EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Tecnológico Nacional de México

Centro Nacional de Investigación y Desarrollo Tecnológico

Tesis de Doctorado

COREOGRAFÍA DE MICROSERVICIOS DE REFACTORIZACIÓN DE SOFTWARE LEGADO

presentada por

M.C. Nelida Barón Pérez

como requisito para la obtención del grado de

Doctora en Ciencias de la Computación

Director de tesis

Dr. René Santaolaya Salgado

Codirectora de tesis

Dra. Blanca Dina Valenzuela Robles

Cuernavaca, Morelos, México. Agosto de 2026.

cenidet
Centro Nacional de Investigación y Desarrollo Tecnológico

cenidet Centro Nacional de Investigación y Desarrollo Tecnológico	ACEPTACIÓN DE IMPRESIÓN DEL DOCUMENTO DE TESIS DOCTORAL	Código: CENIDET-AC-006-D20
		Revisión: O
	Referencia a la Norma ISO 9001:2008 7.1, 7.2.1, 7.5.1, 7.6, 8.1, 8.2.4	Página 1 de 1

Cuernavaca, Mor., a 16 de junio de 2026

DR. CARLOS MANUEL ASTORGA ZARAGOZA
SUBDIRECTOR ACADÉMICO
P R E S E N T E

AT'n: DRA. ANDREA MAGADÁN SALAZAR
PRESIDENTA DEL CLAUSTRO DOCTORAL DEL
DEPARTAMENTO DE CIENCIAS COMPUTACIONALES

Los abajo firmantes, miembros del Comité Tutorial del estudiante **M.C. NELIDA BARÓN PÉREZ**manifiestan que después de haber revisado el documento de tesis titulado **"Coreografía de Microservicios de Refactorización de Software Legado"**, realizado bajo la dirección del **Dr. René Santaolaya Salgado** y Codirigida por la **Dra. Blanca Dina Valenzuela Robles**, el trabajo se ACEPTA para proceder a su impresión.

A T E N T A M E N T E

Excelencia en Educación Tecnológica®
"Conocimiento y Tecnología al Servicio de México"

DR. RENÉ SANTAOLAYA SALGADO
TECNM/CENIDET

DRA. BLANCA DINA VALENZUELA ROBLES
TECNM/CENIDET

DRA. OLIVIA GRACIELA FRAGOSO DÍAZ
TECNM/CENIDET

DR. JUAN CARLOS ROJAS PÉREZ
TECNM/CENIDET

DR. JUAN GABRIEL GONZÁLEZ SERNA
TECNM/CENIDET

DRA. ADRIANA MEXICANO SANTOYO
TECNM/ IT. C.D. VICTORIA

c.c.p: C Verónica Sotelo Boyas/ Jefa del Departamento de Servicios Escolares
c.c.p: C Nóe Alejandro Castro Sánchez / Jefe del Departamento de Ciencias Computacionales
c.c.p: Expediente

Educación
Secretaría de Educación Pública

TECNOLÓGICO NACIONAL DE MÉXICO

Centro Nacional de Investigación y Desarrollo Tecnológico
Subdirección Académica

Cuernavaca Mor, 02/julio/2026
Oficio No. SAC/208/2026
Asunto: Autorización de impresión de tesis

NELIDA BARÓN PÉREZ
CANDIDATA AL GRADO DE DOCTORA EN CIENCIAS DE LA COMPUTACIÓN
P R E S E N T E

Por este conducto, tengo el agrado de comunicarle que el Comité Tutorial asignado a su trabajo de tesis titulado **"Coreografía de Microservicios de Refactorización de Software Legado"**, ha informado a esta Subdirección Académica, que están de acuerdo con el trabajo presentado. Por lo anterior, se le autoriza a que proceda con la impresión definitiva de su trabajo de tesis.

Esperando que el logro del mismo sea acorde con sus aspiraciones profesionales, reciba un cordial saludo.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
"Conocimiento y Tecnología al servicio de México"

EDUCACIÓN | TECNOLÓGICO NACIONAL DE MÉXICO
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
SUBDIRECCIÓN ACADÉMICA

CARLOS MANUEL ASTORGA ZARAGOZA
SUBDIRECTOR ACADÉMICO

c.c.p. Departamento de Ciencias Computacionales
Departamento de Servicios Escolares

CMAZ/lmz

2026 año de Margarita Maza

cenidet
Centro Nacional de Investigación y Desarrollo Tecnológico

Interior Internado Palmira S/N, Col. Palmira,
C.P. 62490, Cuernavaca, Morelos Tel. 01 (777) 3627770, ext. 4106,
e-mail: acad_cenidet@tecnm.mx | cenidet.tecnm.mx

DEDICATORIAS

Esta tesis está dedicada, con todo mi amor, a las personas que han sido mi mayor fortaleza, inspiración y motivo para seguir adelante.

A mis amados hijos, Joan, Emiliano y Alexis, quienes son mi mayor inspiración y lo más importante en mi vida. Quiero que sepan que los amo con todo mi corazón y que cada esfuerzo, cada sacrificio y cada meta alcanzada ha sido pensando en ustedes. Siempre daré lo mejor de mí para que puedan encontrar en su madre un ejemplo de perseverancia, esfuerzo y superación.

A mis amados padres, quienes son el pilar de nuestra familia y la base de todo lo que soy. Gracias por apoyarme en las buenas y en las malas, por confiar siempre en mí, por sostenerme cuando lo he necesitado y por enseñarme que, con esfuerzo, amor y perseverancia, los sueños pueden hacerse realidad. Este logro también es de ustedes. Gracias, Dios, por bendecirme con unos padres tan maravillosos.

A mi hermana Ivón, a quien amo profundamente. Gracias por permanecer siempre a mi lado, por acompañarme en cada momento, por tu cariño, tu paciencia y tu apoyo incondicional. No tengo palabras suficientes para agradecer todo lo que has hecho por mí. Le doy gracias a Dios por haberme bendecido con una hermana como tú. Te amo con todo mi corazón.

A mi hermano César, por estar presente cuando más lo necesito y por enseñarme, con tu ejemplo, que nunca debemos darnos por vencidos. Gracias por tu cariño, tu fortaleza y tu apoyo. Te amo, hermano, y agradezco a Dios por tenerte en mi vida.

A todos ustedes, mi familia, les dedico este logro, porque han sido mi refugio, mi impulso y mi mayor bendición. Sin su amor y apoyo, este sueño no habría sido posible.

AGRADECIMIENTOS

Al Tecnológico Nacional de México por ser una institución de excelencia, formación académica y profesional al servicio de México.

Al Centro Nacional de Investigación y Desarrollo Tecnológico por brindarme el espacio y el tiempo necesario para concluir el programa de Doctorado en Ciencias de la Computación en dicha Institución.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico brindado durante la realización de mis estudios de Doctorado.

A mi director de tesis, Dr. René Santaolaya Salgado, por su amistad, consejos, generosidad y sobre todo por su gran paciencia para la realización de este trabajo.

A mi codirectora de tesis, Dra. Blanca Dina Valenzuela Robles, quien, con su gran carisma y humanidad, siempre me brindó su ayuda y consejos.

A mis revisores, Dra. Olivia Graciela Fragoso Díaz, Dra. Adriana Mexicano Santoyo, Dr. Juan Carlos Rojas Pérez y Dr. Juan Gabriel González Serna, por el tiempo y dedicación, observaciones y comentarios en el desarrollo de esta investigación.

A mis profesores en general, por su enseñanza profesional y académica.

A mis padres, Mercedes y Cesáreo, por todo el gran apoyo que en todo momento me han brindado. Gracias a Dios por tan maravillosos padres.

A mi hermana Ivón, quien siempre está conmigo, apoyándome en cualquier momento de mi vida.

A mis compañeros y amigos que estuvieron conmigo durante todo el recorrido de mi carrera: Alejandra, Derick y Enrique.

RESUMEN

La deuda técnica es una de las principales causas que dificultan el mantenimiento y la evolución de los sistemas legados, ya que se deriva de decisiones apresuradas o incorrectas durante el desarrollo, como el uso excesivo de herencia de implementación, la carencia de abstracción o la debilidad en la protección modular. Estas deficiencias degradan la calidad estructural del software y afectan negativamente atributos como la flexibilidad, la extensibilidad, la reutilización y la mantenibilidad. Para afrontar estos desafíos, la refactorización se ha convertido en una práctica esencial dentro de la ingeniería de software, ya que permite mejorar la estructura interna del código sin alterar su comportamiento externo.

En este trabajo de investigación doctoral se propone un algoritmo Greedy adaptado que permite seleccionar, en cada iteración, la técnica de refactorización más adecuada considerando el estado actual del sistema. Este algoritmo integra precondiciones y postcondiciones para asegurar que las transformaciones aplicadas sean viables y no contradigan las mejoras alcanzadas en pasos anteriores, lo que contribuye a minimizar los conflictos entre técnicas y a procurar que la calidad estructural del código se conserve o mejore durante el proceso.

Como parte de su implementación, el modelo considera una coreografía de microservicios de refactorización, donde cada microservicio ejecuta una técnica específica y mantiene su autonomía. Esta arquitectura permite mayor flexibilidad y evita dependencias rígidas entre técnicas, favoreciendo su integración modular.

La propuesta se enfoca en reducir la deuda técnica en tres dimensiones clave: protección modular, abstracción y herencia de implementación. Estas dimensiones fueron seleccionadas por su relevancia directa en la calidad interna del diseño orientado a objetos y por su influencia en problemas como la fragilidad, la rigidez, la inmovilidad y la baja extensibilidad.

Para validar el modelo, se aplicaron secuencias de refactorización generadas automáticamente sobre 30 sistemas reales seleccionados del repositorio GitHub, todos desarrollados en lenguaje Java. Durante el proceso de experimentación se utilizaron métricas específicas que permitieron medir el impacto de cada secuencia en relación con la mejora de la estructura interna del código. Los resultados obtenidos muestran que el modelo propuesto genera secuencias más estables, con menor interferencia entre técnicas, y favorece la conservación o mejora de la calidad estructural del software, contribuyendo a la reducción de la deuda técnica en los sistemas evaluados

ABSTRACT

Technical debt is one of the main factors that hinder the maintenance and evolution of legacy systems, as it arises from rushed or incorrect decisions made during software development, such as the excessive use of implementation inheritance, the lack of abstraction, or weaknesses in modular protection. These deficiencies degrade the structural quality of software and negatively affect attributes such as flexibility, extensibility, reusability, and maintainability. To address these challenges, refactoring has become an essential practice in software engineering, since it improves the internal structure of the code without altering its external behavior.

This doctoral research proposes an adapted Greedy algorithm that selects, at each iteration, the most appropriate refactoring technique according to the current state of the system. The algorithm incorporates preconditions and postconditions to ensure that the applied transformations are feasible and do not contradict the improvements achieved in previous steps. This contributes to minimizing conflicts among techniques and helps ensure that the structural quality of the code is preserved or improved throughout the process.

As part of its implementation, the model considers a choreography of refactoring microservices, in which each microservice executes a specific technique while maintaining its autonomy. This architecture provides greater flexibility and avoids rigid dependencies among techniques, thereby supporting their modular integration.

The proposal focuses on reducing technical debt in three key dimensions: modular protection, abstraction, and implementation inheritance. These dimensions were selected because of their direct relevance to the internal quality of object-oriented design and their influence on problems such as fragility, rigidity, immobility, and limited extensibility.

To validate the model, automatically generated refactoring sequences were applied to 30 real-world systems selected from GitHub, all developed in the Java programming language. During the experimentation process, specific metrics were used to measure the impact of each sequence with respect to improvements in the internal structure of the code. The results show that the proposed model generates more stable sequences, with less interference among techniques, and promotes the preservation or improvement of software structural quality, thereby contributing to the reduction of technical debt in the evaluated systems.

Contenido

Índice de Figuras

Capítulo 1.- Introducción

Desarrollar software orientado a objetos de calidad implica que el desarrollador aplique de forma armónica principios fundamentales como la abstracción, la modularidad y el encapsulamiento. Sin embargo, en la práctica, estas habilidades no siempre se implementan correctamente, ya sea por falta de experiencia, presiones de tiempo o decisiones de diseño inadecuadas. Como resultado, se generan fragmentos de código mal estructurado o con deficiencias de diseño, comúnmente denominados *code smells*, los cuales contribuyen significativamente al aumento de la deuda técnica.

La deuda técnica se refiere al impacto negativo que provocan decisiones de implementación poco óptimas, afectando la calidad, la evolución y la mantenibilidad del software. Una forma efectiva de reducir esta deuda es mediante la aplicación de técnicas de refactorización, las cuales permiten mejorar la estructura interna del código sin modificar su comportamiento externo. Sin embargo, aplicar múltiples técnicas de refactorización en un sistema complejo puede ocasionar conflictos si no se respeta un orden adecuado, ya que algunas transformaciones pueden anular o interferir con otras. Identificar una secuencia válida y adecuada de refactorizaciones sigue siendo un problema abierto en la literatura.

Con el objetivo de abordar este reto, esta tesis doctoral propone una adaptación al algoritmo Greedy tradicional, orientada a definir una secuencia válida para la selección y ejecución de técnicas de refactorización. Esta versión adaptada del algoritmo considera el estado actual del sistema y evalúa las precondiciones y postcondiciones de cada técnica como criterio de decisión, con la finalidad de reducir los conflictos entre refactorizaciones y procurar que la calidad estructural del código se mantenga o mejore durante el proceso.

Como parte de su implementación, el modelo propuesto se apoya en una arquitectura de microservicios en forma de coreografía, en la cual cada microservicio representa una técnica de refactorización autónoma. Esta arquitectura favorece la independencia y flexibilidad de las técnicas, al mismo tiempo que permite evaluar y aplicar secuencias generadas por el algoritmo adaptado. El modelo se enfoca en reducir la deuda técnica en tres dimensiones clave del diseño orientado a objetos: protección modular, abstracción y herencia de implementación.

Para validar el impacto del modelo, se aplicaron secuencias de refactorización generadas automáticamente sobre 30 proyectos reales escritos en Java, extraídos del repositorio GitHub. Durante este proceso se utilizaron métricas específicas orientadas a la calidad estructural de sistemas orientados a objetos, relacionadas con protección modular, herencia de implementación y abstracción. Estas métricas permitieron medir el efecto de las refactorizaciones sobre aspectos como el encapsulamiento, la herencia de implementación y el nivel de abstracción del sistema.

Los resultados obtenidos muestran que el algoritmo propuesto genera secuencias más estables, con menor interferencia entre técnicas, y favorece que la estructura del software se conserve o mejore durante el proceso de refactorización, contribuyendo a la reducción de la deuda técnica.

Organización de este documento de tesis

El presente documento de tesis se organiza de la siguiente manera:

En el Capítulo 2 se presentan los antecedentes de la investigación, incluyendo el planteamiento del problema, los objetivos, la justificación y el estado del arte relacionado con la refactorización, la deuda técnica y la secuenciación de técnicas.

En el Capítulo 3 se describen los conceptos teóricos que sustentan la investigación, tales como software legado, deuda técnica, orquestación de microservicios, coreografía de microservicios, refactorización y teoría de modelos, los cuales permiten comprender la base conceptual y formal del modelo propuesto.

En el Capítulo 4 se presenta el modelo propuesto de secuenciación de refactorizaciones, el cual representa el sistema como una sucesión de estados estructurales y regula la aplicación de técnicas mediante precondiciones, postcondiciones y métricas de calidad.

En el Capítulo 5 se describe el algoritmo Greedy adaptado para la secuenciación de refactorizaciones. Se analizan las limitaciones del algoritmo Greedy tradicional y se explica la modificación propuesta para seleccionar técnicas de refactorización con base en la evaluación incremental de estados.

En el Capítulo 6 se describe la implementación del modelo y del algoritmo mediante la aplicación MRefactoring Web, detallando su arquitectura, componentes principales, casos de uso y flujo de ejecución.

En el Capítulo 7 se presenta la validación del modelo y del algoritmo, considerando los sistemas de estudio, el plan de validación, las métricas empleadas, los resultados obtenidos y el análisis comparativo entre el algoritmo Greedy tradicional y el algoritmo Greedy adaptado.

Finalmente, en el Capítulo 8 se presentan las conclusiones, las aportaciones principales, las limitaciones identificadas y las líneas de trabajo futuro.

Capítulo 2.- Antecedentes

2.1 Planteamiento del problema

En el dominio de la ingeniería del software, la evolución constante de los sistemas legados se ve gravemente obstaculizada por la acumulación de deuda técnica (DT). Esta deuda se origina en defectos de diseño fundamentales que se perpetúan en el código, tales como la Herencia de Implementación, la Carencia de Abstracción y la Carencia de Protección Modular. La presencia de estos defectos estructurales implica un incumplimiento directo de principios de diseño esenciales, incluyendo el Ocultamiento de Información, el principio Abierto-Cerrado, el de Inversión de Dependencias y el de Sustitución (Liskov).

La consecuencia directa de este código desagradable es una profunda degradación en la calidad estructural del software. Dicha degradación se manifiesta en efectos adversos críticos como la rigidez, fragilidad, inmovilidad y no extensibilidad. Si estos problemas no se corrigen de manera oportuna, la deuda técnica experimenta un crecimiento desmesurado con cada nueva modificación al código.

Este aumento de la DT provoca dificultades operacionales severas, entre las cuales destacan:

- Propagación de defectos: Los cambios o correcciones en una unidad de software se extienden incontrolablemente hacia otras unidades relacionadas, debido a la deficiente protección modular.
- Alto acoplamiento: Se generan dependencias funcionales rígidas entre las jerarquías de clases y las entidades legadas, lo que es un resultado directo del uso excesivo de herencia de implementación funcional.

- Anulación del reuso: La falta de abstracción inutiliza el potencial de reuso de componentes de software y dificulta la implementación de nuevas características.
- Impacto económico: El alto grado de deuda técnica se traduce en una baja productividad, menor calidad del software e incumplimiento de los plazos comprometidos de entrega.

En consecuencia, el crecimiento desmesurado de la DT da como resultado software con grandes problemas para su reuso, mantenimiento y control de evolución, hasta el punto en que puede resultar más costoso modificarlo o introducir nuevos requerimientos que reemplazarlo por un sistema nuevo.

2.2 Objetivos

2.2.1 Objetivo general:

Definir una secuencia de ejecución secuencial de microservicios que reduzca los efectos entre sí. Mediante un modelo de ejecución apropiada, donde cada microservicio de refactorización debe establecer su precondición y su postcondición en la secuencia, para disminuir la deuda técnica en las dimensiones de protección modular, abstracción y herencia de implementación.

2.2.2 Objetivos específicos

- ✓ Disminuir la fragilidad del sistema legado mediante la protección modular aplicando las reglas de ocultamiento de datos del lenguaje Java.
- ✓ Aumentar la flexibilidad y extensibilidad invirtiendo las dependencias funcionales y aplicando el principio de abierto-cerrado mediante el uso correcto de la propiedad de abstracción.
- ✓ Mejorar la modularidad en la dimensión de herencia de implementación mediante la inversión de las dependencias de herencia de implementación por herencia de interfaz.

2.3 Justificación

Desarrollar software demanda la consideración de varios aspectos importantes, entre ellos, su evolución, como lo indican la primera, segunda y sexta ley de la evolución del software de Lehman, las cuales establecen que: 1) un programa que se utiliza en un entorno real debe adaptarse continuamente, de lo contrario resulta menos útil; 2) a medida que un programa evoluciona su complejidad aumenta, a menos que se trabaje para mantener o reducir la complejidad; 3) el contenido funcional de un programa debe incrementarse continuamente para mantener la satisfacción de los usuarios (Lehman, 1968).

Una de las causas sobresalientes que provoca dificultad en el control de la evolución del software es la presencia de DT en el código fuente, la cual además complica el reuso, encarece la producción de software y su mantenimiento. Trabajos de investigación anteriores han expresado que una de las principales razones por la que la DT aparece o incrementa, es debido a las presiones de tiempo para el desarrollo, por lo cual se suelen tomar malas decisiones que tienen un impacto negativo en la calidad del diseño interno, que promueven la presencia de código desagradable, provocando que

el software no se ajuste correctamente después de cada actualización. A este fenómeno se le conoce como entropía o decaimiento del software (Kebir et al., 2017), dicho fenómeno apunta a la presencia de deuda técnica de largo plazo en el código fuente del software. Para abordar este problema, la refactorización surge como una solución posible e interesante.

Hoy en día existen diversos métodos, estrategias y técnicas de refactorización, que se han implementado como herramientas, sistemas o complementos de otra aplicación (plugin). En este trabajo de investigación se pretende desarrollar un modelo que represente la integración y extensión de microservicios de refactorización para eliminar o reducir cierto código desagradable que afecta negativamente a los atributos de calidad de *fragilidad, flexibilidad, extensibilidad, dependencias de herencia y movilidad para reuso*. Lo que, en consecuencia, permitirá disminuir o eliminar la DT y mejorar la evolución del código legado, facilitando su mantenimiento y su movilidad para reuso.

2.4 Estado del Arte

Este apartado tiene como objetivo establecer la base para la investigación y, demostrar la carencia de modelos seguros para la aplicación masiva de refactorizaciones. La revisión de la literatura se estructura en torno a los tres ejes que definen la problemática de esta tesis: la identificación de la deuda técnica (DT), la aplicación de técnicas de refactorización individuales y el desafío de la secuenciación segura de múltiples refactorizaciones.

Si bien la comunidad de Ingeniería del Software ha desarrollado una variedad de técnicas de refactorización para combatir la DT, la complejidad surge cuando estas transformaciones se aplican en conjunto. Los modelos actuales suelen fallar en garantizar que la aplicación de una refactorización no interfiera negativamente con el beneficio obtenido por una anterior, o que no viole las precondiciones necesarias para la ejecución de la siguiente.

Esta sección inicia con la revisión de antecedentes directos, los cuales corresponden a trabajos desarrollados en el CENIDET. Estos estudios han contribuido a definir técnicas que atienden propiedades como la abstracción, el encapsulamiento y la modularidad, aspectos clave para mejorar la calidad del software desde una perspectiva estructural. Posteriormente, se presenta un análisis comparativo de la literatura especializada, focalizado en los trabajos que buscan la automatización de secuencias de refactorización. Este análisis considera diversos enfoques, entre ellos: heurísticas evolutivas, técnicas de aprendizaje automático, análisis de trazabilidad, y propuestas orientadas a patrones de diseño.

2.4.1 Trabajos desarrollados en el CENIDET

Método de refactorización en arquitecturas de software con carencia de abstracción" (Tello, 2021)

En arquitecturas de software que carecen de la propiedad de abstracción, el cliente solicita los servicios funcionales directamente a clases específicas donde su comportamiento es fijo. De esta forma, la interacción entre objetos está fuertemente atada a la implementación funcional definida en sus clases concretas. Es decir, exhiben problemas de a) carencia de flexibilidad que impide el

cambio de comportamiento funcional de diferentes aplicaciones; b) carencia de extensibilidad a nuevos comportamientos por funcionalidades requeridas y c) inmovilidad de código. Estas son causas que previenen su reuso y dificultan su mantenimiento.

(Tello, 2021) propone un método denominado "Método de Refactorización de Carencia de Abstracción" el cual consiste en identificar las instancias de objetos mediante un análisis sintáctico que identifica las clases base (es decir no hereda de otra clase) y toma sus métodos públicos para generar la clase abstracta con la firma de cada método.

Método de refactorización para mejorar la protección modular de arquitecturas orientadas a objetos de sistemas de software existentes" (Barón, 2020)

El desarrollo de software con calidad nos lleva a analizar los principales problemas que se pueden presentar en el desarrollo de aplicaciones orientadas a objetos, que originan deuda técnica. Entre estos problemas está ignorar la protección modular e ignorar el principio de ocultamiento de información en base a las reglas de visibilidad, que previenen el diseño incorrecto de las diferentes entidades de software, ya sean éstas: métodos, clases de objetos, módulos de programa o paquetes. Diseños incorrectos con carencia de protección modular están expuestos a la manipulación inadvertida de agentes externos, lo cual puede originar fragilidad en las entidades de software por la propagación de defectos de un módulo hacia otros módulos.

Con el objetivo de proteger de la manipulación externa a las entidades de software de un sistema, así como para alcanzar un mayor grado de encapsulamiento, (Barón, 2020) propone un método de refactorización denominado "Método de refactorización de calificadores de alcance", el cual identifica y coloca el calificador de alcance correcto a cada una de las funciones (métodos) que se encuentran en las clases de objetos que conforman una aplicación escrita en lenguaje Java.

***Re-Factorización de Código para Reducir el Acoplamiento entre Clases* Relacionadas por Herencia de Implementación en Arquitecturas Orientadas a Objetos" (Ortiz, 2019)**

La programación orientada a objetos es un paradigma de programación que utiliza objetos de datos para desarrollar sistemas de software. Se basa en técnicas como la herencia y abstracción, con la finalidad de crear software de calidad. Además, se apoya en el uso de principios de diseño de software, como lo son los principios de "Abierto-Cerrado" y el de "Inversión de Dependencias". Uno de los problemas que se pueden presentar al no respetar los principios de diseño de software es la herencia de implementación la cual viola los principios de diseño de "Abierto- Cerrado" y el de "Inversión de Dependencias".

Para resolver el problema antes mencionado (Ortiz, 2019) propone un método de refactorización denominado "Reducción de Herencia de Implementación" el cual consiste en invertir las llamadas directas de código de cliente a código de librerías (call forward) por llamadas de código de librerías hacia código de cliente (call back) utilizando el patrón de diseño "template method".

2.4.2 Trabajos relacionados

Para asegurar la exhaustividad de la búsqueda, se emplearon las siguientes cadenas de búsqueda: "métodos y herramientas de refactorización de código fuente y código desagradable (code smell)", "sistemas de análisis de código fuente", "estrategias de eliminación de deuda técnica", "disminución de la deuda técnica", "métricas para la medición de atributos que generan deuda técnica", "aplicación ordenada de refactorizaciones", "secuenciación de técnicas de refactorización", y "orden de técnicas de refactorización".

La búsqueda de artículos se consultó en bases de datos de alta calidad editorial, incluyendo Sciencedirect de Elsevier, IEEE Xplore, ACM Digital Library, Springerlink y Worldscientific. Los artículos relacionados fueron obtenidos principalmente de Elsevier e IEEE Xplore. Es importante destacar que el 85% del total de los trabajos relacionados estudiados son de índice JCR (Journal Citation Reports).

En el ámbito de la refactorización de software, diversos estudios han explorado la identificación y aplicación de secuencias de refactorización con el objetivo de mejorar la calidad del diseño y reducir la Deuda Técnica (DT). Sin embargo, la gran mayoría de estos enfoques se concentra en la optimización de métricas y pocos han puesto énfasis en garantizar que cada técnica aplicada no contradiga ni comprometa las mejoras obtenidas previamente, que es precisamente el objetivo principal de este trabajo.

La literatura relevante se puede clasificar en trabajos que buscan la optimización heurística de métricas y aquellos que se centran en la aplicación segura de refactorizaciones en conjunto.

Optimización Basada en Algoritmos Heurísticos: Investigadores como Kessentini et al. propusieron un enfoque basado en algoritmos genéticos para optimizar métricas de cohesión y acoplamiento. Aunque efectivo para encontrar soluciones, este enfoque no incorpora mecanismos para asegurar la compatibilidad lógica entre refactorizaciones consecutivas, lo que podría resultar en efectos adversos no deseados. De manera similar, Tarwani y Chug investigaron algoritmos heurísticos (Hill-Climbing y variantes) para identificar secuencias, pero no abordan cómo las refactorizaciones posteriores podrían anular la calidad obtenida previamente, lo que reduce la robustez de su enfoque en escenarios de código Java complejo.

Modelos de Predicción y Aprendizaje: Otros trabajos se han centrado en la predicción. Ouni et al. implementaron estrategias de aprendizaje automático para predecir secuencias de refactorización a partir de datos históricos. Si bien logran resultados prometedores, su dependencia de grandes volúmenes de datos y la falta de un mecanismo de verificación de precondiciones limitan su capacidad para abordar explícitamente la posibilidad de contradicciones entre refactorizaciones. Mahouachi y Kessentini emplearon redes bayesianas para modelar el impacto y predecir secuencias, pero este método no valida explícitamente el impacto en los atributos de calidad de diseño (modularidad, abstracción) mediante métricas cuantitativas al nivel que requiere una intervención precisa.

Enfoques en Patrones Específicos: Trabajos como los de Zafeiris et al. y Christopoulou et al. exploraron la automatización de patrones de diseño específicos. Si bien su enfoque es práctico, no

incorporan la lógica de un algoritmo que garantice mejoras progresivas y consistentes a lo largo de una secuencia completa de refactorizaciones, ni ofrecen una solución general para la aplicación de diversas técnicas en conjunto.

La Tabla 1 presenta el listado de trabajos relacionados que fueron identificados a partir de las cadenas de búsqueda definidas para este estudio. Esta tabla tiene como propósito únicamente enumerar las investigaciones revisadas en cada una de las temáticas seleccionadas, mientras que el análisis de sus enfoques, limitaciones y aportaciones se desarrolla en las secciones posteriores.

Tabla 1. Trabajos relacionados

Referencia	Nombre del artículo de investigación
(Tarwani & Chug, 2021)	*Application of AO* Algorithm in Recognizing the Optimum Refactoring Sequence for Examining the Effect on Maintainability: An Empirical Study*
(Tarwani & Chug, 2020)	*Assessment of Optimum Refactoring Sequence to Improve the Software Quality of Object-Oriented Software*
(Othman & Kaya, 2019)	*Refactoring Code Clone Detection*
(Mohan & Greer, 2019)	*Using a Many-Objective Approach to Investigate Automated Refactoring*
(Guggulothu & Moiz, 2019)	*An Approach to Suggest Code Smell Order for Refactoring*
(Lu et al., 2019)	*Automated Refactoring of OCL Constraints with Search*
(Malathi & Sudhakar, 2018)	*Implementation of Software Refactoring Using FODA Tool*
(Nyamawe et al., 2018)	*Recommending Refactoring Solutions Based on Traceability and Code Metrics*
(Khatchadourian & Masuhara, 2017)	*Automated Refactoring of Legacy Java Software to Default Methods*
(Zafeiris, Poulias, Diamantidis, & Giakoumakis, 2017)	*Automated Refactoring of Super-Class Method Invocations to the Template Method Design Pattern*
(Anuradha Chug & Sandhya Tarwani, 2017)	*Determination of Optimum Refactoring Sequence Using A* Algorithm After Prioritization of Classes*
(Tarwani & Chug, 2016)	*Improving Software Maintainability Using Optimum Refactoring Sequence*

(Mahouachi & Kessentini, 2016)	*Refactoring Impact Modeling Using Bayesian Networks for Sequence Prediction*
(Panita Meananeatra, 2012)	*Identifying Refactoring Sequences for Improving Software Maintainability*
(Kessentini et al.)	*A Genetic Algorithm-Based Approach for Software Refactoring Optimization*
(Ouni et al.)	*Search-Based Refactoring Prediction Using Historical Data and Machine Learning*
(Tama, n.d.)	*Scheduling Refactoring Opportunities Using Computational Search*

En resumen, la literatura aborda la optimización métrica y la aplicación de refactorizaciones, pero persiste una carencia crítica de un modelo que resuelva el problema de la interferencia y la seguridad en la secuenciación.

En contraste, este trabajo propone una modificación al algoritmo Greedy, diseñada para aplicarse a técnicas de refactorización de código Java, que busca garantizar que cada refactorización contribuya a una mejora integral y progresiva. A través de la integración formal de precondiciones y postcondiciones, se evita que las refactorizaciones consecutivas generen contradicciones o anulen beneficios, ofreciendo así una garantía adicional de consistencia y seguridad en la calidad del diseño.

En la Tabla 2 se presenta la comparativa de los trabajos con objetivos más semejantes al de este trabajo de tesis, dicha comparativa se basa en los siguientes criterios de evaluación.

Criterios de Evaluación de trabajos relacionados:

Objetivo: Se considera este criterio para establecer una comparativa en cuanto a la diferencia del objetivo de cada una de las investigaciones relacionadas, contra del objetivo que se persigue en este proyecto de tesis.

Proceso: Este criterio establece el grado de participación del usuario en cada producto resultante de las investigaciones relacionadas y este proyecto de tesis.

Se describen como:
Automático (A): Sin ninguna intervención del usuario
Semi-Automático (SA): Con una mínima intervención del usuario
Manual (M): Completa intervención del usuario

Entrada: Este criterio indica cuál es la entrada que requiere cada método para iniciar su proceso.

- Código fuente (CF)
- Modelo de casos de uso (MC)
- Modelo BPMN (MB)
- Microservicios (MI)
- Diseño de la arquitectura (DA)
- Documentación del sistema (DS)

Estrategia de procesamiento de información: Este criterio establece el tipo de estrategia de información que se utilizó en cada trabajo relacionado y la que se utilizará en este proyecto de tesis.

