Universidad Nacional de Loja

Facultad de Energía, las Industrias y los Recursos Naturales no Renovables

Carrera de Computación

Evaluación de mantenibilidad para aplicaciones web en producción de los laboratorios de la carrera de computación de la UNL

Evaluation of the maintainability of web applications in production of the laboratories of the UNL computer science career

Trabajo de Integración Curricular, previo a la obtención del título de Ingeniero en Ciencias de la Computación.

AUTOR:

Gilson Orlando Quezada Guartizaca

DIRECTOR:

Ing. Francisco Javier Álvarez Pineda Mg. Sc.

Loja – Ecuador

2024

Certificación

Loja, 11 de diciembre de 2024

Ing. Francisco Javier Álvarez Pineda, Mg. Sc.

DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

CERTIFICO:

Haber revisado y orientado todo el proceso de la elaboración del Trabajo de Integración Curricular denominado: **Evaluación de mantenibilidad para aplicaciones web en producción de los laboratorios de la carrera de computación de la UNL**, previo a la obtención del título de **Ingeniero en Ciencias de la Computación** de autoría del estudiante: **Gilson Orlando Quezada Guartizaca** con cédula de identidad Nro. **1150006979**, una vez que el trabajo cumple con todos los requisitos exigidos por la Universidad Nacional de Loja, para el efecto, autorizo la presentación del mismo para su respectiva sustentación y defensa.

Firmado electrónicamente por:
FRANCISCO JAVIER
ALVAREZ PINEDA

Ing. Francisco Javier Álvarez Pineda Mg. Sc.
DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

Autoría

Yo, **Gilson Orlando Quezada**, declaro ser autor del presente Trabajo de Integración Curricular y eximo expresamente a la Universidad Nacional de Loja y a sus representantes jurídicos, de posibles reclamos y acciones legales, por el contenido del mismo. Adicionalmente, acepto y autorizo a la Universidad Nacional de Loja la publicación de mi Trabajo de Integración Curricular en el Repositorio Institucional - Biblioteca Virtual.

Firma: 

Cédula de identidad: 1150006979

Fecha: Loja, 15 de diciembre de 2024

Correo electrónico: gilson.quezada@unl.edu.ec

Teléfono: (+593) 959 811 860

Carta de autorización por parte del autor, para la consulta, reproducción parcial y/o total, publicación electrónica de texto completo del Trabajo de Integración Curricular

Yo, **Gilson Orlando Quezada**, declaro ser autor del Trabajo de Integración Curricular denominado: **Evaluación de mantenibilidad para aplicaciones web en producción de los laboratorios de la carrera de computación de la UNL, previo a la obtención del título de Ingeniero en Ciencias de la Computación,** como requisito para optar al título de **Ingeniero en Ciencias de la Computación**, autorizo al sistema Bibliotecario de la Universidad Nacional de Loja para que, con fines académicos, muestre la producción intelectual de la Universidad, a través de la visibilidad de su contenido de la siguiente manera en el Repositorio Institucional.

Los usuarios pueden consultar el contenido de este trabajo en el Repositorio Institucional, en las redes de información del país y del exterior con las cuales tenga convenio la Universidad.

La Universidad Nacional de Loja, no se responsabiliza por el plagio o copia del Trabajo de Integración Curricular que realice un tercero.

Para constancia de esta autorización, suscribo, en la ciudad de Loja, a los once días del mes de diciembre del dos mil veinticuatro.

Firma: 

Autor: Gilson Orlando Quezada Guartizaca

Cédula: 1150006979

Dirección: Loja, Catamayo, Barrio Catamayto Km 6 vía a Gonzanamá

Correo electrónico: gilson.quezada@unl.edu.ec

Teléfono: (+593) 95 981 1860

DATOS COPLEMENTARIOS:

Director del Trabajo de Titulación: Ing. Francisco Javier Álvarez Pineda, Mg. Sc.

Dedicatoria

Para mis padres **Carmen Guartizaca y Roger Quezada** por su apoyo incondicional, que ha sido el motor que me impulsa a seguir adelante y haber estado a mi lado en aquellos momentos difíciles, brindándome palabras de aliento, consejos valiosos y un amor inmenso que me ha motivado a alcanzar mis metas.

Gilson Orlando Quezada Guartizaca.

Agradecimiento

Quiero expresar mi más sincero agradecimiento a mis padres y hermanos, quienes han sido mi mayor apoyo a lo largo de este proceso, su amor incondicional, paciencia y aliento constante me han motivado a superar los desafíos y a perseguir mis sueños. Así mismo, quisiera expresar mi más sincero agradecimiento aquellas personas por su invaluable apoyo a lo largo de estos años, su dedicación y comprensión han sido fundamentales en mi camino, y me siento verdaderamente afortunado de contar con su presencia en mi vida.

A sí mismo a mi director, al Ing. Francisco Alvarez Pineda, le dedico este trabajo con profunda gratitud, por su orientación y apoyo incondicional han sido fundamentales a lo largo de todo el proceso de elaboración del presente TIC. Agradezco infinitamente el tiempo que ha dedicado a cada una de las revisiones, brindándome valiosos consejos y guiándome para superar obstáculos y ver más allá de las limitaciones.

Gilson Orlando Quezada Guartizaca.

Índice de contenidos

Índice de tablas:

Índice de figuras:

Índice de anexos:

1. Título

Evaluación de mantenibilidad para aplicaciones web en producción de los laboratorios de la carrera de computación de la UNL.

Maintainability assessment for web applications in production in the UNL computing career laboratories

2. Resumen

La mantenibilidad es un atributo clave de la calidad del software, ya que facilita la localización y corrección de defectos en la mejora del rendimiento o la adaptación a nuevos requisitos, su omisión puede resultar en un código difícil de comprender y modificar, lo que incrementa costos de desarrollo y mantenimiento. En respuesta a la escasa aplicación de buenas prácticas de programación en el desarrollo de aplicaciones web para los laboratorios de la Carrera de Computación de la UNL, se propone evaluar el grado de mantenibilidad de las aplicaciones web y desarrollar un software para verificar la calidad del código, como una estrategia para identificar y corregir errores de código. La evaluación se centró en investigar y adaptar un modelo de evaluación basado en la familia de las normas ISO/IEC 25000, con el propósito de evaluar el grado de mantenibilidad y obtener defectos del código. El software se desarrolló utilizando la metodología XP, incluyendo actividades como entrevistas y encuestas para recopilar necesidades y expectativas de los usuarios, este enfoque facilitó el levantamiento de historias de usuario y requisitos, lo que a su vez permitió la elaboración de diagramas de caso de uso, clases, componentes y despliegue, esenciales para el desarrollo y pruebas del software. Además, se realizó una revisión bibliográfica de buenas prácticas de programación y utilización de la API de SonarQube para calcular el grado de mantenibilidad del código. Los resultados del proyecto indican que el modelo de evaluación adaptado para los casos de estudio determinó que la aplicación SDLC alcanzó un grado de mantenibilidad del 89.61%, mientras que la aplicación GestionVR obtuvo un 82.34%. La prueba de aceptación de la aplicación web QualityCCode realizada según la escala de Likert, mostró un resultado del 91.60% entre 25 encuestados, lo que refleja que los usuarios perciben la aplicación como útil y fácil de usar. Finalmente, el plan de estrategias implementado para abordar los defectos de mayor prioridad en el backend de la aplicación GestionVR resultó favorable, aumentando el grado de mantenibilidad del producto del 82.34% al 89.62%, clasificado como altamente mantenible (no requerirá esfuerzo adicional para su comprensión y modificación). Contar con una herramienta enfocada en la verificación de la mantenibilidad de aplicaciones web, basada en la evaluación del código y el uso de la API de SonarQube, permite no solo identificar defectos y áreas de mejora, sino también garantizar que el producto final cumpla con los estándares de calidad de código exigidos.

Palabras clave: XP, SDLC, Mantenibilidad, ISO/IEC 25000, QualityCCode, Producción de laboratorios.

Abstract

Maintainability is a key attribute of software quality, as it facilitates the location and correction of defects in order to improve performance or adapt to new requirements. Its omission can result in code that is difficult to understand and modify, which increases development and maintenance costs. In response to the poor application of good programming practices in the development of web applications for the laboratories of the UNL Computer Science Degree, it is proposed to evaluate the degree of maintainability of web applications and develop software to verify code quality, as a strategy to identify and correct code errors. The evaluation focused on researching and adapting an evaluation model based on the ISO/IEC 25000 family of standards, with the purpose of evaluating the degree of maintainability and obtaining code defects. The software was developed using the XP methodology, including activities such as interviews and surveys to collect user needs and expectations. This approach facilitated the collection of user stories and requirements, which in turn allowed the development of use case, class, component and deployment diagrams, essential for software development and testing. In addition, a literature review of good programming practices and use of the SonarQube API was conducted to calculate the degree of maintainability of the code. The results of the project indicate that the evaluation model adapted for the case studies determined that the SDLC application achieved a maintainability degree of 89.61%, while the GestionVR application obtained 82.34%. The acceptance test of the QualityCCode web application carried out according to the Likert scale, showed a result of 91.60% among 25 respondents, reflecting that users perceive the application as useful and easy to use. Finally, the strategy plan implemented to address the highest priority defects in the backend of the GestionVR application was favorable, increasing the degree of maintainability of the product from 82.34% to 89.62%, classified as highly maintainable (it will not require additional effort for its understanding and modification). Having a tool focused on verifying the maintainability of web applications, based on code evaluation and the use of the SonarQube API, allows not only to identify defects and areas for improvement, but also to ensure that the final product meets the required code quality standards.

Key words: XP, Maintainability, ISO/IEC 25000, QualityCCode, Laboratory production.

3. Introducción

La mantenibilidad es uno de los atributos de calidad más importantes en un producto de software, ya que se refiere a la facilidad con la que se puede identificar y solucionar un defecto [1], así como mejorar su rendimiento o adaptarse a nuevos requisitos. Un nivel aceptable de este atributo permite responder rápidamente a los cambios del entorno y a las necesidades de los usuarios, lo que resulta un software más eficiente y efectivo, por el contrario, la falta de este atributo puede resultar en un código difícil de comprender, modificar y extender, lo que a su vez puede incrementar los costos de desarrollo y mantenimiento [2].

En contraste, lo siguiente puede surgir cuando un equipo o encargado del desarrollo del software no implementa buenas prácticas de programación, por ejemplo, la ausencia de pruebas unitarias, código duplicado, documentación inadecuada y otros aspectos similares, lo que puede dar lugar a errores y en consecuencia defectos en el producto final.

La creciente necesidad de abordar defectos en el software y ofrecer soluciones efectivas ha llevado a generar un mayor interés en la mantenibilidad como objeto de estudio. En consecuencia, se ha realizado investigaciones sobre trabajos relacionados en este campo (Ver **Tabla 17**), que subrayan la importancia de la mantenibilidad en el desarrollo de software y su evaluación desde las etapas iniciales del ciclo de vida del mismo. Al igual que los "productos de software que implementan buenas prácticas de programación y además optan por evaluaciones de calidad para la solución de defectos tienden a tener mayor probabilidad de éxito en el mercado" [3], [4], [5].

En el centro de datos de los laboratorios de la Carrera de Computación de la UNL, se ha identificado una falta de aplicación de buenas prácticas de programación el desarrollo de aplicaciones web (Ver **PIC**). Este problema se atribuye a la falta de conocimiento sobre temas relacionados con la calidad y a la ausencia de herramientas que verifiquen la mantenibilidad, como resultado, se ha reportado un alto número de defectos vinculados a la mantenibilidad del código.

Existen diversas teorías que apoyan la importancia de aplicar buenas prácticas de programación para solucionar defectos de mantenibilidad dentro del contexto del control de calidad. Se sostiene que "la calidad se construye a través de un proceso continuo de desarrollo, verificación y optimización, donde el control de calidad debe integrarse en cada fase. La implementación de buenas prácticas de programación, junto con el uso de herramientas de

análisis y pruebas, es esencial para garantizar tanto la mantenibilidad y la calidad del producto final" [6].

En este contexto, y considerando lo mencionado anteriormente, el siguiente trabajo surge como respuesta a la falta de aplicación de buenas prácticas de programación, cuyo propósito es evaluar el grado de mantenibilidad de aplicaciones web y desarrollar un software para verificar la calidad del código, esta herramienta se propone como estrategia para identificar y corregir errores en el código, contribuyendo así a mejorar la calidad del desarrollo de software. Para ello, se llevará a cabo una revisión bibliográfica de buenas prácticas de programación y se utilizará la API de SonarQube, junto con operaciones matemáticas para calcular el grado de mantenibilidad del código. Ante esta situación, se ha planteado la siguiente pregunta: ¿Qué grado de mantenibilidad tienen las aplicaciones web en producción de los laboratorios de la carrera de computación de la UNL, tomando como referencia la familia de la norma ISO/IEC 25000?

Con el propósito de dar respuesta a la pregunta planteada del presente Trabajo de Integración Curricular, se establecen los siguientes objetivos:

- Diagnosticar las aplicaciones web en producción de los laboratorios de la carrera de Computación de la UNL, mediante la evaluación del grado de mantenibilidad basada en la Norma ISO/IEC 25000.
- Desarrollar un software web tomando como referencia XP como metodología de desarrollo, para solucionar el grado de mantenibilidad de una de las aplicaciones web diagnosticadas en el objetivo uno.

El desarrollo del primer objetivo se basa en la investigación y el análisis de documentos científicos y académicos que abordan la familia de las normas ISO/IEC 25000, así como los modelos de calidad aplicados a la mantenibilidad de aplicaciones web, con el propósito de evaluar el grado de mantenibilidad e identificar defectos. En cuanto al segundo objetivo, se emplearon técnicas como la documentación de historias de usuario y el levantamiento de requisitos, siguiendo la norma IEEE 830. A partir de esta información, se diseñaron diagramas de casos de uso, clases, componentes y despliegue, que servirán como base para la codificación del sistema, así como para las pruebas de aceptación y unitarias del mismo, garantizando así la calidad del software. Además, se utilizaron tecnologías de desarrollo como MongoDB y JavaScript. Finalmente, para abordar la solución de defectos, se implementó un plan de

estrategias enfocado en los defectos de mayor prioridad, este proceso se llevó a cabo utilizando la aplicación web desarrollada, junto con herramientas adicionales como GitHub y Excel.

4. Marco teórico

4.1. Antecedentes

En un entorno caracterizado por la rápida evolución tecnológica y la competencia global el desarrollo de aplicaciones web se ha intensificado, demandando soluciones que no solo satisfagan las necesidades de los usuarios, sino que también aseguren la mantenibilidad a lo largo del ciclo de vida del software (SDLC). Tal como lo mencionan, Echeverría en su Revisión Sistemática de Literatura (RSL) recalca que la mantenibilidad "desempeña un papel crucial en el aumento de la calidad del software" [7], en este sentido, un software con un nivel de calidad apropiado desempeña un papel fundamental en la ejecución de operaciones empresariales, debido a su correcto funcionamiento y adaptabilidad.

Así mismo, Adones y Vega [8] destacan la "importancia de evaluar desde etapas tempranas del ciclo de vida del software o inicio de cada proyecto de desarrollo de software", para implementar un producto de software de calidad. Dejando en claro lo mencionado por Martínez [9], que la mantenibilidad es "una de las características de calidad esenciales del producto, debido a que las tareas de mantenimiento consumen una gran proporción del esfuerzo total gastado en el ciclo de vida del software".

