# Catálogo de directrices de sostenibilidad para soportar el desarrollo y operación de software sostenible desde la ingeniería de requisitos

## Catalog of sustainability guidelines to support sustainable software development and operation from requirements engineering

**Rogelio Fernando Hernández Alarcón**
Universidad de Murcia, Murcia, España
rf.hernandezalarcon@um.es
ORCID: 0009-0001-3225-2881

**Begoña Moros Valle**
Universidad de Murcia, Murcia, España
bmoros@um.es
ORCID: 0000-0002-3092-7654

**Joaquín Nicolás Ros**
Universidad de Murcia, Murcia, España
jnr@um.es
ORCID: 0000-0003-1760-3804

doi: https://doi.org/10.36825/RITI.12.27.005

Recibido: Mayo 30, 2024
Aceptado: Julio 23, 2024

**Resumen:** La sostenibilidad constituye uno de los grandes retos de la sociedad actual, en particular la sostenibilidad de las TIC y más concretamente del software. Los ingenieros de requisitos necesitan directrices para abordar con éxito la sostenibilidad en los proyectos de software. Con este objetivo se diseña CRETS4DevOps, un método concebido para gestionar los requisitos en entornos DevOps y que promueve la sostenibilidad, especialmente en su dimensión técnica. CRETS4DevOps incorpora un catálogo de requisitos de sostenibilidad, que ayuda en la reutilización del conocimiento experto para abordar la gestión de atributos de calidad del software que redundan en la sostenibilidad: modularidad, escalabilidad, flexibilidad, reusabilidad, mantenibilidad, fiabilidad, seguridad, interoperabilidad y eficiencia. Este catálogo puede asistir tanto en el desarrollo y operación del software como en una auditoría de sostenibilidad. Actualmente se está desarrollando una herramienta de soporte a CRETS4DevOps, como *plug in* para Microsoft Azure DevOps.

***Palabras clave:*** *Sostenibilidad, Sostenibilidad Técnica, Catálogo de Requisitos, Reutilización de Requisitos, CRETS4DevOps.*

**Abstract:** Sustainability constitutes one of the major challenges of today's society, in particular the sustainability of ICTs and more specifically of software. Requirements engineers need guidelines to successfully address sustainability in software projects. With this goal in mind, the CRETS4DevOps method has been designed to manage requirements in DevOps environments while promoting sustainability, especially technical sustainability.

CRETS4DevOps encompasses a catalog of sustainability requirements, which helps in the reuse of expert knowledge in order to address the management of software quality attributes that result in sustainability, namely: modularity, scalability, flexibility, reusability, maintainability, reliability, security, interoperability and efficiency. This catalog is intended to assist practitioners both in software development and operation as well as in a sustainability audit. A tool to support CRETS4DevOps is currently being developed as a plug in for Microsoft Azure DevOps.

***Keywords:*** *Sustainability, Technical Sustainability, Requirements Catalog, Requirements Reuse, CRETS4DevOps.*

## 1. Introducción

La sostenibilidad se ha convertido en un concepto central para abordar desafíos globales como los relacionados con el cambio climático, incluyendo la eficiencia energética, el uso de materiales y desechos y la reducción de emisiones de CO2 [1, 2]. La sostenibilidad se define tradicionalmente como la capacidad de perdurar sin comprometer las generaciones futuras, y en muchas ocasiones se ha equiparado solamente con temas medioambientales, pero actualmente existe un consenso en que requiere la consideración simultánea de los recursos medioambientales, del bienestar social e individual, de la prosperidad económica y de la viabilidad a largo plazo de la infraestructura técnica [3]. Debido a la dependencia de la sociedad actual de la tecnología y, en particular de los sistemas intensivos en software, numerosos estudios revelan que el sector TIC (Tecnologías de la Información y las Comunicaciones) tiene un impacto nada desdeñable en el medio ambiente [4]. Por tanto, los ingenieros de software deberían tener en cuenta los requisitos de sostenibilidad desde las primeras etapas del ciclo de vida para mitigar los efectos del desarrollo y operación del software [5]. No obstante, los ingenieros de software encuentran dificultades a la hora de abordar la sostenibilidad en sus proyectos principalmente por dos motivos: (1) ausencia de estándares oficiales que la incluyan; y (2) carencia de guías que les ayuden [4]. El uso de catálogos de requisitos de sostenibilidad [6] podría ser una estrategia que contribuya a disponer de estas guías. El catálogo de requisitos recopila la experiencia de los expertos en sostenibilidad y hace explícitos los aspectos de sostenibilidad que los ingenieros tienen que tener en cuenta. El punto de partida es la definición de requisitos de sostenibilidad que sirvan para especificar el comportamiento del sistema (por ejemplo, requisitos que reduzcan el consumo de energía de un sistema), así como para influir en el comportamiento de los usuarios [7].

Para que el uso del catálogo de requisitos sea efectivo debe integrarse en un proceso de desarrollo de software. En la propuesta presentada por Hernández *et al.* [8] se describe CRETS4DevOps (*Continuous Requirements Engineering and Technical Sustainability for DevOps*), un método diseñado para gestionar los requisitos que promueven la dimensión técnica de la sostenibilidad en entornos DevOps. El propósito de esta nueva contribución es presentar el catálogo de requisitos de sostenibilidad utilizado en CRETS4DevOps, explicando su estructura y contenido. Con este catálogo se pretende proporcionar un artefacto útil que permita a los ingenieros de requisitos no solo comprender de forma sencilla los conceptos fundamentales de la sostenibilidad, sino también llevarlos a la práctica de manera efectiva. Al adoptar y seguir estos requisitos, las organizaciones software pueden contribuir de manera significativa a la sostenibilidad, asegurando su viabilidad y éxito a largo plazo en un entorno global cada vez más consciente de la importancia de la sostenibilidad.