- Top-Down (TD): Esta estrategia parte del diseño para la obtención de información o elementos útiles para el estudio en desarrollo.
- Bottom-Up (BU): Esta estrategia parte del análisis del código fuente de sistemas legados para la obtención de información o elementos útiles para el estudio en desarrollo.
- Meet-in-the-middle (MM): Este enfoque es una combinación de las estrategias de Top-Down y Bottom-Up.
- Análisis de trabajos de investigación (ATI)
- Estudio de caso (EC)

Aportación: Este criterio se utiliza para establecer el grado de innovación u originalidad de los diferentes trabajos relacionados.

- Herramienta (HE)
- Algoritmo (AL)
- Método (ME)
- Modelo (MD)
- Arquitectura genérica (AG)
- Marco técnico para la gestión de deuda técnica (MT)
- Métricas nuevas (MN)
- Microservicios de refactorización (MF)

Producto resultante: Este criterio indica el producto derivado de la aportación realizada.

- Un enfoque nuevo (EF)
- Sistema (SI)
- Microservicios (MI)
- Plug-in (PL)
- Software como servicio (SaaS)
- Marco de microservicios (MM)
- Análisis (AN)

Alcance: Este criterio define el alcance de cada aportación en los trabajos analizados.

- Identificado (ID)

- Implementado (MP)
- Desplegado (DP)

Tipo de interacción: Este criterio establece el tipo de interacción entre los microservicios.

- Orquestación (OR)
- Coreografía (CO)

Tabla 2. Tabla comparativa de trabajos relacionados.

Trabajo de investigación	Objetivo	Proceso	Entrada	Producto	Alcance	Aportación	Tipo de interacción
(Kebir, Borne, &Meslati, 2017)	Reducir la presencia de malos olores de software basado en componentes	SA	CF	EF	MP	AL	-
(Al Dallal & Abdin,2018)	Apoyar en la identificación y refactorización de los malos olores en el código fuente	M	-	AN	-	RS	-
(Conejero et al.,2018)	Evaluar la relación de la deuda técnica (DT) causada por anomalías de modularidad, entre los atributos de mantenibilidad que afectan la calidad del software	SA	MC	AN	-	MN	-
(Borges, Barros, & Maia, 2018)	Ayudar a la migración de aplicaciones heredadas a un entorno PaaS	SA	CF	EF	MP	ME	-
(Yli-huumo, 2016)	Prevenir y reducir la deuda técnica (DT) intencional e involuntaria, relacionada con código mal estructurado y código que viola las pautas de codificación	SA	DS	AN	MP	MT	-
Propuesta de tesis	Reducir el código desagradable, derivado de malas prácticas y malas decisiones de diseño arquitectural que afectan los atributos de autonomía, robustez, flexibilidad y extensibilidad en el software legado	A	CF	SS	MP, DP	MF	CO

Trabajo de investigación	Objetivo	Proceso	Entrada	Producto	Alcance	Aportación	Tipo de interacción
(Rana, 2021)	Estudiar el impacto del esfuerzo de mantenimiento y el agregar nuevos requerimientos, al eliminar la deuda técnica (DT) existente en un sistema de software	SA	CF	EF	MP	ME	-
(Seaman & Guo,2011)	Mejorar la calidad del software y facilitar la toma de decisiones en el mantenimiento del software	SA	DS	EF	MP	ME	-
(Ouni, Kessentini, Sahraoui, Inoue, &Deb, 2016)	Automatizar múltiples criterios para sugerir diferentes tipos de refactorizaciones para mejorar la calidad del software	SA	CF	EF	MP	AL	--
(Srirama, Adhikari,& Paul, 2020)	Minimizar el tiempo y el costo de procesamiento de microservicios en la nube	SA	MI	HE	MP	ME	-
(Othman & Kaya,2019)	Identificar el código de clones y reemplazarlo con una sola llamada a la función para facilitar el mantenimiento del software	A	CF	PL	MP	AL	-
Propuesta de tesis	Reducir el código desagradable, derivado de malas prácticas y malas decisiones de diseño arquitectural que afectan los atributos de autonomía, robustez, flexibilidad y extensibilidad en el software legado	A	CF	SS	MP, DP	MF	CO

Trabajo de investigación	Objetivo	Proceso	Entrada	Producto	Alcance	Aportación	Tipo de interacción
(Yoshida, Numata,Choiz, & Inoue, 2019)	La refactorización de clones existentes en el código	SA	CF	SI	MP	ME	-
(Sirqueira, Brandl,Pedro, Silva, & Araújo, 2016)	Apoyar en la identificación y refactorización de los malos olores en el código fuente a alumnos	SA	CF	PL	MP	HE	-
(Dong, Ying, Li, Luo, & Yang, 2020)	Predecir de manera correcta y efectiva el rendimiento y el costo del sistema, y también de determinar la cantidad de núcleos de MVs necesarios para los servicios SaaS a fin de satisfacer los parámetros de calidad de servicio	SA	MI	SS	MP	ME	-
(Kaur & Singh,2017)	Mejorar la capacidad de mantenimiento del código analizando las diferentes técnicas de refactorización y mejorándolas	SA	CF	PL	MP	HE, AL	-
(Mohan & Greer,2019)	Automatizar la refactorización de software para facilitar el mantenimiento del software	SA	CF	PL	MP	HE,AL	-
Propuesta de tesis	Reducir el código desagradable, derivado de malas prácticas y malas decisiones de diseño arquitectural que afectan los atributos de autonomía, robustez, flexibilidad y extensibilidad en el software legado	A	CF	SS	MP, DP	MF	CO

Trabajo de investigación	Objetivo	Proceso	Entrada	Producto	Alcance	Aportación	Tipo de interacciónn
(Zafeiris, Poulias, Diamantidis, & Giakoumakis, 2017)	Refactorizar las instancias del patrón de código “Call super”, cambiando su implementación hacia la estructura del patrón diseño “Template Method”, para sustituir la herencia deimplementación por la herencia de la interfaz	A	CF	PL	MP	ME, AL	-
(Malathi & Sudhakar, 2018)	Mejorar la calidad del software mediante un proceso de refactorización de código	A	CF	SI	MP	HE	-
(Singh & Peddoju, 2017)	Mejorar el despliegue e integración continua de microservicios en la nube computacional	A	AM	SI	MP	MI	-
(Kiss et al., 2019)	Permitir agregar una escalabilidad y orquestación automatizada a las aplicaciones en la nube, sin que estas aplicaciones se sometan a una gran reingeniería de su código	SA	DA	MM	MP	AG	OR
Propuesta de tesis	Reducir el código desagradable, derivado de malas prácticas y malas decisiones de diseño arquitectural que afectan los atributos de autonomía, robustez, flexibilidad y extensibilidad en el software legado	A	CF	SS	MP, DP	MF	CO

Trabajo de investigación	Objetivo	Proceso	Entrada	Producto	Alcance	Aportación	Tipo de interacción
(Malathi & Sudhakar, 2018)	Mejorar la calidad del software mediante un proceso de refactorización de código	A	CF	SI	MP	HE	-
(Singh & Peddoju, 2017)	Mejorar el despliegue e integración continua de microservicios en la nube computacional	A	AM	SI	MP	MI	-
(Kiss et al., 2019)	Permitir agregar una escalabilidad y orquestación automatizada a las aplicaciones en la nube, sin que estas aplicaciones se sometan a una gran reingeniería de su código	SA	DA	MM	MP	AG	OR
Propuesta de tesis	Reducir el código desagradable, derivado de malas prácticas y malas decisiones de diseño arquitectural que afectan los atributos de autonomía, robustez, flexibilidad y extensibilidad en el software legado	A	CF	SS	MP, DP	MF	CO

Capítulo 3.- Marco teórico

Este capítulo presenta los conceptos fundamentales que sustentan la investigación, organizados de manera progresiva desde nociones generales sobre ingeniería de software hasta los elementos teóricos específicos requeridos para el diseño formal del modelo propuesto. Se incluyen temas como calidad del diseño orientado a objetos, deuda técnica, refactorización, secuenciación de técnicas, y la teoría formal utilizada para estructurar y validar el algoritmo Greedy adaptado.

3.1 Software legado

El software legado se refiere a aquellos sistemas que han estado en funcionamiento durante largos periodos de tiempo y que siguen cumpliendo funciones críticas dentro de una organización, pero que presentan serias limitaciones en cuanto a su mantenimiento, adaptación y evolución. A pesar de continuar operativos, estos sistemas enfrentan problemas como falta de documentación actualizada, fuerte dependencia del entorno en el que fueron desarrollados, y estructuras internas complejas que dificultan su comprensión y modificación (Rosik et al., 2008; Bennett et al., 2001).

De acuerdo con Seacord et al. (2003), una de las principales dificultades en los sistemas legados es su arquitectura monolítica, la cual tiende a generar acoplamiento excesivo entre componentes, reduciendo su cohesión y reusabilidad. Además, muchos de estos sistemas han sido intervenidos por múltiples desarrolladores a lo largo del tiempo, lo que provoca fragmentación del conocimiento y aumento de la deuda técnica. Este fenómeno, conocido como entropía del software, se refiere al deterioro progresivo de la estructura interna del sistema conforme se realizan cambios, afectando negativamente su calidad, mantenibilidad y adaptabilidad (Kebir et al., 2017).

Las características más comunes del software legado incluyen:

- Falta de documentación actualizada: lo cual impide comprender el funcionamiento actual del sistema y retrasa cualquier tarea de mantenimiento.
- Arquitectura monolítica y compleja: con fuerte acoplamiento entre componentes que dificulta su análisis y modificación.
- Dependencia del entorno original: como el sistema operativo, base de datos o infraestructura, lo que limita su migración.
- Crucial para el negocio: son sistemas fundamentales para el funcionamiento diario de la organización, por lo que su modificación representa un alto riesgo.
- Dificultades para incorporar nuevos requerimientos: debido a su diseño rígido y falta de modularidad.
- Costos elevados de mantenimiento: asociados a la complejidad técnica, la deuda acumulada y la escasa documentación.

En consecuencia, abordar el mantenimiento y evolución de estos sistemas requiere estrategias bien fundamentadas, que permitan reducir la deuda técnica y facilitar su transformación sin comprometer la funcionalidad crítica que ofrecen.

3.2 Deuda Técnica

La deuda técnica es un concepto introducido por Ward Cunningham en 1992 que se utiliza como metáfora para describir el costo adicional que se incurre en el futuro debido a la adopción de decisiones técnicas rápidas, subóptimas o inadecuadas durante el desarrollo de software. Estas decisiones, si bien pueden acelerar la entrega inicial del producto, generan compromisos en la calidad interna del sistema, lo que dificulta su evolución, mantenimiento y reutilización. La deuda técnica no es únicamente un problema técnico, sino también organizacional, ya que afecta la planificación de recursos y el cumplimiento de plazos.

Diversos autores han clasificado la deuda técnica en varias categorías, entre las que destacan:

- Deuda arquitectónica: Surge cuando la estructura general del sistema o su arquitectura presenta deficiencias, acoplamiento excesivo o falta de modularidad, lo que limita su escalabilidad y mantenibilidad.
- Deuda de diseño: Resulta de malas prácticas en la definición de componentes, clases, interfaces o relaciones, afectando directamente atributos de calidad como la cohesión y el acoplamiento.
- Deuda de código: Se relaciona con problemas localizados en el código fuente, tales como duplicación, malas prácticas de nomenclatura, métodos demasiado largos o complejos y ausencia de comentarios o documentación mínima.
- Deuda de pruebas: Ocurre cuando los casos de prueba son insuficientes, están desactualizados o no existen, dificultando la detección temprana de errores e incrementando los riesgos en el mantenimiento del sistema.

Deuda de documentación: Se produce cuando la documentación técnica, funcional o de usuario está incompleta o desactualizada, complicando el entendimiento del sistema y la incorporación de nuevos desarrolladores.

Cada tipo de deuda técnica puede clasificarse adicionalmente como:

- Deliberada: Cuando las decisiones que la originan se toman de forma consciente, generalmente para cumplir plazos estrictos de entrega.
- Inadvertida: Cuando la deuda surge de desconocimiento, negligencia o falta de experiencia del equipo de desarrollo.

El impacto de la deuda técnica en el software es significativo y multifacético:

- Incremento de costos: Aumenta el esfuerzo y tiempo requerido para realizar modificaciones, correcciones y nuevas implementaciones.
- Disminución de la calidad: Se degradan atributos como la flexibilidad, extensibilidad, mantenibilidad y reutilización del software.
- Aumento de riesgos: Crece la probabilidad de introducir defectos al realizar cambios o integraciones.
- Retrasos en la entrega: Las dificultades para mantener y evolucionar el sistema pueden provocar incumplimiento de plazos y pérdida de competitividad.
- Decaimiento del software (entropía): Si no se atiende, la deuda técnica tiende a incrementarse con el tiempo, reduciendo la vida útil del sistema y llevando eventualmente a costos tan altos que puede ser más viable reemplazarlo que mantenerlo.

3.3 Orquestación de microservicios

La orquestación se entiende como un patrón de composición semejante a un flujo de trabajo, en el cual se organizan actividades de negocio o actividades funcionales implementadas como operaciones de servicios. En este enfoque, existe una definición explícita del proceso que indica cómo se ejecutan las actividades, cuáles son sus entradas y salidas, qué información se transfiere entre ellas y qué orden de ejecución deben seguir. (Wang, 2013) señala que la orquestación de servicios web utiliza un patrón de composición tipo workflow para organizar actividades implementadas como operaciones de servicio y modelar procesos de negocio u otros tipos de procesos.

Desde esta perspectiva, una orquestación permite coordinar un conjunto de unidades funcionales mediante estructuras de control. Estas estructuras pueden establecer ejecución secuencial, selección entre alternativas, ciclos o ejecución paralela. El autor explica que una orquestación contiene actividades atómicas y definiciones de flujo de control que determinan el orden de ejecución de dichas actividades. Además, las actividades pueden recibir datos, modificar estados locales e interactuar con otros servicios o aplicaciones.

En términos generales, la orquestación puede interpretarse como un mecanismo en el que un proceso organiza internamente la ejecución de varias actividades para alcanzar un resultado compuesto. Esto significa que el énfasis no está únicamente en la existencia de servicios independientes, sino en la forma en que estos son integrados dentro de un flujo definido. Por ello, la orquestación resulta adecuada cuando es necesario establecer un orden de ejecución, controlar dependencias entre actividades y administrar el estado interno de un proceso.

Aplicado al contexto de esta tesis, la orquestación puede entenderse como una forma de coordinación en la cual un componente o proceso central define explícitamente el orden de ejecución de las técnicas de refactorización. Bajo este enfoque, las técnicas se ejecutarían siguiendo un flujo previamente determinado, en el que el control del proceso estaría concentrado en un elemento coordinador. Esta idea contrasta con el enfoque propuesto en la investigación, donde la secuencia no se fija de manera rígida desde el inicio, sino que se determina a partir del estado actual del sistema, las precondiciones, las postcondiciones y las métricas de calidad.

3.4 Coreografía de microservicios

La coreografía se refiere a un patrón de composición orientado a describir los comportamientos de interacción externa entre servicios. A diferencia de la orquestación, que se enfoca en el flujo interno de actividades de un proceso, la coreografía describe cómo interactúan los participantes entre sí, cuáles son sus roles, qué mensajes intercambian y bajo qué reglas se produce dicha interacción. (Wang, 2013) indica que la coreografía de servicios web representa un patrón de composición agregado que captura los comportamientos externos de interacción entre servicios y actúa como un contrato o protocolo entre ellos.

En la literatura especializada, la coreografía se describe como una forma de especificar las interacciones entre servicios participantes. Sus elementos principales incluyen la definición de participantes, los roles que asumen, la información intercambiada y los puntos de interacción entre ellos. De esta manera, la coreografía no se centra en controlar internamente todas las actividades de cada servicio, sino en establecer las reglas de comunicación que permiten que los servicios colaboren correctamente.

Wang también señala que una coreografía puede funcionar como un contrato o protocolo entre servicios que interactúan. Desde una perspectiva de integración de negocios, dicho contrato establece derechos y obligaciones entre los participantes; desde una perspectiva tecnológica, puede entenderse como un protocolo de comunicación que coordina los comportamientos de interacción de los servicios involucrados.

En el contexto de esta investigación, la coreografía puede relacionarse con la participación autónoma de los microservicios de refactorización dentro del proceso de transformación del software. Cada microservicio cuenta con una responsabilidad específica, asociada a una técnica de refactorización, al cálculo de métricas, a la evaluación de precondiciones o a la validación de postcondiciones. La interacción entre estos microservicios no se basa únicamente en un flujo rígido preestablecido, sino en las condiciones que determinan si una técnica puede aplicarse y si el estado resultante conserva o mejora la calidad estructural del sistema.

Así, la coreografía permite justificar que los microservicios de refactorización colaboran a partir de reglas de interacción definidas por el modelo. En lugar de imponer una secuencia fija de ejecución, el proceso se construye dinámicamente conforme se evalúan los estados del sistema y las condiciones de aplicabilidad de cada técnica. Esta interpretación es coherente con la idea de Wang (2013), quien considera que la coreografía establece un protocolo de interacción entre servicios participantes.

3.5 Refactorización

La refactorización consiste en modificar un sistema de software con el propósito de mejorar su estructura interna, sin cambiar el comportamiento observable del código. Se trata de una práctica ordenada y controlada que permite depurar el diseño del software y reducir el riesgo de introducir nuevos errores durante el proceso de mejora.

En términos generales, refactorizar implica perfeccionar el diseño del código después de que este ya ha sido desarrollado. (Fowler, 2009) describe esta idea como la "mejora del diseño después de que se haya escrito". Tradicionalmente, se considera que primero se diseña el sistema y posteriormente se codifica; sin embargo, durante la evolución del software, el código suele modificarse constantemente, lo que provoca que la estructura original pierda claridad y que el diseño inicial se deteriore de manera gradual.

La refactorización actúa como una respuesta a ese deterioro. A través de ella, se toma código fuente que ha perdido calidad estructural y se trabaja nuevamente para mejorar su diseño arquitectural. Las modificaciones que se realizan suelen ser pequeñas y controladas, como mover un atributo de una clase a otra, extraer fragmentos de código de un método para formar un nuevo método, o reorganizar elementos dentro de una jerarquía de clases. Aunque cada cambio puede parecer simple, el efecto acumulado de estas transformaciones puede producir una mejora significativa en el diseño general del sistema.

De esta manera, la refactorización representa una práctica contraria al deterioro natural del software. En lugar de permitir que el código se vuelva cada vez más complejo y difícil de mantener, permite aprender del propio proceso de construcción del sistema y ajustar su diseño conforme evoluciona. Como resultado, se obtiene una estructura más clara, mantenible y adaptable, capaz de conservar su calidad a medida que continúa el desarrollo del software.

3.6 Teoría de modelos

La teoría de modelos es una rama de la lógica matemática que estudia la relación entre los lenguajes formales y las estructuras en las que dichos lenguajes son interpretados. En esencia, estudia la relación entre un lenguaje formal (como el que se usa para describir sistemas o propiedades) y las estructuras matemáticas (los "modelos") sobre las que ese lenguaje se interpreta.

La teoría se fundamenta en cuatro conceptos interconectados:

1. **Lenguaje Formal (L):** Es el conjunto de símbolos utilizados para construir afirmaciones. En el contexto de esta investigación, incluye los símbolos lógicos (como "y", "o", "no") y los símbolos no lógicos (como los predicados y funciones que describen el código, por ejemplo: Clase(X), TieneAcoplamientoAlto(Y)).

2. **Estructura (Modelo A):** Es la interpretación del lenguaje. En esta investigación, la estructura corresponde al código fuente del software en un momento determinado. La estructura les da significado real a los símbolos del lenguaje:
3. El Dominio son los elementos del código (clases, métodos, atributos).

4. La interpretación asigna un valor de verdad, verdadero o falso, a los predicados definidos sobre ese código específico.

5. **Satisfacibilidad (A⊨Φ):** Este es el concepto central. Una fórmula o sentencia (Φ) es satisfacible en una estructura (A) si es verdadera bajo la interpretación definida. Es decir, si la propiedad descrita por Φ se cumple en el código A.

6. **Interpretación (I):** Es el puente conceptual que dota de significado a los símbolos del Lenguaje Formal (L) dentro del Dominio (D) de la Estructura (A). La Interpretación asigna una relación, función o valor de verdad (Verdadero o Falso) a cada símbolo no lógico en ese código específico.

La correcta formalización de las precondiciones y postcondiciones en este trabajo se fundamenta en la relación jerárquica de los componentes definidos por la Teoría de Modelos. La Figura 1 ilustra de manera conceptual los elementos de la Teoría de Modelos, mostrando cómo el Lenguaje Formal (L) interactúa con la Estructura (Modelo A) a través de la Interpretación (I) para alcanzar la Satisfacibilidad (el valor de verdad).

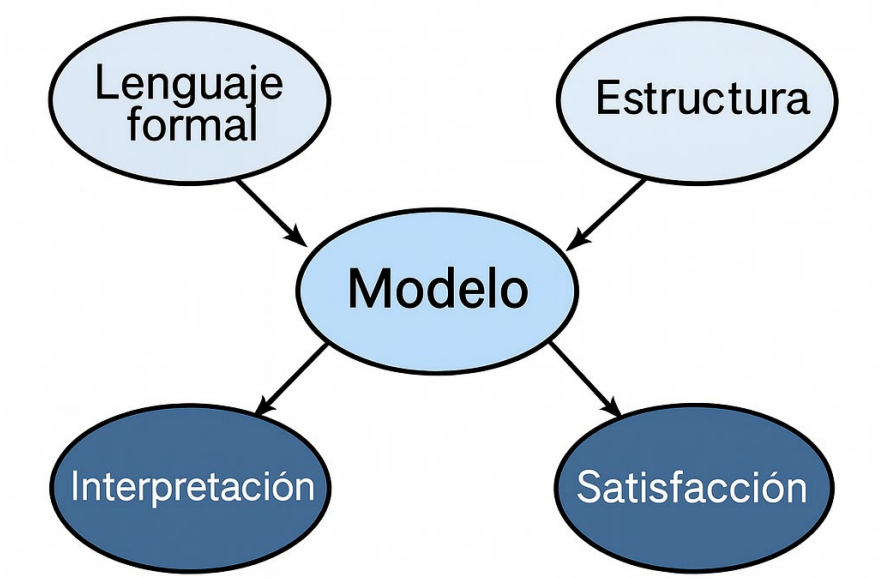


Figura 1. Relación entre lenguaje, estructura e interpretación en la teoría de modelos

Capítulo 4.- Modelo propuesto de secuenciación de refactorizaciones

4.1 Descripción general del modelo

El modelo propuesto parte de la representación formal de un sistema de software orientado a objetos como una estructura compuesta por los elementos fundamentales del paradigma, tales como clases, métodos, atributos y relaciones estructurales entre dichos elementos. Esta formalización se apoya en enfoques previos basados en teoría de modelos, particularmente en trabajos desarrollados en el Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET), donde se han formalizado los elementos estructurales de lenguajes orientados a objetos para modelar procesos de transformación de software. En la presente investigación, dicha formalización se retoma y adapta con el propósito de definir el estado estructural del sistema y modelar su transformación progresiva mediante la aplicación de técnicas de refactorización.

Formalmente, el sistema legado se representa como:

$$S = (C, M, A, Rel)$$

donde:

- C representa el conjunto de clases del sistema.
- M representa el conjunto de métodos definidos en dichas clases.
- A representa el conjunto de atributos asociados a las clases del sistema.
- Rel corresponde al conjunto de relaciones estructurales entre clases, incluyendo herencia, asociación, agregación y composición.

Esta representación permite describir de manera abstracta la organización interna del sistema, independientemente del lenguaje de programación utilizado para su implementación.

A partir de la representación anterior se define el estado estructural del sistema en un instante t como:

$$E_t$$

Donde E_t representa la configuración estructural vigente del sistema en ese momento del proceso de refactorización. En términos generales, el estado estructural del sistema describe cómo está organizado el software en un instante específico, considerando las clases que lo componen, sus atributos, los métodos definidos en cada clase y las relaciones estructurales existentes entre ellas.

Durante el proceso de refactorización, la aplicación de técnicas de transformación modifica la estructura interna del sistema, generando nuevos estados estructurales. De esta forma, el proceso puede interpretarse como una secuencia de transformaciones estructurales, en la cual cada transformación produce un cambio en la organización interna del software.

La obtención de este estado estructural se realiza mediante el análisis sintáctico del código fuente utilizando herramientas basadas en ANTLR, las cuales permiten identificar los elementos estructurales relevantes del sistema. A través de este análisis es posible detectar componentes del software orientado a objetos tales como:

- clases
- métodos
- atributos
- modificadores de acceso
- relaciones de herencia
- dependencias entre clases

A partir de esta información, el sistema puede caracterizarse mediante un conjunto de métricas estructurales que reflejan el grado de cumplimiento de principios fundamentales del diseño orientado a objetos. Entre las métricas utilizadas en esta investigación se encuentran:

- PMFP
- PMFPR
- PMFF
- PM
- TPM
- FHI
- FHIJ
- FHIAC
- FFC
- FMFAC
- A

La descripción detallada de estas métricas, incluyendo su propósito, forma de cálculo e interpretación, se presenta en el Anexo A. Descripción de métricas estructurales. En esta sección se retoman únicamente para establecer su papel dentro del modelo propuesto y su integración en la función de calidad utilizada para comparar los estados estructurales del sistema.

Estas métricas permiten evaluar diferentes propiedades estructurales relacionadas con aspectos clave del diseño orientado a objetos, tales como el encapsulamiento, la utilización de la herencia de implementación a nivel de arquitectura y el nivel de abstracción. Sin embargo, debido a que las métricas consideradas no poseen necesariamente el mismo sentido de interpretación, fue necesario aplicar un proceso de normalización antes de integrarlas en la función de calidad del modelo.

En las métricas asociadas a protección modular, el mejor valor corresponde a 1 y el peor valor corresponde a 0. Por esta razón, las métricas PMFP, PMFPR, PMFF, PM y TPM se conservan con su valor original dentro de la función de calidad. De manera similar, la métrica FMFAC, asociada con la flexibilidad a nivel de arquitectura, y la métrica A, asociada con la abstracción, también se interpretan de forma directa, debido a que un valor cercano a 1 representa una mejor condición estructural del sistema.

Por el contrario, la métrica FHIAC presenta una orientación inversa, ya que en ella el valor deseable es 0, debido a que representa una menor presencia de herencia de implementación en la arquitectura de clases, mientras que el valor 1 representa una condición menos favorable. Para evitar contradicciones durante la comparación de estados estructurales, esta métrica fue transformada mediante una normalización inversa. Es importante señalar que las métricas FHI, FHIJ y FFC también se calculan durante el proceso de evaluación; sin embargo, no se incorporan directamente en la función de calidad global. Estas métricas se utilizan como valores intermedios necesarios para obtener las métricas agregadas FHIAC y FMFAC. De esta manera, se evita duplicar el peso de métricas que ya se encuentran representadas en indicadores de mayor alcance dentro de la función de calidad.

De esta manera, todas las métricas utilizadas directamente en la función de calidad fueron expresadas bajo una misma escala de interpretación, donde el valor 1 representa la mejor condición estructural y el valor 0 representa la condición menos favorable. Para las métricas con orientación positiva, el valor normalizado se conserva sin modificación:

$$M_{norm} = M$$

En esta investigación, esta normalización aplica a las métricas PMFP, PMFPR, PMFF, PM, TPM, FMFAC y A, debido a que en ellas el valor 1 representa la mejor condición estructural.

Para las métricas con orientación inversa, el valor normalizado se obtiene mediante la siguiente expresión:

$$M_{norm} = 1 - M$$

En este caso, la normalización inversa se aplica a FHIAC, ya que su valor deseable es 0, al representar una menor presencia de herencia de implementación en la arquitectura de clases.

Con base en este criterio, la función de calidad utilizada para evaluar el estado estructural del sistema se define como:

$$Q(E_t) = \frac{TPM + (1 - FHIAC) + AF}{3}$$

Donde $Q(E_t)$ representa el valor de calidad estructural del sistema en el estado E_t. Esta función permite comparar de manera homogénea los estados generados durante el proceso de refactorización, integrando métricas de protección modular, herencia de implementación a nivel de arquitectura y abstracción bajo una misma orientación de mejora.

Formalmente, el estado estructural del sistema también puede representarse mediante el conjunto de valores asociados a las métricas estructurales evaluadas:

$$V(E_t) = \{m_1, m_2, m_3, \dots m_n\}$$

donde cada m_i representa una métrica estructural asociada a una propiedad específica del sistema en el estado E_t. Esta representación modela el estado del sistema como un conjunto de indicadores estructurales que describen la calidad interna del software en un momento determinado del proceso de refactorización.

Cada vez que se aplica una técnica de refactorización, el sistema transita desde un estado estructural E_t hacia un nuevo estado E_{t+1}, reflejando el cambio estructural producido por la transformación aplicada. En este sentido, el proceso de refactorización puede interpretarse como una secuencia de transformaciones estructurales del sistema, donde cada transformación modifica la organización interna del software y conduce a una nueva configuración estructural.

Para modelar formalmente este proceso, cada vez que se ejecuta una técnica de refactorización r_i, el sistema pasa de un estado estructural E_t a un nuevo estado E_{t+1}, lo cual puede expresarse formalmente como:

$$E_{t+1} = r_i(E_t)$$

Donde r_i representa la transformación estructural aplicada al sistema. Esta representación permite modelar el proceso de refactorización como una sucesión de transiciones entre estados estructurales, lo cual constituye la base formal del modelo de secuenciación propuesto en esta investigación.
A partir de esta formulación, el proceso de refactorización puede describirse como una secuencia ordenada de transformaciones aplicadas al sistema. Formalmente, una secuencia de refactorización puede representarse como:

$$Seq = \langle r_{i1}, r_{i1}, \dots, r_{i1}, \rangle$$

donde cada r_{ij} corresponde a una técnica de refactorización aplicada durante el proceso de transformación del sistema. Esta secuencia describe el conjunto de transformaciones estructurales

ejecutadas sobre el software y constituye el resultado del proceso de secuenciación desarrollado en esta investigación.

Con el fin de ilustrar de manera general el funcionamiento del modelo propuesto, la Figura 2 presenta una representación conceptual del proceso de secuenciación de refactorizaciones, en la cual se muestran las principales etapas involucradas en el análisis estructural del sistema, la evaluación de las condiciones de aplicabilidad de las técnicas de refactorización y la generación de nuevos estados estructurales del software.

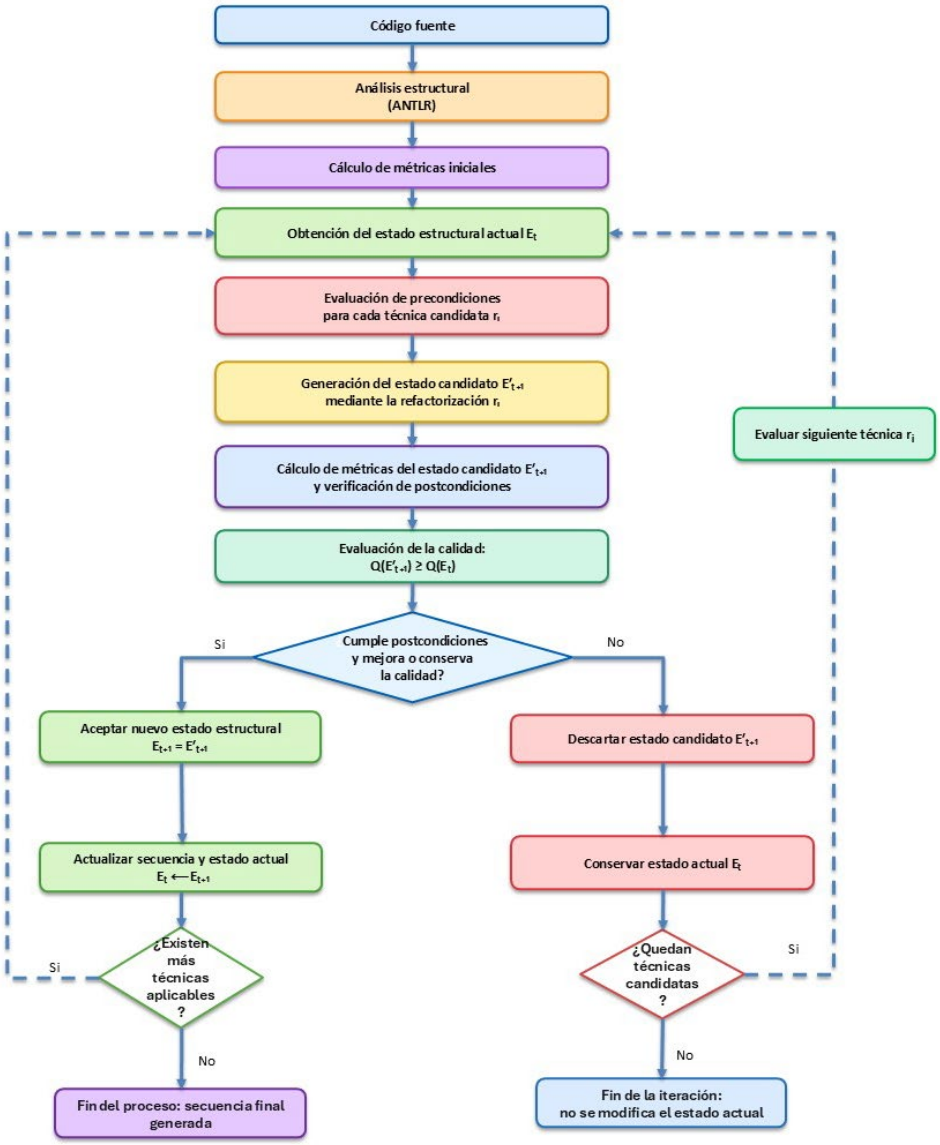


Figura 2. Modelo de secuenciación de refactorizaciones

En la Figura 2, la aplicación de una técnica de refactorización debe entenderse como la generación de un estado candidato del sistema y no como una modificación definitiva del código fuente. Este estado candidato representa una configuración estructural temporal que permite calcular las métricas resultantes

y evaluar las postcondiciones definidas en el modelo. Si el estado candidato cumple con las precondiciones, las postcondiciones y el criterio de preservación de calidad, entonces se acepta como nuevo estado estructural del sistema. En caso contrario, el estado candidato se descarta y el estado actual permanece sin cambios. De esta manera, el modelo evita aplicar directamente transformaciones que puedan degradar la calidad acumulada del sistema.