Para Chávez[3], Bautista [4] y Pardo [5] un grado de mantenibilidad adecuada, se logra cuando los productos de software implementan buenas prácticas de programación y, además, optan por evaluaciones de calidad para la solución de defectos. Este enfoque no solo optimiza el proceso de desarrollo, sino que también incrementa la probabilidad de éxito en el mercado. En la Carrera de Ingeniería en Computación (CIC) de la UNL comprometida con el desarrollo de soluciones tecnológicas en el campo de la Ingeniería de Software [10], [11], bajo los principios de calidad, en un estudio preliminar (PIC), se ha identificado posibles causas que afectan la calidad del software, lo cual resulta en la aparición de numerosos defectos en las aplicaciones desarrolladas.

En particular, durante el desarrollo de aplicaciones web para los laboratorios de computación, según encuestas dirigidas a estudiantes y entrevistas a docentes de la carrera (Ver **Anexo 6-9**), las principales causas que afectan la mantenibilidad de aplicaciones web es el limitado uso de herramientas de control de calidad, desconocimiento en profundidad de temas relacionados con la calidad del software por parte de los directores de tesis y la aplicación escasa de pruebas de calidad y buenas prácticas de programación por parte de los estudiantes.

En la presente investigación, se analizó de manera general cómo aplicar control de calidad en el desarrollo del código mediante la evaluación de mantenibilidad de aplicaciones web. Se comenzó con la identificación de los defectos derivados del diagnóstico del grado de mantenibilidad, posteriormente se investigaron buenas prácticas de programación para implementar listas de verificación digital (aplicación web) e incorporar herramientas de análisis de código estático que permitan solucionar y verificar la: modularidad, reusabilidad, analizabilidad, capacidad de ser modificado y capacidad de ser probado para un caso de estudio. Además, se utilizará normas de calidad, de la familia ISO/IEC 25000, para establecer criterios y objetivos de evaluación y garantizar que el código cumple con los estándares de mantenibilidad establecidos.

La investigación resalta la importancia de adoptar un enfoque integral para la evaluación de aplicaciones web, que no solo se centre en las métricas de calidad del producto en sí, sino que también considere el proceso de evaluación de calidad en su totalidad. De esta manera la norma ISO/IEC 25000 emerge como un pilar fundamental para el presente proyecto, al poseer los siguientes parámetros según lo mencionan Roa [12] y Reina [13] en sus artículos científicos:

- Características según el modelo de calidad ISO/IEC 25010 y métricas según la medición de calidad ISO/IEC 25020, favorece en la claridad de métricas siendo útil para la creación de matrices de prueba cuyo enfoque integral permite evaluar diversos aspectos del software.
- Proceso de evaluación según norma ISO/IEC 25040, detalla actividades claras, tareas, propósitos, entradas, resultados e información adicional para la evaluación de calidad.

En definitiva, se seleccionarán dos casos de estudio en producción para determinar el grado de mantenibilidad, la evaluación se centrará en la calidad interna (código), lo que proporcionará una base sólida para el desarrollo de una aplicación web. Esta aplicación servirá como herramienta para abordar los defectos identificados de un caso de estudio, incorporando listas de verificación y funcionalidades de extraídas de la API de SonarQube, específicamente relacionadas con la mantenibilidad del software.

4.2. Bases teóricas

En la demanda por asegurar la calidad del software, muchas organizaciones proponen metodologías, normas y estándares cuyo objetivo es mejorar, evaluar y proporcionar técnicas y métodos para el desarrollo de software y su evaluación en términos de calidad. Por lo tanto, para alcanzar el objetivo de la investigación, que se centra en el diagnóstico de aplicaciones web con el fin de aportar control de calidad (solución defectos), es esencial comprender diversas definiciones relacionadas con la calidad de software y el propósito del proyecto.

4.2.1. Calidad de software

La calidad de software, según Sifuentes y Peralta en su artículo describen como "el grado en que dicho producto satisface los requisitos de sus usuarios aportando de esta manera un valor" [14]. Otra definición interesante es la de Piattini: en su libro lo describe como "el grado en el que un conjunto de características inherentes de un objeto cumple con los requisitos" [15]. Dicho brevemente, la calidad de software es el grado con el que un producto, componente o proceso cumplen las necesidades de los usuarios forma explícita e implícita.

El grado de un producto de software se puede determinar mediante los siguientes enfoques: calidad interna, externa y en uso, estos son esenciales para garantizar su calidad desde múltiples perspectivas.

En la **Tabla 1** se muestra los tipos de enfoques, cada uno con su respectiva descripción, para medir el grado de calidad de un producto de software.

Tabla 1. Enfoques para la calidad de un producto de software: interna, externa y uso.

Enfoque	Definición	Característica de calidad ISO/IEC 25010
Calidad Interna	Es evaluada desde características propias del software, como la calidad del código y la estructura interna del software [16], se basa en los atributos del software [17] el cual determina si satisface las necesidades representadas en requerimientos bajo ciertas condiciones.	Mantenibilidad, eficiencia y fiabilidad del sistema [18], [19].
Calidad Externa	Es evaluada con características basada en el comportamiento del software, como ejecución en tiempo real [16]. Es decir, hace referencia a la calidad percibida por los usuarios.	Usabilidad, eficacia y satisfacción percibida por los usuarios [18], [19].

Enfoque	Definición	Característica de calidad ISO/IEC 25010
Calidad en Uso	Es evaluada basada en la utilización por el usuario [16]. En cierta forma es la capacidad que un software facilita a los usuarios a lograr sus metas.	Eficacia, productividad, seguridad y satisfacción del usuario final [18] [19].

Por tal efecto, el enfoque para el diagnóstico de las aplicaciones web seleccionadas es la "calidad interna", la cual se enfoca en características propias del software, como la calidad del código y la estructura interna del software para determinar si satisface las necesidades representadas en requerimientos de calidad.

La evaluación del producto de software es fundamental para garantizar su calidad y fiabilidad, no obstante, aunque mejorar el proceso de desarrollo es crucial para producir un software de alta calidad, es la evaluación del producto final la que determina en última instancia su calidad percibida y su utilidad para los usuarios. Según la norma ISO/IEC 25000 [18], la calidad del producto, junto con el proceso, es uno de los aspectos más importantes en el desarrollo de software. Asimismo, Roa [12] señala que evaluar la calidad involucra tanto el producto final como los procesos seguidos para su creación, garantizando así la calidad total del producto. Dado esto demuestra que la evaluación final del producto es indispensable para confirmar que cumple con los estándares y expectativas requeridos.

4.2.2. Evaluación de calidad

La evaluación de un producto de software según Coral, Moraga y Piattini [20] es "comprobar si el producto cumple con los atributos de calidad requeridos", también la norma ISO/IEEC 25000 [18] lo define como "un proceso sistemático para determinar el grado en que un producto de software cumple con los estándares de calidad especificados". En cierta forma, es un proceso que implica una serie de pasos en relación con estándares y criterios específicos, para la medición de las características del software en relación con los criterios de calidad establecidos.

En relación con la definición anteriormente expuesta, la evaluación del software busca determinar si este cumple con los objetivos de calidad establecidos, asegurando su adecuación a las necesidades de los usuarios finales. Este criterio lo respalda Guillen [21], quien enfatiza la importancia de realizar evaluaciones y pruebas de calidad para entregar software de alta

calidad a los usuarios, también de un método preciso para detectar posibles errores, defecto y fallos en el desarrollo del software.

En la evaluación de un producto de software, es importante comprender los conceptos de error, defecto y fallo relacionados con calidad de software, según manual de ISTQB [22]. Un error es la acción humana incorrecta, un defecto es una imperfección en el código provocado por un error, lo que puede causar un mal funcionamiento del sistema y finalmente, un fallo es el resultado de ejecutar un fragmento de código que contiene un defecto. En este caso, aplicar control de calidad el estudio se dirige a detectar y corregir defectos encontrados relacionados específicamente mantenibilidad.

Por lo tanto, la evaluación de producto de software se centra en medir y analizar la calidad global de un producto de software en relación con estándares y criterios específicos, abordando aspectos como la funcionalidad, fiabilidad, usabilidad, mantenibilidad, etc.

Para resolver la evaluación del software se puede aplicar los siguientes métodos [23]: evaluación objetiva, la cual se centra mediciones y la evaluación subjetiva basada en opiniones de personas sobre un hecho o fenómeno.

- **Evaluación objetiva:** La evaluación se centra mediciones.
- **Evaluación subjetiva**: La evaluación es basada en opiniones de personas sobre un hecho o fenómeno.

4.2.3. Gestión de la Calidad

La gestión de calidad incluye todas las actividades que dirigen y controlan una organización con respecto a la calidad, entre las cuales esta aseguramiento y control de calidad para ayudar a identificar áreas de mejora, detectar posibles defectos y tomar decisiones informadas [24].

Galin [25] define al aseguramiento de calidad como “un conjunto, sistemático y planificado, de acciones necesarias para proveer la evidencia adecuada de que el proceso de desarrollo o mantenimiento de un sistema de software cumple los requerimientos técnicos funcionales tan bien como los requerimientos gerenciales para cumplir la planificación y operar dentro del presupuesto confinado”. El control de calidad para Antonio [26] es “comprobar si un producto posee o no posee una determinada característica de calidad en el grado requerido,

así mismo Bahamon [6], menciona que el objetivo del "Control de Calidad es identificar defectos en el producto y corregirlos".

Es importante conocer el principio del control de calidad "calidad se construye a través de un proceso continuo de desarrollo, verificación (revisión) y optimización en diferentes etapas" [6], para facilitar que equipos de desarrollo pueden identificar y corregir defectos de manera temprana.

En la **Tabla 2** se menciona algunas actividades del control de calidad.

Tabla 2. Actividades del control de calidad.

Actividades
Uso de métodos y herramientas de análisis, diseño, codificación y prueba.
Revisiones técnicas formales, que se aplican durante cada paso de la Ingeniería de software.
Estrategia de prueba multiescalada.
Control de la documentación del software y de los cambios realizados.
Procedimientos que aseguren un ajuste a los estándares de desarrollo.
Mecanismos de medida de la calidad ("métricas").

Con base a lo anteriormente expuesto, se evidencia cuál es el objetivo y actividades del control de Calidad en la identificación y corrección de defectos en un producto final, esta premisa está estrechamente vinculada con el propósito del presente proyecto, que se centra en la evaluación de la mantenibilidad de aplicaciones web en producción. Pues sí, el control de calidad se convierte en una parte crucial del proceso, ya que busca garantizar que las aplicaciones web mantengan altos estándares de calidad y que cualquier defecto encontrado sea detectado y corregido de manera oportuna.

Puesto que un producto de software no posee una determinada característica de calidad se dice que tiene un **DEFECTO**.

4.2.4. Métricas de calidad de software

Las métricas de calidad de software desempeñan un papel crucial en la evaluación y mejora continua de los productos de software, las cuales proporcionan una base objetiva para medir diversos aspectos de la calidad del software[27] (fiabilidad, mantenibilidad, eficiencia y usabilidad etc.). Al aplicarlas sobre los procesos del software, posibilita la toma de decisiones informadas para mejorar un atributo específico, el cual se logra comparando indicadores o características relevantes.

Según la norma IEEE [28] define una métrica de calidad de software como “una función cuyas entradas son datos de software y cuya salida es un solo valor numérico que puede ser interpretado como el grado en que el software posee un atributo dado que afecta su calidad”. O interpretar como una medida cuantitativa que proporcionan información objetiva [29], que se utiliza para evaluar diferentes aspectos relacionados con la calidad del software.

Las métricas a nivel de medición, en la calidad del software se categoriza en tres tipos: proyectos, procesos y productos de software, de esta división, la métrica de software es la utilizada para el presente trabajo (calidad interna de un producto), ya que por su cálculo de forma cuantitativo “permite confirmar el correcto funcionamiento del software al evaluar la calidad de análisis, modelos de diseños y código fuente” [24].

4.2.5. Prueba de un producto de software

La prueba de un producto de software, según el estudio de la Universidad de Antioquia, es “un componente crítico del ciclo de vida del desarrollo de software, cuyo objetivo es ayudar a mejorar la calidad y fiabilidad del producto mediante la identificación de defectos, la prevención de errores, y la observación y presentación de reportes” [30], similar a Sommerville [31] en su libro lo define como una técnica para “demostrar que un programa hace lo que se intenta que haga, así como descubrir defectos en el programa antes de usarlo”.

A continuación, se muestra objetivos de pruebas de calidad[22].

- Evalúa áreas de trabajo en el software como: requisitos, historias de usuario, diseño y código.
- Mostrar información a los implicados para tomar decisiones informadas, especialmente en relación con el nivel de calidad del objeto de prueba.
- Prevenir y detectar fallos y defectos.

Desde este punto de vista, las pruebas para el presente proyecto se rigen como una piedra angular, cuya misión es garantizar la calidad y la fiabilidad del producto al detectar y corregir defectos de manera proactiva.

4.2.6. Prueba Estática

Para la evaluación de mantenibilidad, se emplea las pruebas estáticas, cuyo objetivo es examinar el código sin ejecutar el software. Este enfoque ayudará a identificar posibles defectos y mejorar la calidad del producto de software.

De acuerdo con la ISTQB [22], la prueba estática se basa en "evaluación manual de los productos de software (ejemplo basado en la experiencia) o en la evaluación basado en herramientas de código u otros productos de trabajo (ejemplo analizador de código estático)". En consecuencia, la prueba se puede aplicar de forma manual o automatizada de los productos del software, como el código fuente, los requisitos, el diseño y los planos de prueba, cuyo objetivo es identificar defectos directamente en el objeto de prueba[32].

La prueba estática se emplea para mejorar la calidad del software al detectar defectos en etapas tempranas del ciclo de vida del software. La principal ventaja radica en la capacidad de identificar defectos difíciles de detectar mediante pruebas dinámicas, los cuales podrían resultar más costosos de corregir si se descubren en etapas tardías del ciclo de vida del software [33], [22].

A continuación de muestra ventajas de pruebas estáticas [22]:

- Prevenir defectos en el diseño o la programación descubriendo errores, ambigüedades, contradicciones, omisiones, inexactitudes y redundancias en los requisitos.
- Incrementar la productividad de desarrollo.
- Reduce el coste y el tiempo de las pruebas.

4.2.7. Pruebas de caja blanca

La prueba de caja blanca es una técnica de prueba estática que depende del análisis de la estructura interna del software, misma que se basa en la estructura interna del sistema o en su implementación, que puede abarcar aspectos como el código, la arquitectura y los flujos de datos y de trabajo dentro del sistema [34].

El objetivo de esta técnica es verificar que el software se está construyendo de acuerdo a los estándares y las prácticas recomendadas previamente establecidas, y que no hay errores de diseño o lógica.

4.2.8. Mantenibilidad de un Producto de Software

La mantenibilidad es considerada un atributo esencial de la calidad del software[35], [9], [8], debido a los altos costos de mantenimiento, la rapidez y facilidad del producto para ser modificado después de su puesta en producción. Para la norma ISO/IEC 25000 [36], la mantenibilidad es "la capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas", garantizando su relevancia y utilidad continua en un contexto de evolución constante.

La mantenibilidad es una característica crucial, pero a menudo subestimada en el desarrollo de productos de software, a pesar de ser un factor determinante para el éxito a largo plazo, se distingue claramente entre productos creados por desarrolladores aficionados y profesionales [37]. Esta cualidad se pone a prueba cuando necesitamos adaptar la aplicación a cambios en el sistema operativo, resolución de pantalla, o realizar modificaciones en funcionalidades, etc.

Ruiz [38] identifica algunos elementos que inciden directamente en la mantenibilidad de las aplicaciones web.

- Proceso de desarrollo: La mantenibilidad debe ser un componente esencial del proceso de desarrollo de software y un principio fundamental que lo guíe.
- Documentación: En muchas situaciones no se encuentran disponibles, lo que resulta en un aumento en los tiempos y costos asociados con la comprensión, corrección y/o modificación del producto de software.
- Comprensión de Programas: Como resultado por la falta de comprensión del funcionamiento del aplicativo.

