

[12] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proc. 18th Int. Conf. Eval. Assessment Softw. Eng. (EASE '14)*, New York, NY, USA: ACM, 2014. https://doi.org/10.1145/2601248.2601268 ↑

[13] F. Wedyan, R. Morrison, and O. S. Abuomar, "Integration and unit testing of software energy consumption," in *Proc. 10th Int. Conf. Softw. Defined Syst. (SDS)*, San Antonio, TX, USA, 2023, pp. 60–64. https://doi.org/10.1109/SDS59856.2023.10329262 ↑

[14] G. Gharachorlu and N. Sumner, "Avoiding the familiar to speed up test case reduction," in *Proc. IEEE Int. Conf. Softw. Quality, Reliab. Secur. (QRS)*, Lisbon, Portugal, 2018, pp. 426–437. https://doi.org/10.1109/QRS.2018.00056 ↑

[15] E. Y. Y. Kan, "Energy efficiency in testing and regression testing — A comparison of DVFS techniques," in *Proc. 13th Int. Conf. Quality Softw. (QSIC)*, Nanjing, China, 2013, pp. 280–283. https://doi.org/10.1109/QSIC.2013.21 ↑

[16] S. Godboley, A. Dutta, B. Besra, and D. P. Mohapatra, "Green-JEXJ: A new tool to measure energy consumption of improved concolic testing," in *Proc. Int. Conf. Green Comput. Internet of Things (ICGCIoT)*, Greater Noida, India, 2015, pp. 36–40. https://doi.org/10.1109/ICGCIoT.2015.7380424 ↑

[17] R. Verdecchia, P. Lago, C. Ebert, and C. de Vries, "Green IT and green software," *IEEE Softw.*, vol. 38, no. 6, pp. 7–15, Nov.–Dec. 2021. https://doi.org/10.1109/MS.2021.3102254 ↑

[18] D. Di Nucci, A. Panichella, A. Zaidman, and A. De Lucia, "Hypervolume-based search for test case prioritization," in *Search-Based Software Engineering (SSBSE 2015)*, Lecture Notes in Computer Science, vol. 9275, M. Barros and Y. Labiche, Eds. Cham, Switzerland: Springer, 2015. https://doi.org/10.1007/978-3-319-22183-0_11↑

[19] G. Gharachorlu and N. Sumner, "Pardis: Priority aware test case reduction," in *Fundamental Approaches to Software Engineering (FASE 2019)*, Lecture Notes in Computer Science, vol. 11424, R. Hähnle and W. van der Aalst, Eds. Cham, Switzerland: Springer, 2019. https://doi.org/10.1007/978-3-030-16722-6_24 ↑

[20] M. Dick, J. Drangmeister, E. Kern, and S. Naumann, "Green software engineering with agile methods," in *Proc. 2nd Int. Workshop Green Sustain. Softw. (GREENS)*, San Francisco, CA, USA, 2013, pp. 78–85. https://doi.org/10.1109/GREENS.2013.6606425 ↑

[21] J. Larsson and E. P. Enoiu, "Test generation and mutation analysis of energy consumption using UPPAAL SMC and MATS," in *Proc. IEEE Int. Conf. Softw. Test. Verif. Validation Workshops (ICSTW)*, Dublin, Ireland, 2023, pp. 186–189. https://doi.org/10.1109/ICSTW58534.2023.00042 ↑

[22] Md. R. H. Misu, J. Li, A. Bhattiprolu, Y. Liu, E. S. de Almeida, and I. Ahmed, "Test smell: A parasitic energy consumer in software testing," *Inf. Softw. Technol.*, vol. 181, art. 107671, May 2025. https://doi.org/10.1016/j.infsof.2025.107671 ↑

[23] C. Camacho, S. Marczak, and T. Conte, "On the identification of best practices for improving the efficiency of testing activities in distributed software projects: Preliminary findings from an empirical study," in *Proc. 8th IEEE Int. Conf. Global Softw. Eng. Workshops (ICGSEW)*, Bari, Italy, 2013, pp. 1–4. https://doi.org/10.1109/ICGSEW.2013.7 ↑

[24] M. A. Beghoura, A. Boubetra, and A. Boukerram, "Green software requirements and measurement: Random decision forests-based software energy consumption profiling," *Requir. Eng.*, vol. 22, no. 1, pp. 27–40, Mar. 2017. https://doi.org/10.1007/s00766-015-0234-2 ↑

[25] A. Hindle, "Green mining: A methodology of relating software change and configuration to power consumption," *Empir. Softw. Eng.*, vol. 20, pp. 374–409, 2015. https://doi.org/10.1007/s10664-013-9276-6 ↑