4.1.1 Formalización del modelo propuesto basada en teoría de modelos

El modelo propuesto en este trabajo describe el proceso de secuenciación de refactorizaciones como un sistema de transición finito, en el cual cada estado representa una configuración estructural del sistema de software en términos de sus clases, métodos, atributos y relaciones. La evolución del sistema se modela como una sucesión de estados estructurales generados a partir de la aplicación de técnicas de refactorización.

Desde la perspectiva de la teoría de modelos, este proceso puede representarse mediante una estructura formal que permite interpretar los elementos del sistema, las transformaciones aplicadas y las condiciones que regulan su evolución.

Formalmente, el modelo se define como:

$$M = (E, R, Q, \delta, Pre, Post)$$

Donde:

E es el conjunto de estados del sistema, tal que cada estado $E_i \in E$ representa una configuración estructural del software.
R es el conjunto de técnicas de refactorización disponibles, donde cada $R_i \in R$ corresponde a una transformación estructural aplicable al sistema.
$Q: E \longrightarrow \mathbb{R}$ es una función de calidad que asigna a cada estado un valor numérico basado en métricas estructurales, tales como protección modular, abstracción y herencia de implementación.
$\delta: E\ X\ R \longrightarrow E$ es la función de transición que modela la aplicación de una técnica de refactorización sobre un estado del sistema, generando un nuevo estado estructural.
$Pre: R\ x\ E \longrightarrow \{0,1\}$ es la función de precondiciones que determina si una técnica de refactorización puede aplicarse sobre un estado dado.
$Post: R\ x\ E \longrightarrow \{0,1\}$ es la función de postcondiciones que valida el estado resultante de la transformación.

En el modelo propuesto, la aplicación de una técnica de refactorización no se considera definitiva de manera inmediata. Primero se genera un estado candidato, el cual representa una configuración estructural temporal del sistema. Este estado permite calcular las métricas resultantes y verificar si la transformación cumple con las postcondiciones definidas.

Formalmente, si una técnica R_j se evalúa sobre un estado actual E_i, se obtiene un estado candidato:

$$E'_{i+1} = \delta(E_i, R_j)$$

Donde E'_{i+1} representa el estado candidato generado por la aplicación provisional de la técnica R_j. Este estado no reemplaza de forma inmediata al estado actual E_i, sino que se utiliza para evaluar el posible efecto de la refactorización sobre la calidad estructural del sistema.

La transición definitiva entre estados se representa como:

$$E_i \overset{R_j}{\rightarrow} E_{i+1}$$

La cual es válida únicamente si se satisfacen las siguientes condiciones:

$$Pre\left(R_j, E_i\right) = 1$$

$$Post\left(R_j, E_{i+1}\right) = 1$$

$$Q\left(E_{i+1}\right) \geq Q(E_i)$$

Estas condiciones establecen que una técnica de refactorización únicamente puede aplicarse si es válida en el estado actual y si su aplicación no degrada la calidad del sistema. De esta forma, el modelo garantiza la preservación de la calidad acumulativa a lo largo del proceso de refactorización.

El comportamiento del modelo puede expresarse mediante la siguiente implicación lógica:

$$\forall E_i \in E, \forall R_j \in R, (Pre\left(R_j, E_i\right) = 1 \wedge Post\left(R_j, E'_{i+1}\right) = 1 \wedge Q(E'_{i+1}) \geq Q(E_i)) \rightarrow (E_{i+1} = E'_{i+1})$$

Donde:

$$E'_{i+1} = \delta(E_i, R_j)$$

En caso de que alguna de las condiciones no se satisfaga, la transformación no se incorpora a la secuencia y el estado actual se conserva:

$$E_{i+1} = E_i$$

A partir de esta formalización, la secuencia de refactorizaciones puede interpretarse como una sucesión de estados validada mediante funciones de aplicabilidad, postcondiciones y evaluación de calidad. En este contexto, cada transición queda condicionada por restricciones que determinan su validez, lo que permite analizar el proceso de refactorización como un sistema controlado de transformaciones sobre la estructura del software.

4.2 Precondiciones y postcondiciones

Para garantizar la coherencia del proceso de secuenciación y evitar interferencias entre técnicas de refactorización, el modelo propuesto incorpora explícitamente precondiciones y postcondiciones como

mecanismos formales que regulan la aplicabilidad y compatibilidad de las transformaciones estructurales.

Cada técnica de refactorización r_i se asocia a un conjunto de precondiciones, las cuales se definen como predicados que deben cumplirse en el estado estructural actual del sistema para que la técnica pueda aplicarse de manera válida. Formalmente, una precondición puede representarse como:

$$Pre(r_i, E_t) \in \{verdadero, falso\}$$

Donde E_t representa el estado estructural del sistema en el instante t. Si la condición se cumple, la técnica de refactorización puede ejecutarse sobre el sistema; en caso contrario, su aplicación se considera inválida en dicho estado estructural.

Las precondiciones están asociadas a propiedades estructurales del sistema orientado a objetos, tales como la existencia de jerarquías de herencia adecuadas, el nivel de encapsulamiento de atributos y métodos, la presencia de niveles apropiados de abstracción y la ausencia de dependencias estructurales incompatibles. De esta forma, el modelo evita la aplicación de técnicas de refactorización en contextos donde su ejecución podría generar inconsistencias estructurales o deteriorar la calidad interna del software.

De manera complementaria, cada técnica de refactorización genera un conjunto de postcondiciones, las cuales describen los efectos estructurales producidos tras la aplicación de la transformación y caracterizan el nuevo estado estructural del sistema. Formalmente, las postcondiciones pueden representarse como:

$$Post\ (r_i, E_{t+1})$$

Donde E_{t+1} representa el estado estructural resultante después de aplicar la técnica de refactorización r_i.

Las postcondiciones permiten verificar que la transformación aplicada mantiene la coherencia estructural del sistema y que el nuevo estado generado conserva las propiedades necesarias para continuar el proceso de refactorización. En este sentido, las postcondiciones describen el cambio estructural producido en el sistema como resultado de la transformación aplicada.

La relación entre precondiciones y postcondiciones constituye uno de los elementos centrales del modelo de secuenciación propuesto. En particular, el modelo establece que una técnica de refactorización r_j puede aplicarse después de una técnica r_i únicamente si las postcondiciones generadas por r_i satisfacen las precondiciones requeridas por r_j. Esta relación puede expresarse formalmente como:

$$Post\ (r_i) \rightarrow Pre(r_j)$$

Esta condición garantiza que las transformaciones se encadenen de manera coherente y que los cambios estructurales generados en una etapa no comprometan la validez de las transformaciones posteriores.

A partir de esta relación, el modelo define una dinámica de interacción entre las técnicas de refactorización en la cual el orden de ejecución no se establece de manera explícita, sino que emerge del estado estructural actual del sistema. En cada etapa del proceso, el conjunto de técnicas de refactorización aplicables se determina evaluando las precondiciones sobre el estado vigente del sistema.

Formalmente, el conjunto de técnicas viables en el estado E_t puede representarse como:

$$R_t = \{r_i \in R \mid pre(r_i, E_t) = verdadero\}$$

donde R representa el conjunto total de técnicas de refactorización consideradas en el modelo.

El conjunto R_t contiene las transformaciones que pueden aplicarse de manera válida en el estado estructural actual del sistema. La aplicación de una técnica dentro de este conjunto genera un nuevo estado estructural, lo cual puede habilitar o restringir la ejecución de transformaciones posteriores.
De esta manera, el modelo describe una dinámica de transformación estructural del sistema basada en la compatibilidad entre las técnicas de refactorización, en lugar de depender de un orden rígido predefinido.

Es importante señalar que, dentro del modelo propuesto, las precondiciones y postcondiciones se definen como restricciones estructurales que regulan la aplicabilidad y compatibilidad de las técnicas de refactorización. Posteriormente, el algoritmo de secuenciación presentado en el capítulo siguiente utiliza estas condiciones como criterios para identificar el conjunto de refactorizaciones viables y seleccionar, en cada iteración, la técnica más adecuada para continuar el proceso de transformación del sistema.

4.3 Técnicas de refactorización consideradas

El modelo de secuenciación propuesto en esta investigación permite incorporar diversas técnicas de refactorización orientadas a mejorar la calidad estructural de sistemas de software desarrollados bajo el paradigma de programación orientada a objetos. Cada técnica de refactorización se interpreta como una transformación estructural que modifica la organización interna del sistema sin alterar su comportamiento observable.

Desde una perspectiva general, el modelo no restringe el conjunto de técnicas que pueden ser utilizadas. En principio, cualquier técnica de refactorización podría integrarse al modelo siempre que sea posible definir formalmente las precondiciones que determinan cuándo una técnica puede aplicarse, así como las postcondiciones que permiten verificar la validez del estado estructural resultante. De esta forma, el modelo puede extenderse para incorporar nuevas técnicas de refactorización orientadas a mejorar diferentes atributos de calidad del software.

Sin embargo, para efectos de esta investigación se consideran únicamente tres técnicas de refactorización, las cuales han sido previamente estudiadas en trabajos relacionados con la mejora de la calidad estructural de sistemas orientados a objetos. Estas técnicas son:

- herencia de implementación
- protección modular

- abstracción

La selección de estas técnicas se debe a que cada una aborda problemas estructurales comunes en sistemas de software legado, tales como jerarquías de clases mal diseñadas, violaciones al principio de ocultamiento de información o arquitecturas con carencia de abstracción.

Formalmente, el conjunto de técnicas consideradas puede representarse como:

$$R = \{r_1, r_2, r_3\}$$

donde cada r_i representa una técnica de refactorización que puede aplicarse sobre el estado estructural del sistema.

En el modelo propuesto, la aplicación de una técnica de refactorización transforma el estado estructural actual del sistema E_t en un nuevo estado estructural E_{t+1}, lo cual puede representarse como:

$$E_{t+1} = r_i\,(E_t)$$

La aplicación de cada técnica depende de la evaluación de las precondiciones asociadas a la refactorización correspondiente. Una vez aplicada la transformación, las postcondiciones permiten verificar que el nuevo estado estructural del sistema mantiene la consistencia del diseño y preserva el comportamiento observable del software.

4.3.1 Técnica Herencia de implementación

En el trabajo de Ortiz (2020), la herencia de implementación se identifica como una fuente de deuda técnica cuando viola los principios de diseño Abierto-Cerrado e Inversión de Dependencias. Esta mala práctica genera un alto acoplamiento entre las clases del sistema y sus implementaciones concretas, lo que incrementa los costos de mantenimiento y reduce la reutilización.

En este contexto, la Figura 3 se muestra cómo, en una arquitectura inicial, las clases derivadas dependen directamente del comportamiento definido en una clase base, lo que provoca rigidez estructural y fragilidad ante cambios, limitando la evolución del sistema, ya que cualquier modificación en dicha clase impacta en las clases derivadas. Para mitigar esta problemática, la estrategia de refactorización propuesta consiste en sustituir la herencia de implementación por herencia de interfaz, utilizando el patrón de diseño *Template Method* para invertir el flujo de control (*call* forward a *call back),* lo que permite desacoplar la lógica de negocio de las implementaciones concretas.

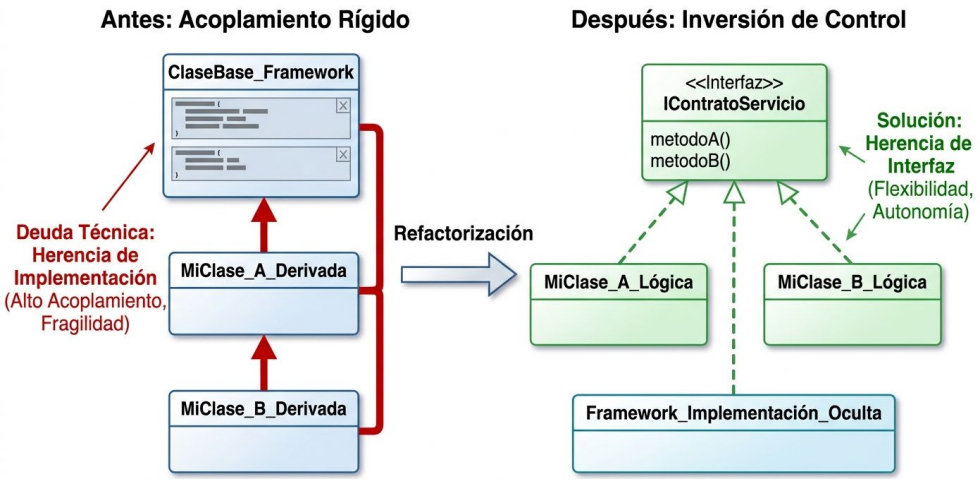


Figura 3. Refactorización de herencia de implementación

4.3.2 Técnica Protección modular

Barón (2020) aborda la deuda técnica desde la perspectiva de la fragilidad del software, causada por una gestión deficiente de la visibilidad de los componentes, definiendo la protección modular como el uso de reglas de visibilidad (calificadores de acceso) para restringir el acceso a los detalles internos de una clase.

En este sentido, la Figura 4 ilustra cómo la ausencia de este mecanismo permite que agentes externos manipulen atributos o métodos que deberían ser privados, facilitando la propagación de defectos entre módulos y comprometiendo el encapsulamiento. Para mitigar esta problemática, el método de refactorización propuesto automatiza la asignación de calificadores de acceso adecuados (como *private* o *protected*), basándose en métricas de uso, lo que fortalece el encapsulamiento y reduce la exposición innecesaria de la lógica interna.

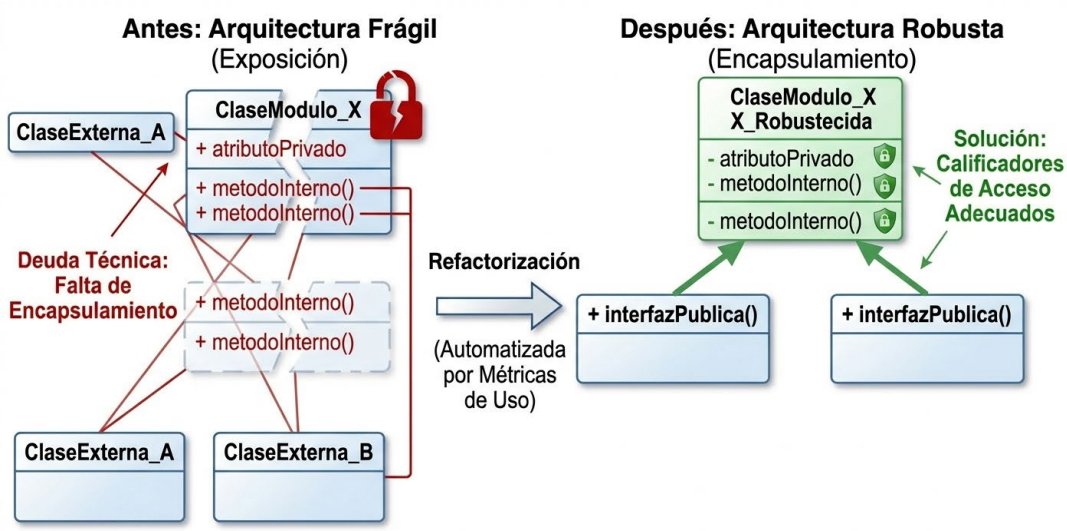


Figura 4. Refactorización de protección modular mediante calificadores de acceso

4.3.3 Técnica Abstracción

Tello (2021) señala que la falta de abstracción constituye uno de los principales obstáculos para la escalabilidad y evolución del software. Cuando las aplicaciones cliente dependen directamente de clases concretas, el diseño se vuelve rígido y difícil de extender. Esta situación provoca que el sistema funcione como un conjunto de componentes fuertemente acoplados, generando duplicación de funcionalidades y una elevada resistencia al cambio. En este contexto, la Figura 5 ilustra cómo la ausencia de mecanismos de abstracción conduce a una arquitectura basada en dependencias concretas, limitando la flexibilidad del sistema.

Para mitigar esta problemática, se propone un método que identifica automáticamente las clases que actúan como proveedores de servicios, con el fin de generar a partir de ellas clases abstractas e interfaces. Esta transformación permite introducir capas de abstracción que reducen la dependencia directa entre los componentes, eliminan la duplicación de código repetido y favorecen una arquitectura más modular, flexible y preparada para la incorporación de nuevas funcionalidades sin necesidad de modificar la base existente.

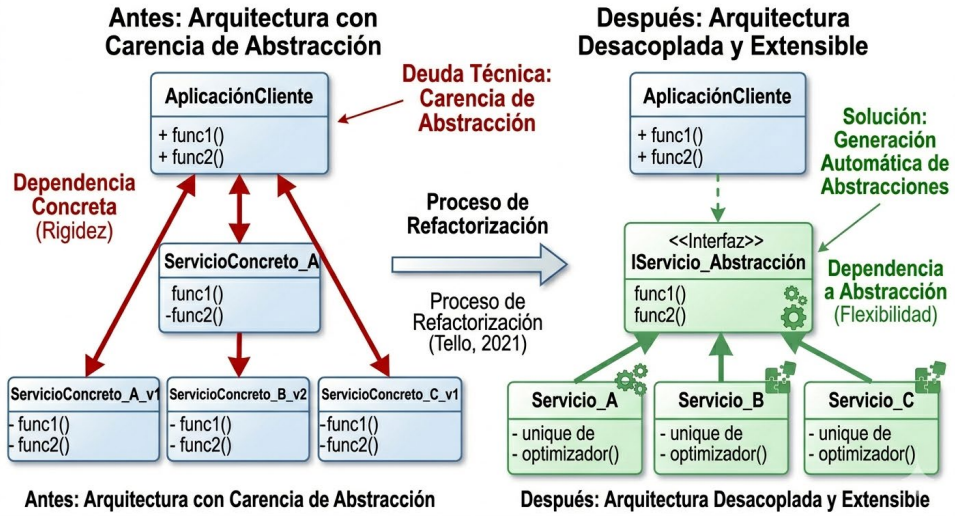


Figura 5. Refactorización de carencia de abstracción mediante clases abstractas e interfaces

4.4 Precondiciones asociadas a las técnicas

Dentro del modelo propuesto, las precondiciones representan el conjunto de condiciones que deben satisfacerse antes de evaluar una técnica de refactorización sobre el estado actual del sistema. Su propósito es determinar si una transformación es viable en una configuración estructural dada y evitar la ejecución de refactorizaciones que puedan comprometer la calidad alcanzada en pasos previos. En la presente investigación, las precondiciones se incorporan como un mecanismo de validación inicial que permite determinar si una técnica de refactorización puede considerarse candidata sobre el estado estructural vigente del sistema y, con ello, controlar su incorporación dentro de la secuencia de ejecución.

En este trabajo, las precondiciones no se limitan únicamente a verificar la existencia de una estructura del sistema susceptible de transformación, sino que además funcionan como un mecanismo de control previo dentro de la secuencia Esto implica que una técnica no debe evaluarse ni incorporarse como candidata solo porque exista una oportunidad estructural para hacerlo, sino porque dicha aplicación resulta compatible con el estado actual del sistema y con la mejora acumulada obtenida hasta ese momento.

Bajo este enfoque, las precondiciones operan sobre el estado actual del sistema E_i y permiten filtrar el conjunto de técnicas disponibles, de manera que únicamente se consideren aquellas cuya evaluación provisional sea válida en ese instante Así, el modelo evita tratar a las técnicas de refactorización como transformaciones universalmente aplicables y establece que su ejecución depende del cumplimiento de condiciones específicas.

En el caso de la técnica orientada a mejorar la protección modular, la precondición consiste en la existencia de al menos un método dentro del sistema. Esta condición permite determinar si existe una base mínima para evaluar y, en su caso, corregir los niveles de visibilidad de los métodos. Por lo tanto, si el sistema no contiene métodos identificables mediante el análisis estructural, la técnica no puede ser considerada como candidata dentro de la secuencia de refactorización.

Para la técnica orientada a mejorar la abstracción, la precondición consiste en la existencia de al menos un método público. Esta condición indica la presencia de comportamiento expuesto que puede ser generalizado o abstraído mediante la introducción de clases abstractas o interfaces. De esta manera, la técnica únicamente se considera aplicable cuando existe una oportunidad estructural que permita elevar el nivel de abstracción del sistema sin alterar su comportamiento observable.

En el caso de la técnica relacionada con herencia de implementación, la precondición se asocia con la existencia de métodos públicos susceptibles de generalización dentro de clases concretas, así como con la presencia de una estructura jerárquica que pueda reorganizarse sin afectar el comportamiento observable del sistema. Esta técnica únicamente puede incorporarse a la secuencia cuando existe una oportunidad estructural real para reducir la dependencia por herencia de implementación y cuando su aplicación no contradice la lógica acumulativa de mejora del modelo.

De manera formal, una precondición puede representarse como una función de validación sobre el estado actual del sistema:

$$Pre\ (R_j, E_i)$$

Donde R_j representa una técnica de refactorización y E_i el estado actual del sistema. Esta función toma el valor de verdadero cuando la técnica R_j puede considerarse candidata en el estado E_i. Esta representación es consistente con la formalización propuesta en esta investigación, donde la precondición general expresa la aplicabilidad de una técnica de refactorización sobre el estado actual del sistema.

A partir de lo anterior, las precondiciones consideradas en esta investigación pueden resumirse de acuerdo con la técnica de refactorización a la que se encuentran asociadas. La Tabla 3 presenta dichas

precondiciones, las cuales funcionan como criterios iniciales para determinar si una técnica puede evaluarse provisionalmente sobre el estado estructural vigente del sistema.

Tabla 3. Precondiciones asociadas a las técnicas de refactorización consideradas

Técnica de refactorización	Precondición
Protección modular	Debe existir al menos un método susceptible de revisión de visibilidad.
Abstracción	Debe existir al menos un método público susceptible de generalización o abstracción.
Herencia de implementación	Debe existir al menos una relación de herencia o dependencia funcional susceptible de reorganización.

4.5 Postcondiciones y efectos estructurales

Las postcondiciones representan el conjunto de condiciones que deben cumplirse después de generar un estado candidato mediante la aplicación provisional de una técnica de refactorización. Su función principal es verificar que la transformación realizada no degrade la calidad estructural previamente alcanzada y que la secuencia de refactorizaciones preserve una mejora acumulativa en el diseño del software.

A diferencia de enfoques tradicionales, donde las refactorizaciones se aplican y evalúan únicamente en términos del resultado final, en este trabajo las postcondiciones se utilizan como un mecanismo de validación inmediata. Esto implica que, después de cada aplicación provisional, se evalúa el estado candidato resultante con el fin de determinar si la refactorización puede formar parte de la secuencia o debe ser descartada.

Dentro del modelo propuesto, las postcondiciones se definen en términos de métricas de calidad estructural. Estas métricas permiten evaluar de manera cuantitativa el impacto de cada refactorización sobre atributos como la protección modular, la abstracción y la herencia de implementación. En este sentido, una refactorización es considerada válida únicamente si, después de su aplicación provisional, los valores de las métricas del estado candidato se mantienen o mejoran con respecto al estado anterior.

Este criterio introduce un mecanismo de control que permite evitar degradaciones intermedias dentro de la secuencia de refactorización. En particular, si la aplicación provisional de una técnica provoca la disminución de alguno de los valores de las métricas evaluadas, dicha técnica es descartada y no forma parte de la secuencia final. De esta manera, se garantiza que el proceso de transformación preserve la calidad acumulada del sistema.

En el caso de la técnica orientada a la protección modular, las postcondiciones establecen que el nivel de protección de los métodos debe mejorar o mantenerse con respecto al estado anterior. Asimismo, se verifica que ninguna de las métricas previamente evaluadas disminuya como consecuencia de la transformación.

Para la técnica orientada a mejorar la abstracción, las postcondiciones garantizan que el nivel de abstracción del sistema aumente o se mantenga. De igual forma, se valida que la aplicación provisional de esta técnica no afecte negativamente otros valores de métricas consideradas dentro del modelo.

En el caso de la técnica basada en herencia de implementación, las postcondiciones se relacionan con la mejora o el mantenimiento de las métricas asociadas a la organización jerárquica del sistema, tales como la flexibilidad y el uso adecuado de la herencia. De igual manera, se verifica que la refactorización no provoque una degradación en las métricas obtenidas previamente.

De manera general, las postcondiciones permiten garantizar que cada transformación contribuya de forma positiva al proceso de mejora estructural del software, asegurando que la calidad del sistema evolucione de manera estable a lo largo de la secuencia de refactorización.

De manera formal, una postcondición puede representarse como una función definida sobre el estado candidato resultante:

$$Post\ (R_j, E'_{i+1})$$

Donde R_j representa la técnica de refactorización aplicada y E'_{i+1} el estado candidato resultante de su aplicación provisional. Esta función toma el valor de verdadero cuando el estado E'_{i+1} cumple con las condiciones de calidad establecidas tras la aplicación de la técnica.
De forma más específica, la validación de las postcondiciones se expresa mediante la siguiente relación:

$$Q\ (E'_{i+1}) \geq Q(E_i)$$

Donde Q representa la función de calidad del sistema. Esta condición asegura que el estado candidato generado por la refactorización evaluada no degrade la calidad acumulada, garantizando que el proceso de transformación preserve o mejore continuamente las propiedades estructurales del software.

A partir de este criterio, las postcondiciones consideradas en esta investigación pueden resumirse de acuerdo con la técnica de refactorización a la que se encuentran asociadas. La Tabla 4 presenta dichas postcondiciones, las cuales funcionan como criterios de validación posterior para determinar si el estado candidato resultante puede aceptarse dentro de la secuencia de refactorización.

Tabla 4. Postcondiciones asociadas a las técnicas de refactorización consideradas

Técnica de refactorización	**Postcondiciones**
Protección modular	La protección de métodos debe mejorar o mantenerse; Ningún valor previo de las métricas debe disminuir
Abstracción	La abstracción debe mejorar o mantenerse; Ningún valor previo de las métricas debe disminuir
Herencia de implementación	La organización jerárquica debe mejorar o mantenerse; Ningún valor previo de las métricas debe disminuir

4.6 Dinámica de transición entre estados

Dentro del modelo propuesto, la dinámica de transición entre estados describe la forma en que se construyen las secuencias de refactorización a partir de la aplicación iterativa de técnicas sobre el sistema.

A partir de un estado inicial E_0, el proceso consiste en identificar el conjunto de técnicas de refactorización aplicables en dicho estado, es decir, aquellas que cumplen con sus precondiciones. Este conjunto define las posibles transformaciones que pueden ejecutarse en ese momento.

Una vez identificado este subconjunto, cada técnica R_j es evaluada de manera independiente mediante la simulación de su aplicación sobre el estado actual E_i, generando un estado candidato E_{i+1}. Este estado es posteriormente validado a través de las postcondiciones y de la función de calidad del sistema. En este proceso, aquellas técnicas cuya aplicación no satisface las postcondiciones o provoca una disminución en la calidad del sistema son descartadas, por lo que no forman parte de la secuencia de refactorización. De esta manera, únicamente se consideran aquellos estados que cumplen con el criterio de preservación de la calidad.

La selección de la técnica a aplicar se realiza a partir del conjunto de estados candidatos válidos, considerando su contribución al mantenimiento o mejora de la calidad del sistema. Este mecanismo permite construir la secuencia de refactorización de manera incremental, asegurando que cada transformación sea consistente con los criterios definidos en el modelo. En caso de que ninguna técnica cumpla simultáneamente con las precondiciones y postcondiciones, el proceso se detiene, indicando que no es posible continuar la secuencia sin comprometer la calidad del sistema.

Desde esta perspectiva, la dinámica de transición entre estados puede representarse mediante un grafo dirigido, en el cual cada nodo corresponde a un estado del sistema y cada arista representa la aplicación de una técnica de refactorización. La Figura 6 ilustra este proceso, mostrando cómo algunas transiciones generan estados válidos, mientras que otras son descartadas cuando no cumplen con las condiciones establecidas por el modelo.

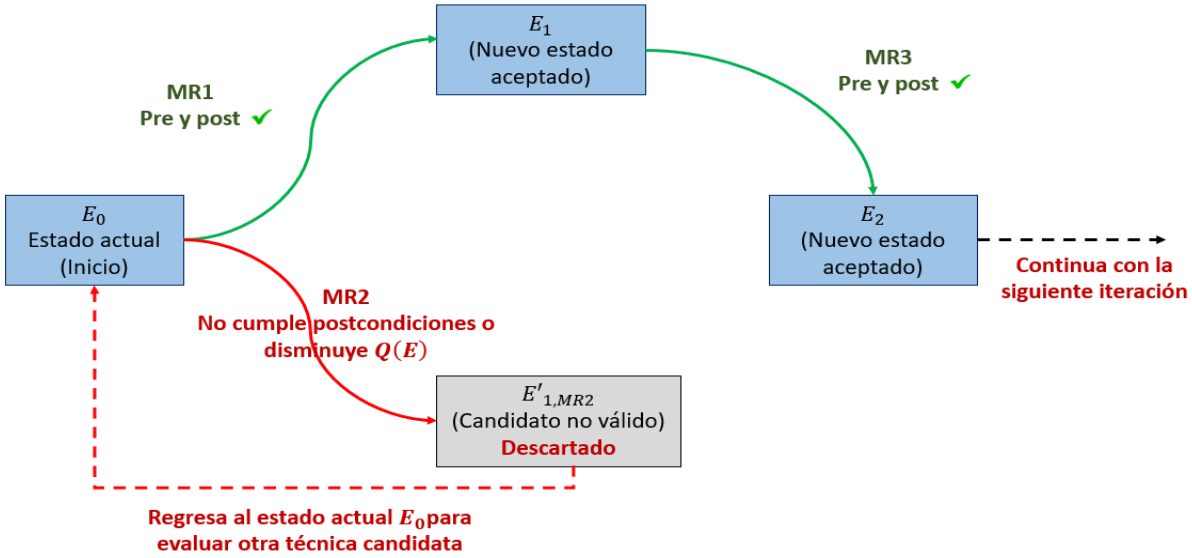


Figura 6. Evaluación y descarte de estados candidatos en el proceso de refactorización

En la Figura 6 se observa que, a partir de un estado inicial E_0, es posible aplicar distintas técnicas de refactorización. Sin embargo, no todas las transiciones conducen a estados válidos. Aquellas transformaciones que cumplen con las precondiciones y postcondiciones generan estados que pueden formar parte de la secuencia.

Por el contrario, las transiciones que no satisfacen estas condiciones son descartadas, lo que impide que dichos estados sean considerados en el proceso de construcción de la secuencia de refactorización.

De esta manera, el proceso puede interpretarse como una exploración controlada del espacio de estados, en la cual únicamente se consideran aquellas trayectorias que cumplen con las condiciones definidas en el modelo.

Capítulo 5.- Algoritmo Greedy adaptado para la secuenciación

5.1 Consideraciones generales del algoritmo

El proceso de secuenciación de refactorizaciones, definido en el capítulo anterior como un problema de exploración dentro de un espacio de búsqueda combinatorio, requiere un mecanismo que permita seleccionar de manera eficiente una trayectoria válida que cumpla con las restricciones del modelo y garantice la preservación de la calidad del sistema.

En este contexto, el uso de algoritmos heurísticos resulta adecuado debido a su capacidad para construir soluciones de forma incremental sin necesidad de evaluar exhaustivamente todas las combinaciones posibles. Entre estos enfoques, el algoritmo Greedy destaca por su simplicidad y eficiencia, ya que selecciona en cada iteración la opción localmente más prometedora.

Sin embargo, cuando se aplica al problema de secuenciación de refactorizaciones, este tipo de algoritmo debe adaptarse para considerar no solo el beneficio inmediato de cada transformación, sino también su compatibilidad con las condiciones previamente establecidas en el sistema. En particular, cada técnica de refactorización debe evaluarse en función del estado actual del sistema, considerando tanto su aplicabilidad como su impacto en la calidad estructural.

A diferencia de un enfoque de búsqueda exhaustiva, en el cual se analizan todas las permutaciones posibles de técnicas, el modelo propuesto se enfoca en la construcción progresiva de una secuencia válida, en donde cada decisión se toma con base en información local, pero sujeta a restricciones globales derivadas de las precondiciones y postcondiciones definidas en el modelo.