4.2.9. Modelo de Calidad

Es un proceso documentado el cual se basa en prácticas para apoyar a las organizaciones en la mejora continua y favorecer su competitividad, mediante actividades para medir la calidad y brindar productos alto nivel [39]. Según Rugel y Chacón [40] un modelo abarca un conjunto de prácticas relacionadas con la gestión y desarrollo de soluciones tecnológicas, estas prácticas son técnicas que buscan alcanzar objetivos estratégicos y cumplir con estándares de calidad del producto, mediante la idealización y aplicación efectiva de métodos y procesos adecuados.

Un concepto importante de modelo de calidad es el de Calero, Piattini, & Moraga el cual lo definen como “conjunto de factores de calidad, y de relaciones entre ellos, que proporciona una base para la especificación de requisitos de calidad y para la evaluación de calidad de componentes software” [20].

El modelo define características de un producto de software, los cuales se pueden evaluar para redimir un criterio de calidad, para esto cada una de las características divide en subcaracterísticas la cual tiene designado una métrica y un indicador que son medibles basadas en los elementos del producto de software [41].

La estructura general de un modelo de calidad comprende de varios factores de calidad que contienen criterios específicos, cada criterio se evalúa mediante métricas, lo que ayuda a reducir la subjetividad en la asignación de un valor, ya sea cuantitativo o cualitativo [39].

En la **Figura 1** se muestra la estructura de un modelo de calidad.

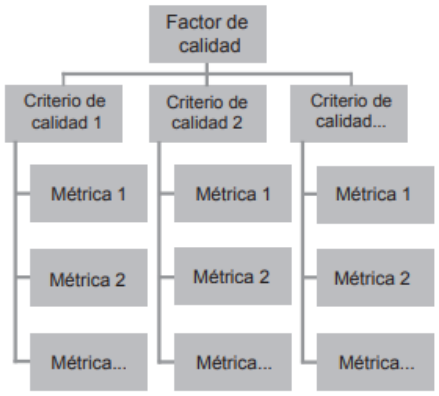


Figura 1. Estructura modelo de calidad del software.
Fuente: Revista Científica: Modelos de calidad [28].

Descripción de los niveles de un modelo de calidad [39], [42], [43]:

- **Factores de calidad**. - Se conoce como las características de calidad, desde la perspectiva del usuario.
- **Criterios de calidad**. - Es la división del factor de calidad(subcaracterísticas), representan la calidad desde la perspectiva del producto.
- **Métricas**. - Se definen para cada criterio de calidad (cuantitativas) que indican el grado de presencia de una característica en el producto.

A continuación, se conceptualiza cada uno de los enfoques del modelo de calidad.

- **Calidad a nivel de proceso. -** La calidad del proceso es la conformidad del proceso de producción de software con los estándares de calidad, así como la mejora continua del proceso para evitar la presencia de errores o defectos al momento de entregar el software [44]. Al planificar la calidad del software desde el inicio de un proyecto y en cada una de las etapas del desarrollo se minimiza los riesgos y ofrece un soporte continuo para llevar a cabo el control y seguimiento de los aspectos de calidad [39]. Lo que garantiza el cumplimiento de los factores de calidad, ya que si pasa por desapercibido la verificación de los factores y criterios es posible tener un nivel de calidad bajo, no solo del proceso además del producto de software [45].
- **Calidad a nivel de producto. -** La finalidad del modelo es especificar y evaluar el cumplimiento de calidad del producto a nivel externo e interno [46], [47], según este criterio, las normas y estándares dividen la calidad del producto en calidad interna, externa y en uso [48], cada enfoque se centra en asegurar que el producto cumpla con las características necesarias para satisfacer los requisitos establecidos al inicio del proceso de desarrollo garantizando así la satisfacción del cliente.
- **Calidad a nivel de uso. -** Es un concepto amplio que abarca elementos de la usabilidad a diferencia de la característica de un producto de software, según Callejas [39] y Baldeon [49] basados en la norma ISO/IEC 25000 lo define como "la perspectiva del usuario respecto a la calidad del producto software cuando el mismo es utilizado en un ambiente específico y un contexto de uso específico".

4.2.10. Norma ISO/IEC 25000

- **ISO/IEC 25010:** Modelo de calidad del producto, enfocado en la característica de la mantenibilidad para la calidad interna del producto.

En la **Tabla 3** se detalla cada una de las subcaracterísticas de la mantenibilidad [50].

Tabla 3. ISO/IEC 25010: 2011 subcaracterísticas mantenibilidad.

Característica	Subcaracterísticas
Mantenibilidad	• **Modularidad**. – Es la capacidad de producto de software para evitar que los cambios en alguno de sus componentes afecten a otr**o.** • **Reusabilidad.** – Es la capacidad de un activo que permita ser utilizado en otros productos de software o construir otro activo.

Característica	Subcaracterísticas
	• **Analizabilidad. -** Es el nivel de facilidad en la que se puede evaluar el impacto de un cambio sobre el resto del producto de software. • **Capacidad para ser modificado. –** Es la capacidad del producto de software que permite ser modificado de manera efectiva, esto sin provocar defectos o reducir el nivel de calidad. • **Capacidad para ser probado. -** Es la facilidad para establecer criterios de prueba para el producto o componente de software para resolver las pruebas y determinar si se cumplen dichos criterios.

- **Norma ISO/IEC 25023:** Modelo de medición de calidad, define métricas a nivel de producto en calidad interna y externa – ISO/IEC 25023.

Las métricas para la calidad Interna y externa según ISO/IEC 25023 permite evaluar las características que se definen en el modelo de calidad según ISO/IEC 25010 del producto de software.

En la **Tabla 4** se muestra las subcaracterísticas de la mantenibilidad (modelo de calidad ISO/IEC 25010) y cada una de sus respectivas métricas (modelo de medición ISO/IEC 25023) enfocada a calidad interna y externa [51]. Además, se conceptualiza cada una de las métricas [51].

Tabla 4. Métricas calidad externa e interna en características de la mantenibilidad.

Subcaracterística	Métricas
Modularidad	• **Capacidad de condensación**: "mide que tan fuerte es la relación entre los componentes del sistema"; determinando cuántos componentes se ven afectados por cambios en otros componentes, en relación con el número total de componentes específicos. • **Acoplamiento de clases:** "mide que tan fuerte es la relación entre una función del sistema con otras clases implementada"; lo cual implica contabilizar el número de relaciones que una función tiene con respecto a otra clase.
Reusabilidad	• **Ejecución de reusabilidad:** "mide cuantos elementos pueden ser reutilizados"; para lo cual cuenta el número de recursos reutilizados y el número total de recursos disponibles en el repositorio.

Subcaracterística	Métricas
Analizabilidad	• **Capacidad de pistas de auditoría:** Determina si los usuarios logran identificar fácilmente la operación específica que causó el fallo; mismo que se determina mediante el análisis del número de datos grabados durante la operación real y comparándolo con la cantidad esperada para controlar el estado del sistema durante dicha operación. • **Diagnóstico de funciones suficientes**: Determinar hasta qué punto las funciones de diagnóstico se encuentran preparadas o hasta qué punto funcionan para el análisis causal; logrando a través de la comparación del número de funciones de diagnóstico implementadas con el número de funciones de diagnóstico requeridas según la especificación de requisitos.
Capacidad de ser modificado	• **Cyclomatic complexity:** (métrica de calidad interna) Determinar cuál es la complejidad estructural de un código; se tienen en cuenta condicionales, bucles, salidas de métodos y cláusulas & y + dentro de los condicionales. • **Profundidad de herencia:** (métrica de calidad interna) Determina qué tan elevada es la jerarquía de la herencia de las clases involucradas de una función específica; para esto cuenta número de jerarquías utilizadas en dicha función o método. • **Grado de localización de corrección de impacto:** (métrica de calidad interna y externa), Determina hasta en que área los problemas causados pueden tener como consecuencia un mantenimiento; para lo cual cuenta el número de fallos después de resolver un problema comparándolo con el número total de fallos resueltos. • **Complejidad de modificación:** (métrica de calidad externa) Determina la facilidad con la que el desarrollador puede modificar el software para solucionar un problema; para lo cual toma el tiempo de trabajo necesario para que el desarrollador realice modificaciones y el número total de cambios realizados. • **Índice de éxito de modificación:** (métrica de calidad externa), Determina hasta que área el sistema puede ser operado sin fallas después aplicar las operaciones de mantenimiento; se calcula contando la cantidad de problemas detectados durante un período específico antes del mantenimiento y comparándolo con la cantidad de problemas identificados en el mismo período después del mantenimiento**.**
Capacidad de ser probado	• **Completitud funcional de funciones de pruebas:** Determinar si los métodos de prueba son completos y sencillos de implementar; el cual implementa un número de procesos de prueba y el número de procesos de prueba requeridas.

Subcaracterística	Métricas
	• **Capacidad de prueba autónoma:** (métrica de calidad interna) Determina que tan independiente es el sistema al ser evaluado; para eso cuenta el número de pruebas que dependen de otros sistemas y el número total de pruebas que dependen con otros sistemas. • **Capacidad de reinicio de pruebas:** (métrica de calidad externa) Determina la facilidad que se puede llevar a cabo las pruebas nuevamente después del mantenimiento; para lo cual cuenta el número de casos en los cuales el mantenedor puede pausar y restaurar las pruebas y contar el número de casos de pausa en la ejecución de pruebas.

- **ISO/IEC 25040:** Modelo de evaluación de calidad, la norma ISO/IEC 25040 contempla un modelo de referencia y un proceso de evaluación de la calidad del producto de software, además de los requisitos para la ejecución de este proceso.

En la **Figura 2**, se puede observar las estradas y resultados del modelo de referencia para la evaluación de calidad de un producto de software.

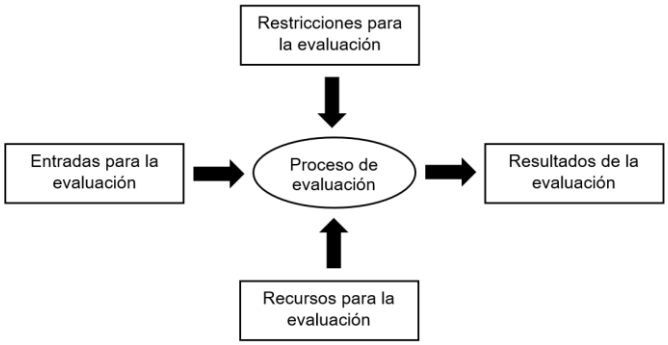


Figura 2. Modelo para proceso de evaluación de calidad.
Fuente: Norma ISO/IEC 25040[52].

A continuación, a modo de ejemplo se muestra las entradas, salidas, recursos y restricciones por cada una función del modelo de referencia para evaluación de calidad:

- **Entrada para la evaluación:** Requisitos de evaluación de calidad del producto de software.
- **Resultado de la evaluación:** Reporte de evaluación, plan de evaluación de calidad del producto de software, criterios de decision definidos por las métricas de calidad,

criterios de decision para la evaluación, planificación de las actividades de evaluación y métricas de calidad.

- **Recursos para la evaluación:** Metodología y herramientas de medida apropiadas, documentos SQuaRE: ISO/IEC 2500n, 2502n, 2503n y 2504n, recursos humanos y económicos para la evaluación, sistemas de información para la evaluación y bases de datos para la evaluación.
- **Restricciones para la evaluación:** Recursos, horarios, costos, entornos, metodología, herramientas e informes para la evaluación.

El proceso de evaluación del producto software según la familia ISO/IEC 25000 envuelve cinco actividades, el cual es reconocido como la norma ISO/IEC 25040, que tiene como objetivo principal evaluar la calidad del producto, cada actividad cumple con sus respectivas tareas y propósitos específicos, en la evaluación se utilizan diferentes entradas, como documentos, datos de pruebas y requisitos de calidad establecidos previamente lo que genera resultados que proporcionan información sobre la calidad del producto, incluyendo posibles áreas de mejora, además, de recopilar información complementaria relacionada con el contexto del producto y su entorno de desarrollo para una evaluación más completa y precisa [53].

4.2.11. Especificación norma ISO/IEC 25040

En el siguiente apartado se detalla cada una de las cinco actividades del proceso de evaluación además de los recursos de entrada y salida de cada actividad. Recuperado de Sifuentes y Peralta [54] en su artículo científico acerca de los **Modelos de medición y evaluación**, la norma la ISO/IEC 25040:2011 [55] y el estudio de la familia de la norma ISO/IEC 25000 [56].

- **Actividad 1:** Determinar los requisitos de la evaluación para definir los requisitos esenciales de la evaluación.

En la **Tabla 5** se detalla los atributos del proceso de entrada y de salida para la determinación de requisitos.

Tabla 5. Atributos de Entrada y salida en la determinación de requisitos.

Entrada	Salida
• Necesidades de la evaluación. • Producto de software a evaluar.	• Especificación del propósito de la evaluación. • Especificación de los requisitos de la evaluación.

Para establecer los requisitos de evaluación es necesario llevar a cabo las siguientes tareas. En la **Tabla 6** se especifica cada una de las tareas.

Tabla 6. Tareas para establecer los requisitos de evaluación.

Tarea	Especificación
Establecer el propósito de la evaluación	Se define el propósito por el cual se requiere evaluar la calidad del producto de software, es decir, es lo que se pretende alcanzar al aplicar un modelo de calidad.
Obtener los requisitos de calidad del producto	Se declara según el modelo de calidad ISO/IEC 25010, cada una de la característica representa un requisito de calidad del producto de software.
Identificar partes del producto para ser incluidas en la evaluación	Para identificar las partes del producto incluidas en la evaluación se debe considerar la entidad objetivo, más de sus detalles. Existen casos, de la cual depende de la entidad objetivo. • **Aseguramiento de la calidad del producto final:** Implica tomar productos finales como entidad objeto de evaluación (especificaciones del producto, código fuente, manuales, descripción del producto final, resultado de pruebas y producto durante en ejecución). • **Mejora en la calidad del producto y la productividad durante el proceso de desarrollo:** La entidad objetivo de evaluación puede ser productos intermedios (documentos del diseño, especificación del producto, código fuente del programa, documentación de las pruebas y el programa ejecutable).

- **Actividad 2:** Especificar la evaluación, en esta sección se especifican métricas, herramientas, técnicas de medición y criterios aplicados en la evaluación.

En la **Tabla 7** se encuentra los respectivos atributos para el proceso de entrada y salida, en la especificación de la evaluación.

Tabla 7. Atributos de Entrada y salida para especificar la evaluación.

Entrada	Salida
Definición de los requisitos de evaluación del producto software	• Especificación de métricas de calidad elegidas. • Especificación de criterios de decisión para evaluar las métricas de calidad. • Especificación de criterios de decisión para valorar la calidad.

En la **Tabla 8** se define cada una de las tareas para la especificación de la evaluación.

Tabla 8. Tareas para establecer los requisitos de evaluación.

Tarea	Especificación
Selección de métricas de calidad	Se realizará la selección de métricas de calidad acorde al objetivo de la evaluación y el tipo de producto bajo evaluación. Estos pueden incluir medidas tanto internas como externas, siguiendo la norma ISO/IEC 25022 Y 25023 o combinarse con las métricas que considere pertinente para la evaluación.
Definir los criterios de decisión para las métricas	Seleccionado las métricas de calidad, se debe determinar los criterios de decision para la misma, que consiste en establecer valores donde se indica el nivel de cumplimiento requerido de las métricas.
Definir los criterios de decisión para la evaluación	Es necesario un procedimiento que divida el criterio con la característica de calidad, en forma de subcaracterísticas individuales para definir un nivel de calidad de la característica.