## 2. Estado del arte

### *2.1. Requisitos de sostenibilidad en la dimensión técnica*

Existe una comprensión limitada de cómo los profesionales del software perciben la sostenibilidad y de cómo el diseño sostenible puede integrarse en el proceso de ingeniería de software, tanto en el desarrollo como en la operación del software. El diseño y la auditoría basados en la sostenibilidad como una cualidad del software siguen siendo pocos conocidos, lo que dificulta la adopción en la industria de prácticas de diseño sostenible en la ingeniería de software [5].

El análisis de requisitos debe considerarse un paso clave en cualquier proyecto de software. A través de este análisis, el ingeniero de requisitos podrá identificar, cuantificar y priorizar los requisitos de sostenibilidad, asegurando que estos se integren de manera efectiva en el diseño y desarrollo del sistema [5, 9, 10, 11]. Hernández *et al.* [12] han señalado que los enfoques actuales de IR en el contexto de DevOps no abordan explícitamente la

 

sostenibilidad. Esto se debe, en parte, a un entendimiento limitado del concepto y de la importancia de la sostenibilidad dentro de la comunidad dedicada a la IR [13], a pesar de que existen propuestas en este campo que proponen guías para su integración [14, 15].

La sostenibilidad se debe implementar desde una perspectiva integral y no sólo en el sentido del software, la tecnología verde y los requisitos funcionales para mejorar la sostenibilidad medioambiental. Esto significa considerar la sostenibilidad desde múltiples perspectivas, asegurando que los sistemas no solo sean eficientes y amables con el medio ambiente, sino también duraderos y capaces de evolucionar de acuerdo con las necesidades futuras [16]. Para diseñar requisitos de sostenibilidad técnica se deben incluir requisitos de no obsolescencia, así como las características de calidad tradicionales de mantenibilidad, soportabilidad, fiabilidad y portabilidad, que conducen a la longevidad de un sistema. De acuerdo con Karita *et al.* [5] los requisitos de calidad del software soportan el desarrollo de software sostenible y se corresponden con requisitos de la dimensión técnica. No obstante, no existe un consenso sobre si la sostenibilidad debe considerarse un atributo de calidad del producto software o no: de hecho, actualmente no forma parte de estándares de calidad como el ISO 25010, aunque algunos autores como Calero *et al.* [17] han propuesto una ampliación del estándar para incluirla. Además, la eficiencia, especialmente la eficiencia energética y la suficiencia del hardware, también deben formar parte de los requisitos de sostenibilidad técnica [18]. Otros aspectos importantes a tener en cuenta en la dimensión técnica son la continuidad, durabilidad, seguridad, usabilidad, escalabilidad, buenas prácticas, evolución del sistema, optimización y uso de código abierto.

### *2.2. CRETS4DevOps*

CRETS4DevOps [8] es un método para la gestión de requisitos continua enfocado a promover la dimensión técnica de la sostenibilidad en entornos DevOps. Los elementos que constituyen CRETS4DevOps se pueden integrar en diferentes métodos de desarrollo ágil, como por ejemplo Scrum. CRETS4DevOps propone combinar las prácticas de IR (las tradicionales y las ágiles) para paliar: (1) el problema de requisitos no explícitos en las especificaciones, al confiar en la comunicación directa entre los interesados (*stakeholders*), en lugar de apoyarse en la documentación [19]; y (2) las limitaciones de las historias de usuario para la especificación de requisitos [20]. CRETS4DevOps promueve la retroalimentación e integración continua de requisitos, otorgando a los interesados la capacidad de analizar y evaluar el progreso del proyecto en las diferentes etapas de su desarrollo alineado a los requisitos.

Siguiendo la analogía de la filosofía de DevOps, poniendo de relieve el enfoque de continuidad, las actividades de CRETS4DevOps se representan en un bucle infinito de elicitación, análisis, especificación, validación, integración continua de requisitos, comunicación continua, retroalimentación continua de requisitos y gestión continua de requisitos. Un aspecto importante en esta propuesta es la reutilización de requisitos, en concreto de los requisitos de sostenibilidad de la dimensión técnica. El soporte a la reutilización se propone mediante el uso de catálogos de requisitos reutilizables de sostenibilidad. Dado que los requisitos se suelen documentar en lenguaje natural en la industria [21], nuestra propuesta antepone esta aproximación en lugar de otras basadas en un lenguaje de dominio, ontologías o modelos iStar. El uso de catálogos de requisitos reutilizables de sostenibilidad permite cubrir la ausencia de expertos en sostenibilidad en los equipos de desarrollo [9]. La integración continua de requisitos a lo largo del proceso, asistida por el ingeniero de requisitos, permite mantener en todo momento la visión global del sistema [8]. Para que la aplicación de CRETS4DevOps sea efectiva requiere soporte automatizado. Por ello, se está desarrollando un *plug* in en la aplicación Azure DevOps (Figura 1), que permita la gestión y explotación del catálogo de requisitos de sostenibilidad.

## 3. Materiales y métodos

La metodología que se ha seguido en esta investigación ha consistido en sucesivas revisiones de la literatura sobre Green IT, cada vez más específicas, de la siguiente manera: (1) La primera revisión de la literatura tuvo como objetivo identificar y conocer las áreas en las que se pueden aplicar los conceptos de sostenibilidad a las TIC, siguiendo las dimensiones del Manifiesto de Kalskrona [22], y se decidió focalizar la investigación en torno a la sostenibilidad técnica, que está íntimamente relacionada con los contextos DevOps de desarrollo y operación del software en los que se centra el método CRETS4DevOps; (2) En una segunda etapa se decidió utilizar los atributos de calidad del software para estructurar en categorías el catálogo de sostenibilidad técnica, en línea con Moreira

et al. [6]; (3) En una tercera fase se realizó una revisión más profunda de la literatura para identificar las principales fuentes para cada categoría de la estructura del catálogo.