Bajo este enfoque, el algoritmo no busca únicamente maximizar la calidad final del sistema, sino garantizar que cada transformación intermedia mantenga o mejore los atributos estructurales evaluados. Esto implica que el proceso de selección se encuentra condicionado por la necesidad de evitar degradaciones en estados intermedios, lo que introduce un criterio adicional de validación en cada iteración.

De esta manera, las consideraciones generales del algoritmo se centran en tres aspectos fundamentales: la selección incremental de refactorizaciones, la validación de su aplicabilidad en función del estado actual del sistema, y la verificación continua de que cada transformación contribuya a una mejora progresiva de la calidad. Estos elementos constituyen la base para la definición del algoritmo Greedy adaptado que se presenta en las siguientes secciones.

5.2 Algoritmo Greedy tradicional

El algoritmo Greedy tradicional es un enfoque heurístico que construye soluciones de manera iterativa mediante la selección de la opción localmente óptima en cada paso. Este tipo de algoritmo se caracteriza por su eficiencia computacional y su simplicidad de implementación, ya que no requiere explorar exhaustivamente el espacio completo de soluciones. En el contexto de la secuenciación de refactorizaciones, el algoritmo Greedy puede emplearse para determinar un orden de aplicación de técnicas seleccionando, en cada iteración, aquella que genere el mayor beneficio inmediato en términos de calidad del sistema. Esta decisión se basa en la evaluación de métricas específicas asociadas al estado actual, sin considerar explícitamente el efecto que dicha elección pueda tener en transformaciones posteriores.

A diferencia de enfoques basados en búsqueda exhaustiva o metaheurísticas, el algoritmo Greedy no construye múltiples secuencias candidatas ni compara resultados globales. En su lugar, avanza de manera directa a través de una única trayectoria dentro del espacio de soluciones, tomando decisiones basadas únicamente en información local disponible en cada iteración.

Este comportamiento implica que el algoritmo selecciona una técnica de refactorización R_j en función de su contribución inmediata a la mejora del sistema, aplicándola sobre el estado actual para generar un nuevo estado. Posteriormente, el proceso se repite sobre el nuevo estado generado, continuando de forma incremental hasta agotar las opciones disponibles o cumplir con un criterio de terminación.

Aunque este enfoque resulta eficiente para reducir la complejidad del problema, su principal característica es que no incorpora mecanismos para validar la coherencia entre decisiones consecutivas. Es decir, cada selección se realiza de manera independiente, sin considerar si una refactorización posterior podría afectar negativamente los beneficios obtenidos previamente.

En consecuencia, el algoritmo Greedy tradicional puede generar secuencias de refactorización que, si bien presentan mejoras en el estado final del sistema, no garantizan una evolución estable durante el proceso de transformación. Esta limitación resulta particularmente relevante en escenarios donde la interacción entre técnicas de refactorización puede provocar efectos contrapuestos en los atributos de calidad del software.

En la Figura 7 se presenta el pseudocódigo del algoritmo Greedy tradicional, donde cada refactorización se evalúa de manera independiente y únicamente se consideran las combinaciones finales, sin analizar si la calidad disminuye durante el proceso de transformación.

```
Inicio

mientras haya tareas pendientes hacer

  seleccionar la mejor opción según el criterio actual

  aplicar la opción seleccionada

fin mientras

Fin
```

Figura 7. Algoritmo Greedy Tradicional (Pseudocódigo).

5.3 Limitaciones del enfoque Greedy clásico

Aunque el algoritmo Greedy tradicional permite construir soluciones de manera eficiente, su aplicación en el contexto de la secuenciación de refactorizaciones presenta limitaciones estructurales que afectan la calidad del proceso de transformación.

Una limitación fundamental radica en la ausencia de control sobre la evolución del sistema en estados intermedios. En el ámbito de la refactorización de software, no es suficiente alcanzar un estado final óptimo; es necesario garantizar que cada transformación preserve la integridad estructural del sistema. Sin embargo, el enfoque Greedy clásico no incorpora mecanismos que permitan verificar este comportamiento, lo que puede derivar en secuencias que introducen inconsistencias temporales.

Asimismo, el algoritmo carece de un modelo explícito que regule la interacción entre técnicas de refactorización. En sistemas reales, las transformaciones no son independientes, ya que la aplicación de una técnica puede modificar las condiciones de aplicabilidad de otras. La falta de un mecanismo que capture estas dependencias limita la capacidad del algoritmo para generar secuencias coherentes.

Otra limitación relevante es la inexistencia de criterios formales de validación durante el proceso de selección. En particular, el algoritmo no establece condiciones que determinen si una refactorización es viable en un estado específico ni verifica si su aplicación compromete propiedades previamente alcanzadas. Esta carencia impide asegurar la consistencia del proceso desde una perspectiva formal.

Adicionalmente, el enfoque Greedy clásico no permite distinguir entre secuencias válidas y secuencias estructuralmente inestables, ya que todas las decisiones se evalúan bajo un mismo criterio local. Esto resulta problemático en escenarios donde la calidad del sistema depende de la preservación simultánea de múltiples atributos, como la modularidad, la abstracción y la herencia de implementación.

En este contexto, las limitaciones identificadas evidencian la necesidad de incorporar mecanismos que permitan controlar el proceso de refactorización de manera más rigurosa, considerando tanto la validez

de cada transformación como su compatibilidad con el estado del sistema. En la siguiente sección se presenta una propuesta que aborda estas limitaciones mediante la integración de precondiciones y postcondiciones dentro del proceso de selección.

5.4 Algoritmo Greedy adaptado

En esta investigación se utiliza el término Greedy adaptado debido a que el algoritmo conserva la lógica de selección local propia del enfoque Greedy, pero se ajusta al problema de secuenciación de refactorizaciones mediante la incorporación de precondiciones, postcondiciones y estados candidatos. Esta adaptación permite que la selección no dependa únicamente de un valor local de calidad, sino también de la validez estructural de cada transición dentro de la secuencia.

Con base en las limitaciones identificadas en el enfoque Greedy tradicional, se propone una adaptación del algoritmo para controlar el proceso de secuenciación de refactorizaciones mediante mecanismos de validación en cada etapa de la transformación. A diferencia del enfoque clásico, en el cual las decisiones se toman principalmente en función de un criterio local inmediato, el algoritmo propuesto introduce un esquema de evaluación incremental que considera tanto la viabilidad de cada refactorización como su impacto en la calidad acumulada del sistema.

El algoritmo Greedy adaptado no se basa en la generación ni evaluación exhaustiva de todas las permutaciones posibles de refactorizaciones. En su lugar, construye la secuencia de manera progresiva, seleccionando en cada iteración únicamente aquellas técnicas que cumplen con las condiciones establecidas en el modelo y que permiten conservar o mejorar la calidad estructural del sistema.

De esta forma, el proceso de selección se transforma en un mecanismo controlado, en el cual cada decisión no solo depende del beneficio inmediato, sino también de su compatibilidad con el estado actual del sistema. Este enfoque permite construir una secuencia consistente y reducir conflictos entre técnicas de refactorización.

En este contexto, la propuesta del algoritmo Greedy adaptado se fundamenta en la incorporación de restricciones formales que regulan la evaluación de cada transformación. El elemento central del algoritmo es la integración de precondiciones y postcondiciones como mecanismos de control durante el proceso de selección de refactorizaciones. Estas condiciones permiten establecer un marco formal que regula la aplicabilidad y validez de cada transformación dentro de la secuencia.

Las precondiciones definen los requisitos que deben cumplirse antes de evaluar una técnica de refactorización sobre el estado actual del sistema. Su función es garantizar que únicamente se consideren transformaciones viables, evitando la evaluación de técnicas que no sean compatibles con la estructura existente.

El algoritmo sigue un proceso iterativo estructurado que permite construir la secuencia de refactorización de manera progresiva:

1. Inicialización: Se calculan las métricas del sistema en su estado inicial E_0 y se identifica el conjunto de refactorizaciones disponibles.

2. Evaluación de precondiciones: En cada iteración se determina qué refactorizaciones pueden aplicarse con base en el estado actual del sistema E_i. Únicamente las técnicas que cumplen sus precondiciones se consideran candidatas.
3. Generación de estados candidatos: Para cada refactorización candidata se genera una aplicación provisional sobre el estado actual, obteniendo un estado candidato E'_{i+1}. Este estado no sustituye todavía al estado actual, sino que permite analizar el posible resultado de aplicar la técnica.
4. Cálculo de métricas del estado candidato: Se calculan las métricas estructurales de cada estado candidato E'_{i+1}, con el fin de estimar el efecto de cada refactorización antes de aceptarla como parte de la secuencia.
5. Evaluación de postcondiciones: Se verifica que cada estado candidato cumpla las postcondiciones definidas y que no degrade la calidad estructural respecto al estado actual. Si algún valor de una métrica disminuye o no se cumplen las condiciones establecidas, el estado candidato se descarta.
6. Construcción del conjunto de candidatos válidos: Los estados candidatos que cumplen las precondiciones, las postcondiciones y el criterio de calidad se almacenan temporalmente como opciones válidas para la iteración actual.
7. Selección de la mejor refactorización disponible: Entre los candidatos válidos, se selecciona la refactorización cuyo estado candidato produce la mejor ganancia local de acuerdo con la función de calidad Q.
8. Aceptación y actualización del estado: El estado candidato seleccionado se acepta como nuevo estado estructural del sistema, es decir, $E_{i+1} = E'_{i+1}$. La refactorización correspondiente se agrega a la secuencia y el nuevo estado se toma como estado actual para la siguiente iteración.
9. Iteración hasta agotar opciones: El proceso continúa mientras exista al menos una refactorización candidata que cumpla precondiciones, postcondiciones y el criterio de preservación de calidad. Cuando no existen candidatos válidos, el algoritmo finaliza y devuelve la secuencia construida.

En la Figura 8 se presenta el pseudocódigo del algoritmo Greedy adaptado, en el cual se integra la evaluación de precondiciones y postcondiciones en cada iteración con el objetivo de garantizar la validez de la secuencia de refactorización generada.

```
Inicio

E_actual = obtener_estado_inicial()
Seq = Ø
R_disponibles = obtener_refactorizaciones_disponibles()

mientras existan R_disponibles hacer

    candidatos_validos = Ø

    para cada refactorización R_i en R_disponibles hacer

        si Pre(R_i, E_actual) = 1 entonces

            E_candidato = δ(E_actual, R_i)
            calcular_metricas(E_candidato)

            si Post(R_i, E_candidato) = 1
               y Q(E_candidato) ≥ Q(E_actual) entonces

                agregar (R_i, E_candidato, Q(E_candidato))
                a candidatos_validos

            fin si

        fin si

    fin para

    si candidatos_validos = Ø entonces
        terminar proceso
    sino
        seleccionar mejor candidato según Q
        E_actual = E_candidato seleccionado
        agregar R_i seleccionada a Seq
        eliminar R_i seleccionada de R_disponibles
    fin si

fin mientras

retornar Seq
```

Figura 8. Algoritmo Greedy adaptado con generación y evaluación de estados candidatos (Pseudocódigo)

Como se observa en la Figura 8, el algoritmo no determina desde el inicio una secuencia completa de refactorizaciones. En su lugar, construye la secuencia de manera incremental a partir del estado actual del sistema. En cada iteración se evalúan las técnicas disponibles, se verifican sus precondiciones y, para aquellas que resultan aplicables, se genera un estado candidato. Cada estado candidato representa una posible evolución estructural del sistema, pero no implica una modificación definitiva del código fuente. Antes de aceptar una técnica, el algoritmo calcula las métricas del estado candidato y verifica sus postcondiciones. Únicamente los estados que conservan o mejoran la calidad estructural son considerados válidos.

Posteriormente, el algoritmo selecciona, entre los estados candidatos válidos, aquel que presenta el mejor valor local de acuerdo con la función de calidad. El estado candidato seleccionado se acepta como nuevo estado actual del sistema y la técnica correspondiente se incorpora a la secuencia. Si ningún candidato cumple las condiciones establecidas, el proceso se detiene y se conserva la secuencia construida hasta ese momento.

De esta manera, el algoritmo no predice a priori el orden completo de refactorización, sino que lo determina paso a paso mediante la evaluación provisional de estados candidatos. Este mecanismo permite decidir cuántas técnicas se aplican y en qué orden, sin comprometer el código fuente antes de validar sus efectos estructurales.

5.5 Formalización matemática del Algoritmo Greedy adaptado

Para formalizar el proceso de selección y ordenamiento de refactorizaciones, se define una función de calidad y un conjunto de precondiciones y postcondiciones. Esta formalización describe cómo el algoritmo Greedy adaptado evalúa, selecciona y valida cada refactorización para construir una secuencia que preserve o mejore la calidad estructural del software.

Con el fin de garantizar precisión y consistencia, se presentan a continuación las definiciones previas, la notación matemática empleada, y las ecuaciones fundamentales que rigen el algoritmo.

Definiciones previas

Sea:

C : conjunto de clases del sistema.
$R = \{R_1, R_2, \ldots, R_n\}$: conjunto de técnicas de refactorización posibles.
S : secuencia parcial de refactorizaciones aplicadas.
$F(S)$: función de calidad basada en métricas.
$PMT(S)$: métrica de protección modular después de aplicar *S*.
$A(S)$: métrica de abstracción después de aplicar *S*.
ΔF_{Ri} : ganancia de calidad obtenida al aplicar la refactorización R_i.
$pre(R_i, S)$: precondición que determina si R_i es aplicable al estado actual.
$post(S \cup \{R_i\})$: postcondición que valida que R_i no degrade la calidad lograda.

Función de calidad

La calidad del sistema después de aplicar la secuencia *S* se define como:

$$F(S) = WPMT * PMT(S) + WA * A(S)$$

Donde los pesos cumplen:

$$WPMT + WA = 1$$

La ganancia de calidad de aplicar una refactorización R_i es:

$$\Delta F_{Ri} = (F(S \cup \{R_i\}) - F(S)$$

Precondiciones y postcondiciones

Precondición general

$pre(R_i, S) \leftrightarrow R_i$

es aplicable en el estado actual del código.

Postcondición general

$post(S \cup \{R_i\}) \leftrightarrow F(S \cup \{R_i\}) \geq F(S)$

Esta condición asegura que ninguna refactorización degrade la calidad acumulada.

Regla de selección del algoritmo Greedy adaptado

En cada iteración, el algoritmo elige la siguiente refactorización:

$a_k = \arg\max_{R_i \in R \setminus S} \{\Delta F_{R_i} | pre(R_i, S) \Delta post(S\{R_i\})\}$

La secuencia final es:

$S^* = [a_1, a_2, \ldots, a_n]$

Restricción lógica de primer orden

Toda refactorización aplicada debe preservar o mejorar la calidad:

$\forall R_i \in R(pre(R_i, S) \wedge post(S \cup \{R_i\}) \rightarrow \Delta F_{R_i} \geq 0)$

Fórmula general del algoritmo

$S^* = \arg\max_{S' \subseteq R} \left(\sum_{R_i \in S'} \blacklozenge F_{R_i} \right)$

sujeto a que:

$\forall R_i \in S' : pre(R_i, S'_{<i}) \wedge post(S'_{\leq i})$,

Donde:

$S'_{<i}$ es la subsecuencia previa al aplicar R_i,
$S'_{\leq i}$ es la subsecuencia que incluye a R_i,

Con el objetivo de facilitar la comprensión de la formalización matemática presentada, a continuación, se resume la notación empleada a lo largo de esta sección. En la Tabla 5 se describen los símbolos utilizados, así como su significado dentro del contexto del algoritmo Greedy adaptado.

Tabla 5. Notación empleada

Símbolo	Significado
R	Conjunto de refactorizaciones disponibles
R_i	Refactorización específica
S	Secuencia acumulada
$F(S)$	Calidad del sistema tras aplicar S
ΔF_{Ri}	Ganancia de calidad al aplicar F_{Ri}
PMT	Protección modular
A	Métrica de abstracción
$pre(R_i, S)$	Precondición de R_i
$post(S \cup \{R_i\})$	Postcondición
a_k	Refactorización elegida en la iteración a_k
S^*	Secuencia generada por el algoritmo Greedy adaptado

Capítulo 6.- Implementación del modelo y del algoritmo

6.1 Descripción general de la aplicación desarrollada

La aplicación desarrollada corresponde a un prototipo funcional orientado a la implementación del modelo propuesto de secuenciación de refactorizaciones, así como del algoritmo Greedy adaptado descrito en el capítulo anterior. Su propósito principal es permitir la ejecución controlada del proceso de refactorización sobre proyectos desarrollados en lenguaje Java, considerando el análisis estructural del código fuente, el cálculo de métricas, la evaluación de precondiciones y postcondiciones, y la generación de una secuencia válida de refactorizaciones.

El prototipo fue diseñado bajo un enfoque basado en microservicios, lo que permitió separar las responsabilidades del sistema en componentes independientes. De esta manera, los procesos relacionados con el análisis del código, el cálculo de métricas, la identificación de técnicas aplicables, la generación de estados candidatos y la evaluación de resultados se implementaron como elementos funcionales diferenciados. Esta organización permite representar de manera práctica la propuesta planteada en los capítulos anteriores, donde cada técnica de refactorización se considera una transformación estructural que modifica el estado actual del sistema sin alterar su comportamiento observable.

Con el propósito de mostrar de manera general el funcionamiento de la aplicación, la Figura 9 presenta la secuencia de ejecución del proceso de refactorización implementado en el prototipo. Esta secuencia inicia con la carga del proyecto Java, continúa con el análisis estructural del código fuente y el cálculo de métricas iniciales, y posteriormente avanza hacia la identificación de técnicas aplicables, la generación de estados candidatos y la evaluación de dichos estados.

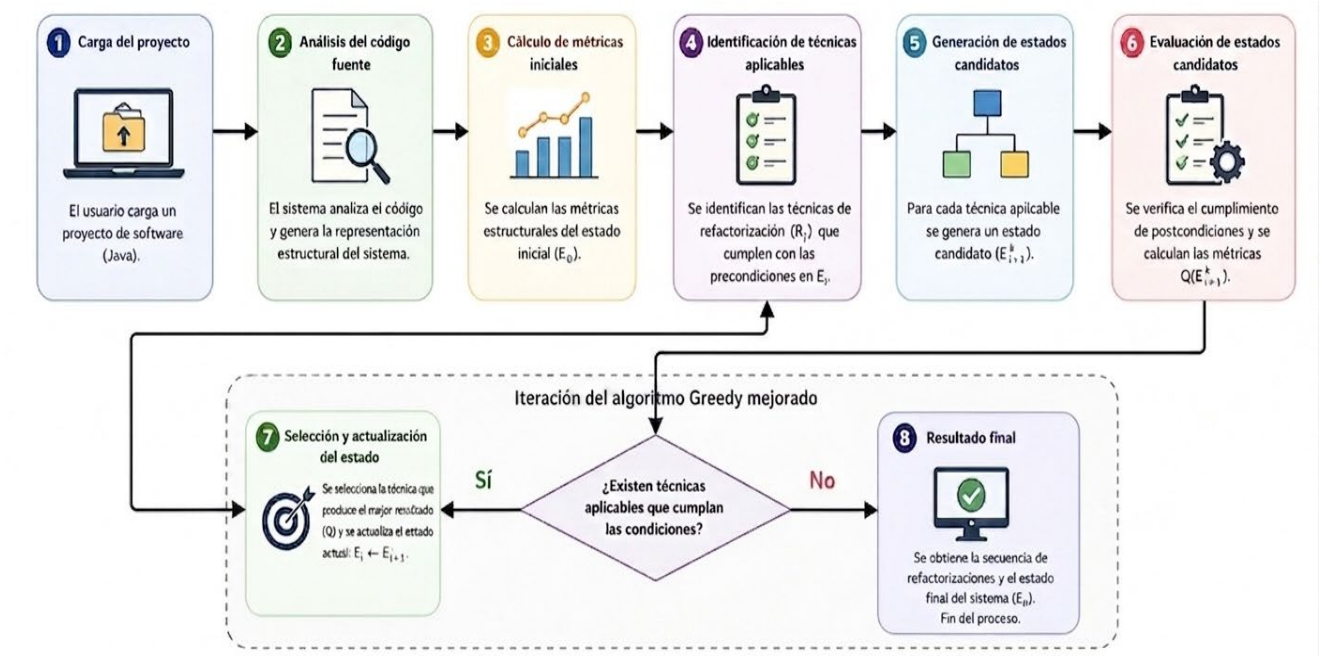


Figura 9. Secuencia de ejecución del proceso de refactorización

A través de una interfaz web, el usuario puede cargar un proyecto de software escrito en Java. Una vez cargado el proyecto, la aplicación realiza el análisis del código fuente con el objetivo de identificar los elementos estructurales relevantes del sistema, tales como clases, métodos, atributos, modificadores de acceso y relaciones entre clases. Con base en esta información, se genera una representación estructural del sistema, la cual sirve como punto de partida para calcular las métricas asociadas a las dimensiones de protección modular, abstracción y herencia de implementación.

Posteriormente, el sistema identifica las técnicas de refactorización que pueden aplicarse sobre el estado actual del proyecto. Para ello, se evalúan las precondiciones definidas para cada técnica, con el fin de determinar si existe una oportunidad válida de transformación. Las técnicas que cumplen con sus precondiciones son consideradas candidatas para la siguiente etapa del proceso. En esta fase, la aplicación genera estados candidatos, los cuales representan posibles configuraciones estructurales del sistema después de aplicar una técnica de refactorización determinada.

Cada estado candidato es evaluado mediante sus respectivas postcondiciones y mediante el cálculo de las métricas estructurales correspondientes. Esta evaluación permite determinar si la transformación propuesta conserva o mejora la calidad del sistema. Si el estado resultante no cumple con las condiciones establecidas o provoca una disminución en los valores de las métricas consideradas, la técnica es descartada y no se incorpora a la secuencia de refactorización.

La parte inferior de la Figura 9 representa la iteración del algoritmo Greedy adaptado. En esta etapa se determina si existen técnicas aplicables que cumplan simultáneamente con las precondiciones y postcondiciones establecidas. En caso afirmativo, se selecciona la técnica que produce el mejor resultado, se actualiza el estado del sistema y el proceso se repite. En caso contrario, se obtiene la secuencia final de refactorizaciones y el estado final del sistema.

El algoritmo Greedy adaptado interviene en la selección de la técnica que produce el mejor resultado en cada iteración. A diferencia de una aplicación directa o desordenada de refactorizaciones, el prototipo no ejecuta una técnica únicamente porque esté disponible, sino que evalúa su impacto sobre el estado actual del sistema y selecciona aquella que genera una mejora estructural válida. Una vez seleccionada la técnica, el estado del sistema se actualiza y el proceso continúa hasta que no existan técnicas aplicables que cumplan simultáneamente con las precondiciones y postcondiciones definidas.

6.2 Arquitectura de la aplicación

La arquitectura de la aplicación desarrollada se fundamenta en un enfoque basado en microservicios, en el cual cada componente implementa una funcionalidad específica dentro del proceso de análisis, evaluación y refactorización del software legado. Esta organización permite mantener una separación clara de responsabilidades, favoreciendo la modularidad, la flexibilidad y la posibilidad de extender el sistema mediante la incorporación de nuevos servicios. Desde el punto de vista conceptual, la solución responde a una coreografía de microservicios de refactorización, debido a que cada servicio participa de manera autónoma dentro del flujo de transformación del sistema. En este esquema, los microservicios no dependen de una estructura monolítica para ejecutar sus funciones, sino que colaboran de acuerdo con las condiciones establecidas por el modelo, principalmente mediante el análisis del código fuente, el cálculo de métricas estructurales, la evaluación de precondiciones, la ejecución de transformaciones y la verificación de postcondiciones.

La Figura 10 presenta el diagrama de contenedores de MRefactoring Web, elaborado con base en el modelo C4. En este nivel se identifican las principales unidades ejecutables que conforman la aplicación: el cliente web, la API de integración, el servicio de análisis de código fuente, los servicios de métricas estructurales y los servicios de refactorización. Estos contenedores representan aplicaciones o servicios ejecutables dentro del sistema, por lo que no deben confundirse con los contenedores Docker utilizados para el despliegue.

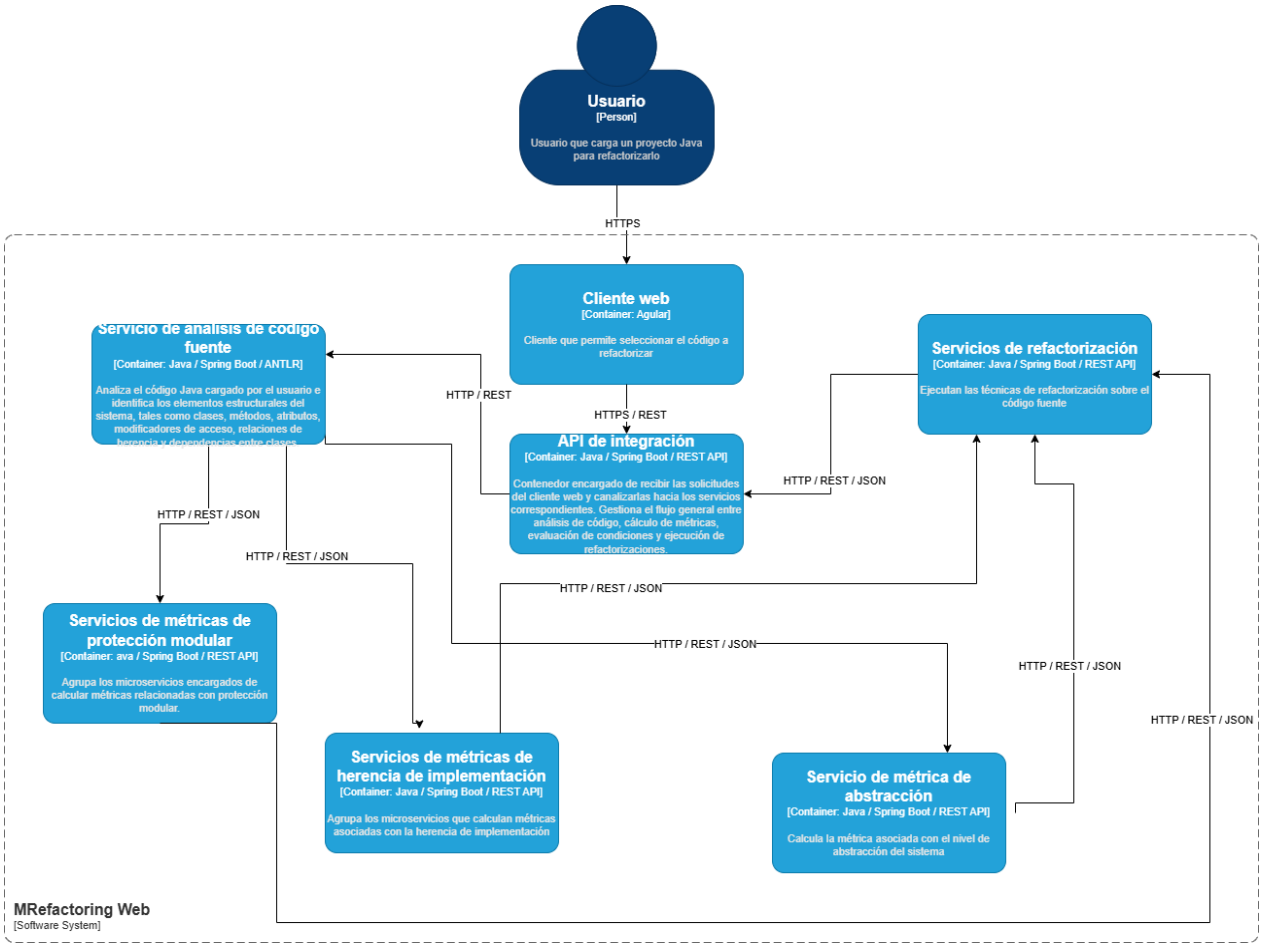


Figura 10. Diagrama de contenedores de MRefactoring Web basado en el modelo C4

La arquitectura se organiza en tres niveles principales:

Capa de presentación. Está representada por el cliente web desarrollado en Angular, mediante el cual el usuario interactúa con la aplicación. Desde esta interfaz es posible cargar el proyecto Java, iniciar el análisis del código fuente, consultar los valores de las métricas estructurales y visualizar los resultados obtenidos durante el proceso de refactorización. Esta capa permite que el usuario observe el comportamiento del sistema sin intervenir directamente en la lógica interna de los microservicios.

Capa de integración. Está implementada mediante una API de integración desarrollada en Spring Boot. Este componente funciona como punto de entrada y salida del sistema, ya que recibe las solicitudes generadas desde el cliente web y canaliza la comunicación inicial hacia los servicios correspondientes. Aunque este elemento facilita la comunicación entre los componentes, su función no debe interpretarse como una orquestación rígida del proceso, debido a que la secuencia de refactorización se determina dinámicamente a partir del estado estructural del sistema, las precondiciones, las postcondiciones y la función de calidad definida en el modelo.

Capa de servicios especializados. Está conformada por los microservicios responsables de ejecutar las funciones principales del sistema. En esta capa se ubican el servicio de análisis de código fuente,

los servicios de métricas estructurales y los servicios de refactorización. El servicio de análisis de código fuente identifica los elementos estructurales del proyecto Java, tales como clases, métodos, atributos, modificadores de acceso, relaciones de herencia y dependencias entre clases. A partir de esta información, los servicios de métricas calculan los valores asociados a protección modular, herencia de implementación y abstracción. Finalmente, los servicios de refactorización ejecutan las transformaciones sobre el código fuente conforme a las condiciones establecidas por el modelo.

Aunque algunas métricas se implementan como servicios independientes dentro del backend, en el diagrama de contenedores se agrupan por dimensión para facilitar la comprensión arquitectónica del sistema. De esta manera, los servicios de métricas de protección modular agrupan las métricas PMFP, PMFPR, PMFF, PM y TPM; los servicios de métricas de herencia de implementación agrupan FHI, FHIJ, FHIAC, FFC y FMFAC; y el servicio de métrica de abstracción calcula A. En este proceso, FHI, FHIJ y FFC se utilizan como métricas intermedias necesarias para obtener FHIAC y FMFAC, mientras que FHIAC y FMFAC son las métricas agregadas consideradas dentro de la evaluación global de la calidad estructural.

La comunicación entre los contenedores se realiza mediante servicios REST, utilizando intercambio de datos en formato JSON. El cliente web se comunica con la API de integración a través de HTTPS, mientras que los servicios internos intercambian información estructural, métricas y resultados mediante peticiones HTTP/REST. Esta forma de comunicación permite que los servicios colaboren de manera desacoplada, manteniendo responsabilidades específicas dentro del proceso de análisis, evaluación y refactorización.

En el flujo general de la arquitectura, el usuario interactúa con el cliente web para cargar un proyecto Java e iniciar el proceso. Posteriormente, la API de integración canaliza la solicitud hacia el servicio de análisis de código fuente, el cual genera la representación estructural del sistema. Esta representación es utilizada por los servicios de métricas para evaluar el estado estructural del software. Con base en los valores obtenidos, las precondiciones, las postcondiciones y la función de calidad, los servicios de refactorización ejecutan las transformaciones aplicables y generan nuevos estados estructurales.

El uso de contenedores Docker permitió empaquetar los servicios de manera independiente, facilitando su ejecución en un entorno local controlado. Esta estrategia contribuyó a mantener aisladas las dependencias de cada componente, evitando conflictos entre servicios y favoreciendo la reproducibilidad del entorno de pruebas. Asimismo, el uso de contenedores permite proyectar la arquitectura hacia escenarios de despliegue más amplios, en los cuales los microservicios podrían ejecutarse de manera distribuida.

La arquitectura implementada permite establecer una relación directa entre el modelo formal y su ejecución práctica. El modelo define los estados, las técnicas, las precondiciones, las postcondiciones y la función de calidad; mientras que la aplicación traduce estos elementos en componentes funcionales que permiten analizar proyectos Java, evaluar posibles transformaciones y construir secuencias de refactorización mediante el algoritmo Greedy adaptado.

6.3 Diseño de los componentes

El diseño de los componentes de la plataforma MRefactoring Web se orienta a implementar el modelo de secuenciación de refactorizaciones mediante unidades funcionales independientes que permiten evaluar, aplicar y validar transformaciones estructurales de manera controlada. A diferencia de la arquitectura general, en este apartado se describen las decisiones de diseño adoptadas para representar el comportamiento del modelo dentro del sistema, particularmente en lo relacionado con la evaluación de técnicas de refactorización sobre distintos estados del software.

Una de las decisiones principales consiste en implementar cada técnica de refactorización como un componente independiente. Esto permite analizar su aplicabilidad de manera individual sobre un estado específico del sistema, generando un estado candidato que puede ser evaluado antes de formar parte de una secuencia de transformación.

Bajo este enfoque, cada componente de refactorización opera sobre una representación estructural del sistema, sin modificar directamente el código fuente original durante la fase de evaluación. Esta decisión facilita la simulación de múltiples alternativas de transformación y permite comparar sus efectos antes de seleccionar una secuencia válida.