- **Actividad 3:** Diseñar la evaluación, se procede a decidir que se va a evaluar y las herramientas a utilizar, es decir se especifica como de resuelve la evaluación.

En la **Tabla 9** se detalla los atributos del proceso de entrada y salida para el diseño de la evaluación.

Tabla 9. Atributos de Entrada y salida para el diseño de la evaluación.

Entrada	Salida
• Especificación de requisitos necesarios para la evaluación. • Definir las métricas de calidad elegidas.	• Descripción exhaustiva del plan para evaluar la calidad. • Técnicas para medir la excelencia del producto.

Entrada	Salida
• Definición de estándares de evaluación para medir la calidad. • Definición de criterios de decisión para valorar la calidad del producto.	

En la **Tabla 10** se especifica cada una de las tareas para diseñar la evaluación:

Tabla 10. Tareas para establecer los requisitos de evaluación.

Tarea	Especificación
Actividades del plan de evaluación	Las actividades deben ser planificadas según la disponibilidad de recursos humanos y materiales, el presupuesto, los métodos de evaluación y estándares aplicados, así como las herramientas disponibles. Las actividades incluidas en un plan de evaluación son: • Propósito de la evaluación de calidad, institución involucrada en la evaluación (desarrollador y evaluador), expectativa del producto a partir de la evaluación, cronograma según las etapas, responsabilidades de los involucrados, entorno de la evaluación, métodos y herramientas, criterios de decision para las métricas y la valoración de la calidad del producto, estándares utilizados y actividades de evaluación.

- **Actividad 4:** Ejecutar la evaluación**,** se lleva a cabo la ejecución de todas las actividades planificadas para la evaluación. obteniendo las métricas de calidad y aplicando los criterios de evaluación.

En la **Tabla 11** se puede identificar los atributos del proceso de entrada y salida para la ejecución de la evaluación.

Tabla 11. Atributos de Entrada y salida para ejecución de la evaluación.

Entrada	Salida
• Especificación del plan de evaluación. • Especificación requisitos de evaluación.	• Resultados de las métricas de calidad. • Resultados de la evaluación.

Entrada	Salida
• Especificaciones métricas seleccionadas. • Especificación criterios de decision para las métricas y valorar la calidad del producto. • Producto a evaluarse.	

En la **Tabla 12** se especifica cada una de las tareas para ejecutar la evaluación:

Tabla 12. Tareas para ejecutar la evaluación.

Tarea	Especificación
Efectuar las mediciones	Se aplica las mediciones del software para obtener los valores de las métricas definidas en el plan de evaluación, además, es importante registrar todos los resultados obtenidos. La medición de la evaluación se efectúa para el producto y sus componentes, cada dato se interpreta para emitir un resultado que será incluido en el informe de evaluación. Es necesario garantizar la confidencialidad de los datos registrados, resultados de la evaluación y la documentación facilitada por la institución.
Aplicar los criterios de decisión para las métricas	En la aplicación de los criterios de decisión es necesario tener los valores resultado de las mediciones, para aplicar los criterios de decision.
Aplicar los criterios de decisión de la evaluación	Permite establecer si los resultados son aceptables o no, para lo cual se debe aplicar criterios de decision a características y subcaracterísticas de calidad, en la forma que permita obtener un grado de valoración que el producto cumple con los requisitos de calidad.

- **Actividad 5:** Concluir la evaluación, son las actividades finales en donde se concluye la evaluación del producto de software mediante la elaboración de un informe con resultados, entonces se entrega al cliente una vez revisado los resultados obtenidos.

En la **Tabla 13** se muestra los atributos del proceso de entrada y salida para concluir con la evaluación.

Tabla 13. Atributos de Entrada y salida para concluir con la evaluación.

Entrada	Salida
• Especificación resultados reales del plan de evaluación. • Especificación métodos de la evaluación de calidad. • Resultados evaluación.	Informe de evaluación de calidad del producto.

En la **Tabla 14** se especifica cada una de las tareas para concluir la evaluación.

Tabla 14. Tareas para concluir la evaluación.

Tarea	Especificación
Revisión de resultados de evaluación	El evaluador y el cliente revisan los resultados de la evaluación para mejorar la interpretación y detectar errores de manera más efectiva, a medida que se: • Crea un informe de evaluación, al examinar los resultados, se genera un informe que detalla los requisitos, los resultados obtenidos, así como las limitaciones, restricciones y el personal involucrado en la evaluación. • Revisa la calidad de la evaluación, el evaluador revisa los resultados de la evaluación y la validez de los indicadores y métricas utilizadas, proporcionando retroalimentación para mejorar el proceso de evaluación.
Tratamiento los datos de evaluación	Después de la evaluación, es necesario llevar a cabo un proceso de gestión de datos acordado con el cliente, realizado por el evaluador, que trae como consecuencia devolverlos, modificarlos, guardarlos, etc.

4.2.12. Verificación del software

Las razones detrás de un defecto pueden derivarse de un problema en el hardware o de un error durante su desarrollo, lo que deriva en un comportamiento inesperado en relación con las especificaciones en una situación particular. Es común, considerar durante la etapa del desarrollo que se está llevando la lógica correcta, pero en muchos casos no es correcto, por tal razón, es importante tener en claro que los defectos vienen de errores (acción humana) que hacen que el software no siga la lógica definida.

En este caso, la investigación se dirige hacia la **verificación del software**. Para Lluna[57] en su artículo científico, menciona que es “posible eliminar defectos del sistema verificando que el código desarrollado hace exactamente lo que se ha especificado que debe hacer”, para la cual, existen diferentes técnicas como análisis estático y análisis dinámico, ensayos formales etc. [58], [59], [60], en este caso, el estudio es para análisis de código estático.

4.2.13. Herramienta para Análisis de código estático

El análisis de código estático es una técnica para la evaluación del código desarrollada para buscar defectos sin necesidad de ejecutar el código, siendo recomendada por muchas normas de desarrollo de software [57].

Aplicar el análisis de código estático, en la evaluación de un producto de software basada en algún criterio (características) de calidad, permite mediante el uso de métricas indicar el grado de presencia de una característica en el producto [61]. Igualmente, el análisis permite identificar posibles problemas y vulnerabilidades, como código muerto, duplicado o complejidad excesiva, que pueden afectar la mantenibilidad a largo plazo del software [57].

Uno de los aspectos importantes del análisis de código estático es la detección de malas prácticas de programación, mediante el uso de herramientas especializadas, se pueden identificar patrones de código que no siguen las mejores prácticas establecidas, como el acoplamiento excesivo, la falta de modularidad o la falta de comentarios, estas malas prácticas pueden dificultar la comprensión y el mantenimiento del código en el futuro, por lo que el análisis de código estático permite corregirlas y mejorar la calidad del software [62].

SonarQube es una plataforma de código abierto que ofrece revisiones automáticas del código mediante análisis estático, su objetivo es detectar errores, problemas de diseño de código (conocidos como 'code smells') y vulnerabilidades de seguridad en una amplia gama de lenguajes de programación, que incluyen Java, C#, JavaScript, TypeScript, C/C++, COBOL, Python, entre otros. Al mismo tiempo, SonarQube promueve un enfoque proactivo para mejorar la calidad del código durante el proceso de desarrollo [63].

SonarQube aborda aspectos fundamentales de la calidad del software, al analizar un proyecto proporciona información detallada sobre diversos aspectos, como la arquitectura y el diseño, comentarios en el código, duplicación de código, cumplimiento de reglas de programación específicas del lenguaje, posibles bugs y sugerencias de solución, complejidad

del proyecto y resultados de las pruebas unitarias, incluyendo el número de pruebas unitarias exitosas y el porcentaje de cobertura de estas pruebas [64].

En el siguiente apartado se podrá identificar los siete ejes de la calidad del código fuente, en cierta forma son los principales errores de los desarrolladores [64], [65], [63].

- Tener líneas de código duplicado («Duplicated code»).
- No respetar los estándares de codificación y las mejores prácticas establecidas («Coding standards»).
- Tener una cobertura baja (o nula) de pruebas unitarias, especialmente en partes complejas del programa («Unit tests»).
- Tener componentes complejos y/o una mala distribución de la complejidad entre los componentes («Complex code»).
- Dejar fallos potenciales sin analizar («Potential bugs»).
- Falta de comentarios en el código fuente, especialmente en las APIs públicas («Comments»).
- Tener el temido diseño spaghetti, con multitud de dependencias cíclicas («Design and architecture»).

SonarQube se basa en ciertos criterios técnicos a partir de los cuales se definen las métricas con las cuales se lleva mediciones.

En la **Tabla 15** se muestra los criterios técnicos y su respectiva descripción [63], [66].

Tabla 15. Criterios técnicos de Sonarqube.

Criterios técnicos	Descripción
Reglas de codificación	Dispone más 600 reglas incorporadas las cuales realizan desde verificaciones rápidas a cálculos complejos.
Errores potenciales	Identifica código que conducir a violaciones de seguridad para prevenir vulnerabilidades.
Documentación y comentarios	Muestra información de documentación, especialmente de las API públicas y del código fuente.
Código Duplicado	Identifica duplicaciones.
Cobertura de Tests	Identifica el código sin probar tanto a nivel de línea como en las ramas de control, considerando todos los posibles resultados de las operaciones condicionales.
Diseño y arquitectura	Se centra en minimizar dependencias.

4.2.14. Estudios de Sonarqube basado en la norma ISO/IEC 25000

En la **Tabla 16** se presenta una recopilación de autores que exploran y analizan la relación entre Sonarqube, una herramienta de análisis estático de código, y la norma ISO/IEC 25000 específicamente norma ISO/IEC 25010/ISO/IEC 25023, que establece estándares para la evaluación de la calidad del software en la característica de la mantenibilidad.

Tabla 16. Principios de Sonarqube basado en la norma ISO/IEC 25000.

Autor	Título de la Investigación	Resultado del estudio
Albea [67]	Normativa de Análisis del Código con SONAR | Pruebas Estáticas.	SonarQube proporciona un conjunto de métricas relacionadas con la calidad y mantenibilidad del código, de las cuales menciona: Tamaño de la aplicación, grado de documentación del código, grado de duplicidad de código, complejidad, deuda técnica y ratio de deuda técnica, número de defectos, vulnerabilidades y code smells.
Vera [68]	Estudio Funcional de la plataforma "SONARQUBE" para la evaluación de código fuente con respecto a la calidad de producto de software.	Las características y subcaracterísticas de la ISO 25000 se relacionan con la plataforma SonarQube.
Galán [69]	Estudio de la plataforma sonarqube para la implementación de ISO/IEC 25000.	La plataforma Sonarqube muestra una cobertura en su totalidad de todas las subcaracterísticas de la mantenibilidad basados en el modelo de calidad de la ISO/IEC 25010.

Después de examinar la herramienta Sonarqube basado en todos los aspectos mencionados anteriormente, se plantea como un eje secuencial para el presente proyecto de evaluación.

4.2.15. Buenas prácticas de programación

Las buenas prácticas de programación son un conjunto de técnicas y enfoques que se utilizan para desarrollar software de alta calidad y fácilmente mantenible [70], estas prácticas se basan en estándares y mejores prácticas reconocidas por la industria y se centran en asegurar que el código sea legible, modular y reutilizable. Al seguir buenas prácticas de programación, los desarrolladores pueden mejorar la calidad del software y reducir la cantidad de errores y problemas que pueden surgir durante el ciclo de vida del software[71].

Uno de los aspectos importantes de las buenas prácticas de programación en relación a la evaluación de mantenibilidad es la claridad del código, un código limpio y bien estructurado facilita la comprensión y el mantenimiento a largo plazo, según MacConnell [72] en su libro **Código Completo** implica utilizar nombres de variables descriptivas, comentar el código de manera adecuada y seguir convenciones de codificación establecidas. Al igual que, el uso de técnicas como el modularidad y la abstracción ayuda a dividir el código en partes más pequeñas y manejables, lo que facilita su mantenimiento y mejora la escalabilidad del software.

Otra consideración importante en las buenas prácticas de programación relacionadas con la calidad del software es la realización de pruebas exhaustivas, las pruebas unitarias, de integración y de rendimiento, son esenciales para garantizar que el software funcione correctamente y se mantenga estable a lo largo del tiempo.

4.2.16. Metodología XP

La Programación Extrema (XP) es una metodología ágil que se centra en la colaboración, la adaptabilidad y la entrega continua de software funcional, esto a su serie de principios, como la retroalimentación continua, la simplicidad, la comunicación efectiva y la aceptación del cambio [73]. Con la participación activa de todos los miembros del equipo de desarrollo, está fomenta la comunicación constante y la toma de decisiones [74], cuyo objetivo es obtener entregas tempranas y frecuentes de un producto de software funcional, lo que permite lograr una retroalimentación valiosa de los usuarios y adaptar el producto.

Según Maida y Pacienzia [75], uno de los aspectos importantes de XP es la práctica de la programación en parejas, en donde los desarrolladores trabajan colaborando estrechamente en la escritura de código, esta práctica fomenta la revisión y mejora continua del código, así como el intercambio de conocimientos y la resolución conjunta de problemas para ayudar a reducir los errores y mejorar la calidad del software.

4.2.16.1. Fases: Las fases de la metodología XP son las siguientes:

- **Planeación o Planificación:** Esta etapa se inicia con una reunión entre los clientes o usuarios y el equipo de desarrollo, donde se generan historias de usuario que detallan las características y funcionalidades del software. Por lo cual, el cliente asigna un valor o prioridad a cada historia según su importancia, luego, el equipo de desarrollo evalúa y asigna un costo a cada historia, generalmente en semanas de

desarrollo. Existe una condición, si una historia supera las 3 semanas, se divide en partes más pequeñas, cada una con su valor y costo, es importante considerar que, a lo largo del proyecto las historias pueden ser modificadas, añadidas, descompuestas o eliminadas según sea necesario.

- **Diseño:** En la siguiente fase se realizan los diseños tanto de arquitectura como de prototipos de interfaz, estos diseños permiten tener una visión de la estructura y la parte visual de los aplicativos.
- **Codificación:** Es la fase que implica la programación codificación de las diversas funcionalidades y necesidades acordadas con el cliente, una buena aplicación de las fases 1 y 2 se logra un desarrollo organizado y bien planificado.
- **Pruebas:** XP bajo el cambio constante, las pruebas se integran con el desarrollo, no al final, esto por la metodología que se desarrolla en ciclos temporales relativamente cortos. Todo el código debe pasar pruebas unitarias antes de ser publicado, lo que permite detectar y corregir errores tempranamente (el cliente puede desempeñar el rol de Tester, especialmente si posee habilidades en programación).

4.2.17. Requisitos de software

Los requisitos de software son fundamentales para el desarrollo de productos de software, puesto que son la base sobre la que se construye el producto final. Según [76], [77], son la descripción detallada de lo que se espera que el software haga, cómo debe funcionar y qué características debe tener, además de ser la guía para los desarrolladores y otros involucrados en el proceso de desarrollo garantizan que el software se ajuste a las necesidades y expectativas de los usuarios.

Los requisitos de sistema pueden ser clasificados en dos categorías fundamentales: requisitos funcionales y requisitos no funcionales, estas categorías son clave para entender las necesidades y expectativas del sistema conforme se menciona a continuación:

- Los requisitos funcionales se refieren a las características y funcionalidades específicas que el software debe tener para satisfacer las necesidades y objetivos del usuario, y definitivamente son fundamentales garantizar que el software cumpla con las expectativas del usuario y proporcione una experiencia de usuario satisfactoria [78], [79].

- Los requisitos no funcionales se centran en los atributos y características del software que no están directamente relacionados con su funcionalidad, sin embargo, son importantes para su éxito y sostenibilidad, cada requisito abarca aspectos clave como el rendimiento, la seguridad, la escalabilidad, la usabilidad y la compatibilidad con diferentes plataformas, entre otros. Sin lugar a dudas los requisitos son fundamentales para garantizar que el software sea confiable, eficiente y fácil de usar [78], [80].