El punto de entrada para la primera revisión de literatura fue el trabajo *Introduction to Green in Software Engineering* de Calero y Piattini [23]. Otros trabajos utilizados como punto de entrada fueron los estudios de caso sobre Green IT desarrollados por Penzenstadler y Venters [24] y por Capra *et al.* [25].

Los requisitos se desarrollaron utilizando el modelo de referencia de requisitos de la metodología CREST4DEVOPS. La herramienta utilizada para elaborar el documento de requisitos ha sido Microsoft Word 365, mientras que Mendeley fue la herramienta utilizada para gestionar la información bibliográfica. El catálogo se especificó utilizando Azure DevOps en tanto se desarrolla un plugin para CRESTS4DEVOPS.

## 4. Resultados

Como resultado se propone un catálogo de requisitos de sostenibilidad que busca contribuir al desarrollo de software sostenible proporcionando guías a los equipos de desarrollo en la dimensión técnica de la sostenibilidad. Las pautas y buenas prácticas recogidas en el catálogo pueden servir de guía para la sostenibilidad en el desarrollo y operación del software y para la auditoría de sostenibilidad, ayudando a que el producto o servicio software perdure de manera sostenible en el tiempo. Nótese que no es obligatorio aplicar todos los requisitos definidos en el catálogo: la selección de requisitos dependerá del proyecto a desarrollar.

El catálogo de requisitos de sostenibilidad está estructurado en categorías que se corresponden con los atributos de calidad del producto software, de la siguiente forma:

1) Modularidad: se refiere a la capacidad de un sistema a ser diseñado por módulos que sean independientes entre sí, lo que a menudo proporciona flexibilidad y posibilidad de utilizarlo de diversas maneras en el proyecto en desarrollo con un impacto mínimo en otros componentes [6, 26].
2) Escalabilidad: implica la capacidad de gestionar un mayor volumen de datos o usuarios a medida que el proyecto en desarrollo crece y se expande [27].
3) Flexibilidad: se refiere a la capacidad de adaptarse a cambios en los requisitos del negocio o del usuario sin necesidad de reescribir el código del proyecto en desarrollo [26, 28].
4) Reusabilidad: el sistema debe ser diseñado por componentes para ser utilizados en otros proyectos [26, 27].
5) Mantenibilidad: se refiere a cómo mantener el proyecto en el tiempo y actualizarlo a medida que surjan nuevos requisitos [6, 29].
6) Fiabilidad: se relaciona con la capacidad del sistema para funcionar sin errores durante un periodo de tiempo, y a la recuperación rápida en caso de fallos [6, 27, 30].
7) Seguridad: este atributo de calidad es de crucial importancia, ya que garantiza que el producto final no solo ofrezca seguridad, sino que también resguarde de manera proactiva la información personal y los datos sensibles del usuario contra cualquier amenaza potencial. La seguridad sostiene que deben integrar medidas de seguridad robustas para proteger la integridad, confidencialidad y disponibilidad de la información [6, 29].
8) Interoperabilidad: garantiza que el sistema desarrollado debe ser capaz de trabajar con otros sistemas y tecnologías [6, 26].
9) Eficiencia: se centra en el usuario es la capacidad de realizar tareas en un tiempo razonable y con un consumo de recursos aceptable [26, 30].

Todas las categorías del catálogo tienen una estructura escalonada, donde existen requisitos hijo que se refinan a partir de requisitos padre. En algunos casos es necesario un atributo Justificación para eliminar cualquier incertidumbre que pueda surgir durante la lectura del requisito. Independientemente del formato que tenga la categoría, todo requisito tiene un atributo Discusión, en el cual se justifica su relación (contradictoria o no) con otros requisitos o con aspectos que puedan estar relacionados con la sostenibilidad, usabilidad, accesibilidad, etc. En la Figura 1 se muestra el catálogo de requisitos de sostenibilidad en la aplicación Azure DevOps, a través del *plug* in en desarrollo. El contenido del catálogo se detalla a continuación, una categoría por cada atributo de calidad.


**Figura 1.** Soporte automatizado mediante un *plug* in para Microsoft Azure.

### *4.1. Modularidad*

Los requisitos contenidos en esta categoría describen un diseño modular del servicio o infraestructura que permite la ampliación o reducción del sistema de manera eficiente, adaptándose a las necesidades cambiantes sin requerir una reestructuración completa. La modularidad permite la reutilización de componentes en diferentes sistemas o proyectos, optimizando los recursos, lo cual es fundamental para la sostenibilidad al minimizar la necesidad de producir nuevos componentes y perdurar en el tiempo. Permite actualizaciones específicas en lugar de reemplazar sistemas completos, se pueden reducir significativamente los costos a lo largo del ciclo de vida del producto, dando cabida a sistemas flexibles, eficientes y fáciles de mantener, prolongando su vida útil y reduciendo el impacto ambiental.