De manera complementaria, el cálculo de métricas se realiza mediante servicios encargados de evaluar propiedades estructurales del sistema en cada estado. Estos servicios permiten medir el impacto de una transformación sin intervenir en el proceso de modificación, lo que contribuye a mantener la separación entre análisis y ejecución.

Asimismo, se incorporan servicios de validación cuya función es verificar el cumplimiento de las condiciones definidas en el modelo. Estos servicios permiten evaluar la viabilidad de cada transformación antes de su aplicación y validar el estado resultante, asegurando que únicamente se consideren aquellas alternativas que satisfacen las restricciones establecidas.

En la Tabla 6 se presentan los componentes principales de la plataforma, así como sus responsabilidades, entradas y salidas dentro del proceso de secuenciación. Esta descripción permite identificar la función específica de cada elemento y su participación dentro del modelo propuesto.

Tabla 6. Componentes principales de MRefactoring Web

Componente	Responsabilidad principal	Entrada	Salida
Cliente web	Permitir la interacción del usuario con la plataforma y la ejecución de funcionalidades del sistema	Proyecto Java y solicitudes del usuario	Visualización de métricas, resultados y secuencias de refactorización

Componente de integración	Coordinar el flujo de ejecución del sistema, gestionando el análisis, evaluación y aplicación de técnicas de refactorización	Solicitudes del cliente web y representación estructural del sistema	Control del flujo de ejecución y envío de solicitudes a los servicios
Servicios de métricas	Evaluar propiedades estructurales del sistema en cada estado	Representación estructural del sistema	Valores de métricas asociados al estado evaluado
Servicio de refactorización de protección modular	Evaluar y ajustar los calificadores de acceso de los métodos para mejorar el encapsulamiento	Estado actual del sistema	Estado candidato con modificaciones en la visibilidad de métodos
Servicio de refactorización de abstracción	Generar estructuras de abstracción mediante la introducción de interfaces o clases abstractas	Estado actual del sistema	Estado candidato con mayor nivel de abstracción
Servicio de refactorización de herencia de implementación	Reorganizar jerarquías de clases para mejorar la reutilización y reducir el acoplamiento	Estado actual del sistema	Estado candidato con mejor organización jerárquica
Servicios de validación	Verificar el cumplimiento de precondiciones y postcondiciones en cada transformación	Estado actual y estado candidato, junto con valores de métricas	Aprobación o descarte de la transformación

6.3.1 Casos de uso del sistema

La interacción del usuario con la plataforma MRefactoring Web se describe mediante un conjunto de casos de uso que representan las principales funcionalidades del sistema. Estos casos de uso permiten identificar las operaciones disponibles y el comportamiento del sistema ante cada solicitud.

A nivel general, el usuario interactúa con la plataforma mediante la carga de un proyecto desarrollado en lenguaje Java, a partir del cual puede ejecutar el análisis estructural del código fuente, calcular métricas y aplicar el proceso de secuenciación de refactorizaciones. Estas acciones reflejan las funcionalidades principales que soporta el sistema. Entre los casos de uso más relevantes se encuentran la carga del proyecto, el cálculo de métricas estructurales y la ejecución del proceso de refactorización. En cada uno de estos casos, el sistema procesa la información proporcionada y genera resultados que pueden ser visualizados a través de la interfaz. Con el objetivo de mantener claridad en la representación, en esta sección se presentan los casos de uso a nivel general. La especificación detallada de los casos de uso asociados al cálculo individual de métricas y a las técnicas de refactorización se incluye en el Anexo B.

La Figura 11 presenta el diagrama general de casos de uso de la plataforma MRefactoring Web, en el cual se muestran las principales funcionalidades disponibles para el usuario y las relaciones necesarias para su ejecución. El usuario puede cargar un proyecto, calcular métricas estructurales y ejecutar una secuencia de refactorización.

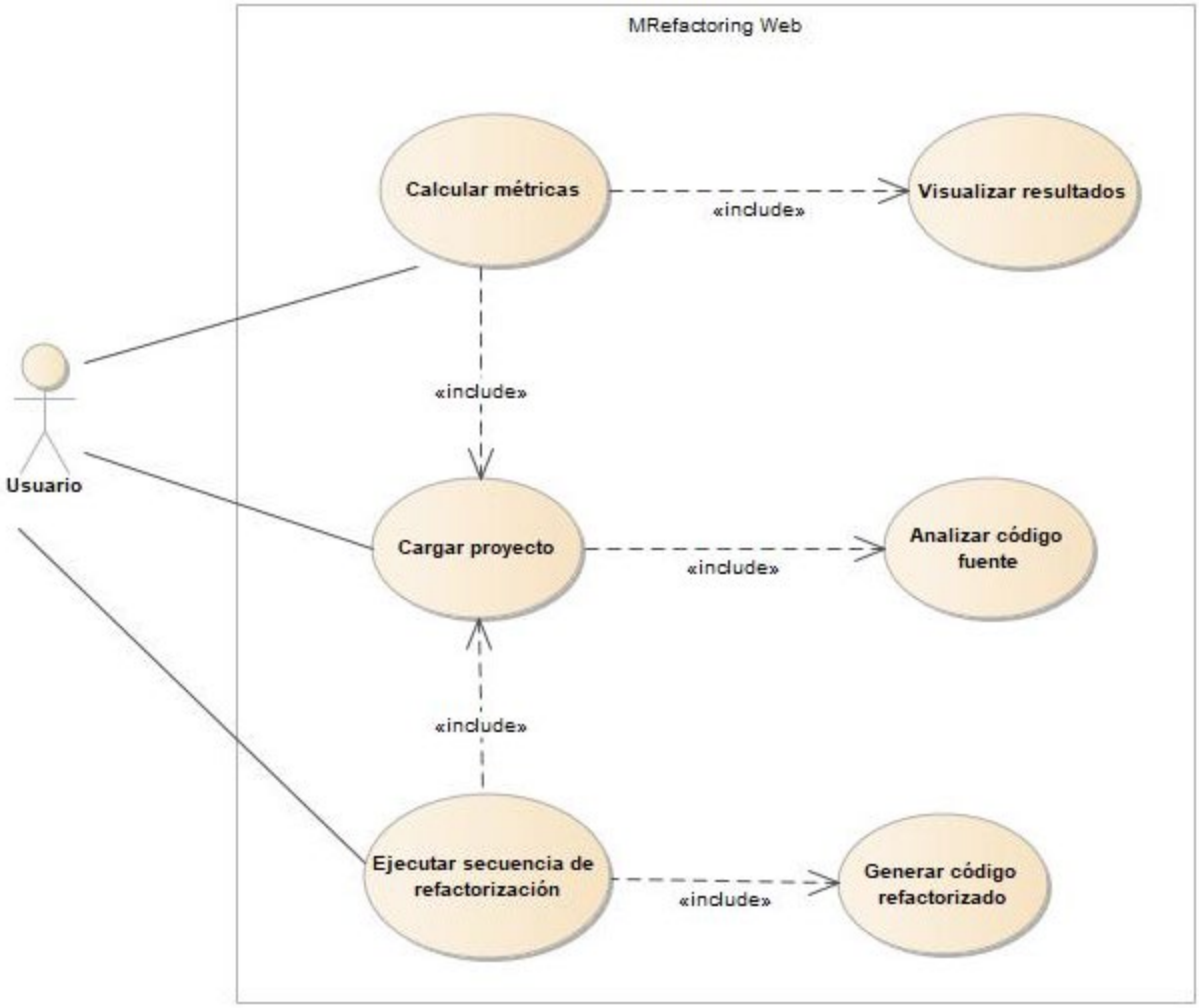


Figura 11. Diagrama general de casos de uso de la plataforma MRefactoring Web.

Capítulo 7.- Validación del modelo y del algoritmo

7.1 Objetivo de la validación

El objetivo de este capítulo es validar el comportamiento del modelo propuesto y del algoritmo Greedy adaptado para la secuenciación de refactorizaciones. La validación se centra en analizar si el algoritmo permite construir secuencias de refactorización que mantengan o mejoren la calidad estructural del software durante cada etapa del proceso, evitando que una técnica aplicada posteriormente degrade los beneficios obtenidos por una transformación previa.

La validación se orienta a comprobar tres aspectos principales. El primero consiste en verificar que el modelo propuesto permite representar el proceso de refactorización como una sucesión de estados estructurales, donde cada transición se encuentra condicionada por el cumplimiento de precondiciones y postcondiciones. El segundo aspecto consiste en evaluar si el algoritmo Greedy adaptado selecciona secuencias de refactorización distintas a las generadas por el Greedy tradicional, debido a que considera no solo el resultado final, sino también la estabilidad de la calidad en cada paso. El tercer aspecto consiste en analizar si las métricas estructurales utilizadas permiten evidenciar la mejora o preservación de la calidad del sistema después de aplicar las secuencias generadas.

En este sentido, la validación no se limita a comprobar la ejecución técnica de una aplicación, sino que busca analizar el comportamiento del modelo y del algoritmo frente a proyectos Java reales. Para ello, se emplearon sistemas seleccionados del repositorio GitHub, con diferentes tamaños y características estructurales, con el propósito de observar el desempeño del algoritmo en escenarios heterogéneos.

La validación se realizó considerando las tres dimensiones abordadas en esta investigación: protección modular, abstracción y herencia de implementación. Estas dimensiones fueron evaluadas mediante

métricas estructurales que permiten caracterizar el estado inicial del sistema, el impacto de las técnicas de refactorización y el estado final alcanzado después de aplicar la secuencia generada.

7.2 Sistemas de estudio

Para validar el modelo y el algoritmo propuesto se seleccionaron 30 proyectos desarrollados en lenguaje Java, extraídos del repositorio GitHub. La selección de estos sistemas tuvo como propósito contar con un conjunto diverso de aplicaciones que permitiera observar el comportamiento del algoritmo en diferentes escenarios de análisis y refactorización.

Los sistemas seleccionados presentan variaciones en tamaño, estructura, cantidad de clases, métodos, atributos, relaciones entre clases y estilo de implementación. Esta diversidad permitió evaluar el comportamiento del algoritmo Greedy adaptado tanto en sistemas pequeños como en proyectos con mayor complejidad estructural. La inclusión de sistemas con características distintas contribuye a fortalecer la validación, ya que permite analizar si el modelo mantiene su funcionamiento bajo diferentes configuraciones del código fuente.

Para caracterizar los sistemas de estudio no solo se consideró el tamaño del código fuente, sino también los elementos estructurales relacionados con las precondiciones de las técnicas de refactorización evaluadas. Por esta razón, además del número de archivos Java, clases, métodos, atributos y líneas de código fuente, se identificaron métodos públicos, clases abstractas, interfaces, enumeraciones, relaciones de herencia mediante *extends* y relaciones de implementación mediante *implements*. Estos elementos permiten establecer la condición inicial de cada sistema y justificar la aplicabilidad de las técnicas de refactorización consideradas en esta investigación.

Los criterios considerados para seleccionar los sistemas de estudio fueron los siguientes:

a) Que el proyecto estuviera desarrollado principalmente en lenguaje Java.
b) Que el código fuente pudiera analizarse estructuralmente para identificar clases, métodos, atributos y relaciones entre clases.
c) Que el sistema contara con elementos suficientes para calcular las métricas relacionadas con protección modular, abstracción y herencia de implementación.
d) Que el proyecto permitiera aplicar al menos una de las técnicas de refactorización consideradas en esta investigación.
e) Que el sistema pudiera analizarse antes y después de la refactorización, con el propósito de comparar los valores métricos obtenidos.

La Tabla 7 presenta las características estructurales iniciales de los sistemas considerados en la validación. En ella se muestran los proyectos Java utilizados para analizar el comportamiento del algoritmo Greedy tradicional y del algoritmo Greedy adaptado. Asimismo, se incluyen los elementos estructurales que permiten justificar la evaluación inicial de precondiciones antes de aplicar las secuencias de refactorización.

Tabla 7. Características estructurales iniciales de los sistemas Java considerados para la validación

No.	Programa	Clases	Interfaces	Clases abstractas	Métodos	Constructores	Métodos public	Métodos private	Métodos protected	Métodos friendly	Métodos static	Métodos abstractos*	Atributos aprox.	Rel. extends	Rel. implements
1	Blue-Block-master	13	0	0	104	10	87	6	0	0	10	0	75	4	1
2	chess-system-java-master	16	0	2	88	14	50	18	6	0	9	1	41	9	0
3	CompanyManagementSystem-master	14	0	3	28	2	26	0	0	0	3	2	18	6	0
4	CustomerManagerApp-master	6	1	0	28	3	24	1	0	0	12	0	21	0	1
5	FileOperations-master	10	1	0	22	4	18	0	0	0	2	0	9	0	5
6	HospitalProject-master	32	3	3	176	30	144	0	0	0	15	2	79	6	6
7	InventoryManagementSystem-master	6	0	0	47	6	23	0	7	0	9	0	16	0	0
8	Java_Console-Course-Management-System-main	12	3	1	56	4	43	9	0	0	15	0	36	4	3
9	Java_Imobiliaria-master	5	0	0	16	2	14	0	0	0	11	0	9	2	0
10	Java-Console-Card-Game-master	10	0	0	40	5	36	0	0	0	15	0	37	0	0
11	Juego-El-ahorcado-en-Java-master	3	0	0	6	1	5	0	0	0	2	0	8	0	0
12	Library_Management_System_11-Jan-2022-master	5	0	0	37	4	16	3	0	0	31	0	35	0	1
13	Movie-Service-Review-Java-main	5	0	0	20	3	17	0	0	0	1	0	14	0	1
14	online-quiz-application-in-java-with-jdbc-using-swing-login-registration-time...	6	0	0	24	1	14	2	6	2	2	0	54	5	0
15	Password-Generator	4	0	0	24	4	7	8	0	9	1	0	28	0	0
16	PasswordManager-main	21	3	2	76	3	54	4	0	18	31	2	24	4	4
17	Polinom	10	0	0	36	8	24	0	5	7	1	0	32	4	0
18	qurbani-animals-market-main	15	4	3	104	6	96	2	0	6	30	1	88	3	11
19	radio	6	0	0	48	8	40	0	0	8	1	0	15	0	0
20	real-estate-console-app-master	8	0	0	36	1	34	1	0	1	11	0	29	0	0
21	SchoolManagement-master	13	1	2	48	4	43	0	0	5	3	4	25	4	2
22	shapp-master	6	1	0	30	4	25	0	1	4	1	0	18	1	0
23	Simple-Java-Calculator-master	4	0	0	35	1	10	2	0	23	1	0	47	0	1
24	TetrisForDesktopWithJava-master	9	6	0	87	7	56	22	0	9	2	0	69	4	0
25	tic-tac-toe-console-master	7	0	0	17	0	1	2	0	14	17	0	58	0	0
26	tiny-compiler-master	13	0	4	88	12	39	37	0	12	1	0	76	3	0
27	treinaweb-java-oo-master	12	3	1	45	6	27	0	0	18	1	1	12	6	5
28	UniversalCalculator-master	15	2	0	57	2	20	24	2	11	3	0	52	7	4
29	Llaveros-master	4	0	0	15	2	13	0	0	0	7	0	9	0	0
30	AI-Chat-Bot-master	6	0	0	38	4	34	0	0	0	4	0	22	6	6

La información presentada en la Tabla 7 permite observar que los sistemas seleccionados poseen diferentes niveles de tamaño y complejidad estructural. La cantidad de archivos Java, clases, métodos y atributos permite estimar la magnitud de cada programa. Por su parte, los calificadores de alcance de los métodos, las clases abstractas, las interfaces y las relaciones extends e implements permiten identificar condiciones relevantes para la aplicación de las técnicas de refactorización. En particular, los métodos públicos son relevantes para la técnica de protección modular y para la identificación de carencia de abstracción; las relaciones de herencia permiten observar escenarios asociados con herencia de implementación; mientras que las interfaces y clases abstractas permiten reconocer el nivel inicial de abstracción presente en cada sistema.

7.3 Plan de validación

El plan de validación se diseñó con el propósito de evaluar el comportamiento del modelo propuesto y del algoritmo Greedy adaptado mediante su implementación en la aplicación MRefactoring Web, utilizando un conjunto de proyectos Java. Para ello, se definió un proceso experimental dividido en etapas, de manera que cada proyecto pudiera analizarse bajo las mismas condiciones.

La primera etapa consistió en la carga y análisis del proyecto Java. En esta fase, cada proyecto fue procesado con el fin de identificar sus elementos estructurales principales, tales como clases, métodos, atributos, relaciones de herencia, modificadores de acceso y dependencias entre componentes. Esta información permitió construir el estado inicial correspondiente.

La segunda etapa consistió en el cálculo de métricas iniciales. Estas métricas permitieron establecer una línea base para cada proyecto antes de aplicar cualquier técnica de refactorización. La línea base fue necesaria para comparar los valores posteriores y determinar si las transformaciones aplicadas mantenían, mejoraban o degradaban la calidad estructural del software.

La tercera etapa consistió en la identificación de técnicas de refactorización aplicables. A partir de las características estructurales identificadas en los sistemas de estudio, se evaluó el cumplimiento inicial de las precondiciones asociadas con los métodos de refactorización considerados. Esta evaluación permitió determinar si cada sistema contenía los elementos mínimos necesarios para aplicar las técnicas de protección modular, herencia de implementación y carencia de abstracción. Debido a que el algoritmo Greedy adaptado requiere validar si una técnica puede aplicarse sobre el estado actual del sistema, cuando una técnica no cumplía las precondiciones establecidas, el estado candidato era descartado y el algoritmo continuaba evaluando otra alternativa de secuenciación. En caso contrario, la técnica se consideraba candidata para generar un nuevo estado.

La cuarta etapa consistió en la generación de estados candidatos. Cada técnica aplicable fue simulada o ejecutada sobre el estado actual del sistema, generando un posible estado resultante. Estos estados no fueron aceptados automáticamente, ya que debían ser evaluados mediante postcondiciones y métricas estructurales.

La quinta etapa consistió en la evaluación de postcondiciones. En esta fase se verificó que cada estado candidato cumpliera las condiciones estructurales establecidas para la técnica aplicada y que conservara o mejorara la calidad alcanzada en el estado anterior. Si el estado resultante incumplía

alguna postcondición o producía una degradación en las dimensiones evaluadas, era descartado y la técnica no se incorporaba a la secuencia.

La sexta etapa consistió en la selección de la técnica mediante el algoritmo Greedy adaptado. Entre los estados candidatos válidos, el algoritmo seleccionó aquel que presentaba el mejor resultado local de acuerdo con la función de calidad definida para las dimensiones de protección modular, herencia de implementación y abstracción. Posteriormente, el estado del sistema se actualizó y el proceso se repitió hasta que no existieran técnicas aplicables que cumplieran simultáneamente con las precondiciones y postcondiciones establecidas.

La séptima etapa consistió en la comparación con el algoritmo Greedy tradicional. Para cada sistema se registró la secuencia seleccionada por ambos enfoques. Esta comparación permitió identificar diferencias en la forma en que cada algoritmo construye la secuencia de refactorización.
El plan de validación permitió organizar el proceso experimental de forma sistemática, garantizando que todos los sistemas fueran evaluados bajo el mismo procedimiento.

Para evaluar los resultados obtenidos durante la validación, se calcularon las métricas estructurales consideradas en el modelo propuesto. Estas métricas se mencionan de manera general en la sección 4.1 y se describen con mayor detalle en el Anexo A, donde se presenta su propósito, forma de cálculo e interpretación. Su aplicación permitió analizar las dimensiones de protección modular, herencia de implementación y abstracción durante el proceso de refactorización.

Aunque se calcularon las métricas específicas correspondientes a cada dimensión, la función de calidad $Q(E_t)$ retomó únicamente una métrica representativa de cada una: TPM para protección modular, $1 - FHIAC$ para herencia de implementación y A para abstracción, de acuerdo con la definición establecida previamente en el modelo.

Las demás métricas se utilizaron como valores específicos, base o intermedios necesarios para obtener las métricas representativas, pero no se incorporaron directamente en $Q(E_t)$. De esta manera, se evitó asignar un peso mayor a una dimensión por la inclusión simultánea de varias métricas relacionadas, y el valor de calidad obtenido permitió comparar los estados candidatos y guiar la selección incremental realizada por el algoritmo Greedy adaptado.

7.3.1 Convención de nombres

La convención de nombres mostrada en la Figura 12 se utiliza a lo largo del proceso de validación realizado mediante la aplicación MRefactoring Web.

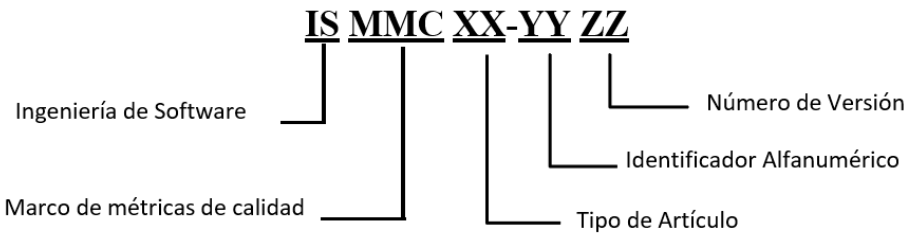


Figura 12. Convención de nombres.

Tipo de Artículo	
01	Módulos de programa.
02	Programas de control.
05	Casos de prueba.

Programas de Control	
ISMMC01 - nn	Módulos de programa

Documentación de pruebas	
ISMMC03 - YYZZ	Plan de pruebas.
ISMMC05 - YYZZ	Especificación de casos de prueba.

Introducción

El plan de pruebas fue diseñado para evaluar el sistema MRefactoring Web. Este sistema implementa un modelo formal de una secuencia de ejecución de métodos de refactorización, con el fin de mejorar la calidad y mantenibilidad de código legado escrito en lenguaje Java.

La correcta ejecución de este plan de pruebas permitirá identificar posibles fallos o inconsistencias en el sistema MRefactoring Web, con el fin de corregirlos antes de su implementación final. Además, contribuirá a verificar que el sistema cumpla con los requerimientos definidos y que sus funcionalidades respondan adecuadamente a los objetivos establecidos.

Objetivo

Asegurar el correcto desempeño del sistema MRefactoring Web mediante una serie de pruebas y evaluaciones integrales, con el propósito de validar el cumplimiento de los requisitos establecidos.

Artículos de prueba

Módulo	Función	Nomenclatura
Cálculo de métricas	Cálculo de métricas: PMP, PMPR, PMF, PM, TPM, FHI, FHIJ, FHIAC, FMFAC y A	ISSRWC01 – 01
Refactorización de Código	Refactorización de código escrito en lenguaje Java.	ISSRWC01 – 02

Características que no serán evaluadas

Los casos de prueba no incluirán todas las posibles construcciones sintácticas y combinaciones de éstas, para código escrito en lenguaje Java.

La validación no contempla la corrección semántica completa del programa Java. Se asume que los proyectos de entrada no presentan errores léxicos o sintácticos que impidan su análisis estructural.

Casos de prueba

ID y nombre	CP1 - Verificación de la selección de archivos en lenguaje Java
Ejecutor	M.C. Nelida Barón Pérez
Cliente	Dr. René Santaolaya Salgado
Fecha y hora	28 /10/2024
Prioridad	Alta
Objetivo: Verificar que el sistema MRefactoring Web permita la selección únicamente de archivos escritos en lenguaje Java para el proceso de refactorización.	
Precondiciones: 1. Ingreso al Sistema MRefactoring Web. 2. Seleccionar la opción “Seleccionar archivos”.	
Ejecución: 1. Acceder a la interfaz del sistema MRefactoring Web. 2. Navegar hasta la sección de selección de archivos. 3. Intentar seleccionar un archivo con una extensión no correspondiente al lenguaje Java, por ejemplo, un archivo de texto (.txt) o una imagen (.jpg). 4. Verificar que el sistema MRefactoring Web muestra un mensaje de error indicando que solo se permiten archivos escritos en lenguaje Java. 5. Intentar seleccionar un archivo con una extensión válida para el lenguaje Java, por ejemplo, un archivo con extensión .java. 6. Verificar que el sistema MRefactoring Web acepta la selección del archivo y lo muestra en la lista de archivos a refactorizar. 7. Repetir los pasos del 3 al 6 con varios archivos de diferentes extensiones para asegurarse de que únicamente se permita la selección de archivos en lenguaje Java. 8. Registrar los resultados obtenidos.	
Valores/Datos de entrada: archivos con extensión .java	
Criterio pasa / no pasa: Este caso de prueba será considerado válido si el sistema MRefactoring Web permite únicamente la selección de archivos con extensión .java y rechaza archivos con extensiones diferentes.	

ID y nombre	CP2 - Verificación del cálculo de métricas
Ejecutor	M.C. Nelida Barón Pérez
Cliente	Dr. René Santaolaya Salgado
Fecha y hora	29/10/2024
Prioridad	Alta
Objetivo: Verificar que el sistema MRefactoring Web realice correctamente el cálculo de las métricas.	
Precondiciones: Haber seleccionado un programa escrito en lenguaje Java.	
Ejecución: 1. Acceder a la interfaz del sistema MRefactoring Web. 2. Seleccionar un proyecto escrito en lenguaje Java. 3. Aplicar las métricas de calidad. 4. Verificar que el sistema MRefactoring Web muestra los valores correctos de las métricas PMP, PMPR, PMF, PM, TPM, FHI, FHIJ, FHIAC, FMFAC y A. 5. Realizar pruebas adicionales con diferentes códigos de entrada para asegurarse de que el cálculo de métricas se realiza correctamente en diferentes casos. 6. Registrar los resultados obtenidos.	
Valores/Datos de entrada: Aplicación escrita en lenguaje Java	
Criterio pasa / no pasa: El criterio de evaluación de esta prueba se realizará tomando en cuenta los resultados obtenidos manualmente. Este caso de prueba será considerado válido cuando los resultados obtenidos manualmente coincidan con los resultados calculados de forma automática.	

ID y nombre	CP3 - Verificación de la selección del método de refactorización
Ejecutor	M.C. Nelida Barón Pérez
Cliente	Dr. René Santaolaya Salgado
Fecha y hora	29/10/2024
Prioridad	Alta
Objetivo: Verificar que el sistema MRefactoring Web permita al usuario seleccionar el método de refactorización a aplicar al código de entrada.	
Precondiciones: Haber seleccionado un programa escrito en lenguaje Java.	
Ejecución: 1. Seleccionar un proyecto escrito en lenguaje Java. 2. Aplicar las métricas de calidad. 3. Revisar las opciones disponibles de métodos de refactorización. 4. Seleccionar uno de los métodos de refactorización disponibles. 5. Verificar que el sistema MRefactoring Web registra la selección del método correctamente. 6. Repetir los pasos 1 a 5 con diferentes métodos de refactorización y probar las distintas combinaciones posibles de selección. 7. Registrar los resultados obtenidos.	
Valores/Datos de entrada: Aplicación escrita en lenguaje Java	
Criterio pasa / no pasa: Este caso de prueba será considerado válido si MRefactoring Web permite seleccionar los métodos de refactorización disponibles y registrar correctamente las distintas combinaciones de ejecución.	

ID y nombre	CP4 - Verificar la refactorización de código por los tres métodos de refactorización
Ejecutor	M.C. Nelida Barón Pérez
Cliente	Dr. René Santaolaya Salgado
Fecha y hora	29/10/2024
Prioridad	Alta
Objetivo: Verificar que el sistema MRefactoring Web realiza la refactorización de código correctamente utilizando los tres métodos de refactorización disponibles.	
Precondiciones: 1. Ingreso al Sistema MRefactoring Web. 2. Seleccionar la opción "Seleccionar archivos".	
Ejecución: 1. Acceder a la interfaz del sistema MRefactoring Web. 2. Seleccionar un archivo con extensión Java para refactorizar. 3. Aplicar las métricas de calidad. 4. Seleccionar uno o más métodos de refactorización. 5. Ejecutar el proceso de refactorización. 6. Verificar que el código se ha refactorizado correctamente según los métodos seleccionados. 7. Repetir los pasos 1 al 6 con diferentes aplicaciones escritas en lenguaje Java hasta verificar que el proceso de refactorización se realiza con éxito al seleccionar uno o más métodos de refactorización. 8. Registrar los resultados obtenidos.	
Valores/Datos de entrada: Aplicación escrita en lenguaje Java	
Criterio pasa / no pasa: Este caso de prueba será considerado válido si el sistema MRefactoring Web realiza correctamente la refactorización aun cuando solo se seleccione uno o exista una combinación de ejecución de los métodos de refactorización.	

ID y nombre	CP5 - Verificar si el sistema muestra alertas en caso de error o al finalizar un proceso de manera exitosa
Ejecutor	M.C. Nelida Barón Pérez
Cliente	Dr. René Santaolaya Salgado
Fecha y hora	30/10/2024
Prioridad	Baja
Objetivo: Verificar que el sistema MRefactoring Web muestra alertas al usuario en caso de error durante la refactorización o al finalizar un proceso con éxito.	
Precondiciones: Haber llevado a cabo al menos una de las acciones proporcionadas por el sistema, como la selección de archivos, el cálculo de métricas y la refactorización de código	
Ejecución: 1. Acceder a la interfaz del sistema MRefactoring Web. 2. Realizar una acción que genere un error durante el proceso de refactorización, por ejemplo, seleccionar un archivo no válido. 3. Verificar que el sistema MRefactoring Web muestra una alerta indicando el error ocurrido. 4. Realizar una refactorización exitosa. 5. Verificar que el sistema MRefactoring Web muestra una alerta informando al usuario que el proceso ha finalizado exitosamente. 6. Registrar los resultados obtenidos.	
Valores/Datos de entrada: Aplicación escrita en lenguaje Java	
Criterio pasa / no pasa: Este caso de prueba será considerado válido si MRefactoring Web muestra una alerta de error cuando ocurre una situación no válida y una notificación de éxito cuando el proceso concluye correctamente.	

7.4 Escenarios de ejecución

Para validar el comportamiento del modelo propuesto y del algoritmo Greedy adaptado, se definieron escenarios de ejecución basados en las combinaciones posibles de las tres técnicas de refactorización consideradas en esta investigación: protección modular, abstracción y herencia de implementación. Debido a que el modelo evalúa el efecto del orden de aplicación de las técnicas, fue necesario analizar las distintas secuencias posibles y observar el impacto que cada una produce sobre las métricas estructurales del sistema.

La Figura 13 presenta el árbol de combinaciones utilizado para representar las posibles secuencias de ejecución de las técnicas de refactorización. Cada trayectoria del árbol corresponde a una secuencia distinta, identificada como S1, S2, S3, S4, S5 o S6. Estas secuencias permiten analizar cómo cambia la calidad estructural del sistema cuando las mismas técnicas se aplican en distinto orden.

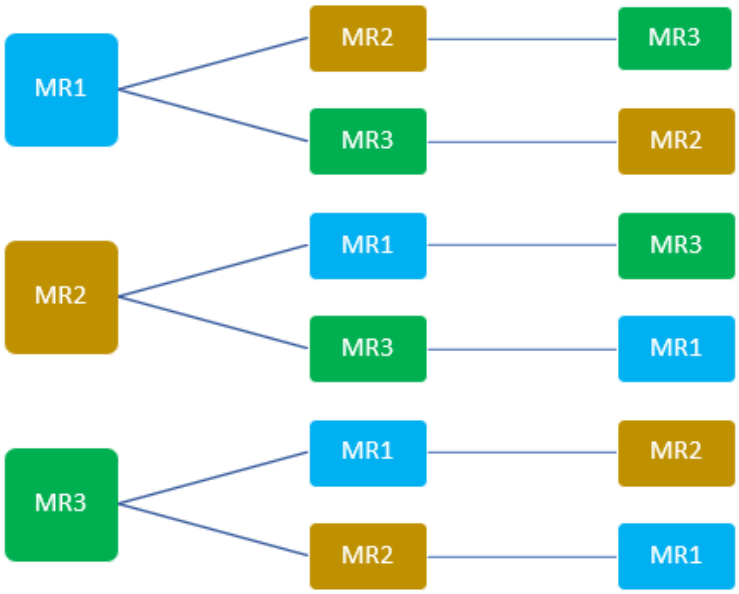


Figura 13. Árbol de combinaciones de las secuencias de refactorización

Las siglas desglosadas en el árbol de combinaciones tienen los siguientes significados correspondientes:

MR1 = Método de refactorización de protección modular
MR2 = Método de refactorización de herencia de implementación
MR3 = Método de refactorización de carencia de abstracción

A fin de gestionar de manera efectiva los resultados obtenidos tras la aplicación de cada secuencia en los 30 proyectos Java, se asignó una identificación única a cada una de ellas. La Tabla 8 proporciona los identificadores que se utilizarán de ahora en adelante para referenciar cada secuencia de ejecución de métodos de refactorización.

Tabla 8. Secuencias de refactorización evaluadas

ID de secuencia	Secuencia de ejecución de métodos de refactorización
S1	MR1-MR2-MR3
S2	MR1-MR3-MR2
S3	MR2-MR1-MR3
S4	MR2-MR3-MR1
S5	MR3-MR1-MR2
S6	MR3-MR2-MR1

Durante la validación, cada una de estas secuencias fue aplicada sobre los sistemas de estudio con el propósito de comparar los efectos producidos en las métricas estructurales. Esta comparación permitió identificar si el orden de ejecución de las técnicas influye en la conservación o mejora de la calidad acumulada del sistema.