4.2.18. Arquitectura cliente servidor

El desarrollo de software se basa en principios y normas que guían la construcción de sistemas, lo que se traduce en la creación de **arquitecturas de software** que definen la estructura y organización del código. Apoyado en la idea anterior, la arquitectura muestra una estructura fundamental de un producto de software, que define cómo se organiza y se relacionan sus componentes [81], según [82] se definen los componentes, interacciones e interfaces del software, así como los principios y patrones de diseño que definen en la construcción de software. De algún modo, proporciono una visión global del sistema.

"La arquitectura cliente servidor es una estructura muy utilizada en el diseño de sistemas de software distribuidos dividido en dos componentes principales, un cliente que se encarga de enviar las peticiones al servidor y mostrar la información al usuario; y un servidor que proporciona los servicios y datos solicitados por el cliente" [83], [84]. Comúnmente es utilizada en aplicaciones que requieren la gestión de grandes cantidades de datos y la comunicación entre diferentes sistemas.

En la **Figura 3** se muestra la comunicación entre el cliente y el servidor, el cual se lleva a cabo a través de una red, ya sea internet o una red de área local (LAN).

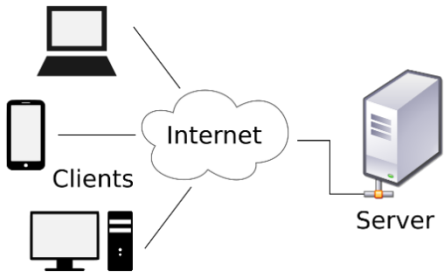


Figura 3. Arquitectura cliente servidor.

- **Backend:** Parte del servidor, es la parte encargada de administrar las operaciones, según [85], [86] es un elemento del producto de software que se encarga de procesar y gestionar los datos y servicios solicitados por el frontend, además se comunica con la base de datos y proporciona los datos y servicios necesarios para que el frontend funcione correctamente.
- **Frontend:** Tecnología del lado del cliente, son todas aquellas herramientas que están ejecutándose en el navegador del usuario final para gestionar la comunica con el backend para obtener los datos y servicios necesarios para mostrar la información y realizar las operaciones solicitadas por el usuario [85], [87].

La elección de la presente arquitectura se justifica por su capacidad para distribuir eficientemente las tareas entre el cliente y el servidor, lo que mejora la escalabilidad y la performance del sistema.

4.2.19. Tecnologías de desarrollo de Software

- **Node JS:** Es un entorno de ejecución de código abierto para JavaScript, diseñado específicamente para desarrollar aplicaciones del lado del servidor [88], a diferencia de otros entornos de ejecución, Node.js se basa en el motor V8 de Google Chrome, lo que le permite tener un rendimiento excepcionalmente rápido. Su arquitectura se fundamenta en un modelo de entrada/salida (E/S) no bloqueante y basado en eventos, lo que lo convierte en una opción ideal para aplicaciones en tiempo real y con alta escalabilidad.
- **Vite Js**: Es un entorno de desarrollo y un empaquetador de aplicaciones web contemporáneo que prioriza la velocidad y la eficiencia. Utiliza la técnica conocida como **hot module replacement** (HMR), que permite que los cambios en el código se reflejen de manera instantánea en el navegador, eliminando la necesidad de recargar toda la página[89], está característica mejora notablemente la experiencia de desarrollo.

  Técnicamente, Vite es especialmente útil para proyectos que utilizan frameworks modernos como Vue, React y Svelte.
- **MongoDb:** Es una base de datos NoSQL orientada a documentos que permite el almacenamiento y la gestión de datos de manera flexible y escalable, permite organizar la información en tablas y filas. MongoDB adopta un modelo de datos

basado en documentos en formato BSON (Binary JSON), cuya estructura permite que cada documento contenga datos complejos y anidados, facilitando así la representación de información diversa y en constante evolución [90].

4.2.20. Muestreo

Basado en los objetivos del presente proyecto, la recolección de información es un elemento esencial para llevar a cabo cualquier estudio en particular para respaldar las hipótesis y preguntas de investigación que requiera investigación o validación a través de pruebas. En este contexto, el muestreo es una de las técnicas ampliamente utilizadas para la recolección de información, según Acharya [91] el muestreo se define como el "proceso de seleccionar un subconjunto reducido de elementos de una población específica con el fin de realizar observaciones y análisis de un problema concreto".

En lugar de estudiar cada elemento de la población, el muestreo permite trabajar con una muestra representativa, esto garantiza la veracidad y confiabilidad de los resultados obtenidos, facilitando así el análisis e interpretación de los datos para su aplicación en una población general. Para la investigación del presente proyecto se utilizó.

- **Muestreo no probabilístico por conveniencia:** Facilita la posibilidad de obtener una muestra aleatoria de toda la población, según Hernández "la muestra se elige de acuerdo con la conveniencia de investigador, le permite elegir de manera arbitraria cuántos participantes puede haber en el estudio" [92].

4.2.21. Tipos de pruebas de software

- **Pruebas unitarias:** Son reconocidas como pruebas de caja blanca de modulo a modulo, durante el desarrollo de software es una componente esencial para la evaluación de componentes individuales reconocidos como funciones o métodos [79], [93]. En pocas palabras tiene como objetivo es evaluar determinadas áreas del sistema y verificar si se han satisfecho las especificaciones del usuario final [76]. En efecto este tipo de prueba permite identificar y corregir errores de forma temprana cuyos principios son el aseguramiento de calidad y mantenimiento seguro.
- **Pruebas de aceptación:** Es un tipo de prueba de caja negra, comúnmente es utilizada al finalizar el proceso de desarrollo del software, para verificar el cumplimiento de las necesidades representada en los requerimientos por el usuario

final previo a su entrega [76]. En resumidas, se encarga de la verificación del comportamiento general del software desde la perspectiva del usuario final, para el presente proyecto se hace uso de las **Pruebas de aceptación del usuario (UAT)** la cual es "realizada por el usuario final simulando un entorno real de uso, con el objetivo de evaluar la facilidad de uso y cumplimiento de funcionalidades" [94].

4.2.22. Metodología para revisiones bibliográficas

Las metodologías para revisiones bibliográficas son esenciales en el ámbito de la investigación, ya que permiten sistematizar y analizar la literatura existente sobre un tema específico. Se considera que las metodologías pueden variar en función de los objetivos de la investigación y la naturaleza del material a revisar. A continuación, se hace énfasis en la metodología de Barbara Kitchenham utilizada en el presente proyecto.

Barbara Kitchenham es un enfoque riguroso y estructurado para identificar, evaluar e integrar la evidencia de estudios primarios relevantes a una pregunta de investigación específica, la misma que se ha convertido en un estándar en el campo de la ingeniería de software [95].

Para la formulación de la pregunta de investigación se necesario utilizar el método PICOC, el cual permite formular preguntas de investigación de manera clara y concisa [96].

Las principales fases de la metodología de Kitchenham son [97]:

- Planificación de la revisión.
- Conducción de la revisión.
- Reporte de la revisión.

4.3. Trabajos relacionados

Al revisar la bibliografía pertinente, se destaca que el criterio a tener en cuenta al considerar trabajos relacionados para el presente proyecto de investigación es a través de artículos científicos y proyectos que hayan hecho uso de la norma ISO/IEC 25000 para la evaluación de aplicaciones web respecto a la mantenibilidad.

En la **Tabla 17** se presenta de manera concisa los aspectos más destacados de cada uno de los estudios y modelos de evaluación identificados en relación con el proyecto actual.

Tabla 17. Trabajos relacionados.

Titulo	UM	SD	Descripción
Evaluación de calidad en empresas de desarrollo de software aplicando la norma ISO/IEC 25000	SI	NO	Este proyecto [98] de investigación tiene como objetivo evaluar la mantenibilidad y portabilidad del framework React Native mediante un caso de estudio en la aplicación de gestión de eventos OAQ, utilizando la norma ISO/IEC 25010. La metodología empleada se basa en métricas de calidad interna y externa para obtener resultados cuantitativos de la evaluación. A través del análisis de los resultados, se identifican las buenas prácticas de desarrollo de software en React Native y se ofrecen recomendaciones para mejorar la portabilidad y mantenibilidad de los productos desarrollados con este framework, además proporciona una propuesta valiosa para la evaluación del framework.
Mantenibilidad según el modelo Square ISO/IEC 25000.	SI	NO	Esta investigación [99] aborda el problema de la industria del software respecto al desconocimiento de la calidad interna de los productos de software, un aspecto que a menudo es desestimado por muchos usuarios y clientes. El objetivo principal del estudio es medir el grado de mantenibilidad de los productos de software utilizando el modelo de calidad de la norma ISO/IEC 25000. Para lograr esto, seleccionó un modelo, identificando las propiedades de calidad y las herramientas necesarias para realizar el análisis del código fuente.
Análisis de mantenibilidad del Framework react native aplicando la norma iso/iec 25010.	NO	NO	Este proyecto [100] de investigación tiene como objetivo evaluar la mantenibilidad y portabilidad del framework React Native mediante un caso de estudio en la aplicación de gestión de eventos OAQ, aplicando la norma ISO/IEC 25010. Se ha utilizado una metodología basada en métricas de calidad interna y externo para obtener resultados cuantitativos de la evaluación. Según los resultados, se pudo identificar la propuesta de aplicar buenas prácticas de desarrollo de software en React Native y ofrecer recomendaciones para mejorar la portabilidad y mantenibilidad de los productos desarrollados en este framework. Además, el trabajo ofrece una propuesta valiosa para evaluar el framework React Native y mejorar la calidad de los productos de software desarrollados con él.

Titulo	UM	SD	Descripción
Investigación y selección de norma IEEE e ISO para aportar control de calidad en la fase de análisis de desarrollo de Software, en aplicaciones desarrolladas por el área de TI de CoopMego.	NO	NO	El estudio [101] se enfoca en la calidad de software desde la fase de análisis y diseño de aplicaciones web, en si el proyecto propone aplicar normas de control de calidad de software en la fase de análisis, utilizando una aplicación de prueba que facilitará a los grupos de trabajo minimizar los defectos y no heredar errores o retrasos en las siguientes fases de desarrollo de aplicaciones web en las diferentes. metodologías aplicadas. El proyecto buscó mejorar la calidad del software producido por el área de TI de CoopMego a través de la selección y aplicación de normas y estándares de calidad internacionales reconocidos y aceptados en la industria del software, al implementar estos estándares, mostraron mejorar significativamente la calidad del software y reducir los errores, fallas y defectos en las aplicaciones desarrolladas

Utilización de modelo (UM); Solución de defectos (SD).

5. Metodología

En esta sección se proporciona detalles sobre el procedimiento y recursos empleados en la realización del proyecto de Trabajo de Integración Curricular denominado **Evaluación de mantenibilidad para aplicaciones web en producción de los laboratorios de la carrera de computación de la UNL** se fundamentó en una investigación descriptiva explicativa, cuantitativa y mejora, con el propósito de resolver los objetivos previamente establecidos.

5.1. Área de estudio

El área de estudio académico del presente TIC se desarrolló en la CIC de la UNL. Según la **Figura 4** de Google Maps, se encuentra ubicada en la Ciudad de Loja en la dirección XR92+M4M, C. 9, Loja. La Carrera de Computación es una de las 47 de la UNL y **analiza los fundamentos científicos para desarrollar soluciones en el contexto de sistemas inteligentes, ingeniería de software, computación aplicada**.

Figura 4. Área de estudio, laboratorios de la CIC de la UNL.

5.2. Procedimiento

En esta sección se presenta el proceso llevado a cabo para alcanzar el objetivo general del proyecto TIC, donde se abordan de manera detallada los objetivos específicos y las actividades asociadas a cada uno de ellos.

5.2.1. Objetivo 1: Diagnosticar las aplicaciones web en producción de los laboratorios de la carrera de Computación de la UNL, mediante la evaluación del grado de mantenibilidad basado en la norma ISO/IEC 25000

Para garantizar la calidad y la evaluación de los productos de software para la identificación de defectos basado en la norma ISO/IEC 25000, se propone desarrollar las siguientes fases.

5.2.1.1. Fase 1: Investigar y seleccionar un modelo de evaluación.

- **Recopilar fundamentos científicos para selección de un modelo de evaluación**

Se recopiló dos artículos científicos, cuyo propósito es abordar y resolver las dificultades que surgen al evaluar la calidad del producto conforme a la norma ISO/IEC 25000, específicamente en relación a la característica de la mantenibilidad. Las investigaciones se utilizaron como base para la recopilación de criterios para seleccionar un modelo de evaluación adecuado (Ver **Tabla 23**).

- **Análisis de los artículos científicos**

Se realizó un análisis descriptivo de los dos artículos con el objetivo de identificar: requisitos para poder llevar a cabo la evaluación del producto software y certificación del producto software basado en la familia de la norma ISO/IEC 25000. Los requisitos propuestos en cada una de las investigaciones son elementos fundamentales que garantizan que la evaluación de un producto de software se realice conforme a los estándares de calidad establecidos por la norma ISO/IEC 25000 (Ver **Tabla 24**).

- **Investigar y seleccionar el modelo de evaluación**

Se aplicó un proceso investigativo para buscar, excluir y seleccionar un modelo de evaluación, en la **Tabla 18** se detalla cada una de las fases para cumplir con la selección del modelo de evaluación.

Tabla 18. Fases para seleccionar el modelo de evaluación.

Fase	Descripción	Técnica
Búsqueda	Se analizaron artículos científicos, papers y documentos relacionados con la evaluación producto software, basado en la ISO/IEC 25000, específicamente en la característica de la mantenibilidad.	Palabras clave y matriz bibliográfica

Fase	Descripción	Técnica
Criterios de selección	Se establecieron condiciones que debe cumplir los documentos encontrados. En este caso, abarcar los requisitos para poder llevar a cabo la evaluación de producto de software: • Modelo de calidad. • Proceso de evaluación. • Entorno tecnológico. • Función de mantenibilidad	Criterios de inclusión
Selección	Se analizó cada uno de los modelos recolectados, el propósito es aplicar el cumplimento a los criterios de inclusión para obtener un modelo de evaluación aplicable al TIC, de acuerdo a los criterios de selección definidos.	Matriz de criterios de inclusión

A partir de los resultados obtenidos en el proceso de investigación, se eligió el modelo de evaluación propuesto por Galán Pausic. Esta selección se realizó con el fin de cumplir con el primer objetivo del proyecto (Ver **Tabla 35**).

5.2.1.2. Fase 2: Aplicación del modelo de evaluación a los casos de estudio.

El modelo de evaluación seleccionado, propuesto por Galán Pausic, ha sido denominado **Modelo Galán** para el objetivo de la presente evaluación.

Para determinar el grado de mantenibilidad de los casos de estudio, se adaptó un **modelo de medición de mantenibilidad**, este modelo se basó entre el Modelo Galán y tareas específicas descritas por la norma ISO/IEC 25040.

En los siguientes ítems se presenta el modelo de medición de mantenibilidad propuesto, que detalla los requisitos necesarios para llevar a cabo el proceso de evaluación de calidad de los casos de estudio.

- **Modelo de calidad**

El modelo Galán se fundamenta en la norma ISO/IEC 25010, con un enfoque particular en la característica de mantenibilidad, la cual se descompone en cinco subcaracterísticas que permiten evaluar la calidad del producto de software de manera detallada.

La **Figura 5** se muestra la característica de la mantenibilidad de un producto de software, de las cuales se utilizó las cinco subcaracterísticas enfocado en la calidad interna.