#### 4.1.1. Requisitos de modularidad

- **Mod.1.** Compatibilidad y conectividad.
- **Mod.1.1.** El software debe utilizar APIs (Interfaces de Programación de Aplicaciones) y protocolos de comunicación estandarizados que aseguren la interoperabilidad con otras aplicaciones. Por ejemplo, RESTful APIs, SOAP, gRPC, y protocolos de trasferencia de datos como HTTP o HTTPS.
- **Mod.1.2.** Utilizar formatos de datos comunes y ampliamente aceptados (JSON, XML, CSV) para asegurar que los datos puedan ser fácilmente intercambiados y comprendidos por otros sistemas.
- **Mod.1.3.** Realizar pruebas de regresión para asegurar que las nuevas versiones del software no introduzcan errores ni problemas de compatibilidad con versiones anteriores. Herramientas de integración continua (CI) como Jenkins, CircleCI, o GitHub Actions.
- **Mod1.4.** Implementar herramientas de monitoreo de red como Nagios y Zabbix para supervisar la conectividad y el rendimiento de las comunicaciones entre sistemas.
- **Mod1.5.** Gestionar las conexiones activas y los recursos de red para asegurar una operación eficiente. Esto incluye el uso de patrones como la conexión persistente y el manejo de conexiones en espera para mejorar el rendimiento.
- **Mod.2.** Integración y Actualización.
- **Mod2.1.** Crear manuales y guías de integración que expliquen claramente cómo agregar nuevos componentes y servicios al sistema existente.
- **Mod2.2.** Dividir el sistema en microservicios independientes que se comuniquen entre sí mediante APIs. Esto permite desarrollar, desplegar y escalar cada microservicio de manera autónoma.

- **Mod2.3.** Implementar sistemas de control de configuración que permitan gestionar y versionar los cambios en la infraestructura y el software, facilitando las actualizaciones y revertiendo cambios si es necesario.
- **Mod2.4.** Implementar pruebas de regresión para asegurar que las actualizaciones no introduzcan errores ni afecten funcionalidades existentes.
- **Mod2.5.** Configurar pipelines de CI/CD usando herramientas como Jenkins, GitLab CI, o GitHub Actions.

### *4.2. Escalabilidad*

La escalabilidad es el grado en que el software puede adaptarse al crecimiento horizontal, es decir la capacidad de distribuir o expandir a más servidores o nodos para balancear la carga de los procesos dentro de un sistema, mientras que el crecimiento vertical implica mejorar la capacidad de un solo servidor o sistema para manejar más carga de trabajo, lo cual se logra mediante la adición de recursos como CPU, memoria y almacenamiento a un servidor existente [27].

#### 4.2.1. Requisitos de escalabilidad

- **Esc.1.** Proceso de Diseño e Implementación.
- **Esc.1.1.** Los programadores deben escribir algoritmos eficientes.
- **Esc.1.2.** La arquitectura del software debe tener el menor número de capas posible.
    - **Justificación:** El uso de librerías dinámicas, módulos independientes y cualquier otro sistema software que añada capas a la arquitectura del sistema final, aumenta la complejidad y, por extensión, la entropía del software.
- **Esc.2.** Proceso de análisis *green*.
- **Esc.2.1.** Se deberá utilizar hardware de alta eficiencia energética que minimice el uso de energía a medida que aumente su capacidad.
- **Esc.2.2.** Implementar políticas de gestión de energía, como el uso de modos de bajo consumo y el ajuste de la frecuencia del CPU.
- **Esc.3.** Distribución de carga.
- **Esc.3.1.** Usar técnicas de balanceo de cargas para distribuir equitativamente los procesos entre varios servidores, optimizando el uso de recursos.
- **Esc.3.2.** Implementar una arquitectura de microservicios que permita escalar diferentes partes de la aplicación de manera independiente.
- **Esc.3.3.** Implementar sistemas de monitoreo continuo y análisis de escenarios para anticipar la demanda y ajustar dinámicamente la capacidad, evitando una saturación en el sistema.

### *4.3. Flexibilidad*

La flexibilidad se puede utilizar en contextos de la especificación inicial de requisitos, permitiendo ajustes y refinamientos conforme al avance del proyecto. Esto es especialmente útil en entornos ágiles, donde los requisitos pueden evolucionar rápidamente en respuesta a la retroalimentación de los interesados.

#### 4.3.1. Requisitos de flexibilidad

- **Fle.1.** Metodologías agiles.
- **Fle.1.1.** Crear y mantener una biblioteca con componentes reutilizables que pueden integrarse en diferentes proyectos, promoviendo la eficiencia y la sostenibilidad.
- **Fle.1.2.** Implementar metodologías ágiles como Scrum que permitan iteraciones cortas con revisiones continuas de requisitos.
- **Fle.1.3.** Establecer procesos de control de cambios que incluyan la evaluación del impacto de los cambios en los requisitos.

- **Fle.1.4.** Utilizar técnicas que permitan priorizar de forma dinámica los requisitos y que permitan identificar requisitos críticos en función de los cambios.

*4.4. Reusabilidad*

La reusabilidad es el grado en el que el software puede reutilizarse en otras aplicaciones sin necesidad de modificar su escritura, permitiendo una mayor eficiencia en el desarrollo de nuevos proyectos, lo que permite reducir costos, minimizar errores y acelerar el tiempo en el desarrollo.

4.4.1. Requisitos de reusabilidad

- **Reu.1.** Proceso de Eliminación.
- **Reu.1.1.** Cuando se elimina un producto o servicio, se debe guardar el código para reusarlo en futuros proyectos.
    - **Justificación:** Esto minimizará los costes de desarrollo.
    - **Discusión:** Reusar el código para otros proyectos no se debe hacer solamente en este proceso, pues también es posible (y recomendable) hacerlo durante el desarrollo y despliegue.
- **Reu.2.** Biblioteca de componentes.
- **Reu.2.1.** Desarrollar una biblioteca de componentes en una herramienta tecnológica como GitHub que puedan ser reutilizados en múltiples aplicaciones y proyectos.
- **Reu.2.2.** Crear estándares para la creación y uso de componentes reutilizables para asegurar su consistencia.
- **Reu.2.3.** Crear y mantener una documentación de los componentes desarrollados para su reutilización.
- **Reu.3.** Uso de plantillas.
- **Reu.3.1.** Crear y utilizar plantillas que puedan ser adaptadas a diferentes proyectos.
- **Reu.3.2.** Desarrollar o adaptar *frameworks* que faciliten la construcción de aplicaciones reutilizables.
- **Reu.4.** Control de cambios.
- **Reu.4.1.** Crear un sistema de versionado para los componentes reutilizables, permitiendo mejoras y correcciones a errores encontrados que permitan su fácil integración.
- **Reu.4.2.** Mantener actualizado el historial de cambios.