7.5 Resultados de la ejecución

En esta sección se presentan los resultados obtenidos durante la ejecución de las secuencias de refactorización aplicadas sobre los sistemas de estudio. El propósito de esta etapa fue analizar el comportamiento de las técnicas de refactorización cuando son ejecutadas en distinto orden, así como evaluar si el algoritmo Greedy adaptado permite construir secuencias más estables que el algoritmo Greedy tradicional.

La validación se realizó a partir de las secuencias definidas en la sección anterior, correspondientes a las combinaciones posibles de los métodos de refactorización considerados en esta investigación: protección modular, herencia de implementación y carencia de abstracción. Para facilitar el análisis de los resultados, cada combinación fue identificada mediante los códigos S1, S2, S3, S4, S5 y S6, los cuales corresponden a los distintos órdenes de ejecución de los métodos de refactorización.

Antes de aplicar las secuencias de refactorización, se calcularon los valores iniciales de las métricas estructurales de los sistemas analizados. Estos valores permitieron establecer la condición de partida de cada programa y sirvieron como referencia para comparar los cambios obtenidos después de aplicar las combinaciones de refactorización. Debido a la extensión del conjunto de sistemas, los resultados iniciales se distribuyen en las Tablas 9 y 10.

Para interpretar los resultados se retomó la función de calidad definida en el modelo propuesto, la cual integra únicamente métricas representativas de las dimensiones evaluadas. En este sentido, la calidad estructural de cada estado se evaluó considerando TPM para protección modular, $1 - FHIAC$ para herencia de implementación y A para abstracción. Las demás métricas calculadas durante el proceso se utilizaron como valores base o intermedios, de acuerdo con lo establecido previamente en el modelo.

Tabla 9. Resultados iniciales de las métricas estructurales consideradas en la función de calidad

No. Programa	Nombre	URL	No. Clases	PMMP	PMMPr	PMMF	PM	PMT	FHIAC	FMFAC	PF	A
1	Blue-Block-master	https://github.com/abc013/Blue-Block	13	0.062	0	0.037	0.092	0.001	0	0	0	0
2	chess-system-java-master	https://github.com/acenelio/chess-system-java	16	0.097	0.259	0	0.27	0.004	0.675	0.013	0.182	0.063
3	CompanyManagementSystem-master	https://github.com/fdeniz07/CompanyManagementSystem	14	0	0	0	0	0	0.444	0.048	0.714	0.214
4	CustomerManagerApp-master	https://github.com/Sepobanz/CustomerManagerApp	6	0	0	0	0	0	0	0	0	0
5	FileOperations-master	https://github.com/berkaydursun/FileOperations	10	0.05	0	0.043	0.087	0.003	0	0	0	0.086
6	HospitalProject-master	https://github.com/fdeniz07/HospitalProject	32	0	0	0.069	0.069	0	0.267	0.035	0.476	0.188
7	InventoryManagementSystem-master	https://github.com/twilsonn/InventoryManagementSystem	6	0	0	0.028	0.222	0	0	0	0	0
8	Java_Console-Course-Management-System-main	https://github.com/DrNeonsy/Java_Console-Course-Management-System	12	0.049	0	0.302	0.395	0.003	0.5	0	1.667	0.364
9	Java_Imobiliaria-master	https://github.com/Marcello-Goncalves/Java_Imobiliaria	5	0	0	0	0	0	0.6	0	0	0
10	Java-Console-Card-Game-master	https://github.com/dnavas77/Java-Console-Card-Game	10	0	0	0	0	0	0	0	0	0
11	Juego-El-ahorcado-en-Java-master	https://github.com/davidmazparrote/Juego-El-ahorcado-en-Java	3	0	0	0	0	0	0	0	0	0
12	Library_Management_System_11-Jan-2022-master	https://github.com/hariprakash2113/Library_Management_System_11-Jan-2022	5	0	0	0.8	0.8	0.04	0	0	0	0
13	Movie-Service-Review-Java-main	https://github.com/lakshygupta/Movie-Service-Review-Java	5	0	0	0	0	0	0	0	0	0
14	online-quiz-application-in-java-with-jdbc-using-swing-login-registration-timer-master	https://github.com/srilekhadasari/online-quiz-application-in-java-with-jdbc-using-swing-login-registration-timer	6	0.017	0	0.132	0.342	0.001	0	0	0	0
15	Password-Generator	https://github.com/KZarzour/Password-Generator	4	0	0	0	0	0	0	0	0	0

Tabla 10. Resultados iniciales de las métricas estructurales consideradas en la función de calidad continuación

No. Programa	Nombre	URL	No. Clases	PMMP	PMMPr	PMMF	PM	PMT	FHIAC	FMFAC	PF	A
16	PasswordManager-main	https://github.com/codershiyar/PasswordManager	21	0.04	0	0.194	0.25	0.001	0.5	0.024	0.167	0.19
17	Polinom	https://github.com/taijased/univers_cours2_Java/tree/master/Polinom	10	0	0	0.154	0.282	0.001	0	0	0	0
18	qurbani-animals-market-main	https://github.com/MSarmadQadeer/qurbani-animals-market	15	0.016	0	0.019	0.038	0	0.833	0	0.083	0.5
19	radio	https://github.com/KI7MT/java-app-examples/blob/master/ConsoleApps/src/beam/example/radio/station/RadioStation.java	6	0	0	0	0	0	0	0	0	0
20	real-estate-console-app-master	https://github.com/Leo-Chan01/real-estate-console-app	8	0.031	0	0	0.028	0.01	0	0	0	0
21	SchoolManagement-master	https://github.com/fdeniz07/SchoolManagement	13	0	0	0.021	0.021	0	0.6	0.062	0.5	0.231
22	shapp-master	https://github.com/vicianm/shapp	6	0	0	0	0	0	0	0	0	0
23	Simple-Java-Calculator-master	https://github.com/pH-7/Simple-Java-Calculator.git	4	0.092	0	0	0.154	0.008	0	0	0	0
24	TetrisForDesktopWithJava-master	https://github.com/kkikkodev/TetrisForDesktopWithJava	9	0.149	0	0	0.278	0.002	0	0	0	0.429
25	tic-tac-toe-console-master	https://github.com/rohandebsarkar/tic-tac-toe-console	7	0	0	0	0	0	0	0	0	0
26	tiny-compiler-master	https://github.com/almirfilho/tiny-compiler	13	0.203	0	0	0	0.473	0.005	0	0	0.25
27	treinaweb-java-oo-master	https://github.com/treinaweb/treinaweb-java-oo	12	0	0	0.154	0.154	0.001	0.882	0.005	0.0235	0.333
28	UniversalCalculator-master	https://github.com/dima666Sik/UniversalCalculator	15	0.21	0	0.08	0.6	0.009	0.987	0	0.133	0.133
29	Llaveros-master	https://github.com/JHORJE18/Llaveros	4	0	0	0	0	0	0	0	0	0
30	AI-Chat-Bot-master	https://github.com/gazalpatel/AI-Chat-Bot	6	0	0	0	0	0	0	0	0	0

Posteriormente, se aplicaron las seis secuencias de refactorización definidas en el escenario de ejecución. Para cada secuencia se calcularon nuevamente las métricas estructurales y el valor de la función de calidad. En la Tabla 11 se presentan los valores de calidad $Q(E_t)$ obtenidos para cada programa después de aplicar las seis secuencias de refactorización S1, S2, S3, S4, S5 y S6. Además, se muestra la secuencia seleccionada por el algoritmo Greedy tradicional y la secuencia seleccionada por el modelo propuesto. Esta comparación permite identificar si la secuencia con mayor valor final coincide o no con la secuencia determinada por el modelo, el cual considera la evaluación incremental de precondiciones, postcondiciones y preservación de la calidad estructural durante el proceso de refactorización.

Tabla 11. Valores de calidad obtenidos por secuencia de refactorización y selección final del modelo

No.	Programa	(Q(S1))	(Q(S2))	(Q(S3))	(Q(S4))	(Q(S5))	(Q(S6))	Secuencia seleccionada por Greedy tradicional	Secuencia seleccionada por el modelo	Valor (Q) de la secuencia seleccionada
1	Blue-Block-master	0.262	0.262	0.262	0.485	0.423	0.423	S4	S5	0.423
2	chess-system-java-master	0.493	0.493	0.493	0.489	0.489	0.489	S1	S2	0.493
3	CompanyManagementSystem-master	0.393	0.393	0.393	0.393	0.393	0.393	S1	S3	0.393
4	CustomerManagerApp-master	0.282	0.282	0.282	0.242	0.242	0.242	S1	S4	0.242
5	FileOperations-master	0.282	0.313	0.313	0.282	0.282	0.313	S2	S6	0.313
6	HospitalProject-master	0.222	0.466	0.222	0.191	0.191	0.191	S2	S6	0.191
7	InventoryManagementSystem-master	0.270	0.270	0.070	0.200	0.200	0.200	S1	S5	0.200
8	Java_Console-Course-Management-System-main	0.282	0.282	0.297	0.352	0.352	0.352	S4	S3	0.297
9	Java_Imobiliaria-master	0.270	0.211	0.270	0.264	0.284	0.284	S5	S6	0.284
10	Java-Console-Card-Game-master	0.293	0.307	0.307	0.293	0.293	0.307	S2	S6	0.307
11	Juego-El-ahorcado-en-Java-master	0.097	0.097	0.097	0.084	0.084	0.084	S1	S1	0.097
12	Library_Management_System_11-Jan-2022-master	0.160	0.160	0.160	0.160	0.160	0.160	S1	S6	0.160
13	Movie-Service-Review-Java-main	0.080	0.080	0.080	0.115	0.115	0.115	S4	S4	0.115

No.	Programa	(Q(S1))	(Q(S2))	(Q(S3))	(Q(S4))	(Q(S5))	(Q(S6))	Secuencia seleccionada por Greedy tradicional	Secuencia seleccionada por el modelo	Valor (Q) de la secuencia seleccionada
14	online-quiz-application-in-java-with-jdbc-using-swing-login-registration-timer-master	0.058	0.058	0.058	0.058	0.058	0.058	S1	S5	0.058
15	Password-Generator	0.294	0.294	0.294	0.240	0.240	0.240	S1	S1	0.294
16	PasswordManager-main	0.172	0.172	0.172	0.172	0.185	0.185	S5	S6	0.185
17	Polinom	0.297	0.297	0.297	0.287	0.290	0.287	S1	S3	0.297
18	qurbani-animals-market-main	0.115	0.115	0.115	0.085	0.085	0.085	S1	S1	0.115
19	radio	0.285	0.285	0.285	0.267	0.267	0.267	S1	S3	0.285
20	real-estate-console-app-master	0.341	0.341	0.341	0.298	0.298	0.298	S1	S1	0.341
21	SchoolManagement-master	0.298	0.298	0.298	0.185	0.185	0.185	S1	S6	0.185
22	shapp-master	0.249	0.249	0.249	0.208	0.208	0.208	S1	S2	0.249
23	Simple-Java-Calculator-master	0.350	0.284	0.284	0.223	0.226	0.226	S1	S4	0.223
24	TetrisForDesktopWithJava-master	0.336	0.336	0.336	0.298	0.298	0.298	S1	S6	0.298
25	tic-tac-toe-console-master	0.365	0.365	0.365	0.365	0.365	0.365	S1	S1	0.365
26	tiny-compiler-master	0.339	0.339	0.339	0.337	0.337	0.337	S1	S3	0.339
27	treinaweb-java-oo-master	0.328	0.328	0.328	0.328	0.328	0.328	S1	S2	0.328
28	UniversalCalculator-master	0.627	0.627	0.627	0.549	0.549	0.549	S1	S6	0.549
29	llaveros-master	0.355	0.355	0.355	0.344	0.344	0.344	S1	S5	0.344
30	AI-Chat-Bot-master	0.425	0.425	0.425	0.425	0.425	0.425	S1	S1	0.425

Como se observa en la Tabla 11, la comparación de los valores obtenidos permite identificar cómo varía la calidad estructural del software cuando las técnicas se ejecutan en diferente orden. Al utilizar únicamente métricas representativas dentro de la función de calidad, se evita duplicar el peso de una misma dimensión. En particular, TPM concentra la evaluación de la protección modular, $1 - FHIAC$ permite interpretar la reducción de herencia de implementación bajo una orientación positiva y A representa el nivel de abstracción del sistema.

En los resultados se aprecia que las técnicas de protección modular y herencia de implementación presentan menor interferencia entre sí, sin generar una degradación significativa cuando se aplican de manera consecutiva, debido a que atienden propiedades estructurales diferentes. La protección modular se enfoca en mejorar el encapsulamiento mediante el ajuste de calificadores de acceso, mientras que la herencia de implementación se orienta a reorganizar dependencias jerárquicas y reducir acoplamientos rígidos.

Por el contrario, la técnica de carencia de abstracción produce modificaciones estructurales más amplias, ya que puede incorporar clases abstractas, interfaces o nuevas relaciones de generalización. Debido a ello, su posición dentro de la secuencia tiene un efecto relevante sobre el comportamiento posterior de las métricas. En este sentido, los resultados permiten observar que las secuencias donde interviene MR3 pueden modificar de forma importante la estructura sobre la cual posteriormente actúan las demás técnicas.

Además del análisis de los valores de calidad por secuencia, se comparó el comportamiento del algoritmo Greedy tradicional con la selección realizada por el modelo propuesto. Esta comparación permitió observar si la incorporación de precondiciones, postcondiciones y evaluación incremental de calidad modificaba la selección de las secuencias. Como se muestra en la Tabla 11, la secuencia seleccionada por el modelo no siempre coincide con la secuencia seleccionada por el algoritmo Greedy tradicional, debido a que ambos enfoques utilizan criterios de decisión diferentes.

Mientras que el algoritmo Greedy tradicional selecciona con base en el mejor valor final disponible, el modelo propuesto, mediante el algoritmo Greedy adaptado, evalúa cada transición entre estados. Para ello, considera si la técnica cumple sus precondiciones, si el estado resultante satisface las postcondiciones y si la calidad estructural se conserva o mejora durante el proceso. Esta diferencia permite explicar por qué, en algunos programas, el modelo selecciona una secuencia cuyo valor final no es necesariamente el más alto, pero cuya trayectoria resulta más estable y consistente con las condiciones definidas.

En los casos en los que dos o más secuencias presentaron el mismo valor de calidad, se aplicó un criterio de desempate para mantener un comportamiento determinista. En el algoritmo Greedy tradicional se seleccionó la primera secuencia, de acuerdo con el orden S1, S2, S3, S4, S5 y S6, que presentó el valor final máximo de calidad. En el algoritmo Greedy adaptado se seleccionó el primer estado candidato que cumplió las precondiciones y postcondiciones establecidas y que conservó o mejoró la calidad estructural alcanzada.

A partir de la comparación anterior, se obtuvo la frecuencia con la que cada secuencia fue seleccionada por ambos enfoques, como se muestra en la Tabla 12. Esta información permite observar

de manera resumida el comportamiento de selección del algoritmo Greedy tradicional y del modelo propuesto.

Tabla 12. Frecuencia de selección de secuencias por algoritmo

Secuencia	Frecuencia en Greedy tradicional	Frecuencia en Greedy adaptado
S1	22	6
S2	3	3
S3	0	5
S4	3	3
S5	2	4
S6	0	9
Total	30	30

La frecuencia de selección muestra que el algoritmo Greedy tradicional concentró sus resultados principalmente en la secuencia S1, mientras que no seleccionó la secuencia S6 en ninguno de los sistemas evaluados. En contraste, el algoritmo Greedy adaptado seleccionó S6 en nueve de los treinta sistemas, lo que representa el 30% de los casos. Este comportamiento es relevante porque S6 corresponde al orden MR3→MR2→MR1, es decir, carencia de abstracción, herencia de implementación y protección modular.

La selección recurrente de S6 por parte del algoritmo Greedy adaptado sugiere que la evaluación incremental favorece trayectorias más estables durante el proceso de refactorización. Esta secuencia inicia con la incorporación de abstracción, lo que permite reorganizar la estructura general del sistema; posteriormente, se aplica la técnica de herencia de implementación, que ajusta las relaciones jerárquicas; finalmente, se aplica protección modular, fortaleciendo el encapsulamiento de los elementos resultantes. Esta interpretación coincide con el propósito del modelo, ya que cada transformación se evalúa en función del estado estructural generado y no únicamente como una mejora aislada.

Los resultados también permiten observar que el algoritmo Greedy adaptado no acepta una técnica únicamente porque produzca una mejora local inmediata, sino porque su aplicación mantiene la consistencia de la secuencia. Si una técnica produce un estado candidato que no satisface las postcondiciones o que disminuye la calidad previamente alcanzada, dicho estado es descartado. Esta característica permite reducir la interferencia entre técnicas y evita que una refactorización posterior anule los beneficios alcanzados por una transformación previa.

En conjunto, la ejecución de las secuencias y la comparación entre algoritmos muestran que el modelo propuesto, al ser operado mediante el algoritmo Greedy adaptado, permite construir secuencias de refactorización más controladas. La validación evidencia que el uso de precondiciones, postcondiciones y función de calidad contribuye a seleccionar transiciones válidas entre estados estructurales, preservando la mejora acumulada del sistema y reduciendo el riesgo de degradación durante el proceso. Los resultados detallados por sistema se presentan en el Anexo C, donde se incluyen las tablas individuales de ejecución por proyecto Java y los valores correspondientes a las combinaciones de refactorización aplicadas sobre cada sistema. Estos resultados complementan el

análisis presentado en este capítulo y permiten consultar de manera específica el comportamiento de cada secuencia evaluada.

7.6 Análisis de la validación

El análisis de la validación permite identificar que la principal diferencia entre el algoritmo Greedy tradicional y el algoritmo Greedy adaptado se encuentra en el criterio de aceptación de cada técnica de refactorización. Mientras el enfoque tradicional se orienta principalmente al resultado final, el algoritmo propuesto evalúa cada paso del proceso, considerando el estado actual del sistema, las precondiciones de las técnicas y las postcondiciones del estado resultante.

Esta diferencia es relevante porque en un proceso de refactorización no basta con obtener un resultado final favorable. También es necesario asegurar que las transformaciones intermedias no comprometan la calidad estructural del sistema. En sistemas legados, donde la deuda técnica suele estar relacionada con múltiples deficiencias de diseño, una refactorización aplicada sin control puede mejorar una dimensión de calidad, pero afectar negativamente otra.

El algoritmo Greedy adaptado atiende este problema mediante una evaluación incremental. Cada técnica se analiza antes de aplicarse y después de generar un estado candidato. Si las precondiciones no se cumplen, la técnica no se considera viable. Si las postcondiciones no se satisfacen o algún valor de las métricas disminuye, el estado candidato se descarta. Este comportamiento permite reducir el riesgo de interferencia entre técnicas de refactorización.

Los resultados obtenidos en los 30 sistemas analizados muestran que el algoritmo propuesto genera secuencias distintas a las del Greedy tradicional. Esto evidencia que la incorporación de precondiciones y postcondiciones modifica sustancialmente el proceso de selección. La diferencia entre secuencias indica que el algoritmo no solo busca maximizar una función de calidad final, sino mantener la estabilidad del proceso de transformación.

Desde el punto de vista del modelo formal, la validación confirma que el sistema puede representarse como una sucesión de estados estructurales. Cada estado corresponde a una configuración del software y cada técnica de refactorización representa una transición entre estados. La validez de cada transición depende del cumplimiento de las condiciones definidas. Esta interpretación permite analizar el proceso de refactorización como una exploración controlada del espacio de estados.

Desde el punto de vista de la implementación, la validación confirma que la arquitectura basada en microservicios permite separar las responsabilidades del proceso de refactorización. Los componentes encargados del análisis, cálculo de métricas, evaluación de condiciones y generación de resultados permiten materializar el modelo en una solución funcional.

El análisis también permite observar que la propuesta no elimina la necesidad de evaluar el comportamiento funcional del sistema después de la refactorización. Aunque las métricas estructurales permiten medir el impacto interno de las transformaciones, la preservación del comportamiento observable debe complementarse con pruebas funcionales, pruebas unitarias o mecanismos de verificación adicionales. Esto resulta especialmente importante cuando las refactorizaciones se aplican sobre sistemas reales.

En conjunto, la validación permite concluir que el algoritmo Greedy adaptado ofrece una alternativa más segura que el enfoque tradicional para construir secuencias de refactorización, ya que incorpora mecanismos explícitos de control antes y después de cada transformación. Esta característica fortalece la confiabilidad del proceso y contribuye a reducir la deuda técnica de manera progresiva.

Capítulo 8.- Conclusiones y trabajos futuros

8.1 Conclusiones

Esta tesis presentó un modelo de secuenciación de microservicios de refactorización orientado a reducir deuda técnica en sistemas legados desarrollados en lenguaje Java. La propuesta se centró en tres dimensiones estructurales del diseño orientado a objetos: protección modular, abstracción y herencia de implementación. Estas dimensiones fueron seleccionadas debido a su relación directa con problemas de calidad interna como fragilidad, rigidez, bajo reuso, dificultad de extensión y propagación de defectos.

El problema abordado en esta investigación surge de la necesidad de aplicar técnicas de refactorización de manera ordenada y controlada. Aunque la refactorización permite mejorar la estructura interna del software sin modificar su comportamiento observable, la aplicación de múltiples técnicas sin un criterio de secuenciación puede generar interferencias entre transformaciones. Por ello, esta tesis propuso incorporar precondiciones, postcondiciones y estados candidatos como mecanismos de control dentro del proceso de secuenciación.

El modelo propuesto representa el sistema de software como una sucesión de estados estructurales. Cada estado describe la configuración del sistema en un momento determinado, considerando elementos como clases, métodos, atributos, relaciones y métricas estructurales. Bajo esta perspectiva, cada técnica de refactorización se interpreta como una transición entre estados. Dicha transición solo se considera válida si la técnica cumple con sus precondiciones y si el estado candidato satisface las postcondiciones establecidas.

Respecto al objetivo general, se concluye que fue atendido mediante la definición de un modelo de secuenciación de microservicios de refactorización basado en precondiciones, postcondiciones,

estados candidatos y evaluación de calidad. Este modelo permitió establecer una forma controlada de seleccionar y ejecutar técnicas de refactorización, considerando el estado actual del sistema antes de aceptar cada transformación. De esta manera, cada microservicio de refactorización se integra a la secuencia únicamente cuando cumple las condiciones establecidas, lo que contribuye a reducir interferencias entre técnicas y a conservar o mejorar la calidad estructural del sistema.

En relación con el primer objetivo específico, orientado a disminuir la fragilidad del sistema legado mediante protección modular, se concluye que fue atendido a través de la técnica MR1 Protección modular. Esta técnica permitió analizar los modificadores de acceso de los métodos y evaluar oportunidades para fortalecer el encapsulamiento del código. Al controlar la exposición innecesaria de métodos públicos y favorecer una mayor protección de los elementos internos, el modelo contribuye a disminuir la fragilidad asociada con la propagación de cambios o defectos entre componentes.

Respecto al segundo objetivo específico, enfocado en aumentar la flexibilidad y extensibilidad mediante abstracción, se concluye que fue atendido mediante la técnica MR3 Carencia de abstracción. Esta técnica permitió identificar elementos susceptibles de generalización y evaluar la incorporación de estructuras abstractas, como clases abstractas o interfaces, cuando las condiciones del sistema lo permitieron. Con ello, el modelo favorece una organización más flexible del código y reduce la dependencia directa hacia implementaciones concretas.

En cuanto al tercer objetivo específico, relacionado con mejorar la modularidad en la dimensión de herencia de implementación, se concluye que fue atendido mediante la técnica MR2 Herencia de implementación. Esta técnica permitió evaluar relaciones jerárquicas y dependencias funcionales susceptibles de reorganización, con el propósito de reducir el acoplamiento derivado de la herencia de implementación. La evaluación de esta técnica dentro de la secuencia permitió controlar que los cambios realizados no afectaran negativamente los valores de calidad previamente alcanzados.

También se concluye que la adaptación realizada al algoritmo Greedy tradicional permitió construir secuencias de refactorización de manera incremental. El algoritmo Greedy adaptado no selecciona únicamente la alternativa con mayor valor final de calidad, sino que evalúa cada paso del proceso. En cada iteración, identifica las técnicas aplicables, genera estados candidatos, valida las postcondiciones y selecciona la técnica que produce el mejor resultado local sin degradar la calidad acumulada. De esta forma, la secuencia se construye de manera progresiva y controlada.

La validación realizada sobre 30 proyectos Java permitió observar que el algoritmo Greedy adaptado genera secuencias distintas a las del algoritmo Greedy tradicional. Esta diferencia evidencia que la incorporación de precondiciones, postcondiciones, estados candidatos y función de calidad modifica el proceso de selección. Mientras el enfoque tradicional puede elegir secuencias con base en el resultado final, el algoritmo adaptado prioriza trayectorias que preservan la calidad estructural en cada transición.

La implementación del modelo mediante una arquitectura basada en microservicios permitió materializar los elementos teóricos de la propuesta en un prototipo funcional. La separación de responsabilidades entre componentes facilitó el análisis del código fuente, el cálculo de métricas, la evaluación de condiciones y la construcción de secuencias. Asimismo, la coreografía de microservicios

permitió representar la interacción entre técnicas de refactorización autónomas, evitando una dependencia rígida entre ellas.

En conjunto, los resultados permiten afirmar que los objetivos de la investigación fueron atendidos, ya que el modelo propuesto no solo define una secuencia de aplicación de microservicios de refactorización, sino que también establece mecanismos de control para decidir cuándo una técnica puede ser evaluada, aceptada o descartada. La validación evidencia que el uso de precondiciones, postcondiciones, estados candidatos y función de calidad contribuye a seleccionar transiciones válidas entre estados estructurales, procurando que la calidad del software se conserve o mejore durante el proceso de refactorización.

8.2 Aportaciones de la investigación

La primera aportación de esta investigación es la definición de un modelo de secuenciación de refactorizaciones basado en estados estructurales del sistema. Este modelo permite representar formalmente el proceso de transformación del software como una sucesión de estados y transiciones, donde cada transición corresponde a la aplicación de una técnica de refactorización.

La segunda aportación es la incorporación explícita de precondiciones y postcondiciones como mecanismos de control dentro del proceso de refactorización. Las precondiciones permiten determinar si una técnica puede aplicarse sobre el estado actual del sistema, mientras que las postcondiciones permiten validar que el estado resultante no degrade la calidad alcanzada previamente.

La tercera aportación es la modificación del algoritmo Greedy tradicional para adaptarlo al problema de secuenciación de refactorizaciones. El algoritmo Greedy adaptado construye la secuencia de manera incremental y selecciona, en cada iteración, la técnica que produce el mejor resultado local sin comprometer la calidad acumulada del sistema.

La cuarta aportación es la integración de métricas estructurales como criterio de evaluación durante el proceso de secuenciación. Estas métricas permiten medir el impacto de cada técnica sobre dimensiones específicas de la calidad orientada a objetos, particularmente protección modular, abstracción y herencia de implementación.

La quinta aportación es la implementación de un prototipo funcional basado en microservicios. Esta implementación permite llevar el modelo propuesto a un entorno práctico, donde los servicios se encargan de tareas específicas como análisis del código, cálculo de métricas, evaluación de condiciones, generación de estados candidatos y presentación de resultados.

La sexta aportación es la validación del modelo y del algoritmo mediante 30 proyectos Java seleccionados del repositorio GitHub. Esta validación permitió comparar el comportamiento del algoritmo Greedy tradicional y del algoritmo Greedy adaptado, evidenciando diferencias en la selección de secuencias debido al control incremental de calidad.

La séptima aportación es la construcción de una base metodológica para extender el modelo hacia nuevas técnicas de refactorización. Debido a que el modelo se basa en la definición de precondiciones,

postcondiciones y métricas, puede ampliarse para incorporar otras transformaciones estructurales, siempre que estas puedan formalizarse bajo el mismo enfoque.

8.3 Limitaciones identificadas

Una de las principales limitaciones de esta investigación es que la validación se realizó únicamente sobre proyectos desarrollados en lenguaje Java. Aunque Java es un lenguaje ampliamente utilizado y adecuado para analizar propiedades del paradigma orientado a objetos, la aplicación del modelo a otros lenguajes requeriría adaptar el análisis sintáctico, las métricas y las técnicas de refactorización consideradas.

Otra limitación se relaciona con el número de técnicas de refactorización incluidas en el modelo. Esta investigación se centró en tres dimensiones: protección modular, abstracción y herencia de implementación. Si bien estas dimensiones son relevantes para la reducción de deuda técnica, existen otras técnicas que también podrían contribuir a mejorar la calidad del software, tales como extracción de métodos, extracción de clases, movimiento de métodos, eliminación de duplicación o reorganización de paquetes.

Una tercera limitación corresponde al conjunto de métricas utilizadas. Las métricas seleccionadas permiten evaluar aspectos estructurales específicos, pero no cubren todos los atributos posibles de calidad de software. Aspectos como rendimiento, seguridad, facilidad de prueba, comprensión del código o esfuerzo de mantenimiento podrían requerir métricas adicionales.

Una cuarta limitación se relaciona con la evaluación funcional del sistema después de la refactorización. Aunque las técnicas de refactorización deben preservar el comportamiento observable del software, esta investigación se enfocó principalmente en la evaluación estructural mediante métricas. Por ello, resulta necesario complementar el modelo con pruebas unitarias, pruebas de regresión o mecanismos de verificación funcional que permitan asegurar que el comportamiento externo del sistema se mantiene después de cada transformación.

Otra limitación se encuentra en el entorno de ejecución del prototipo. La implementación se realizó en un entorno controlado con contenedores Docker, lo cual permitió validar la separación de servicios y el funcionamiento general de la arquitectura. Sin embargo, no se evaluó de manera profunda el comportamiento del sistema en escenarios de despliegue distribuido a gran escala, con alta concurrencia, balanceo de carga o tolerancia a fallos.

Finalmente, el algoritmo Greedy adaptado mantiene la naturaleza heurística del enfoque Greedy. Esto significa que la secuencia generada corresponde a una solución construida mediante decisiones locales controladas por precondiciones y postcondiciones, pero no necesariamente representa una solución óptima global en todos los casos posibles. Aun así, el algoritmo propuesto ofrece una ventaja importante al evitar degradaciones intermedias y preservar la calidad acumulada durante el proceso.

8.4 Líneas de trabajo futuro

Como línea de trabajo futuro se propone ampliar el conjunto de técnicas de refactorización consideradas en el modelo. La incorporación de nuevas técnicas permitiría evaluar el comportamiento del algoritmo en un espacio de búsqueda más amplio y analizar su capacidad para construir secuencias más complejas.

Otra línea futura consiste en extender el modelo hacia otros lenguajes de programación orientados a objetos, como C#, Kotlin o Python. Para ello sería necesario adaptar los mecanismos de análisis sintáctico, las métricas y las reglas de refactorización a las características particulares de cada lenguaje. También se propone integrar pruebas unitarias y pruebas de regresión dentro del proceso de validación. Esto permitiría complementar la evaluación estructural con una verificación funcional, asegurando que las refactorizaciones aplicadas no alteren el comportamiento observable del sistema.
Una línea adicional consiste en incorporar métricas relacionadas con otros atributos de calidad, como mantenibilidad, complejidad ciclomática, cohesión, acoplamiento, seguridad y facilidad de prueba. Esto permitiría construir una función de calidad más robusta y adaptable a diferentes objetivos de refactorización.

Otra posible extensión consiste en comparar el algoritmo Greedy adaptado con otros enfoques heurísticos y metaheurísticos, tales como algoritmos genéticos, búsqueda tabú, recocido simulado, algoritmos A*, AO* o técnicas basadas en aprendizaje automático. Esta comparación permitiría analizar con mayor profundidad las ventajas y limitaciones del enfoque propuesto.
También se propone avanzar hacia un despliegue distribuido real de los microservicios de refactorización en una infraestructura de nube. Esto permitiría evaluar aspectos como escalabilidad, disponibilidad, tolerancia a fallos, comunicación entre servicios y rendimiento del sistema bajo diferentes condiciones de carga.

Finalmente, se plantea como trabajo futuro el desarrollo de una interfaz más completa para la visualización de resultados. Esta interfaz podría incluir gráficas comparativas de métricas antes y después de la refactorización, representación visual de la secuencia generada, trazabilidad de las decisiones tomadas por el algoritmo y explicación de las técnicas descartadas por incumplimiento de precondiciones o postcondiciones.

Referencias|

Al Dallal, J., & Abdin, A. (2018). Empirical Evaluation of the Impact of Object-Oriented Code Refactoring on Quality Attributes: A Systematic Literature Review. *IEEE Transactions on Software Engineering*, *44*(1), 44–69. https://doi.org/10.1109/TSE.2017.2658573

Barón, N. (2020). *Método de refactorización para mejorar la protección modular de arquitecturas orientadas a objetos de sistemas de software existentes*.