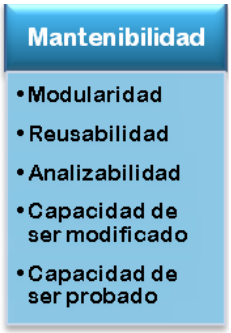


Figura 5. Subcaracterísticas de la Mantenibilidad.
Fuente: Norma ISO/IEC 25010 [50].

- **Métricas de calidad o instrumentos de medición**

El Modelo Galán proporcionó un total de 10 métricas enfocado en la calidad interna, cada métrica está relacionada con las subcaracterísticas de la mantenibilidad definida por la norma ISO/IEC 25010; estas métricas son: mala práctica, redundante, complejidad cognitiva, obsoleto, confuso, diseño, dificultad encontrada, densidad de comentarios, duplicaciones y cobertura.

- **Proceso de evaluación de calidad**

Se utilizó como guía la norma ISO/IEC 25040, la cual proporcionó un marco general de evaluación por fases, detallando actividades, tareas, propósitos, entradas, resultados e información complementaria para la evaluación de calidad.

La **Figura 6** muestra los procesos generales y las actividades correspondientes que se aplica durante el proceso de evaluación de calidad según la norma ISO/IEC 25040.

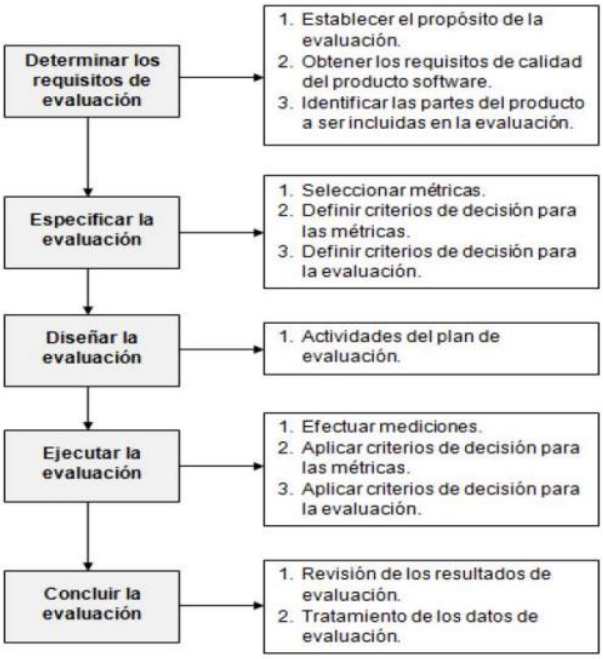


Figura 6. Proceso de evaluación de calidad.
Fuente: Norma ISO/IEC 25040 [55].

Con base los lineamientos de la norma ISO/IEC 25040, se planificaron actividades específicas para el proceso de evaluación en conjunto con el Modelo Galán. Como apoyo a este proceso, se empleó la herramienta de Sonarqube como requisito (entorno tecnológico **Ver Tabla 24**) para optimizar el análisis del código, permitiendo una evaluación más precisa y alineada con las recomendaciones de la norma.

En la **Tabla 19** se muestra las actividades seleccionadas para proceso a evaluación, cuyo propósito es obtener las métricas del código fuente y, posteriormente, evaluar la mantenibilidad del software.

Tabla 19. Actividades del proceso de evaluación.

Obtención de métricas del código	Proceso de evaluación
- Determinar los requisitos de evaluación. - Especificar la evaluación. - Diseñar la evaluación.	- Ejecutar la evaluación. - Concluir la evaluación.

- **Funciones de calidad**

El Modelo Galán incluye fórmulas para calcular el grado de mantenibilidad y de cada subcaracterística definida en la norma ISO/IEC 25010, utilizando operaciones matemáticas específicas. Estas fórmulas permiten cuantificar y evaluar objetivamente la mantenibilidad,

facilitando un análisis más preciso de la calidad interna del software en función de sus componentes clave.

La fórmula para calcular el grado de mantenibilidad (**GM**) utiliza los niveles de calidad de las subcaracterísticas definidas por la norma ISO/IEC 25010 (Ver **Figura 5**).

Formula 1. Medición del grado mantenibilidad.

$$\boldsymbol{GM} = \frac{A + M + R + CM + CP}{5}$$

Subcaracterísticas:
A= Analizabilidad
M= Modularidad
R= Reusabilidad
CM= Capacidad de ser Modificado
CP= Capacidad de ser Probado

Las fórmulas para calcular el grado de calidad de las subcaracterísticas (**GS**), utilizan los niveles de las métricas (Ver **Instrumentos**), obtenidos a partir de operaciones matemáticas (Ver **Funciones**) que emplean las mediciones de las propiedades de calidad del entorno tecnológico (SonarQube), como líneas duplicadas, líneas de cobertura, olores de código, complejidad cognitiva y líneas de comentadas.

A continuación, se presenta las funciones **GS** utilizadas para evaluar el grado de calidad de las subcaracterísticas de mantenibilidad:

Formula 2: Modularidad (M)

$$\boldsymbol{M} = \frac{DE + DP + CB}{3}$$

Métricas:
DE = Dificultad encontrada
DP = Duplicaciones
CB = Cobertura

Formula 3: Reusabilidad (R)

$$R = \frac{MP + O + D + DP}{4}$$

Métricas:
MP = Mala Práctica
O = Obsoleto
D = Diseño
DP = Duplicaciones

Formula 4: Analizabilidad (A)

$$A = \frac{MP + R + CC + C + D + DE + FI}{7}$$

Métricas:
MP= Mala Práctica
R = Redundante
CC = Cognitive Complexity
C = Confuso
D = Diseño
DE = Dificultad Encontrada
FI = Densidad de comentarios

Formula 5: Capacidad de ser modificado (CM)

$$CM = \frac{MP + R + D + DP}{4}$$

Métricas:
MP = Mala Práctica
R = Redundante
D = Diseño
DP = Duplicaciones

Formula 6: Capacidad de probado (CP)

$$CP = \frac{R + D + DE + CB}{4}$$

Métricas:
R = Redundante.
D = Diseño.
DE = Dificultad Encontrada.
CB = Cobertura.

A partir de los resultados obtenidos del proceso de medición de mantenibilidad (Ver **Aplicación modelo de evaluación**), se identificaron cuatro defectos en la aplicación del modelo en los dos casos de estudio, que se detallan a continuación: Código duplicado, falta de pruebas unitarias, malas prácticas de codificación, comentarios desactualizados o innecesarios.

5.2.2. Objetivo 2: Desarrollar un software web tomando como referencia XP como metodología de desarrollo, para solucionar el grado de mantenibilidad de una de las aplicaciones web diagnosticadas en el objetivo uno

Para abordar el objetivo 2, se tomaron en cuenta los defectos identificados en el objetivo 1 del caso de estudio de Gestión VR, que fue seleccionado como el primer caso de estudio (Ver **Anexo 3**) del presente proyecto. El enfoque del presente objetivo consiste en desarrollar una aplicación web e implementar un plan de estrategias dirigido a abordar los defectos considerados de mayor prioridad. Este plan no solo busca corregir los defectos, sino también optimizar los procesos que garantizan la mantenibilidad del software en el área de estudio, utilizando la aplicación web que permita verificar la calidad del código con los estándares de calidad requeridos.

Para cumplir con el objetivo 2, se estableció alcanzar un grado de mantenibilidad dentro del rango de valoración del 86% al 100% (Ver **Tabla 42**), este rango se considera altamente mantenible o no existe esfuerzo necesario para entender y modificar un producto de software, según lo definido en el modelo de Galán y respaldado por los autores Rodríguez y Piattini [102].

Para el desarrollo de la aplicación web, se propuso adoptar ciertas prácticas de la metodología XP, así como el desarrollo de una revisión bibliográfica de buenas prácticas de programación utilizando tareas de la metodología de Kitchenham, para recopilar pautas de mantenibilidad que serán utilizadas para la implementación de listas de verificación y utilización de la API de SonarQube para calcular el grado de calidad del código mediante el análisis de código estático. En este contexto, el software tiene como propósito verificar la calidad del código.

A continuación, se describen las fases con sus actividades para el desarrollo de la aplicación web como herramienta para abordar la solución de los defectos.

5.2.2.1. Fase de planeación.

- **Revisión bibliográfica para recopilar pautas de mantenibilidad**

La planificación y ejecución de la revisión bibliográfica utilizando la metodología Kitchenham (Ver **Revisión bibliográfica)** se llevó a cabo con el propósito de recopilar buenas

prácticas de programación como pautas de mantenibilidad que influyan positivamente en la calidad del software, contribuyendo así a la reducción de defectos.

- **Preparación de la entrevista y encuesta para recopilar los requisitos de sistema**

La planificación de las preguntas de la encuesta se llevó a cabo con el propósito de recopilar de información asociada con los requisitos de la aplicación web y la entrevista para recopilar especificaciones técnicas por parte de docentes y gestor académico de la carrera de computación (Ver **Planificación**), en relación a la aplicación de control de calidad en el desarrollo del código. El proceso de revisión y aprobación de las preguntas elaboradas la realizó el director del TIC, Ing. Francisco Alvarez Mg.

- **Ejecución de entrevista para recopilar los requisitos de sistema**

La realización de las entrevistas y encuestas proporcionó información relevante para orientar el proceso de desarrollo del software y especificaciones técnicas para aplicar control de calidad (Ver **Ejecución**). En este sentido, la entrevista se llevó a cabo con el Ing. Edison Coronel, Ing. Valeria Herrera, Ing. Óscar Cumbicus y Ing. Pablo Ordóñez, como docentes expertos en desarrollo de software.

- **Desarrollo de las historias de usuario**

El desarrollo de historias de usuario permitió identificar las necesidades y el flujo de funcionalidades, para llevar un control adecuado en el desarrollo del software. Para documentar los resultados se elaboró un documento de historias de usuario (Ver **Historias**), el documento fue revisado y aprobado por el Ing. Francisco Alvarez Pineda, Mg., en su rol como director del TIC.

En la **Figura 7** muestra el modelo utilizado para el desarrollo de las historias de usuario, además se describe cada uno de los atributos.

Historia de usuario			
ID:			
Usuario/Rol			
Prioridad en negocio		Riesgo en desarrollo	
Descripción			
Número	Escenario	Criterio de aceptación	

Figura 7. Modelo de historia de usuario.
Fuente: Google Académico.

A continuación, se muestra la descripción de cada atributo de la historia de usuario para facilitar la comprensión:

- **ID**: Asignación de un ID a cada Historia de Usuario.
- **Nombre de la historia de usuario:** Descripción breve de la HU.
- **Usuario/Rol:** Nombre de la persona responsable de la actividad mencionada en la HU.
- **Prioridad en negocio:** Se asigna según la relevancia de la actividad.
- **Iteración:** Asignación de la iteración en la que se llevará a cabo la ejecución.
- **Riesgo en desarrollo:** Se asigna según la relevancia de la actividad.
- **Referencia de requerimiento:** Se asigna la referencia del requisito abordado.
- **Descripción:** Detalle de las actividades en las Historias de Usuario**.**
- **Nro.:** Identificación del escenario**.**
- **Criterios de aceptación:** Aspectos importantes en cada Historias de Usuario**s.**
- **Escenario:** Especificación del escenario donde se llevan a cabo los criterios de aceptación.

- **Levantamiento de requerimientos según el estándar IEEE 830**

La identificación de necesidades mediante la encuesta y entrevista permitió la determinación de los requisitos funcionales y no funcionales de la herramienta, esto para asegurar que la aplicación no solo cumpla con las expectativas funcionales, sino que también sea eficiente, segura y fácil de mantener. El informe se elaboró con la documentación respectiva siguiendo las normativas del estándar IEEE 830 (Ver **Requerimientos**).

5.2.2.2. Fase de diseño.

Para el diseño de la arquitectura de la aplicación web, se utilizaron diagramas como el diagrama de caso de uso, diagrama de clases, diagrama de componentes y diagrama de despliegue, cuyo propósito es representar la estructura y el comportamiento de los componentes individuales, lo que a su vez permite definir la estructura global de la aplicación basada en el modelo cliente-servidor (Ver **Diagramas**).

5.2.2.3. Fase de codificación.

- **Planificación de iteraciones previo a la construcción**

Para guiar el proceso de codificación de las funcionalidades de la aplicación web, se implementaron tareas basada en historias de usuario (Ver **Tabla 84**), este enfoque tiene como objetivo establecer un proceso de desarrollo (diseño, ejecución y prueba) centrado en cumplir las funcionalidades solicitadas.

- **Codificación del frontend y backend de la aplicación web**

Definida la arquitectura de software, a continuación, se lleva a cabo el desarrollo backend y el frontend de la aplicación web en la gestión de funcionalidades y control de datos, esto de cada una de las tareas plasmadas en la actividad anterior, para lo cual se empleó framework Node y Express para el backend mientras que para el frontend se utilizó el Vite Js y Tailwind CSS.

5.2.2.4. Fase de pruebas.

Para la fase de pruebas de la aplicación de la aplicación web se planificaron las siguientes actividades:

- **Evaluación de la aplicación web mediante pruebas de aceptación – director del proyecto**

Se planificaron y ejecutaron las pruebas de aceptación del director, para verificar el cumplimiento de las funcionalidades del software, esto basado en los criterios de aceptación de las historias de usuario (Ver **Historias**). La prueba se llevó a cabo en un entorno controlado, donde se utilizó el software (Ver **Pruebas aceptación**) y datos de prueba.

- **Evaluación de la aplicación web mediante pruebas de aceptación – usuario final**

Se planificaron y ejecutaron las pruebas de aceptación del usuario final para evaluar el nivel de satisfacción en factores como la facilidad de uso y la utilidad percibida del software (Ver **Pruebas aceptación**), cuyo propósito es obtener una medida precisa y cuantificable de la percepción de los usuarios, para esto se utilizó una escala de Likert, la cual permitió analizar en detalle las opiniones de los usuarios sobre su experiencia con la aplicación.

5.2.2.5. Solución de defectos

Para abordar la solución de los defectos en el backend del estudio Gestión VR, área responsable de la lógica de la aplicación web, se desarrolló un plan de estrategias enfocado en los defectos de mayor prioridad, el plan emplea las herramientas como GitHub, Excel y la herramienta desarrollada QualityCCode (Ver **Plan de estrategias**). QualityCCode, se utilizó con objetivo de recopilar soluciones efectivas como pautas y calcular el grado respecto a la mantenibilidad del software, SonarQube para identificar áreas específicas en donde se presenta el defecto y Excel para registrar y documentar el grado de mantenibilidad del caso de estudio.

5.3. Recursos

Para alcanzar los objetivos establecidos en este proyecto de TIC, se utilizaron los siguientes recursos.

5.3.1 Recursos científicos

- **Método analítico:** Consiste en desglosar un objeto en componentes individuales para comprender de manera más efectiva su estructura, funcionamiento y relaciones internas. El en presente TIC se utilizó para alcanzar el objetivo general del proyecto, al desglosar actividades que permiten ejecutar los objetivos específicos.
- **Revisión bibliográfica:** Se utilizó para la búsqueda de información en diversas fuentes científicas tales como IEEE Xplore, Scopus, Scielo y Dialnet; incluyendo tesis de master y otras fuentes de información científica dentro del marco del área de estudio. Esto permitió la construcción del marco teórico (Ver **Bases teóricas**) y recolectar modelo de evaluación (Ver **Modelo de medición)** para la evaluación de los casos de estudio.
- **Investigación descriptiva explicativa:** Se centra en describir y comprender las características y propiedades de una situación específica. En el contexto del TIC,

permitió describir, interpretar y comprender modelos de evaluación basados en la norma ISO/IEC 25000.