*4.5. Mantenibilidad*

Es importante garantizar el mantenimiento de un sistema a lo largo de su ciclo de vida, permitiendo que se realicen actualizaciones, correcciones de errores y mejoras de manera eficientes. Un sistema mantenible es fácil de comprender y modificar, lo que reduce el tiempo de mantenimiento. Esto permite prolongar la vida útil de un sistema, además de mejorar su rendimiento y confiabilidad.

4.5.1. Requisitos de mantenibilidad.

- **Man.1.** Proceso de Mantenimiento.
- **Man.1.1.** El código desarrollado debe ser legible.
    - **Justificación:** Si un programa es fácil de entender, será más rápido, fácil y energéticamente eficiente hacer cambios en él.
- **Man.1.2.** La documentación del código debe estar siempre disponible, además de ser coherente con el propio código.
- **Man.1.3.** Se deben crear actualizaciones de seguridad, para evitar posibles fallos del sistema software.
- **Man.2.** Hacer uso de programas y entornos de desarrollo *open source*.
    - **Justificación:** Este tipo de programas no son desarrollados por compañías que solo buscan un beneficio, sino que son mantenidos, en la mayoría de los casos, por una comunidad de desarrolladores que buscan eliminar los errores que aparecen, para mejorar el producto. La

diferencia radica en que las compañías buscarán que se compren sus nuevas versiones, teniendo en mente siempre generar un producto nuevo que mantener.

- **Man.3.** Cambio Divergente.
- **Man.3.1.** Aplicar el *refactoring* "Extraer clase" o "Extraer superclase" cuando haya que aplicar muchos cambios a una sola clase para introducir una nueva funcionalidad.
    - **Justificación:** Mejora la organización del código, reduciendo el código duplicado y simplificando su mantenimiento.
- **Man.4.** *Shotgun Surgery*.
- **Man.4.1.** Aplicar el *refactoring* "Mover campo" y "Mover método" a una clase apropiada (creada a tal efecto si fuera necesario) cuando un pequeño cambio en una clase implica modificaciones en otras clases y métodos.
    - **Justificación:** Mejora la organización del código, reduciendo el código duplicado y simplificando su mantenimiento.
- **Man.5.** *Feature Envy.*
- **Man.5.1.** Aplicar el *refactoring* "Mover método" si es posible, o "Extraer método", si solo una parte del código accede a los datos de la otra clase, cuando un método tiene más interés en otra clase que en la suya propia: lee atributos, datos, etc.
    - **Justificación:** Hay menos código duplicado y la organización del código mejora, al aproximar los métodos que manejan los datos de aquellos que los requieren.
- **Man.6.** Obsesión Primitiva.
- **Man.6.1.** Evitar el uso de tipos de datos primitivos para representar estados, objetos o almacenar información.
- **Man.6.1.1.** Implementar clases que agrupen los tipos de datos primitivos de manera que sea un objeto el que represente la información.
- **Man.6.1.2.** Si las variables codifican datos complejos, intentar reemplazarlos por clases, subclases o haciendo uso de los patrones Estado/Estrategia.
    - **Justificación:** El código es más flexible y aumenta su comprensibilidad.
- **Man.7.** Clase de información.
- **Man.7.1.** Los atributos de una clase de información (solo tiene atributos y métodos *get/set*) solo deben ser modificados utilizando los métodos ofrecidos por la clase.
    - **Justificación:** Mejora la comprensibilidad del código.

### *4.6. Fiabilidad*

Conjunto de atributos que influyen en la capacidad del software para mantener su nivel de rendimiento en las condiciones establecidas durante un periodo de tiempo determinado. Esto incluye la capacidad del software para funcionar correctamente y de forma consistente sin fallos, garantizando que puede manejar las cargas de trabajo.

#### 4.6.1. Requisitos de fiabilidad

- **Fia.1.** Se debe hacer uso de las reglas F.I.R.S.T. en los *tests* que se desarrollen.
- **Fia.1.1.** Rapidez: Los *tests* deben ser rápidos.
- **Fia.1.1.1.** Si no pudieran ejecutarse en un tiempo aceptable, deberán programarse para ser ejecutados con menos frecuencia.
- **Discusión:** Los *tests* se deben ejecutar lo máximo posible, siempre y cuando su ejecución no genere un consumo mayor de los recursos y, por ende, un mayor gasto de energía. Si los *tests* requieren muchos recursos para ser ejecutados, hay que programarlos para que se ejecuten con menor frecuencia. De esta manera, se busca un equilibrio entre su ejecución y el consumo de energía.
- **Fia.1.2.** Independencia: Un *test* no debe depender de la ejecución de otro *test*.
- **Fia.1.3.** Repetición: Se deben programar los *tests* para ejecutarse en cualquier entorno y en cualquier momento.

- **Fia.1.4.** Validación automática: Los *tests* deben devolver solamente una salida de tipo booleano: acierto o fallo.
- **Fia.1.5.** Puntualidad: Los *tests* deben ser escritos justo antes del código de producción que los cumplirá.
- **Fia.2.** Proceso de Reciclabilidad.
- **Fia.2.1.** Si el producto o servicio deja de funcionar, deben existir mecanismos que ayuden al usuario a su reparación.
- **Fia.2.2.** Se debe implementar un mecanismo de recolección de opiniones de los usuarios sobre la durabilidad del producto, basándose en encuestas.
- **Fia.3.** Cuando exista un proceso ejecutándose y requiera un tiempo entre 0,2-1 segundos, se debe implantar una pantalla de carga con un mensaje de aviso de espera.
- **Fia.4.** Fuentes y Tipografía.
- **Fia.4.1.** Usa un máximo de dos fuentes distintas en tu servicio.
- **Fia.4.2.** Haz uso de tipos de letra óptimos para la pantalla y no para su impresión, como Georgia o Verdana.
- **Fia.4.3.** Haz uso de un máximo de 3 tamaños distintos para las fuentes de texto.