Baškarada, S., Nguyen, V., & Koronios, A. (2020). Architecting Microservices: Practical Opportunities and Challenges. *Journal of Computer Information Systems*, *60*(5), 428–436. https://doi.org/10.1080/08874417.2018.1520056

Borges, M., Barros, E., & Maia, P. H. (2018). Cloud restriction solver: A refactoring-based approach to migrate applications to the cloud. *Information and Software Technology*, *95*, 346–365. https://doi.org/10.1016/j.infsof.2017.11.014

Christopoulou, A., Giakoumakis, E. A., Zafeiris, V. E., & Soukara, V. (2012).Automated refactoring to the Strategy design pattern. *Information and Software Technology*, *54*(11), 1202–1214. https://doi.org/10.1016/j.infsof.2012.05.004

Conejero, J. M., Rodríguez-Echeverría, R., Hernández, J., Clemente, P. J., Ortiz-Caraballo, C., Jurado, E., & Sánchez-Figueroa, F. (2018). Early evaluation oftechnical debt impact on maintainability. *Journal of Systems and Software*, *142*, 92–114. https://doi.org/10.1016/j.jss.2018.04.035

Dong, B., Ying, S., Li, L., Luo, H., & Yang, Z. (2020). Impact Analysis about Response Time Considering Deployment Change of SaaS Software. *International Journal of Software Engineering and Knowledge Engineering, 30(7),* 977–1004. https://doi.org/10.1142/S0218194020500266

Freitas, M. A. de. (2020). Identifying self-admitted technical debt through code comment analysis with a contextualized vocabulary.

Furda, A. (2018). On Multitenancy, Statefulness, and Data Consistency. *IEEESoftware*, 63–72.

Hernández, O. H. L. (2018). *Método para Migrar Servicios Web bajo el Protocolo SOAP hacia Microservicios Basados en REST*. Centro Nacional de Investigación y Desarrollo Tecnológico.

Jamshidi, P., Pahl, C., Mendonca, N. C., Lewis, J., & Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. *IEEE Software*, *35*(3), 24–35. https://doi.org/10.1109/MS.2018.2141039

Kaur, G., & Singh, B. (2017). Improving the quality of software by refactoring. *Proceedings of the 2017 International Conference on Intelligent Computingand Control Systems, ICICCS 2017*, *2018-Janua*, 185–191.ttps://doi.org/10.1109/ICCONS.2017.8250707

Kebir, S., Borne, I., & Meslati, D. (2017). A Genetic Algorithm-Based Approach forAutomated Refactoring of Component-Based Software. *Information and Software Technology*. https://doi.org/10.1016/j.infsof.2017.03.009

Khatchadourian, R., & Masuhara, H. (2017). Automated Refactoring of LegacyJava Software to Default Methods. *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering, ICSE 2017*, 82–93. https://doi.org/10.1109/ICSE.2017.16

Kiss, T., Kacsuk, P., Kovacs, J., Rakoczi, B., Hajnal, A., Farkas, A., … Terstyanszky, G. (2019). MiCADO—Microservice-based Cloud Application- level Dynamic Orchestrator. *Future Generation Computer Systems*, *94*, 937–946. https://doi.org/10.1016/j.future.2017.09.050

Laobel, I., Betancourt, C., & Martínez, N. S. (2020). ONTOLOGÍA PARA LADESCRIPCIÓN DE CODE SMELLS.

Larrucea, X., Santamaria, I., Colomo-palacios, R., & Ebert, C. (2021). Microservices.

Lehman, M. M. (1968). Laws of S o f t w a r e E v o l u t i o n R e v i s i t e d.

Lenarduzzi, V., Lomio, F., Saarimäki, N., & Taibi, D. (2020). Does migrating a monolithic system to microservices decrease the technical debt? *Journal of Systems and Software*, *169*, 110710. https://doi.org/10.1016/j.jss.2020.110710

Li, S., Zhang, H., Jia, Z., Li, Z., Zhang, C., Li, J., … Shan, Z. (2019). A dataflow-driven approach to identifying microservices from monolithic applications. *Journal of Systems and Software*, *157*. https://doi.org/10.1016/j.jss.2019.07.008

Mahesh Kumar. (2014). Software As a Service for Efficient Cloud Computing. *International Journal of Research in Engineering and Technology*, *03*(01), 178–181. https://doi.org/10.15623/ijret.2014.0301028

Malathi, S., & Sudhakar, P. (2018). Implementation of Software Refactoring Using FODA Tool. *Proceedings of the 3rd International Conference on Communication and Electronics Systems, ICCES 2018*, (Icces), 839–842. https://doi.org/10.1109/CESYS.2018.8723986

Matoussi, W., & Hamrouni, T. (2021). A new temporal locality-based workload prediction approach for SaaS services in a cloud environment. *Journal of KingSaud University - Computer and Information Sciences*, (xxxx). https://doi.org/10.1016/j.jksuci.2021.04.008

Mazlami, G., Cito, J., & Leitner, P. (2017). Extraction of Microservices from Monolithic Software Architectures. *Proceedings - 2017 IEEE 24th International Conference on Web Services, ICWS 2017*, 524–531. https://doi.org/10.1109/ICWS.2017.61

Mohan, M., & Greer, D. (2019). Using a many-objective approach to investigate automated refactoring. *Information and Software Technology*, *112*(April), 83–101. https://doi.org/10.1016/j.infsof.2019.04.009

Nick Antonopoulos. (2011). *Cloud Computing*. 정보법학 (Second Edi, Vol. 15). Ortiz, O. (2019). Re-Factorización De Código Para Reducir El Acoplamiento Entre Clases Relacionadas Por Herencia De Implementación En ArquitecturasOrientadas A Objetos, 2020.

Othman, Z. S., & Kaya, M. (2019). Refactoring code clone detection. *7th International Symposium on Digital Forensics and Security, ISDFS 2019*, 1–6. https://doi.org/10.1109/ISDFS.2019.8757479

Ouni, A., Kessentini, M., Sahraoui, H., Inoue, K., & Deb, K. (2016). Multi-criteria code refactoring using search-based software engineering: An industrial case study. *ACM Transactions on Software Engineering and Methodology*, *25*(3). https://doi.org/10.1145/2932631

Potdar, A. M., Narayan, D. G., Kengond, S., & Mulla, M. M. (2020). Performance Evaluation of Docker Container and Virtual Machine. *Procedia Computer Science*, *171*(2019), 1419–1428. https://doi.org/10.1016/j.procs.2020.04.152

Rana, : Arooj Arif y Zeeshan Ali. (2021). Refactoring of Code to Remove Technical Debt and Reduce Maintenance Effort.

Rani, D., & Ranjan, R. K. (2014). A Comparative Study of SaaS , PaaS and IaaS inCloud Computing. *International Journal of Advanced Research in Computer Science and Software Engineering*, *4*(6), 458–461.

Rodero-Merino, L., Vaquero, L. M., Caron, E., Muresan, A., & Desprez, F. (2012). Building safe PaaS clouds: A survey on security in multitenant softwareplatforms. *Computers and Security*, *31*(1), 96–108. https://doi.org/10.1016/j.cose.2011.10.006

Seaman, C., & Guo, Y. (2011). *Measuring and Monitoring Technical Debt*. *Advances in Computers* (1st ed., Vol. 82). Elsevier Inc. https://doi.org/10.1016/B978-0-12-385512-1.00002-5

Singh, V., & Peddoju, S. K. (2017). Container-based microservice architecture for cloud applications. *Proceeding - IEEE International Conference on Computing,Communication and Automation, ICCCA 2017*, *2017-Janua*, 847–852. https://doi.org/10.1109/CCAA.2017.8229914

Sirqueira, T. F. M., Brandl, A. H. M., Pedro, E. J. P., Silva, R. D. S., & Araújo, M. A. P. (2016). Code Smell Analyzer: A Tool To Teaching Support Of Refactoring Techniques Source Code. *IEEE Latin America Transactions*, *14*(2), 877–884. https://doi.org/10.1109/TLA.2016.7437235

Srirama, S. N., Adhikari, M., & Paul, S. (2020). Application deployment using containers with auto-scaling for microservices in cloud environment. *Journal of Network and Computer Applications*, *160*(March), 102629. https://doi.org/10.1016/j.jnca.2020.102629

Surianarayanan, C., & Chelliah, P. R. (2019). *Essentials of Cloud Computing: AHolistic Perspective*.

Taibi, D., Lenarduzzi, V., & Pahl, C. (2017). Processes, Motivations, and Issues forMigrating to Microservices Architectures: An Empirical Investigation. *IEEE Cloud Computing*, *4*(5), 22–32. https://doi.org/10.1109/MCC.2017.4250931

Tello, R. (2021). *Método de refactorización en arquitecturas de software concarencia de abstracción*.

Tsai, W. L. (2021). Constructing Assessment Indicators for Enterprises EmployingCloud IaaS. *Asia Pacific Management Review*, *26*(1), 23–29. https://doi.org/10.1016/j.apmrv.2020.06.001

Valderas, P., Torres, V., & Pelechano, V. (2020). A microservice composition approach based on the choreography of BPMN fragments. *Information andSoftware Technology*, *127*(July 2019). https://doi.org/10.1016/j.infsof.2020.106370

Verdecchia, R. (2021). Building and evaluating a theory of architectural technical debt in software-intensive systems. *Journal of Systems and Software*.

Yli-huumo. (2016). How do software development teams manage technical debt ? – An empirical study. *The Journal of Systems & Software*, *0*, 1–24. https://doi.org/10.1016/j.jss.2016.05.018

Yoshida, N., Numata, S., Choiz, E., & Inoue, K. (2019). Proactive clone recommendation system for extract method refactoring. *Proceedings - 2019IEEE/ACM 3rd International Workshop on Refactoring, IWOR 2019*, 67–70.https://doi.org/10.1109/IWoR.2019.00020

Zafeiris, V. E., Poulias, S. H., Diamantidis, N. A., & Giakoumakis, E. A. (2017). Automated refactoring of super-class method invocations to the Template Method design pattern. *Information and Software Technology*, *82*, 19–35.https://doi.org/10.1016/j.infsof.2016.09.008

Anexo A.- Descripción de métricas de calidad

Métrica FHI (Factor de Herencia de Implementación)

La métrica FHI mide el factor de herencia de implementación que tiene una clase con respecto a otra. Dada una clase derivada, esta métrica consiste en la suma de los métodos virtuales y no virtuales implementados en su clase base, entre el número total de métodos de dicha clase base.

La expresión matemática de la métrica FHI se muestra a continuación:

$$\text{FHI} = \frac{\sum MV + \sum MnV}{\text{Tm}}$$

Donde:

Mv Métodos virtuales (no abstractos) implementados en una clase base. Estos métodos pueden/deben ser redefinidos en las clases derivadas.

Mnv Métodos no virtuales (no abstractos) implementados en una clase base. Estos métodos no se pueden redefinir en las clases derivadas.

Tm Total de métodos en una clase base.

La métrica fue normalizada para que los valores obtenidos no dependan de la cantidad de métodos de una clase dada. El mejor valor es 0, lo cual significa que una clase derivada no hereda implementación alguna desde los métodos de su clase base. En caso contrario, el valor menos deseado es el 1, lo que significa que todos los métodos de la clase derivada heredan implementación desde los métodos de clase base.

Métrica FHIJ (Factor de Herencia de Implementación de una Jerarquía de Clases)

La métrica FHIJ mide el factor de herencia de implementación que tiene una jerarquía de clases. Dada una jerarquía de clases, FHIJ se calcula de la siguiente manera: Primero se realiza la sumatoria de los cocientes, donde el numerador es la suma acumulada de métodos virtuales en la jerarquía de clases, y el divisor es la suma acumulada del total de métodos en la jerarquía de clases. Posteriormente, este resultado es dividido entre el número total de clases menos uno.

La expresión matemática de la métrica FHIJ se muestra a continuación:

$$FHIJ = \frac{\sum_{i}^{n}\left(\sum_{j=0}^{i=i-1} MVC_j \Big/ \sum_{j=0}^{j=i-i} MC_j\right)}{\text{n} - 1}$$

Donde:

Mv Cantidad de Métodos Virtuales de la Clases.

M Cantidad de Métodos Totales de las Clases.

i, j Representan el recorrido de la "i, j-*esima"* clase.

n Total de clases en la jerarquía.

La métrica fue normalizada para que los valores obtenidos no dependan de la cantidad de métodos de una clase dada. El mejor valor es 0, lo cual significa que no se está heredando implementación alguna de los métodos en la jerarquía, mientras que 1 es el valor menos deseado, que significa que cada clase en una jerarquía tiene el máximo de herencia de implementación.

Métrica FHIAC (Factor de Herencia de Implementación de una Arquitectura de Clases)

La métrica FHIAC mide el factor de herencia de implementación que tiene una arquitectura de clases. Dada una arquitectura de clases, esta métrica consiste en la suma de los FHIJ de las jerarquías de clases en la arquitectura, entre el número total de jerarquías de clases en la arquitectura (NOH).

La expresión matemática de la métrica FHIAC se muestra a continuación:

$$\text{FHIAC} = \frac{\sum FHIJ}{\text{NOH}}$$

Donde:
ΣFHIJ Sumatoria de *FHIJ* de una jerarquía de clases.
NOH Es el conteo de las jerarquías de clase.

La métrica fue normalizada para que los valores obtenidos no dependan de la cantidad de métodos de una clase dada. El mejor valor es 0, lo cual significa que ninguna de las jerarquías de la arquitectura, hereda implementación desde los métodos, mientras que 1 es el valor menos deseado, lo que significa que cada jerarquía de clases en la arquitectura tiene el máximo de herencia de implementación.

Métrica FFC (Factor de Flexibilidad de Clases)

La métrica FFC mide el factor de flexibilidad que tiene una clase con respecto a otra. Dada una clase base, esta métrica consiste en la suma del NOP, entre el número total de sus métodos.

La expresión matemática de la métrica FFC se muestra a continuación:

$$\text{FFC} = \frac{\sum NOP}{\text{T}_m}$$

Donde:
NOP Métodos virtuales que exhiben un comportamiento polimórfico (Bansiya & Davis, 2002).
Tm Total de métodos en una clase base.

La métrica fue normalizada para que los valores obtenidos no dependan de la cantidad de métodos de una clase dada. El mejor valor es 1, lo cual significa que todos los métodos de la clase son abstractos y/o virtuales redefinidos en las clases derivadas, por lo tanto, todos exhiben un comportamiento polimórfico. En caso contrario, el valor menos deseado es el 0, lo cual significa que no se permite la redefinición de ningún método. Es decir, ninguno de los métodos es abstracto.

Métrica FMFAC (Factor Medio de Flexibilidad de Arquitecturas de Clases)

Se definió una métrica para medir el factor de flexibilidad que tiene una arquitectura de clases. Esta métrica consiste en la suma de los FFC de la arquitectura de un módulo entre el número total de clases.

La expresión matemática de la métrica FMFAC se muestra a continuación:

$$\text{FMFAC} = \frac{\sum FFC}{\text{T}_C}$$

Donde:
FFC Factor de Flexibilidad de Clase.
Tc Total de clases en la arquitectura.

El valor óptimo de FMFAC es 1, lo cual significa que todos los métodos de las clases de la arquitectura son virtuales y/o abstractos, por lo tanto, todos exhiben un comportamiento polimórfico, y el valor menos deseado de FMFAC es el 0, el cual significa que ningún método de las clases de la arquitectura es abstracto y ninguno es virtual redefinido en clases derivadas, por lo tanto, ninguno tiene un comportamiento polimórfico.

Métrica Factor de Abstracción (A)

La métrica de factor de abstracción (A) mide la proporción de abstracción en un sistema de software, reflejando si el sistema es carente de usar la propiedad de herencia. Esta métrica consiste de una división del total de clases abstractas e interfaces entre el número total de clases.

La expresión matemática de la métrica A se muestra a continuación:

$$A = \frac{Na}{NC}$$

Dónde:
TCA total de clases abstractas e interfaces.
TC total de clases

A medida que esta razón tienda al 1 se indicará que hay tendencia a tener clases abstractas e interfaces y por lo tanto se presenta un alto factor de abstracción. Una medida que tienda al 0 indica que hay pocas clases abstractas e interfaces, por lo que el factor de abstracción se encuentra en un nivel bajo. El valor deseable es 1 y el peor valor es 0.

Métrica PMP (Factor de Protección Modular de Método Privadas)

La métrica PMFP mide el grado de protección modular con respecto a los métodos que han sido declarados con el calificador de alcance "*private"*. Esta métrica consiste en detectar en cada una de las clases los métodos que han sido declarados como "*private" y* realizar una suma de todas estas Método, entre el número total de Método, posteriormente realizar una suma de todos los valores obtenidos y dividirlo entre el número total de clases.

La expresión matemáticas de la métrica PMFP se muestra a continuación:

$$\mathrm{PMP} = \frac{\sum_{ci=1}^{ci=n} \left(\frac{\sum_{fi=0}^{fi=m} \mathrm{FP}}{\mathrm{FTC}} \right)}{\mathrm{NTC}}$$

Donde:

FP Método con calificador private.
Fi "*i-esima*" método.
FTC Número total de métodos de la clase.
Ci "i-*esima*" clase.
NTC Número total de clases.
PMP Factor de protección modular por métodos privados.

A medida que esta razón tienda al 1 se indicara que hay una tendencia a tener métodos "*private"* y por lo tanto se presenta una alta protección de la información. Una medida que tienda al 0 indica que hay pocos métodos "*private",* por lo que la protección de la información se encuentra con un nivel bajo. El mejor valor será el 1, el peor valor será 0.

Métrica PMPR (Protección Modular de Método Protegidas)

La métrica PMPR mide el grado de protección modular con respecto a los métodos que han sido declarados con el calificador de alcance "*protected"*. Esta métrica consiste en sumar los métodos que han sido declarados "*protected*" de cada una de las jerarquías entre el número total de métodos, posteriormente realizar la suma de todos los valores obtenidos y dividirlo entre el número total de jerarquías

La expresión matemáticas de la métrica PMPR se muestra a continuación:

$$\mathrm{PMFPr} = \frac{\sum_{ji=1}^{ji=n} \left(\frac{\sum_{fi=0}^{fi=m} \mathrm{FPr}}{\mathrm{NTF}} \right)}{\mathrm{NTJ}}$$

Donde:

FPr Método con calificador "*protected".*
Fi "*i-esima*" método *"protected"* de la jerarquía.
NTF Número total de métodos de la jerarquía.
ji "*i-esima*" jerarquía.
NTJ Número total de jerarquías.

A medida que esta razón tienda al 1 se indicara que hay una tendencia a tener métodos *protected* y por lo tanto se presenta una alta protección de la información. Una medida que tienda al 0 indica que hay pocos métodos *protected,* por lo que la protección de la información se encuentra con un nivel bajo. El mejor valor será el 1, el peor valor será 0.

Métrica PMF (Protección Modular de Método Friendly)

PMF mide el grado de protección modular con respecto a los métodos que han sido declarados con el calificador de alcance "*friendly".* Esta métrica consiste en sumar los métodos que han sido declarados "*friendly*" o son calificador de alcance (*default*) entre el número total de métodos.

La expresión matemáticas de la métrica PMF se muestra a continuación:

$$\mathrm{PMF} = \frac{\sum_{i=0}^{i=n} \mathrm{FF}}{\mathrm{NTF}}$$

Donde:
FF Método con calificador "*Friendly" o "default".*
i "*i-esima*" método "*Friendly" o "default".*
NTF Número total de métodos.
PMF Es el nivel de protección modular de métodos "*friendly".*

A medida que esta razón tienda al 1 se indicara que hay una tendencia a tener métodos "*friendly"* y por lo tanto se presenta una alta protección de la información en cuanto método "*friendly"* o *default*. Una medida que tienda al 0 indica que hay pocos métodos "*friendly"* o *default,* por lo que la protección de métodos "*friendly*" se encuentra con un nivel bajo. El mejor valor será el 1, el peor valor será 0.

Métrica PM (Protección Modular)

La métrica PM mide el grado de protección modular de un software legado. Esta métrica consiste en, la suma de los métodos que han sido declarados con el calificador de alcance diferente a "*public"*, entre el número total de métodos.

La expresión matemática de la métrica PM se muestra a continuación:

$$\mathrm{PM} = \frac{\sum_{i=0}^{i=n} \mathrm{FNP}}{\mathrm{TF}}$$

Donde:
FNP Métodos que no son "public"
N, TF Número total de métodos
PM Nivel de protección modular

A medida que esta razón tienda a 0 mayor es el problema, indicaría que hay una tendencia a tener muchos métodos públicos que no inician capacidades o requerimientos de la aplicación en evaluación y por lo tanto tienen poca protección modular y un encapsulamiento incorrecto. Una medida que tienda al 1 indica que hay pocos métodos públicos que no son iniciadores de requerimientos o capacidades de la aplicación en evaluación, por lo tanto, tienden a tener más capacidad de protección modular y mejor nivel de encapsulamiento. El mejor valor será de 1, el peor valor será de 0.

Métrica TPM (Total de Protección Modular)

TPM mide el grado total de protección modular que tiene una arquitectura de clases. Esta métrica consiste en la suma de PMFP, PMPR y PMF entre tres.

La expresión matemática de la métrica TPM se muestra a continuación:

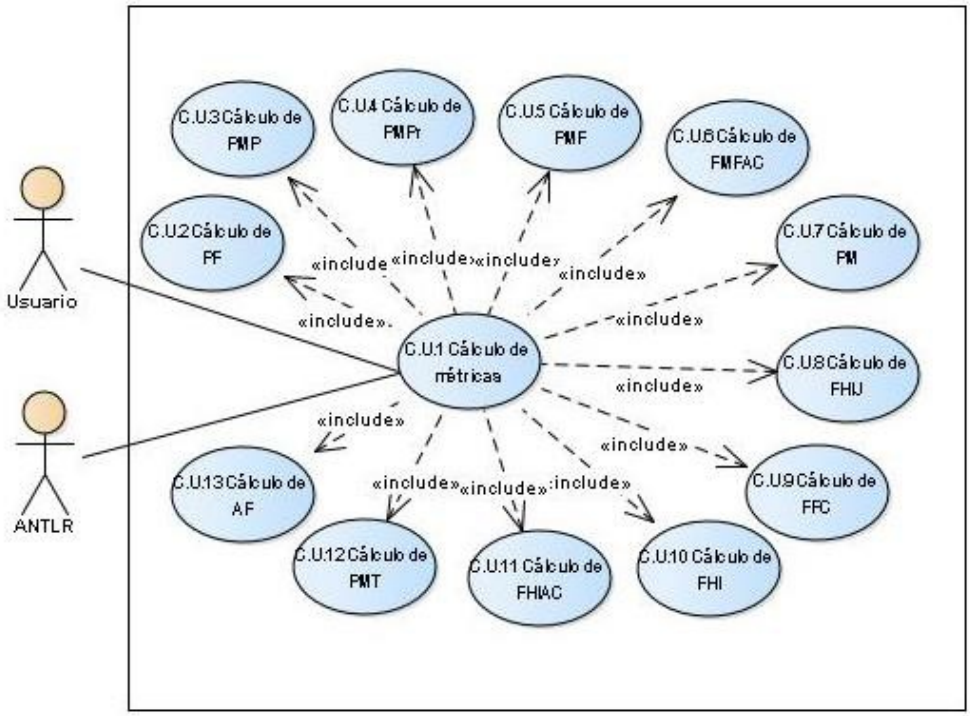
$$\text{TPM} = \frac{((\text{PMFP} * 1) \ + (\text{PMFPr} * 0.75) \ + (\text{PMF} * 0.25))}{NTF}$$

Donde:
PMFP Grado de protección modular de métodos "*private"*.
PMPR Grado de protección modular de métodos "*protected"*.
PMF Es el grado de protección modular de métodos "*friendly" o "default"*.
NTF Es el número total de métodos.
TPM Es el nivel total de protección modular.

La métrica fue normalizada con el objetivo de que los valores obtenidos se encuentren dentro del rango del 0 al 1. A medida que esta razón tienda a 0 mayor es el problema, indicaría que hay una tendencia a tener muchos métodos públicos que no inician capacidades o requerimientos de la aplicación en evaluación y por lo tanto tienen poca protección modular y un encapsulamiento incorrecto.

Una medida que tienda al 1 indica que hay pocos métodos públicos que no son iniciadoras de requerimientos o capacidades de la aplicación en evaluación, por lo tanto, tienden a tener más capacidad de protección modular y mejor nivel de encapsulamiento. El mejor valor será de 1, el peor valor será de 0.

Anexo B.- Casos de uso



Nombre:	C.U.1: Cálculo de métricas
Descripción:	Evalúa niveles de los factores de calidad: abstracción, protección modular y herencia de implementación de unsistema escrito en lenguaje Java.
Actores:	Usuario físico, subsistema ANTLR
Precondiciones:	Lista de información resultante del subsistema ANTLR.
Postcondiciones:	Resultados de las métricas: PMP, PMPR, PMF, PM, TPM, FHI, FHIJ, FHIAC, FFC, FMFAC y A.
Escenario principal de éxito:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se identifican las características a medir de cada una de las métricas. 3. Se realiza el cálculo de cada una de las métricas. 4. El sistema informa al usuario que el cálculo se realizó con éxito. 5. El sistema muestra al usuario el nivel de los factores de calidad correspondientes. 6. Termina caso de uso.

Escenario de fracaso:	7. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 8. El sistema detecta una situación no anticipada y no puede seguir realizando los cálculos 9. El sistema informa al usuario la existencia de un error durante el proceso. 10. Termina caso de uso.

Nombre:	C.U.3: Cálculo de PMP
Descripción:	El cálculo de PMP consiste en detectar en cada una de las clases las funciones que han sido declaradas como "*private*" y realizar una suma de todas estas funciones, entre el número total de funciones, posteriormente realizar una suma de todos los valores obtenidos y dividirlo entre el número total de clases. obtenidos y dividirlo entre el número total, de clases.
Actores:	Usuario físico, subsistema ANTLR
Precondiciones:	Lista de información resultante del subsistema ANTLR
Postcondiciones:	Resultado de la métrica PMP
Escenario principalde éxito:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria resultante de la división de la sumatoria de todos los métodos *private* entre el número total de métodos de cada clase. 3. Se realiza la división de la sumatoria obtenida en el paso 2 entre la el número total de clases. 4. Se envía resultado de PMP. 5. Termina el caso de uso.
Excepciones:	
Escenario de fracaso1:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria de todos los métodos sobreescritos de cada clase. 3. El sistema informa al usuario la presencia de un error durante el proceso. 4. Termina el caso de uso.

Nombre:	C.U.4: Cálculo de PMPr
Descripción:	Consiste en sumar los métodos que han sido declarados "*protected*" de cada una de las jerarquías entre el número total de métodos, posteriormente realizar la suma de todos los valores obtenidos y dividirlo entre el número total de jerarquías
Actores:	Usuario físico, subsistema ANTLR
Precondiciones:	Lista de información resultante del subsistema ANTLR.
Postcondiciones:	Resultado de la métrica PMPr
Escenario principal de éxito:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria de los métodos *protected* de cada jerarquía entre en número de total de métodos. 3. Se realiza la división entre la sumatoria obtenida en el paso entre el número de jerarquías. 4. El sistema informa al usuario termino exitoso del cálculo. 5. Se envía resultado de PMPr. 6. Termina el caso de uso.
Excepciones:	
Escenario de fracaso1:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria de los métodos *protected* de cada jerarquía entre en número de total de métodos. 3. Se realiza la división entre la sumatoria obtenida en el paso entre el número de jerarquías. 4. El sistema detecta un problema durante el proceso. 5. El sistema informa al usuario la presencia de un error durante el proceso. 6. Termina el caso de uso.

Nombre:	C.U.5: Cálculo de PMF
Descripción:	Mide el grado de protección modular con respecto a las funciones que han sido declaradas con el calificador de alcance "*friendly*". Esta métrica consiste en sumar las funciones que han sido declaradas "*friendly*" o son calificador de alcance (default) entre el número total funciones.
Actores:	Usuario físico, subsistema ANTLR
Precondiciones:	Lista de información resultante del subsistema ANTLR
Postcondiciones:	Resultado de la métrica PMF

Escenario principal de éxito:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria de los métodos *friendly*. 3. Se realiza la división entre la sumatoria obtenida en el paso 2 entre el número total de métodos. 4. El sistema informa al usuario termino exitoso del cálculo. 5. Se envía resultado de PMF. 6. Termina el caso de uso.
Excepciones:	
1. Escenario de fracaso 1:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria de los métodos *friendly*. 3. El sistema detecta un problema durante el proceso. 4. El sistema informa al usuario la presencia de un error durante el proceso. 5. Termina el caso de uso.

Nombre:	C.U.6: Cálculo de PM
Descripción:	Consiste en, la suma de los métodos que han sido declaradas con el calificador de alcance diferente a "*public*", entre el número total de métodos.
Actores:	Usuario físico, subsistema ANTLR
Precondiciones:	Lista de información resultante del subsistema ANTLR
Postcondiciones:	Resultado de la métrica PM
Escenario principalde éxito:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria de los métodos no públicos. 3. Se realiza la división entre la sumatoria obtenida en el paso 2 entre el número total de métodos. 4. El sistema informa al usuario termino exitoso del cálculo. 5. Se envía resultado de PM. 6. Termina el caso de uso.
Excepciones:	

Escenario de fracaso 1:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria de los métodos no públicos. 3. Se realiza la división entre la sumatoria obtenida en el paso 2 entre el número total de métodos. 4. El sistema detecta un problema durante el proceso. 5. El sistema informa al usuario la presencia de un error durante el proceso. 6. Termina el caso de uso.

Nombre:	C.U.7: Cálculo de PMT
Descripción:	Consiste en la suma de PMFP, PMFPr y PMFF entre el número total de métodos.
Actores:	Usuario físico, subsistema ANTLR
Precondiciones:	Lista de información resultante del subsistema ANTLR
Postcondiciones:	Resultado de la métrica PMT
Escenario principalde éxito:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria de los resultados de las métricas PMP, PMPr y PMF. 3. Se realiza la división entre la sumatoria obtenida en el paso 2 entre el número total de métodos. 4. El sistema informa al usuario termino exitoso del cálculo. 5. Se envía resultado de PMT. 6. Termina el caso de uso.
Excepciones:	
Escenario de fracaso1:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria de los resultados de las métricas PMP, PMPr y PMF. 3. Se realiza la división entre la sumatoria obtenida en el paso 2 entre el número total de métodos. 4. El sistema detecta un problema durante el proceso. 5. El sistema informa al usuario la presencia de un error durante el proceso. 6. Termina el caso de uso.

Nombre:	C.U.8: Cálculo de FFC ()
Descripción:	Consiste en la suma de los métodos virtuales, entre el número total de sus métodos

Actores:	Usuario físico, subsistema ANTLR
Precondiciones:	Lista de información resultante del subsistema ANTLR
Postcondiciones:	Resultado de la métrica FFC
Escenario principalde éxito:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria de los métodos virtuales. 3. Se realiza la división entre la sumatoria obtenida en el paso 2 entre el número total de métodos en una clase. 4. El sistema informa al usuario termino exitoso del cálculo. 5. Se envía resultado de FFC. 6. Termina el caso de uso.
Postcondiciones:	Resultado de la métrica FFC
Escenario de fracaso1:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria de los métodos virtuales. 3. Se realiza la división entre la sumatoria obtenida en el paso 2 entre el número total de métodos en una clase. 4. El sistema detecta un problema durante el proceso. 5. El sistema informa al usuario la presencia de un error durante el proceso. 6. Termina el caso de uso.

Nombre:	C.U.9: Cálculo de FHI ()
Descripción:	Consiste en la suma de los métodos virtuales y no virtuales implementados en su clase base, entre el número total de métodos de dicha clase base.
Actores:	Usuario físico, subsistema ANTLR
Precondiciones:	Lista de información resultante del subsistema ANTLR
Postcondiciones:	Resultado de la métrica FHI
Escenario principal de éxito:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria de los métodos virtuales y no virtuales. 3. Se realiza la división entre la sumatoria obtenida en el paso 2 entre el número total de métodos de la clase. 4. El sistema informa al usuario termino exitoso del cálculo. 5. Se envía resultado de FHI. 6. Termina el caso de uso.

Excepciones:	
Escenario de fracaso1:	1. Recibe como entrada la lista resultante análisis delcódigo realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria de los métodos virtuales y novirtuales. 3. Se realiza la división entre la sumatoria obtenida en el paso 2 entre el número total de métodos de la clase. 4. El sistema detecta un problema durante el proceso. 5. El sistema informa al usuario la presencia de un errordurante el proceso. 6. Termina el caso de uso.