- **Investigación de mejora**: El propósito es realizar mejoras en un determinado fenómeno o área de estudio. En el presente TIC, se buscó mejorar el proceso de verificar la calidad del software de aplicaciones web en los laboratorios de la CIC de la UNL a través de la implementación de una aplicación web. Esta iniciativa tiene como objetivo optimizar tecnológicamente el proceso tradicional, permitiendo reducir causas que afectan la calidad del software, específicamente grado de mantenibilidad.
- **Investigación cuantitativa**: Se basa en la recolección y análisis de datos numéricos para entender y explicar fenómenos. En este caso para el presente TIC, se busca seleccionar buenas prácticas de programación relacionadas con la calidad de software y solucionar los defectos encontrados resultado de la evaluación de la mantenibilidad.

5.3.2 Recursos de Hardware y Software

Para el desarrollo del presente TIC se utilizaron implementos de hardware y software, estos elementos se describen en la **Tabla 20** y **Tabla 21**.

Tabla 20. Hardware requerido.

Cantidad	Dispositivo	Características
1	Laptop	Marca: Asus Procesador: Intel(R) Core (TM) i5-8250U CPU @ 1.60GHz 1.80 GHz Memoria RAM: 8,00 GB Disco Duro: 250 TB Tarjeta Gráfica: Intel(R) UHD Graphics 620 SO: Windows 11

Tabla 21. Software requerido.

Nombre	Versión	Descripción
Word	2016	Herramienta que permite crear, editar y compartir documentos
Sonarqube	9.9	Análisis de código estático. Administrador de recursos de Azure. Detectar errores y vulnerabilidades básicas. Revisar los puntos de acceso de seguridad. Realice un seguimiento de los olores del código y solucione su deuda técnica. Métricas e historial de calidad del código.

5.3.3 Recursos técnicos:

- **Entrevista:** Se empleó para obtener información respecto a la aplicación de control de calidad en productos de software desarrollados por estudiantes en TIC para los laboratorios de la carrera de computación, con el objetivo de conocer los problemas respecto a la calidad del software. Se aplicó al Gestor de la carrera como responsable del centro de datos del laboratorio de computación y docentes, quienes son encargados de guiar el desarrollo de aplicaciones web para conocer defectos de relacionados con mantenibilidad de aplicaciones web en producción.
- **Encuesta:** Se aplicó a los estudiantes encargados de la implementación de aplicaciones web en el centro de datos de los laboratorios de Computación, con el propósito de recopilar información respecto a errores en la aplicación de control de calidad en los productos desarrollados.

5.3.4 Participantes

Tabla 22. Participantes en el desarrollo del presente TIC.

Nombre	Responsabilidad
Gilson Orlando Quezada	Autor del presente TIC
Ing. Francisco Javier Álvarez Pineda, Mg.Sc.	Director del presente TIC

5.3.5 Procesamiento y análisis de datos

- **Muestreo no probabilístico por conveniencia**: Para diagnosticar las aplicaciones web y ejecutar pruebas de aceptación, se utilizó este método, la elección se fundamentó en la disponibilidad de las aplicaciones potenciales y usuarios que pudieran contribuir directamente al objetivo de la evaluación y prueba.

5.3.6 Acrónimos y abreviaturas

Durante el desarrollo del presente trabajo se utilizaron los siguientes acrónimos y abreviaturas:

- UNL: Universidad Nacional de Loja.
- TIC: Trabajo de Integración Curricular.

6. Resultados

En la presente sección se expone los resultados obtenidos para cada objetivo específico, junto con la descripción de las actividades realizadas para su cumplimiento. En la sección **6.1**, alcanzado mediante dos fases orientadas a la selección y aplicación de un modelo de evaluación para la identificación de defectos. Por su parte, la sección **6.2** aborda el cumplimiento del segundo objetivo específico, enfocado en la resolución de los defectos identificados.

6.1. Objetivo 1: Diagnosticar las aplicaciones web en producción de los laboratorios de la carrera de Computación de la UNL, mediante la evaluación del grado de mantenibilidad basada en la Norma ISO/IEC 25000

El cumplimiento del presente objetivo se desarrolló mediante dos fases, en la uno investigar y seleccionar un modelo de evaluación y la dos aplicación del modelo de evaluación (modelo de medición de mantenibilidad) a los casos de estudio.

6.1.1. Fase 1: Investigar y seleccionar un modelo de evaluación

6.1.1.1. Fundamento para la selección de un modelo de evaluación.

Cada una de las investigaciones, ha demostrado su propósito en la construcción de modelos completos destinados a la evaluación y certificación de la calidad del producto de software.

En la **Tabla 23** se muestra atributos importantes de cada investigación, correspondiente a los artículos científicos considerados.

Tabla 23. Investigaciones científicas, para mejora y certificación de la mantenibilidad de un producto software.

Autores	Propuesta	Implicación	Resultado
Rodríguez, Pedreira y Fernández[103]	Modelo completo, que permiten realizar la evaluación del producto software y Certificación de calidad en la característica de la mantenibilidad.	Expone un caso de estudio de evaluación, mejora y certificación de la mantenibilidad de un producto software.	Presentación de un caso práctico de evaluación, mejora y certificación de un producto software basado en la familia ISO/IEC 25000, que ofrece una propuesta completa para la evaluación y certificación de la calidad del producto software y demuestra. su aplicación práctica.

Autores	Propuesta	Implicación	Resultado
Rodríguez y Piattini [102]	Ecosistema para la evaluación y certificación de la calidad del Producto Software en la característica de la mantenibilidad.	Presenta un enfoque para evaluar y certificar la calidad del producto software, y se muestra cómo se lleva a cabo el proceso de evaluación y certificación del producto.	Presentación de un enfoque para la evaluación y certificación de la calidad del producto software, con un ecosistema completo que ha permitido la realización de los primeros proyectos de evaluación y certificación de la calidad.

6.1.1.2. Análisis de artículos científicos.

De acuerdo con cada grupo de autores, los requisitos son elementos esenciales para realizar una evaluación de calidad basado en la familia de normas ISO/IEC 25000, cada requisito implica actividades fundamentales definir un modelo de calidad automatizado, apoyado por herramientas tecnológicas que permite medir y evaluar la mantenibilidad de un producto de software. Este enfoque facilita la obtención de indicadores precisos sobre la calidad interna, específicamente en lo que respecta al código.

En la **Tabla 24**, se muestra los requisitos y sus respectivas descripciones, que debe cumplir un modelo para evaluar la calidad de un producto de software.

Tabla 24. Requisitos para la evaluación de la calidad.

Requisito	Descripción	Norma
Modelo de Calidad	Define las características y subcaracterísticas de más bajo nivel y a su vez como indicadores y métricas que se puedan medir a partir de los elementos del producto software.	ISO/IEC 25010:2011
Proceso de Evaluación	Define las actividades que se deben realizar para poder llevar a cabo la evaluación. Selección de partes del producto software que se van a evaluar. Identificar características se van a evaluar. Establecer el proceso que el evaluador debe seguir, determina las herramientas a emplear, así como identificar los participantes y las actividades en las que se involucrarán.	ISO/IEC 25040:2011

Requisito	Descripción	Norma
Entorno tecnológico	El propósito del entorno tecnológico es apoyar tanto el proceso de evaluación como la recopilación de las métricas y umbrales establecidos en el modelo de calidad. Las herramientas que constituyen este entorno están diseñadas para simplificar la recopilación de datos. Estas herramientas deben ser capaces de automatizar los algoritmos de medición, permitiendo que las métricas base obtenidas se escalen para generar los indicadores de calidad (métricas).	Formado por varios niveles de herramientas que permiten automatizar en gran medida las tareas de medición y evaluación del producto.

Nota: Cada autor menciona, al cumplir estos requisitos, se puede realizar una evaluación exitosa y certificación de la calidad del producto software, lo que, a su vez, puede mejorar la satisfacción del cliente.

6.1.1.3. Selección de un modelo de evaluación considerando los requisitos de evaluación de calidad.

A continuación, se muestra las actividades ejecutadas para la selección de un modelo de evaluación:

a. Búsqueda.

Para la búsqueda de información se utilizaron palabras claves que se indican en la **Tabla 25**, en portales científicos: SCIELO, IEEE Xplore, DIALNET, ISO/IEC y dspace: UCLM, UDC, AENOR, en los cuales se obtuvieron documentos claves para seleccionar un modelo de evaluación.

Tabla 25. Palabras claves.

Palabras Claves
• Mantenibilidad de un producto de software. • Calidad producto software. • Modelo de calidad. • Garantizar la mantenibilidad. • Certificar la calidad del producto. • Errores en la evaluación de calidad. • Control de calidad en el desarrollo del código.

Además, se utilizó la matriz bibliográfica, en donde se registró cada artículo relevante al tema (Ver **Anexo 1**), en esta matriz se almacenó información clave respecto a cada

documento encontrado, la cual permitió tener una visión general de cada uno de los mismos como: Tipo documento; Referencia; Resumen; Tema; Propósito; Aspectos de interés del documento; Año.

En la **Tabla 26** se muestra un resumen de los modelos de evaluación recolectados en cada uno de los documentos propuestos por cada autor.

Tabla 26. Modelos de evaluación investigados.

Modelo	Autor	Descripción
1	Adones y Vega	Presenta algunas consideraciones importantes sobre el proceso de mantenimiento de software, que se deben tener en cuenta a la hora de especificar el atributo de mantenibilidad de tal forma que se pueda validar que efectivamente el producto es mantenible [8].
2	Rodríguez y Piattini	Presenta un conjunto de experiencias en la evaluación y certificación de productos software en la industria, junto con el ecosistema de entidades que han participado en el proceso. También se describe el proceso de evaluación [102].
3	Irrazábal	Desarrolla y propone un entorno basado en software libre, denominado KEMIS, para medir y evaluar la mantenibilidad del software en niveles operativo, táctico y estratégico. Además, busca ofrecer un soporte metodológico que permita calcular el valor y el retorno de inversión de las refactorizaciones necesarias, ayudando a las empresas a controlar la calidad de sus productos de software de manera automatizada y panificable, especialmente en contextos de desarrollo externalizado [104].
4	Galán Pausic	El propósito del trabajo es evaluar la calidad de un sistema en función de su capacidad para satisfacer las necesidades de los interesados, utilizando el modelo SQuaRE de la familia de normas ISO/IEC 25000. Se analizan las métricas de calidad y se emplea la herramienta SonarQube para medir propiedades del código fuente estáticamente. El trabajo propone fórmulas para calcular la mantenibilidad, fiabilidad, vulnerabilidad y rendimiento del software, basándose en la norma ISO/IEC 25010, y se ilustra la metodología con un ejemplo práctico [69].

b. Criterios de selección.

Para seleccionar el modelo de evaluación se tomó en consideración los cuatro modelos descritos en la **Tabla 26** y como criterios de inclusión los 3 requisitos (Ver **Tabla 24**) que debe cumplir un modelo para la evaluación de calidad de un producto de software (Modelo de calidad, proceso de evaluación y el entorno tecnológico).

Aunque el entorno tecnológico, implícitamente calcula el grado de mantenibilidad según sus fórmulas y escala de valoración, es necesario incorporar un nuevo criterio de inclusión denominado **Función de la mantenibilidad**. Para Irrazábal [104], este criterio permite calcular el grado la mantenibilidad de los productos de software, mediante los valores obtenidos de las subcaracterísticas de la mantenibilidad, la función en sí, es una forma de documentar el grado de mantenibilidad que muestra la herramienta tecnológica.

De esta manera, a continuación, se muestra los criterios de inclusión.

- Modelo de Calidad (ISO/IEC 25010).
- Proceso de Evaluación o Métricas de calidad.
- Entorno tecnológico.
- Función de la mantenibilidad.

c. Selección del modelo de evaluación.

A continuación, en los siguientes puntos se presenta el análisis de cada modelo de evaluación recolectado para determinar el cumplimiento de los criterios de inclusión, detallando aspectos como el modelo de calidad, el proceso de evaluación o métricas de calidad, el entorno tecnológico, y la función de mantenibilidad. Finalmente, en la **Tabla 35** se resume los resultados, indicando qué modelo cumple los criterios de inclusión, permitiendo así seleccionar el modelo de evaluación.

- **Modelo de calidad 1:** Propuesto por Adones y Vega

En la **Tabla 27** se muestra el análisis del modelo de evaluación.

Tabla 27. Análisis del modelo 1.

Modelo de Calidad	Proceso de Evaluación o Métricas de calidad	Entorno tecnológico	Función de mantenibilidad
ISO/IEC 25010	No abarca todas las subcaracterísticas de la mantenibilidad de la norma ISO/IEC 25010. Contempla de métricas de calidad para las subcaracterísticas de la mantenibilidad (Ver **Tabla 28**).	No muestra	No muestra

En la **Tabla 28** se muestra las métricas de calidad para cada subcaracterística de la mantenibilidad.

Tabla 28. Métricas de calidad.

	Métricas			
Subcaracterísticas	**Analizabilidad**	**Cambiabilidad**	**Estabilidad**	**CSP**
Densidad de Cyclomatic complexity	x	x		x
Cantidad de instrucciones	x			
Promedio tamaño de instrucciones	x	x		x
Frecuencia de comentarios	x			
Peso de los métodos por clase	x			
Numero de clases base	x			
Numero de saltos incondicionales		x	x	x
Numero de niveles anidados	x	x		x
Acoplamiento entre objetos	x	x	x	x
Ausencia de cohesión	x	x	x	x
Profundidad del árbol de herencia	x	x	x	x
Componentes llamados directamente			x	
Número de hijos	x	x	x	x
N. de salidas de estructuras condicionales				x
Respuestas de una clase				x
Numero de mensajes enviados en un método	x	x	x	x
Cobertura de pruebas unitarias			x	x
N. de errores en pruebas unitarias			x	x
Violaciones de código fuente	x			
Violaciones de estilo	x			
Numero de dependencias cíclicas	x	x		
Acoplamiento aferente y eferente		x	x	
Código duplicado	x	x		

Capacidad de ser probado (CSP).

- **Modelo de calidad 2:** Propuesto por Rodríguez y Piattini

En la **Tabla 29** se muestra el análisis del modelo de evaluación.

Tabla 29. Análisis del modelo 2.

Modelo de Calidad	Proceso de Evaluación o Métricas de calidad	Entorno tecnológico	Función de mantenibilidad
ISO/IEC 25010	Abarcar todas las subcaracterísticas de la mantenibilidad de la norma ISO/IEC 25010. Contempla de métricas de calidad para las subcaracterísticas de la mantenibilidad (Ver **Tabla 30**).	No muestra	No muestra

En la **Tabla 30** se muestra las métricas de calidad para cada subcaracterística de la mantenibilidad.

Tabla 30. Métricas de calidad.

	Métricas				
Subcaracterísticas	**Analizabilidad**	**Modularidad**	**CSM**	**Reusabilidad**	**CSP**
Incumplimiento de reglas	x		x	x	x
Documentación del código	x			x	
Complejidad	x		x		x
Balance inestabilidad abstracción / acoplamiento		x	x	x	x
Ciclos		x	x	x	x
Cohesión	x	x			
Estructuración de paquetes	x	x		x	x
Estructuración de clases	x	x			x
Tamaño de métodos	x	x			
Código duplicado	x		x		x

Capacidad de ser modificado (CSM); Capacidad de ser probado (CSP).

- **Modelo de calidad 3:** Propuesto por Irrazábal

En la **Tabla 31** se muestra el análisis del modelo de evaluación.

Tabla 31. Análisis del modelo 3.

Modelo de Calidad	**Proceso de Evaluación o Métricas de calidad**	**Entorno tecnológico**	**Función de mantenibilidad**
ISO/IEC 25010	No abarca en totalidad las características de la mantenibilidad, tres subcaracterísticas en total. Contempla de métricas de calidad para cada subcaracterística (Ver **Tabla 32**)	Java, CSS, PWD, Junit, Jdepend	Si muestra

En la **Tabla 32** se muestra las métricas de calidad para cada subcaracterística de la mantenibilidad.