[26] A. Zaidman, "An inconvenient truth in software engineering? The environmental impact of testing open source Java projects," in *Proc. IEEE/ACM Int. Conf. Autom. Softw. Test (AST)*, Lisbon, Portugal, 2024, pp. 214–218. https://doi.org/10.1145/3644032.3644461 ↑

[27] S. Godboley, S. Panda, A. Dutta *et al.*, "An automated analysis of the branch coverage and energy consumption using concolic testing," *Arab. J. Sci. Eng.*, vol. 42, pp. 619–637, 2017. https://doi.org/10.1007/s13369-016-2284-2 ↑

[28] S. Herfert, J. Patra, and M. Pradel, "Automatically reducing tree-structured test inputs," in *Proc. 32nd IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE)*, Urbana, IL, USA, 2017, pp. 861–871. https://doi.org/10.1109/ASE.2017.8115697 ↑

[29] D. Li, C. Sahin, J. Clause, and W. G. J. Halfond, "Energy-directed test suite optimization," in *Proc. 2nd Int. Workshop Green Sustain. Softw. (GREENS)*, San Francisco, CA, USA, 2013, pp. 62–69. https://doi.org/10.1109/GREENS.2013.6606423 ↑

[30] L. V. Povoa, P. W. Bignatto, C. E. Monteiro, D. Mueller, C. A. C. Marcondes, and H. Senger, "A model for estimating energy consumption based on resources utilization," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Split, Croatia, 2013, pp. 1–6. https://doi.org/10.1109/ISCC.2013.6754957 ↑

[31] S. Godboley, A. Dutta, and D. P. Mohapatra, "Reduced energy consumption for MC/DC testing," *Int. J. Bus. Inf. Syst.*, vol. 28, no. 4, pp. 447–467, 2018. https://doi.org/10.1504/IJBIS.2018.093657 ↑

[32] D. Di Nucci, "Methods and tools for focusing and prioritizing the testing effort," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Madrid, Spain, 2018, pp. 722–726. https://doi.org/10.1109/ICSME.2018.00089 ↑

[33] R. Jabbarvand, A. Sadeghi, H. Bagheri, and S. Malek, "Energy-aware test-suite minimization for Android apps," in *Proc. 25th Int. Symp. Softw. Test. Analysis (ISSTA 2016)*, New York, NY, USA: ACM, 2016, pp. 425–436. https://doi.org/10.1145/2931037.2931067 ↑

[34] Sahar Tahvili, Mehrdad Saadatmand, and M. Bohlin, "Multi-Criteria Test Case Prioritization Using Fuzzy Analytic Hierarchy Process," *ResearchGate*, Nov. 15, 2015. https://www.researchgate.net/publication/281593743_Multi-Criteria_Test_Case_Prioritization_Using_Fuzzy_Analytic_Hierarchy_Process (accessed May 28, 2026)

[35] J.-W. Lin, R. Jabbarvand, J. Garcia, and S. Malek, "Nemo: Multi-criteria test-suite minimization with integer nonlinear programming," in *Proc. 40th IEEE/ACM Int. Conf. Softw. Eng. (ICSE)*, Gothenburg, Sweden, 2018, pp. 1039–1049. https://doi.org/10.1145/3180155.3180174 ↑

[36] D. Di Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman, and A. De Lucia, "PETRA: A software-based tool for estimating the energy profile of Android applications," in *Proc. 39th IEEE/ACM Int. Conf. Softw. Eng. Companion (ICSE-C)*, Buenos Aires, Argentina, 2017, pp. 3–6. https://doi.org/10.1109/ICSE-C.2017.18 ↑

[37] S. Song, F. Wedyan, and Y. Jararweh, "Empirical evaluation of energy consumption for mobile applications," in *Proc. 12th Int. Conf. Inf. Commun. Syst. (ICICS)*, Valencia, Spain, 2021, pp. 352–357. https://doi.org/10.1109/ICICS52457.2021.9464579 ↑

[38] M. Mohankumar and M. Anand Kumar, "An empirical study on green and sustainable software engineering," 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:36428799. ↑

[39] A. Dutta, "Green-J3 model: A novel approach to measure energy consumption of modified condition/decision coverage using concolic testing," *CSI Trans. ICT*, 2017. https://doi.org/10.1007/S40012-017-0157-9 ↑

[40] D. Li, Y. Jin, C. Sahin, J. Clause, and W. G. J. Halfond, "Integrated energy-directed test suite optimization," in *Proc. Int. Symp. Softw. Test. Analysis (ISSTA 2014)*, New York, NY, USA: ACM, 2014, pp. 339–350. https://doi.org/10.1145/2610384.2610414 ↑