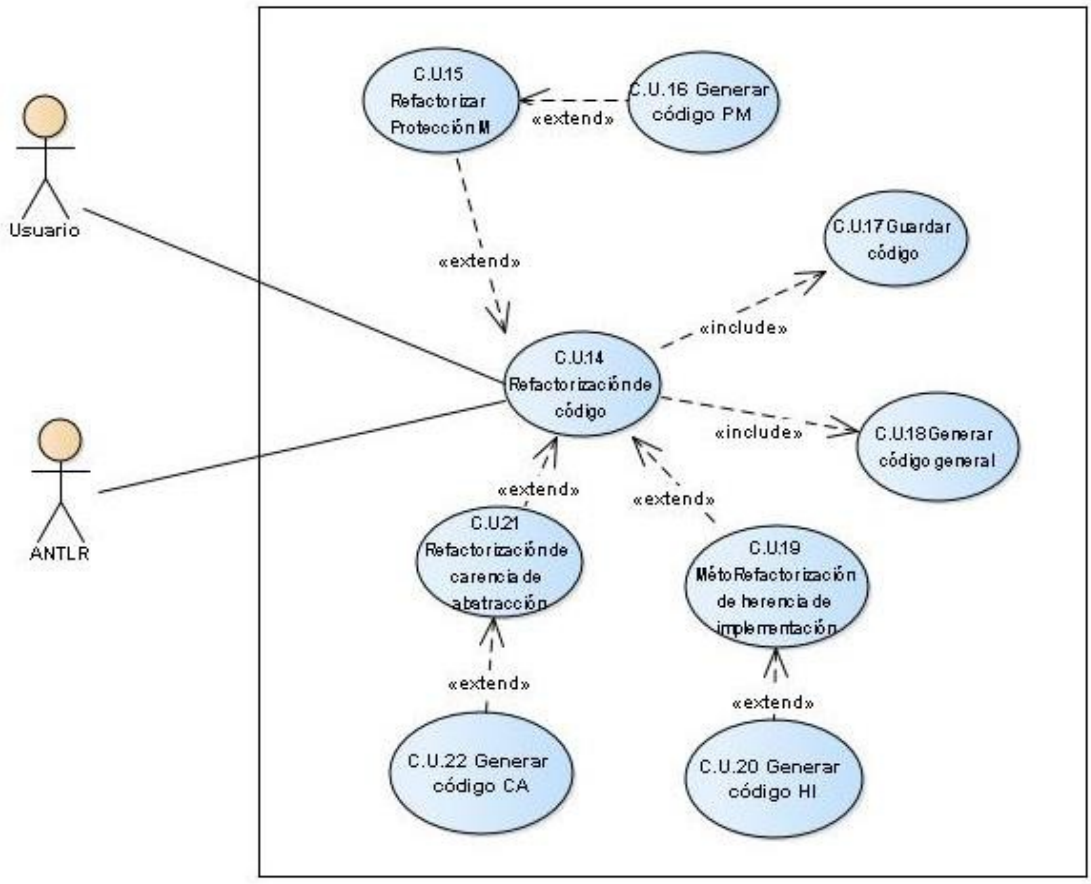
Nombre:	C.U.10: Cálculo de FHIJ ()
Descripción:	Mide el factor de herencia de implementación que tiene unajerarquía de clases. Dada una jerarquía de clases.
Actores:	Usuario físico, subsistema ANTLR
Precondiciones:	Lista de información resultante del subsistema ANTLR
Postcondiciones:	Resultado de la métrica FHIJ
Escenario principal de éxito:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria de los métodos virtuales entre el total de métodos presentes en la jerarquía. 3. Se realiza la división entre la sumatoria obtenida en el paso 2 entre el número total d clases de la jerarquía menos uno. 4. El sistema informa al usuario termino exitoso del cálculo. 5. Se envía resultado de FHIJ. 6. Termina el caso de uso.
Excepciones:	
Escenario de fracaso1:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria de los métodos virtuales entre el total de métodos presentes en la jerarquía. 3. Se realiza la división entre la sumatoria obtenida en el paso 2 entre el número total de clases de la jerarquía menos uno. 4. El sistema detecta un problema durante el proceso. 5. El sistema informa al usuario la presencia de un error durante el proceso. 6. Termina el caso de uso.

Nombre:	C.U.11: Cálculo de FHIAC ()
Descripción:	Consiste en la suma de los FHIJ de las jerarquías de clases en la arquitectura, entre el número total de jerarquías de clases en la arquitectura
Actores:	Usuario físico, subsistema ANTLR
Precondiciones:	Lista de información resultante del subsistema ANTLR
Postcondiciones:	Resultado de la métrica FHIAC
Escenario principalde éxito:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria de los FHIJ. 3. Se realiza la división entre la sumatoria obtenida en el paso 2 entre el número total de jerarquías. 4. El sistema informa al usuario termino exitoso del cálculo. 5. Se envía resultado de FHIAC. 6. Termina el caso de uso.
Excepciones:	
Escenario de fracaso1:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria de los FHIJ. 3. Se realiza la división entre la sumatoria obtenida en el paso 2 entre el número total de jerarquías. 4. El sistema detecta un problema durante el proceso. 5. El sistema informa al usuario la presencia de un error durante el proceso. 6. Termina el caso de uso.

Nombre:	C.U.12: Cálculo de A ()
Descripción:	Mide la proporción de abstracción en un sistema de software, reflejando si el sistema es carente de usar la propiedad de herencia.
Actores:	Usuario físico, subsistema ANTLR
Precondiciones:	Lista de información resultante del subsistema ANTLR
Postcondiciones:	Resultado de la métrica A
Escenario principalde éxito:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria de las clases abstractas e interfaces. 3. Se realiza la división entre la sumatoria obtenida en el paso 2 entre el número total de clases. 4. El sistema informa al usuario termino exitoso del cálculo. 5. Se envía resultado de A. 6. Termina el caso de uso.

Excepciones:	
Escenario de fracaso1:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria de las clases abstractas e interfaces. 3. Se realiza la división entre la sumatoria obtenida en el paso 2 entre el número total de clase. 4. El sistema detecta un problema durante el proceso. 5. El sistema informa al usuario la presencia de un error durante el proceso. 6. Termina el caso de uso.

Nombre:	C.U.13: Cálculo de FMFAC ()
Descripción:	Mide el factor de flexibilidad que tiene una arquitectura de clases
Actores:	Usuario físico, subsistema ANTLR
Precondiciones:	Lista de información resultante del subsistema ANTLR
Postcondiciones:	Resultado de la métrica FMFAC
Escenario principalde éxito:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria de los FFC. 3. Se realiza la división entre la sumatoria obtenida en el paso 2 entre el número total de clases. 4. El sistema informa al usuario termino exitoso del cálculo. 5. Se envía resultado de FMFAC. 6. Termina el caso de uso.
Excepciones:	
Escenario de fracaso1:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se obtiene la sumatoria de los FFC. 3. Se realiza la división entre la sumatoria obtenida en el paso 2 entre el número total de clases. 4. El sistema detecta un problema durante el proceso. 5. El sistema informa al usuario la presencia de un errordurante el proceso. 6. Termina el caso de uso.



Nombre:	C.U.14: Refactorización de código
Descripción:	Aplica los métodos de refactorización: carencia de abstracción,protección modular y reducción de herencia de implementación.
Actores:	Usuario físico, subsistema ANTLR
Precondiciones:	Lista de información resultante del subsistema ANTLR.
Postcondiciones:	Lista de código refactorizada por los métodos derefactorización carencia de abstracción, protección modular y reducción de herencia de implementación.
Escenario principal de éxito:	1. Recibe como entrada la lista resultante análisis del códigorealizado por el subsistema ANTLR. 2. Se identifican las características necesarias para laejecución de los métodos de refactorización. 3. Se ejecutan los métodos de refactorización seleccionados. 4. El sistema informa al usuario que el proceso derefactorización se realizó con éxito. 5. Termina caso de uso.

Excepciones:	
Postcondiciones:	Lista compleja de datos refactorizada.
Escenario de fracaso:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR. 2. Se identifican las características necesarias para la ejecución de los métodos de refactorización. 3. Se ejecutan los métodos de refactorización seleccionados. 4. El sistema detecta una situación no anticipada y no puede continuar con el proceso de refactorización. 5. El sistema informa al usuario la existencia de un error durante el proceso. 6. Termina caso de uso.

Nombre:	C.U.15: Refactorización de protección modular
Descripción:	Este método de refactorización, identifica el modificador de alce de cada uno de los métodos presente en el código de unaaplicación escrita en lenguaje Java.
Actores:	Usuario físico, subsistema ANTLR
Precondiciones:	Lista de información resultante del subsistema ANTLR.
Postcondiciones:	Lista compleja de datos refactorizada por el método derefactorización de protección modular.
Escenario principal de éxito:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR 2. Se recorre la lista compleja de datos, identificando el calificador de alcance de cada método. 3. El sistema informa al usuario que el proceso de refactorización se realizó con éxito. 4. Termina caso de uso.
Excepciones:	
Escenario de fracaso:	1. Recibe como entrada la lista resultante análisis del código realizado por el subsistema ANTLR 2. Se recorre la lista compleja de datos, identificando el calificador de alcance de cada método. 3. El sistema detecta una situación no anticipada y no puede continuar con el proceso. 4. El sistema informa al usuario la existencia de un error durante el proceso. 5. Termina caso de uso.

Nombre:	C.U.16: Generar código PM
Descripción:	Genera los archivos nuevos que contienen el código de refactorizado por el método de refactorización de protección modular.
Actores:	Usuario físico, subsistema ANTLR
Precondiciones:	Lista de información resultante del subsistema ANTLR.
Postcondiciones:	Archivos Java con el código refactorizado por el método de refactorización de protección modular.
Escenario principal de éxito:	1. Recibe como entrada la lista resultante de la refactorización de protección modular. 2. Se crea cada archivo con el código refactorizado de cadaclase. 3. El sistema resguarda los nuevos archivos con el nuevocódigo. 4. El sistema informa al usuario que los archivos fueronresguardados con éxito. 5. Termina caso de uso.
Excepciones:	
Escenario de fracaso:	1. Recibe como entrada la lista resultante de la refactorización de protección modular. 2. Se crea cada archivo con el código refactorizado de cadaclase. 3. El sistema detecta un problema con la generación decódigo. 4. El sistema informa al usuario la presencia de un error durante el proceso. 5. Termina el caso de uso.

Nombre:	C.U.17: Guardar código
Descripción:	Guarda los archivos java nuevos que contiene el códigorefactorizado.
Actores:	Usuario físico, subsistema ANTLR
Precondiciones:	Archivos Java con el código refactorizado.
Postcondiciones:	Archivos Java con el código refactorizado resguardado.
Escenario principal de éxito:	1. Recibe los nuevos archivos Java con el nuevo código 2. Resguarda cada uno de los nuevos archivos Java 3. El sistema informa al usuario que los archivos se guardaron correctamente. 4. Termina caso de uso.
Excepciones:	

Escenario de fracaso:	1. Recibe los nuevos archivos Java con el nuevo código 2. El sistema detecta un problema al guardar los archivos Java. 3. El sistema informa al usuario la presencia de un errordurante el proceso. 4. Termina el caso de uso.

Nombre:	C.U.18: Generar código general
Descripción:	Genera los archivos nuevos que contienen el código de refactorizado por los métodos de refactorización de carencia de abstracción, protección modular y reducción de herencia de implementación.
Actores:	Usuario físico, subsistema ANTLR
Precondiciones:	Lista compleja de datos previamente refactorizada por losmétodos de carencia de abstracción, protección modular y herencia de implementación.
Postcondiciones:	Archivos Java con el código refactorizado los métodos derefactorización de carencia de abstracción, protección modular y herencia de implementación
Escenario principal de éxito:	1. Recibe como entrada la lista compleja de datos refactorizadapor los tres métodos de refactorización. 2. Se crea cada archivo con el código refactorizado de cadaclase. 3. El sistema resguarda los nuevos archivos con el nuevocódigo. 4. El sistema informa al usuario que los archivos fueron resguardados con éxito. 5. Termina caso de uso.
Excepciones:	
Escenario de fracaso:	1. Recibe como entrada la lista compleja de datos refactorizadapor los tres métodos de refactorización. 2. Se crea cada archivo con el código refactorizado de cadaclase. 3. El sistema detecta un problema con la generación de código. 4. El sistema informa al usuario la presencia de un errordurante el proceso. 5. Termina el caso de uso.

Nombre:	C.U.19: Refactorización de herencia de implementación
Descripción:	Este método de refactorización identifica escenarios en donde presenta herencia de implementación para su refactorización.
Actores:	Usuario físico, subsistema ANTLR
Precondiciones:	Lista de información resultante del subsistema ANTLR.

Postcondiciones:	Lista compleja de datos refactorizada por el método derefactorización de herencia de implementación.
Escenario principal de éxito:	1. Recibe como entrada la lista resultante análisis del códigorealizado por el subsistema ANTLR 2. Se recorre la lista compleja de datos, identificando herencia de implementación. 3. Se refactoriza el código. 4. El sistema informa al usuario que el proceso de refactorización se realizó con éxito. 5. Termina caso de uso.
Excepciones:	
Escenario de fracaso:	1. Recibe como entrada la lista resultante análisis del códigorealizado por el subsistema ANTLR 2. Se recorre la lista compleja de datos, identificando herencia de implementación. 3. El sistema detecta una situación no anticipada y no puedecontinuar con el proceso. 4. El sistema informa al usuario la existencia de un errordurante el proceso. 5. Termina caso de uso.

Nombre:	C.U.20: Generar código HI
Descripción:	Genera los archivos nuevos que contienen el código de refactorizado por el método de refactorización de reducción de herencia de implementación.
Actores:	Usuario físico, subsistema ANTLR
Precondiciones:	Lista de información resultante del subsistema ANTLR.
Postcondiciones:	Archivos Java con el código refactorizado por el método de refactorización de reducción de herencia de implementación.
Escenario principal de éxito:	1. Recibe como entrada la lista resultante de la refactorizaciónde reducción de herencia de implementación. 2. Se crea cada archivo con el código refactorizado de cadaclase. 3. El sistema resguarda los nuevos archivos con el nuevocódigo. 4. El sistema informa al usuario que los archivos fueronresguardados con éxito. 5. Termina caso de uso.
Excepciones:	

Escenario de fracaso:	1. Recibe como entrada la lista resultante de la refactorización de reducción de herencia de implementación. 2. Se crea cada archivo con el código refactorizado de cadaclase. 3. El sistema detecta un problema con la generación de código. 4. El sistema informa al usuario la presencia de un errordurante el proceso. 5. Termina el caso de uso.

Nombre:	C.U.21: Refactorización de carencia de abstracción
Descripción:	Este método de refactorización identifica los métodos públicospresentes en una clase, de tal manera determina la creación de una clase abstracta conformada por los métodos públicosidentificados.
Actores:	Usuario físico, subsistema ANTLR
Precondiciones:	Lista de información resultante del subsistema ANTLR.
Postcondiciones:	Lista compleja de datos refactorizada por el método derefactorización de carencia de abstracción.
Escenario principal de éxito:	1. Recibe como entrada la lista resultante análisis del códigorealizado por el subsistema ANTLR 2. Se recorre la lista compleja de datos, identificando los métodos calificados como públicos. 3. Se determina la creación de clases abstractas. 4. El sistema informa al usuario que el proceso de refactorización se realizó con éxito. 5. Termina caso de uso.
Excepciones:	
Escenario de fracaso:	1. Recibe como entrada la lista resultante análisis del códigorealizado por el subsistema ANTLR 2. Se recorre la lista compleja de datos, identificando los métodos calificados como públicos. 3. Se determina la creación de clases abstractas. 4. El sistema detecta un problema durante el proceso derefactorización. 5. El sistema informa al usuario la presencia de un errordurante el proceso. 6. Termina el caso de uso.

Nombre:	C.U.22: Generar código CA
Descripción:	Genera los archivos nuevos que contienen el código de refactorizado por el método de refactorización de carencia de abstracción.

Actores:	Usuario físico, subsistema ANTLR
Precondiciones:	Lista de información resultante del subsistema ANTLR.
Postcondiciones:	Archivos Java con el código refactorizado por el método de refactorización carencia de abstracción.
Escenario principal de éxito:	1. Recibe como entrada la lista resultante de la refactorizacióncarencia de abstracción. 2. Se crea cada archivo con el código refactorizado de cadaclase. 3. El sistema resguarda los nuevos archivos con el nuevocódigo. 4. El sistema informa al usuario que los archivos fueron resguardados con éxito. 5. Termina caso de uso.
Excepciones:	
Escenario de fracaso:	1. Recibe como entrada la lista resultante de la refactorizacióncarencia de abstracción. 2. Se crea cada archivo con el código refactorizado de cadaclase. 3. El sistema detecta un problema con la generación de código. 4. El sistema informa al usuario la presencia de un errordurante el proceso. 5. Termina el caso de uso.

Anexo C.- Resultados de las combinaciones de ejecución de métodos de refactorización en los 30 sistemas

En este anexo se presentan los resultados detallados obtenidos mediante la aplicación de las seis secuencias de métodos de refactorización sobre los 30 sistemas Java utilizados durante la validación. Su propósito es proporcionar la evidencia completa de los valores estructurales generados durante la experimentación y complementar el análisis global desarrollado en el Capítulo 7.

Para cada sistema se incluye una tabla que contiene el nombre del proyecto, la dirección de su repositorio y los valores de las métricas obtenidos al aplicar las secuencias S1, S2, S3, S4, S5 y S6. Cada secuencia corresponde a un orden específico de ejecución de los métodos de refactorización MR1, MR2 y MR3, lo que permite observar las variaciones producidas en las dimensiones estructurales evaluadas.

Las métricas presentadas permiten observar el comportamiento de las dimensiones de protección modular, herencia de implementación y abstracción ante los distintos órdenes de ejecución. Los valores obtenidos constituyen la base para calcular la función de calidad correspondiente a cada secuencia y realizar la comparación entre la selección del algoritmo Greedy tradicional y la del algoritmo Greedy adaptado.

Debido a que el procedimiento experimental y la estructura de las tablas son los mismos para todos los sistemas, los resultados se presentan de manera consecutiva, sin repetir una interpretación individual para cada proyecto. El análisis de los patrones generales, las frecuencias de selección y las diferencias entre ambos enfoques se desarrolla en las secciones 7.5 y 7.6. De esta manera, el presente anexo funciona como evidencia detallada y permite consultar los valores específicos de cualquiera de los sistemas evaluados.

No. Programa	Nombre				URL	
1	Blue-Block-master				https://github.com/abc013/Blue-Block	
Valores de las métricas aplicando S1						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.404	0	0.311	0.66	0.007	1	0.778
Valores de las métricas aplicando S2						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.404	0	0.311	0.66	0.007	1	0.778
Valores de las métricas aplicando S3						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.404	0	0.311	0.66	0.007	1	0.778
Valores de las métricas aplicando S4						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.213	0	0.054	0.306	0.003	0.325	0.778
Valores de las métricas aplicando S5						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.213	0	0.054	0.306	0.003	0.325	0.591
Valores de las métricas aplicando S6						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.213	0	0.054	0.306	0.003	0.325	0.591

	Nombre				URL	
2	chess-system-java-master				https://github.com/acenello/chess-system-java	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.293	0	0.079	0.438	0.005	0.325	0.8
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.293	0	0.079	0.438	0.005	0.325	0.8
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.293	0	0.079	0.438	0.005	0.325	0.8
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.213	0	0.054	0.306	0.003	0.325	0.789
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.213	0	0.054	0.306	0.003	0.325	0.789
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.213	0	0.054	0.306	0.003	0.325	0.789

No. Programa	Nombre				URL	
3	CompanyManagementSystem-master				https://github.com/idemi2077/CompanyManagementSystem	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.186	0.322	0.033	0.6	0.019	0.556	0.716
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.186	0.322	0.033	0.6	0.019	0.556	0.716
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.186	0.322	0.033	0.6	0.019	0.556	0.716
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.186	0.322	0.033	0.6	0.019	0.556	0.716
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.186	0.322	0.033	0.6	0.019	0.556	0.716
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.186	0.322	0.033	0.6	0.019	0.556	0.716

No. Programa	Nombre				URL	
4	CustomerManagerApp-master				https://github.com/Sepoban z/CustomerManagerApp	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.424	0	0.242	0.727	0.029	1	0.818
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.455	0	0.458	0.667	0.028	1	0.818
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.424	0	0.242	0.727	0.029	1	0.818
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.395	0	0.058	0.519	0.011	1	0.714
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.395	0	0.058	0.519	0.011	1	0.714
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.395	0	0.058	0.519	0.011	1	0.714

No. Programa	Nombre				URL	
5	FileOperations-master				https://github.com/berkayd ursun/FileOperations	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.5	0	0.522	0.739	0.033	1	0.907
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.5	0	0.522	0.739	0.033	1	0.907
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.5	0	0.522	0.739	0.033	1	0.907
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.455	0	0.458	0.667	0.028	1	0.818
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.455	0	0.458	0.667	0.028	1	0.818
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.455	0	0.458	0.667	0.028	1	0.818

No. Programa	Nombre				URL	
6	HospitalProject-master				https://github.com/fdeniz07/HospitalProject	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.429	0	0.026	0.524	0.005	1	0.66
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.429	0	0.026	0.524	0.005	0.267	0.66
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.429	0	0.026	0.524	0.005	1	0.66
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.134	0	0.007	0.121	0.001	1	0.571
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.134	0	0.007	0.121	0.001	1	0.571
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.134	0	0.007	0.121	0.001	1	0.571

No. Programa	Nombre				URL	
7	InventoryManagementSystem-master				https://github.com/twilson/InventoryManagementSystem	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.139	0	0.694	0.833	0.01	1	0.8
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.139	0	0.694	0.833	0.01	1	0.8
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.139	0	0.694	0.833	0.01	1	0.2
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0	0	0.028	0.222	0	1	0.6
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0	0	0.028	0.222	0	1	0.6
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0	0	0.028	0.222	0	1	0.6

No. Programa	Nombre				URL	
8	Java_Console-Course-Management-System-main				https://github.com/DrNeons/Java_Console-Course-Management-System	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.182	0	0.333	0.5	0.007	0.545	0.385
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.182	0	0.333	0.5	0.007	0.545	0.385
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.182	0	0.333	0.5	0.007	0.5	0.385
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0	0	0.778	0.778	0.011	0.5	0.545
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0	0	0.778	0.778	0.011	0.5	0.545
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0	0	0.778	0.778	0.011	0.5	0.545

No. Programa	Nombre				URL	
9	Java_Imobiliaria-master				https://github.com/Marcello-Goncalves/Java_Imobiliaria	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.465	0	0.438	0.688	0.044	0.4	0.167
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.465	0	0.438	0.688	0.044	0.577	0.167
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.465	0	0.438	0.688	0.044	0.4	0.167
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.354	0	0.368	0.526	0.026	0.4	0.167
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.354	0	0.368	0.526	0.026	0.34	0.167
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.354	0	0.368	0.526	0.026	0.34	0.167

No. Programa	Nombre				URL	
10	Java-Console-Card-Game-master				https://github.com/dnavas77/Java-Console-Card	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.388	0	0.39	0.902	0.022	1	0.9
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.388	0	0.39	0.902	0.022	1	0.9
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.388	0	0.39	0.902	0.022	1	0.9
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.288	0	0.27	0.722	0.003	1	0.875
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.288	0	0.27	0.722	0.003	1	0.875
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.288	0	0.27	0.722	0.003	1	0.875

No. Programa	Nombre				URL	
11	Juego-El-ahorcado-en-Java-master				https://github.com/davidmazparrote/Juego-El-ahorcado-en-Java	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.583	0	0	0.667	0.292	1	0
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.583	0	0	0.667	0.292	1	0
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.583	0	0	0.667	0.292	1	0
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0	0	0.111	0.111	0.003	1	0.25
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0	0	0.111	0.111	0.003	1	0.25
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0	0	0.111	0.111	0.003	1	0.25

No. Programa	Nombre				URL	
12	Library_Management_System_11-Jan-2022-master				https://github.com/hariprakash2113/Library_Management_System_11-Jan-2022	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.483	0	0.081	0.865	0.036	1	0.444
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.483	0	0.081	0.865	0.036	1	0.444
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.483	0	0.081	0.865	0.036	1	0.444
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.345	0	0.081	0.657	0.013	1	0.467
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.345	0	0.081	0.657	0.013	1	0.467
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.345	0	0.081	0.657	0.013	1	0.467

No. Programa	Nombre				URL	
13	Movie-Service-Review-Java-main				https://github.com/lakshyGupta/Movie-Service-Review-Java	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.133	0	0.7	0.8	0.017	1	0.222
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.133	0	0.7	0.8	0.017	1	0.222
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.133	0	0.7	0.8	0.017	1	0.222
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.013	0	0.04	0.03	0.001	1	0.344
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.013	0	0.04	0.03	0.001	1	0.344
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.013	0	0.04	0.03	0.001	1	0.344

No. Programa	Nombre	URL
14	online-quiz-application-in-java-with-jdbc-using-swing-login-registration-timer-master	https://github.com/srilekhadasari/online-quiz-application-in-java-with-jdbc-

Valores de las métricas aplicando S1						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.55	0	0	0.588	0.032	1	0.143
Valores de las métricas aplicando S2						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.55	0	0	0.588	0.032	1	0.143
Valores de las métricas aplicando S3						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.55	0	0	0.588	0.032	1	0.143
Valores de las métricas aplicando S4						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.471	0	0	0.566	0.028	1	0.146
Valores de las métricas aplicando S5						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.471	0	0	0.566	0.028	1	0.146
Valores de las métricas aplicando S6						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.471	0	0	0.566	0.028	1	0.146

No. Programa	Nombre	URL
15	Password-Generator	https://github.com/KZar20ul/Password-Generator

Valores de las métricas aplicando S1						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.457	0	0.105	0.737	0.048	1	0.833
Valores de las métricas aplicando S2						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.457	0	0.105	0.737	0.048	1	0.833
Valores de las métricas aplicando S3						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.457	0	0.105	0.737	0.048	1	0.833
Valores de las métricas aplicando S4						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.104	0	0	0.32	0.006	1	0.714
Valores de las métricas aplicando S5						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.104	0	0	0.32	0.006	1	0.714
Valores de las métricas aplicando S6						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.104	0	0	0.32	0.006	1	0.714

No. Programa	Nombre				URL	
16	PasswordManager-main				https://github.com/coderslinver/PasswordManager	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.25	0	0.25	0.694	0.006	0.75	0.261
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.25	0	0.25	0.694	0.006	0.75	0.261
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.25	0	0.25	0.694	0.006	0.75	0.261
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.018	0	0.197	0.5	0.005	0.75	0.261
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.018	0	0.197	0.5	0.005	0.71	0.261
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.018	0	0.197	0.5	0.005	0.71	0.261

No. Programa	Nombre				URL	
17	Polinom				https://github.com/taijased/univers_cours2_Java/tree/	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.347	0	0.132	0.684	0.016	1	0.875
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.347	0	0.132	0.684	0.016	1	0.875
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.347	0	0.132	0.684	0.016	1	0.875
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.234	0	0.12	0.455	0.08	1	0.78
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.234	0	0.12	0.455	0.08	1	0.79
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.234	0	0.12	0.455	0.08	1	0.78

No. Programa	Nombre				URL	
18	qurbani-animals-market-main				https://github.com/MSarmadQadeer/qurbani-animals-market	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.338	0.135	0.241	0.782	0.011	1	0.333
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.338	0.135	0.241	0.782	0.011	1	0.333
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.338	0.135	0.241	0.782	0.011	1	0.333
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.245	0.135	0.241	0.567	0.005	1	0.25
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.245	0.135	0.241	0.567	0.005	1	0.25
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.245	0.135	0.241	0.567	0.005	1	0.25

No. Programa	Nombre				URL	
19	radio				https://github.com/KI7MT/java-app-	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.638	0	0.083	0.813	0.037	1	0.818
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.638	0	0.083	0.813	0.037	1	0.818
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.638	0	0.083	0.813	0.037	1	0.818
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0	0	0	0	0	1	0.8
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0	0	0	0	0	1	0.8
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0	0	0	0	0	1	0.8

No. Programa	Nombre	URL
20	real-estate-console-app-master	https://github.com/Leo-Chan01/real-estate-console-app

Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.32	0	0.194	0.861	0.023	1	1
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.32	0	0.194	0.861	0.023	1	1
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.32	0	0.194	0.861	0.023	1	1
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.194	0	0.071	0.196	0.004	1	0.889
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.194	0	0.071	0.196	0.004	1	0.889
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.194	0	0.071	0.196	0.004	1	0.889

No. Programa	Nombre	URL
21	SchoolManagement-master	https://github.com/idenz077/SchoolManagement

Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.153	0.059	0	0.383	0.005	1	0.889
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.153	0.059	0	0.383	0.005	1	0.889
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.153	0.059	0	0.383	0.005	1	0.889
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.194	0	0.071	0.196	0.004	1	0.55
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.194	0	0.071	0.196	0.004	1	0.55
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.194	0	0.071	0.196	0.004	1	0.55

No. Programa	Nombre				URL	
22	shapp-master				https://github.com/vicfanm/shapp	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.561	0	0.118	0.735	0.033	1	0.714
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.561	0	0.118	0.735	0.033	1	0.714
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.561	0	0.118	0.735	0.033	1	0.714
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.0294	0	0.211	0.684	0.014	1	0.611
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.0294	0	0.211	0.684	0.014	1	0.611
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.0294	0	0.211	0.684	0.014	1	0.611

No. Programa	Nombre				URL	
23	Simple-Java-Calculator-master				https://github.com/pH-7/Simple-Java	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.467	0	0.385	0.846	0.051	1	1
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.467	0	0.385	0.846	0.051	1	0.8
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.467	0	0.385	0.846	0.051	1	0.8
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0	0	0	0	0	1	0.67
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.144	0	0.056	0.333	0.009	1	0.67
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.144	0	0.056	0.333	0.009	1	0.67

No. Programa	Nombre				URL	
24	TetrisForDesktopWithJava-master				https://github.com/kkikkodev/TetrisForDesktopWithJava	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.144	0	0.056	0.333	0.009	1	1
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.144	0	0.056	0.333	0.009	1	1
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.144	0	0.056	0.333	0.009	1	1
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.004	0	0.04	0.025	0.006	1	0.889
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.004	0	0.04	0.025	0.006	1	0.889
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.004	0	0.04	0.025	0.006	1	0.889

No. Programa	Nombre				URL	
25	tic-tac-toe-console-master				https://github.com/rohandeb...	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.583	0	0.294	0.941	0.094	1	1
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.583	0	0.294	0.941	0.094	1	1
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.583	0	0.294	0.941	0.094	1	1
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.583	0	0.294	0.941	0.094	1	1
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.583	0	0.294	0.941	0.094	1	1
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.583	0	0.294	0.941	0.094	1	1

No. Programa	Nombre				URL	
26	tiny-compiler-master				https://github.com/alimfinho/tiny-compiler	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.454	0	0.085	0.744	0.016	1	1
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.454	0	0.085	0.744	0.016	1	1
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.454	0	0.085	0.744	0.016	1	1
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.452	0	0.085	0.732	0.012	1	0.999
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.452	0	0.085	0.732	0.012	1	0.999
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.452	0	0.085	0.732	0.012	1	0.999

No. Programa	Nombre				URL	
27	treinaweb-java-oo-master				https://github.com/treinaweb/treinaweb-java-oo	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.18	0.136	0	0.605	0.009	1	0.975
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.18	0.136	0	0.605	0.009	1	0.975
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.18	0.136	0	0.605	0.009	1	0.975
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.18	0.136	0	0.605	0.009	1	0.975
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.18	0.136	0	0.605	0.009	1	0.975
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.18	0.136	0	0.605	0.009	1	0.975

No. Programa	Nombre	URL
28	UniversalCalculator-master	https://github.com/dima666Sik/UniversalCalculator

Valores de las métricas aplicando S1						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.406	0.114	0.143	0.816	0.028	0.013	0.867
Valores de las métricas aplicando S2						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.406	0.114	0.143	0.816	0.028	0.013	0.867
Valores de las métricas aplicando S3						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.406	0.114	0.143	0.816	0.028	0.013	0.867
Valores de las métricas aplicando S4						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.308	0	0.017	0.55	0.011	0.013	0.65
Valores de las métricas aplicando S5						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.308	0	0.017	0.55	0.011	0.013	0.65
Valores de las métricas aplicando S6						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.308	0	0.017	0.55	0.011	0.013	0.65

No. Programa	Nombre	URL
29	Llaveros-master	https://github.com/JHORJE18/Llaveros

Valores de las métricas aplicando S1						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.381	0	0.267	0.8	0.064	1	1
Valores de las métricas aplicando S2						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.381	0	0.267	0.8	0.064	1	1
Valores de las métricas aplicando S3						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.381	0	0.267	0.8	0.064	1	1
Valores de las métricas aplicando S4						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.37	0	0.267	0.5	0.032	1	1
Valores de las métricas aplicando S5						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.37	0	0.267	0.5	0.032	1	1
Valores de las métricas aplicando S6						
PMMP	**PMMPr**	**PMMF**	**PM**	**PMT**	**FHIAC**	**A**
0.37	0	0.267	0.5	0.032	1	1

No. Programa	Nombre				URL	
30	AI-Chat-Bot-master				https://github.com/gazalpat el/AI-Chat-Bot	
Valores de las métricas aplicando S1						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.55	0	0	0.857	0.275	1	1
Valores de las métricas aplicando S2						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.55	0	0	0.857	0.275	1	1
Valores de las métricas aplicando S3						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.55	0	0	0.857	0.275	1	1
Valores de las métricas aplicando S4						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.55	0	0	0.857	0.275	1	1
Valores de las métricas aplicando S5						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.55	0	0	0.857	0.275	1	1
Valores de las métricas aplicando S6						
PMMP	PMMPr	PMMF	PM	PMT	FHIAC	A
0.55	0	0	0.857	0.275	1	1