*4.7. Seguridad*

Es un requisito fundamental, ya que garantiza que los sistemas y datos estén protegidos contra accesos no autorizados, vulnerabilidades y amenazas externas e internas. Se debe garantizar la confidencialidad, integridad y disponibilidad de la información junto con los recursos tecnológicos.

4.7.1. Requisitos de seguridad

- **Seg.1.** Credenciales.
- **Seg.1.1.** Los usuarios deberán ingresar con un nombre de usuario y contraseña única y segura.
- **Seg.1.2.** Las contraseñas deben cumplir con los requisitos de complejidad (mínimo de caracteres, combinación de letras, números y caracteres especiales), y deben ser cambiadas periódicamente.
- **Seg.1.3.** Cuando se usan aplicaciones móviles se deberá tener autentificación de dos pasos (a través de un mensaje sms o correo electrónico).
- **Seg.1.4.** Todos los dispositivos móviles utilizados deberán ser autorizados y registrados por el administrador del sistema.
- **Seg.1.5.** Realizar capacitación sobre prácticas para proteger sus credenciales.
- **Seg.1.6.** Realizar auditorías continuas sobre accesos al software y servidores.
- **Seg.2.** Se debe aplicar el Principio de Ocultación de la Información: Los detalles de implementación de un módulo deben estar ocultos a otros módulos y sólo se debe acceder a través de la interfaz pública de ese módulo.
- **Seg.3.** Si la aplicación o servicio se va a desarrollar siguiendo el paradigma de Programación Orientada a Objetos, se debe aplicar la Ley de Demeter: Un método f de la clase C solo debería llamar a métodos de C, de un objeto creado (o pasado como argumento) por f y de una variable de instancia de C.
- **Seg.4.** Validación de datos.
- **Seg.4.1.** Implementar mecanismos de validación y control de datos de entrada para asegurar que sean correctos y no contengan alguna inconsistencia.
- **Seg.4.2.** Utilizar firma digital para verificar la integridad de los datos y asegurar que no han sido modificados sin autorización.
- **Seg.4.3.** Asegurar que los datos estén cifrados desde la transferencia y almacenamiento, protegidos de accesos no autorizados y robo de información.
- **Seg.4.4.** Utilizar algoritmos de cifrado fuertes, como AES-256, para garantizar la protección de la información crítica.

*4.8. Interoperabilidad*

La interoperabilidad garantiza que el sistema en desarrollo sea capaz de trabajar con otros sistemas y tecnologías de manera eficiente, sin problemas, facilitando la integración. A su vez permite al sistema evolucionar y adaptarse a medida que los requisitos del negocio y las tecnologías cambian.

4.8.1. Requisitos de interoperabilidad

- **Int.1.** Estándares abiertos.
- **Int.1.1.** Implementar estándares abiertos como HTTP, HTTPS, REST, SOAP, JSON, XML, y otros protocolos que permitan la compatibilidad entre sistemas diferentes.
- **Int.1.2.** Utilizar formatos de datos estandarizados, como JSON, XML o CSV, para facilitar el intercambio de información entre sistemas dispares.
- **Int.1.3.** Desarrollar APIs definidas y documentadas que permitan a otros sistemas interactuar con el software de manera clara.
- **Int.2.** Monitoreo continuo.
- **Int.2.1.** Implementar sistemas de monitoreo continuo que supervisen la interoperabilidad y el rendimiento de los sistemas, detectando problemas.
- **Int.2.2.** Configurar alertas y notificaciones para informar a los administradores sobre cualquier problema de interoperabilidad.
- **Int.2.3.** Ofrecer soporte técnico para ayudar en la resolución de problemas de interoperabilidad y para facilitar la integración continua de nuevos sistemas.

*4.9. Eficiencia*

Se centra en la capacidad de realizar tareas en un tiempo razonable y con un consumo de recursos aceptable, garantizando una experiencia de usuario eficiente y satisfactoria. Esto implica que el software debe estar optimizado para responder rápidamente a las interacciones del usuario, minimizando los tiempos de espera y evitando el uso excesivo de recursos del sistema.

4.9.1. Requisitos de eficiencia

- **Efi.1.** *Frameworks.*
- **Efi.1.1.** Se deben crear distintos perfiles o modos para ahorrar energía, apagando o hibernando aplicaciones que no estén siendo usadas durante un periodo de inactividad.
- **Efi.2.** Computación *Green.*
- **Efi.2.1.** Si el sistema necesita activar dispositivos externos o periféricos, se debe programar de manera que se activen solo cuando vayan a ser requeridos.
    - **Justificación:** Así evitaremos un uso abusivo de los mismos, y reduciremos el coste energético de mantenerlos activos, pero en desuso.
- **Efi.3.** Métricas del rendimiento.
- **Efi.3.1.** Se debe usar la información de frecuencia de uso de las CPUs para cuantificar la disipación de la energía que se produce durante el desarrollo.
- **Efi.4.** Se debe implantar un mecanismo de apagado o hibernación de los sistemas de desarrollo tras un periodo de inactividad.
- **Efi.5.** Si no fuese posible dicho sistema de apagado o hibernación, se debe utilizar en su caso un salvapantallas de color blanco, sin texto ni gráficos, en los sistemas que usen la tecnología de visualización basada en rayos catódicos o LCD.
    - **Justificación:** Los salvapantallas sin texto ni gráficos generan menos conexiones con la GPU, reduciendo el gasto energético.
- **Efi.6.** Si no fuese posible dicho sistema de apagado o hibernación, se debe utilizar en su caso un salvapantallas de color negro, sin texto ni gráficos, en los sistemas que usen la tecnología OLED.

    - **Justificación:** En la tecnología OLED, cada píxel genera su propia luz, por lo que, si el color que se busca representar es el negro, no se activará ningún píxel.
- **Efi.7.** Si los monitores de los usuarios de la aplicación usan la tecnología de rayos catódicos o LCD, se debe usar el color blanco para representar las pantallas de carga.
- **Efi.8.** Si los monitores de los usuarios de la aplicación usan la tecnología OLED, se debe usar el color negro para representar las pantallas de carga.
- **Efi.9.** Representación de imágenes.
- **Efi.9.1.** Si es necesario representar un diagrama o un gráfico plano que no necesite escalarse, se debe usar el formato PNG o GIF.
- **Efi.9.2.** Se debe usar el formato JPG para representar fotos.
- **Efi.9.3.** Se debe hacer uso del formato SVG para mantener la nitidez en aquellos gráficos que permitan escalado.
- **Efi.9.4.** Se deben usar *Sprites* CSS en vez de imágenes JPG para mostrar botones de navegación e imágenes para iconos.
    - **Justificación:** Los navegadores cachean las imágenes CSS (tales como flechas de navegación hacia atrás o hacia delante, icono clásico de página de inicio, iconos para guardar, imprimir, etc.). De esta forma se eliminan las peticiones HTTP que debería hacer el navegador en caso de usar imágenes alojadas en un servidor.

## 5. Conclusiones

CRETS4DevOps es un método en curso diseñado para gestionar requisitos de sostenibilidad en la dimensión técnica, combinando prácticas tradicionales y ágiles de IR en el marco de la IR continua. Este enfoque se apoya en la reutilización de requisitos para fomentar la sostenibilidad técnica, con lo cual surge la necesidad de un catálogo reutilizable de requisitos para la sostenibilidad técnica, que puede ayudar en la integración de la sostenibilidad en proyectos de software aún en ausencia de un experto en sostenibilidad. El catálogo permite potenciar la innovación y la capacidad de otorgar una ventaja en el desarrollo de aplicaciones que promuevan la sostenibilidad, al contar con una estructura clara de requisitos que promueve la toma de decisiones consensuadas. Actualmente se sigue con los trabajos para finalizar el desarrollo de la aplicación en Azure DevOps que soporte el catálogo de requisitos de sostenibilidad. CRETS4DevOps se pretende validar en un caso real en la industria o, en su defecto, mediante una encuesta a expertos en DevOps.

## 6. Agradecimientos

Esta investigación forma parte del proyecto OASSIS-UMU (PID2021-122554OB-C32) y de la Red de Excelencia en Calidad y Sostenibilidad del Software (RED2022-134656-T), ambos financiados por el Ministerio de Ciencia, Innovación y Universidades y la Agencia Estatal de Investigación de España (/10.13039/501100011033/) y por el Fondo Europeo de Desarrollo Regional (FEDER), *A way to make Europe*.

## 7. Referencias

[1] Penzenstadler, B. (2013). *Towards a definition of sustainability in and for software engineering*. ACM Symposium on Applied Computing, Coimbra, Portugal. https://doi.org/10.1145/2480362.2480585

[2] Silveira, C., Santos, V., Reis, L., Mamede, H. (2022). CRESustain: Approach to Include Sustainability and Creativity in Requirements Engineering. *Journal of Engineering Research and Sciences, 1* (8), 27–34. https://doi.org/10.55708/js0108004

[3] Becker, C., Betz, S., Chitchyan, R., Duboc, L. (2016). Requirements: The key to sustainability. *IEEE Software, 33* (1), 56–65. https://doi.org/10.1109/MS.2015.158

[4] Noman, H., Mahoto, N., Bhatti, S., Rajab, A., Shaikh, A. (2024). Towards sustainable software systems: A software sustainability analysis framework. *Information and Software Technology, 169*, 107411. https://doi.org/10.1016/J.INFSOF.2024.107411

 

[5] Karita, L., Mourão, B. C., Machado, I. (2022). *Towards a common understanding of sustainable software development*. XXXVI Brazilian Symposium on Software Engineering (SBES), Virtual Event Brazil. https://doi.org/10.1145/3555228.3555236

[6] Moreira, A., Araújo, J., Gralha, C., Goulão, M., Brito, I. S., Albuquerque, D. (2023). A social and technical sustainability requirements catalogue. *Data & Knowledge Engineering*, *143*, 1-16. https://doi.org/10.1016/j.datak.2022.102107

[7] Venters, C. C., Seyff, N., Becker, C., Betz, S., Chitchyan, R., Duboc, L., Mcintyre, D., Penzenstadler, B. (2017). *Characterising sustainability requirements: A new species red herring or just an odd fish?* IEEE/ACM 39th International Conference on Software Engineering in Society Track (ICSE-SEIS). Buenos Aires, Argentina. https://doi.org/10.1109/ICSE-SEIS.2017.2

[8] Hernández Alarcón, R. F., Moros Valle, B., Nicolás Ros, J. (2024). *Método de gestión de requisitos para promover la sostenibilidad en DevOps: CRETS4DevOps*. XXVII Ibero-American Conference on Software Engineering (CIbSE). Curitiba, Brasil.

[9] Mahaux, M., Heymans, P., Saval, G. (2011). *Discovering sustainability requirements: An experience report*. 17th International Working Conference (REFSQ). Essen Germany. https://doi.org/10.1007/978-3-642-19858-8_3

[10] Alsaqaf, W., Daneva, M., Wieringa, R. (2019). Quality requirements challenges in the context of large-scale distributed agile: An empirical study. *Information and Software Technology., 110* (2019), 39–55. https://doi.org/10.1016/j.infsof.2019.01.009

[11] Penzenstadler, B., Femmer, H. (2013). *A Generic Model for Sustainability with Process- and Product-specific Instances*. Workshop on Green in Software Engineering and Green by Software Engineering. Fukuoka, Japan. https://doi.org/10.1145/2451605.2451609

[12] Hernández, R., Moros, B., Nicolás, J. (2023). Requirements management in DevOps environments: a multivocal mapping study. *Requirements Engineering, 28*, 317-346. https://doi.org/10.1007/s00766-023-00396-w

[13] Seyff, N., Betz, S., Groher, I., Stade, M., Chitchyan, R., Duboc, L., Penzenstadler, B. (2018). *Crowd-focused semi-automated requirements engineering for evolution towards sustainability*. IEEE 26th International Requirements Engineering Conference (RE). Banff, Canada. https://doi.org/10.1109/RE.2018.00-23

[14] Dewi Saputri, T. R., Lee, S. W. (2021). Integrated framework for incorporating sustainability design in software engineering life-cycle: An empirical study. *Information Software Technology, 129*. https://doi.org/10.1016/j.infsof.2020.106407

[15] Oyedeji, S., Seffah, A., Penzenstadler, B. (2018). A catalogue supporting software sustainability design. *Sustainability, 10* (7), 1-30. https://doi.org/10.3390/su10072296

[16] Christoph, B. (2014). *Sustainability and longevity: Two sides of the same quality?* Third International Workshop on Requirements Engineering for Sustainable Systems. Karlskrona, Sweden. https://ceur-ws.org/Vol-1216/paper1.pdf

[17] Calero, C., Moraga, M. Á., Bertoa, M. F. (2013). *Towards a Software Product Sustainability Model*. First Workshop on Sustainable Software for Science: Practices and Experiences. Denver, CO, USA. https://alarcos.esi.uclm.es/ALARNET2/FILES/Congresos/2013-VSSSPE-Calero.pdf

[18] Raturi, A., Penzenstadler, B., Tomlinson, B., Richardson, D. (2014). *Developing a sustainability non-functional requirements framework*. 3rd International Workshop Green Sustainable Software. Hyderabad, India. https://doi.org/10.1145/2593743.2593744

[19] Moreira, A., Schneider, K. (2022). Editorial. *Requirements Engineering, 27* (4), 403–404. https://doi.org/10.1007/s00766-022-00392-6

[20] Kasauli, R., Knauss, E., Horkoff, J., Liebel, G., de Oliveira Neto, F. G. (2021). Requirements engineering challenges and practices in large-scale agile system development. *Journal of Systems and Software, 172*, 1-26. https://doi.org/10.1016/j.jss.2020.110851

[21] Franch, X., Palomares, C., Quer, C., Chatzipetrou, P., Gorschek, T. (2023). The state-of-practice in requirements specification: an extended interview study at 12 companies. *Requirements Engineering, 28* (3), 377–409. https://doi.org/10.1007/s00766-023-00399-7

[22] Becker, C., Chitchyan, R., Duboc, L., Easterbrook, S., Penzenstadler, B., Seyff, N., Venters, C. C. (2015). *Sustainability Design and Software: The Karlskrona Manifesto*. IEEE/ACM 37th IEEE International Conference on Software Engineering. Florence, Italy. https://doi.org/10.1109/ICSE.2015.179

 

[23]Calero, C., Piattini, M. (2015). Introduction to green in software engineering. En C. Calero, M. Piattini (Eds.) *Software Engineering* (pp. 3–27). Springer International Publishing. https://doi.org/10.1007/978-3-319-08581-4_1

[24]Penzenstadler, B., Duboc, L., Venters, C. C., Betz, S., Seyff, N., Wnuk, K., Chitchyan, R., Easterbrook, S. M., Becker, C. (2017). *Software engineering for sustainability*. Digital Technology and Sustainability: Engaging the Paradox.

[25]Capra, E., Francalanci, C., Slaughter, S. A. (2012). Is software "green"? Application development environments and energy efficiency in open source applications. *Information and Software Technology, 54* (1), 60–71. https://doi.org/10.1016/j.infsof.2011.07.005

[26]Condori-Fernandez, N., Lago, P. (2018). Characterizing the contribution of quality requirements to software sustainability. *Journal of Systems and Software, 137*, 289–305. https://doi.org/10.1016/j.jss.2017.12.005

[27]Venters, C. C., Jay, C., Lau, L. M. S., Griffiths, M. K., Holmes, V., Ward, R. R., Austin, J., Dibsdale, C. E., Xu, J. (2014). *Software sustainability: The modern tower of babel*. Third International Workshop on Requirements Engineering for Sustainable Systems. Karlskrona, Sweden. https://ceurspt.wikidata.dbis.rwth-aachen.de/Vol-1216/paper2.html

[28]Condori-Fernandez, N., Lago, P., Luaces, M. R., Places, Á. S. (2020). An action research for improving the sustainability assessment framework instruments. *Sustainability, 12* (4), 1–25. https://doi.org/10.3390/su12041682

[29]Condori-Fernandez, N., Lago, P. (2019). *Towards a software sustainability-quality model: Insights from a multi-case study*. 13th International Conference on Research Challenges in Information Science (RCIS). Brussels, Belgium. https://doi.org/10.1109/RCIS.2019.8877084

[30]Koçak, S. A., Alptekin, G. I., Bener, A. B. (2014). *Evaluation of software product quality attributes and environmental attributes using ANP decision framework*. Third International Workshop on Requirements Engineering for Sustainable Systems (RE4SuSy). Karlskrona, Sweden. https://ceur-ws.org/Vol-1216/paper7.pdf