Tabla 32. Métricas de calidad.

	Métricas		
Subcaracterísticas	**Analizabilidad**	**CSM**	**CSP**
Densidad de Cyclomatic complexity	x		
Densidad de código repetido	x		
Densidad de comentarios	x		
Densidad de los defectos de la capacidad para ser analizado	x		
Densidad de los defectos de la capacidad para ser cambiado		x	

	Métricas		
Subcaracterísticas	**Analizabilidad**	**CSM**	**CSP**
Densidad de ciclos		x	
Densidad de defectos de estabilidad		x	
Densidad de defectos de pruebas unitarias			x
Densidad de pruebas unitarias			x
Cobertura de pruebas unitarias			x
Tasa de fallos de pruebas unitarias			x
Tasa de errores de pruebas unitarias			x

Capacidad de ser modificado (CSM); Capacidad de ser probado (CSP).

- **Modelo de calidad 4:** Propuesto por Galán Pausic

En la **Tabla 33** se muestra el análisis del modelo de evaluación.

Tabla 33. Análisis del modelo 4.

Modelo de Calidad	Proceso de Evaluación	Entorno tecnológico	Función de mantenibilidad
ISO/IEC 25010	Abarcar todas las subcaracterísticas de la mantenibilidad de la norma ISO/IEC 25010. Contempla métricas de calidad para las subcaracterísticas de la mantenibilidad (Ver **Tabla 34**).	Sonarqube	Si muestra

En la **Tabla 34** se muestra las métricas de calidad para cada subcaracterística de la mantenibilidad.

Tabla 34. Métricas de calidad.

	Métricas				
Subcaracterísticas	**Analizabilidad**	**Modularidad**	**CSM**	**Reusabilidad**	**CSP**
Mala practica	x		x	x	x
Redundante	x			x	
Cognitive Complexity	x		x		x
Obsoleto		x	x	x	x
Confuso		x	x	x	x
Diseño	x	x			
Dificultad encontrada	x	x		x	x
Densidad de comentarios	x	x			x
Duplicaciones	x	x			
Cobertura	x		x		x

Capacidad de ser modificado (CSM); Capacidad de ser probado (CSP).

En la **Tabla 35** se muestra un resumen de la aplicación de los criterios de inclusión, los cuales permitieron analizar y determinar qué modelo de evaluación aplicar a los casos de estudio.

Tabla 35. Aplicación de criterios de inclusión a los modelos de evaluación.

Modelo de Calidad	Modelo de Calidad (ISO/IEC 25010)	Proceso de Evaluación o Métricas de calidad	Entorno tecnológico	Función de Mantenibilidad
modelo 1	✓	✓	x	x
modelo 2	✓	✓	x	x
modelo 3	x	x	✓	✓
modelo 4	✓	✓	✓	✓

El análisis de los modelos recolectados permitió identificar cuál de ellos cumple con los criterios necesarios para evaluar la calidad del producto conforme a la norma ISO/IEC 25000. En particular, el **modelo 4** propuesto por **Galán Pausic,** destacó por satisfacer todos los criterios, incluyendo aquellos relacionados con el entorno tecnológico y la función de mantenibilidad. Estos requisitos lo hacen especialmente adecuado para su aplicación en los casos de estudio seleccionados.

6.1.2. Fase 2: Aplicación del modelo de evaluación a los casos de estudio

Antes de aplicar el proceso de evaluación de los casos de estudio, se procedió a seleccionar los casos de estudio e identificar los dos lenguajes de programación que se utilizarían como enfoques de estudio.

Basado en el análisis y la selección de los lenguajes de programación predominantes en la carrera (Ver **Anexo 2**), se seleccionaron JavaScript y Python como los lenguajes de programación objeto de estudio para esta evaluación. Esta selección se realizó con el propósito de enfocar la evaluación y la propuesta de soluciones de mantenibilidad en lenguajes que son ampliamente utilizados y enseñados en malla curricular de la carrera.

Como resultado del análisis y selección de los casos de estudio mediante el muestreo no probabilístico por conveniencia aplicando criterios de inclusión y exclusión, se seleccionaron las aplicaciones web "Gestión VR" y "SDLC" (Ver **Anexo 3**), esto para asegurar que los casos de estudio fueran representativos y pertinentes para los objetivos del proyecto de evaluación de mantenibilidad. Además, se incluyeron los diagramas de caso de uso, secuencia y clases

correspondientes a estas aplicaciones web (ver **Anexo 5**), proporcionando una visión clara de su estructura y funcionalidad.

En la **Tabla 36** se muestra el proceso de evaluación de los casos de estudio, se llevó a cabo siguiendo un enfoque estructurado, que consiste en obtener las métricas del código fuente para posteriormente evaluar la calidad del producto software respecto a la mantenibilidad.

Tabla 36. Modelo de medición de mantenibilidad.

Proceso de evaluación			
Subfase 1		**Subfase 2**	
Actividad		**Actividad**	
Obtención de métricas del código	- Determinar los requisitos de evaluación. - Especificar la evaluación. - Diseñar la evaluación.	**Proceso de evaluación**	- Ejecutar la evaluación. - Concluir la evaluación.

El enfoque estructurado incluye tareas vinculadas a las actividades del **modelo de medición de mantenibilidad** adaptado (Ver **Aplicación del modelo de evaluación**), cuyo propósito es determinar el grado de mantenibilidad de aplicaciones web e identificación de defectos.

El proceso de evaluación de los casos de estudio para obtener el grado de mantenibilidad se lleva a cabo mediante el análisis de código estático, el cual se ejecuta a continuación.

6.1.2.1. Obtención de métricas del código.

a. Determinar los requisitos de evaluación.

El objetivo de la presente actividad es establecer un marco claro de evaluación de la calidad para asegurar que la evaluación sea integral y alineada al estándar de calidad establecido.

- **Establecer el propósito de la evaluación:** El propósito de la evaluación es verificar la calidad del código de las aplicaciones web **SDLC** y **Gestion VR** cumpla con la norma de calidad ISO/IEC 25000, mediante un modelo de calidad adaptado, el cual debe de cumplir la característica de la mantenibilidad.
- **Obtener los requisitos de calidad:** Según el objetivo del proyecto, el requisito de calidad es la mantenibilidad basada en la familia de la norma ISO/IEC 25000.

- **Identificar las partes del producto a ser incluidas en la evaluación:** Las partes de los productos a evaluar es el código mediante el análisis de código estático de las aplicaciones web **SDLC** y **Gestion VR**.

b. Especificar la evaluación.

El propósito de la siguiente actividad es especificar detalladamente cómo se llevó a cabo la evaluación de la calidad del código.

Para la evaluación del código para aportar control de calidad, se emplearon métricas conforme al modelo de medición de mantenibilidad, en donde cada una de las subcaracterísticas basada en la norma ISO/IEC 25010 se relacionan con una o varias métricas propuestas por el Modelo Galán.

En la **Tabla 37** se muestra las subcaracterísticas del modelo de medición de mantenibilidad según norma la ISO/IEC 25010.

Tabla 37. Subcaracterísticas del modelo de medición de mantenibilidad.

Subcaracterísticas
Modularidad
Reusabilidad
Analizabilidad
Capacidad de ser modificado
Capacidad de ser probado

- **Seleccionar las métricas**

El modelo Galán atribuyó con las métricas de calidad interna relacionada a cada una de las subcaracterística del requerimiento de calidad.

En la **Tabla 38** se muestra las métricas de calidad interna relacionadas a cada una de las subcaracterísticas.

Tabla 38. Métricas de calidad interna.

Analizabilidad	Modularidad	CSM	Reusabilidad	CSP
Mala practica	Diseño	Mala practica	Mala practica	Redundante
Redundante	Duplicaciones	Redundante	Obsoleto	Diseño
Complejidad cognitiva	Cobertura	Diseño	Diseño	Dificultad encontrada
Confuso		Duplicaciones	Duplicaciones	Cobertura
Diseño				

Analizabilidad	Modularidad	CSM	Reusabilidad	CSP
Dificultad encontrada				
Densidad comentarios				

Capacidad de ser modificado (CSM); Capacidad de ser probado (CSP).

- **Definir los criterios de decisión para subcaracterísticas**

En esta sección, se especificó la escala de valoración utilizada para representar la calidad de las subcaracterísticas, así como también, el nivel de aceptación de las propiedades de calidad del producto de software.

Nota: Para la herramienta de Sonarqube una propiedad es un atributo de un caso de estudio, el cual le se asigna un valor numérico como resultado del análisis de código estático al evaluar su calidad.

En la **Tabla 39** se definió la escala de valoración utilizada para representar el grado de calidad de las subcaracterísticas y realizar el análisis de calidad correspondiente.

Tabla 39. Escala de valoración de las subcaracterísticas.

Valor (%)	Representación
1 – 25	Deficiente
26 – 60	Insatisfactorio
61 – 90	Aceptable
91 – 100	Excelente

En la identificación y análisis de defectos, se utilizaron las propiedades de calidad de Sonarqube. En la **Tabla 40** se describe cada una de las propiedades de calidad utilizadas de Sonarqube.

Tabla 40. Descripción de propiedades de calidad de Sonarqube.

Propiedades de Mantenibilidad	Descripción
Líneas duplicadas (Duplicated Lines)	Esta propiedad mide la cantidad de código duplicado en el proyecto y es un indicador de mala práctica de programación.
Líneas de cobertura (Coverage lines)	Esta propiedad indica el porcentaje de líneas de código que están cubiertas por pruebas unitarias, es decir, determina cuánto del código ha sido probado.
Complejidad ciclomática (Cyclomatic complexity)	Evalúa la complejidad del flujo de control del código (cuantitativa), para identificar código complejo.

Propiedades de Mantenibilidad	Descripción
Olores de código (Code smells)	Esta propiedad indica de manera general indicadores de posibles problemas en el código, los cuales se podrían mejorar para aumentar la mantenibilidad.
Complejidad cognitiva (Cognitive Complexity)	Esta propiedad mide cuán difícil es entender el flujo de control de un programa (cualitativa), basada en el esfuerzo mental requerido para comprender el código.
Líneas de comentadas (Comment lines)	Esta propiedad mide las líneas de código que contienen comentarios,

En la **Tabla 41**, se definieron porcentajes aceptables para cada propiedad utilizada en la identificación de defectos, los cuales tienen influencia directa en la medición de las métricas.

Tabla 41. Escala de aceptación de las propiedades.

Propiedades de Mantenibilidad	Representación	Interpretación
Duplicated Lines	> 5%	Defecto
Coverage lines	< 70%	Defecto
Cyclomatic complexity	> 15%	Defecto
Code smells	> 5	Defecto
Cognitive Complexity	> 15%	Defecto
Comment lines	>= 20%	Defecto

- **Definir criterios de decisión para la evaluación**

A continuación, se detalla la escala de valoración empleada para representar el grado de mantenibilidad del producto de software, así como para la identificación de defectos.

En la **Tabla 42** se definió la escala de valoración adaptada para representar el grado de mantenibilidad del código de los productos de software, esta escala también es utilizada para el respectivo análisis de calidad de los casos de estudio, en relación con el requerimiento de calidad.

Tabla 42. Escala de valoración de la mantenibilidad.

Valor de mantenibilidad	Descripción
0 - 25	Deficiencia en la mantenibilidad
26 - 64	Dificultad en la mantenibilidad
65 - 85	Moderadamente mantenible
86 - 100	Altamente mantenible

Nota: Para representar el grado de mantenibilidad del producto de software, el modelo Galán incluye una escala de valoración. Sin embargo, se consideró el estudio de Rodríguez y Piattini [102]

para desarrollar una escala de valoración más detallada de la mantenibilidad, quienes establecen cuatro líneas de valoración, lo que permite una evaluación más precisa y diferenciada. La adaptación se realizó con base en las especificaciones técnicas recabadas durante la entrevista con la Ing. Valeria Herrera, lo que garantizó la pertinencia y aplicabilidad de la escala en el contexto de la investigación.

La evaluación del grado de mantenibilidad e identificación de defectos se definió de la siguiente manera:

- **Evaluación del grado de mantenibilidad**

La herramienta Sonarqube mediante el análisis de código estático recupera mediciones en datos numéricos para las propiedades de calidad (descritas **Tabla 40),** cada una de las propiedades tiene un criterio de evaluación asignado que es operado matemáticamente para establecer un valor para cada una de las métricas (descritas **Tabla 38),** mediante las fórmulas matemáticas (Ver **Anexo 4**). Así mismo, las métricas permiten calcular y definir el grado de calidad de cada una de las subcaracterísticas (descritas **Tabla 37**), para lo cual se utiliza las funciones de calidad (Ver **Función GS**) y escala de valoración (descrito **Tabla 39**). Finalmente, para determinar y definir el grado de mantenibilidad del producto de software se utilizó el grado de calidad de las subcaracterísticas operando mediante funciones matemáticas (Ver **Función GM**) y representar mediante la escala de valoración (descritas **Tabla 42**).

- **Identificación de defectos**

La identificación de defectos consiste en comparar las propiedades de calidad (Ver **Tabla 40)** con los porcentajes de aceptación establecidos en la **Tabla 41**. Este proceso permite determinar si alguna propiedad no cumple con los requisitos de mantenibilidad, lo que indicaría que dicha propiedad debe considerarse un defecto.

Nota: Los valores de las métricas matemáticamente contribuyen en la ponderación de la mantenibilidad.

c. Diseñar la evaluación.

El propósito de la presente actividad es planificar y estructurar el proceso de evaluación del código fuente, para lo cual se definió las siguientes actividades:

- Identificar los archivos de código fuente a evaluar.
- Aplicación de métricas y criterios de evaluación de calidad.

- Analizar los resultados en base a las métricas.

6.1.2.2. Proceso de evaluación.

d. Ejecutar la evaluación.

El objetivo de esta actividad es aplicar las métricas seleccionadas, evaluar el cumplimiento de los criterios establecidos, y analizar los resultados obtenidos, con el fin de determinar la calidad del producto de software en relación con los estándares definidos.

Para la evaluación, conforme el modelo de medición, se utilizó SonarQube que tiene como propósito resolver el análisis de código estático para obtener las mediciones de las propiedades de calidad, posteriormente, se aplicaron los criterios de decisión para determinar los porcentajes y niveles de cumplimiento. Finalmente, se analizaron los resultados utilizando la escala de valoración de las subcaracterísticas, proporcionando un diagnóstico claro del estado de calidad del código.

- **Repositorio del código fuente de los casos de estudio**

Los casos de estudio se recopilaron de la página oficial de la Carrera de Computación, de los repositorios de código abierto en Git Lab y Git Hub.

- **SDLC**: https://gitlab.com/franzflores21/pm-frontend, https://gitlab.com/franzflores21/pm-backend.
- **Gestion VR**: https://github.com/gilsonOrlando/vr-backend_SinModificaciones.

- **Efectuar mediciones**

  - **Caso de estudio: SDLC**

En la **Tabla 43** se muestra las mediciones recopiladas de cada propiedad de calidad, resultado del análisis de código estático de la herramienta de Sonarqube.

Se muestra un total de 5300 líneas de código backend y 17000 líneas de código frontend.

Tabla 43. Resultado de las mediciones del caso de estudio SDLC.

Propiedad	Backend	Frontend
Duplicated lines	707	689
Coverage lines	1388	2668
Cyclomatic complexity	525	1613
Code smells	60	114
Cognitive complexity	4	1
Comment lines	1,70%	1,90%

- **Caso de estudio: Gestion VR**

En la **Tabla 44** se muestra las mediciones recopiladas de cada propiedad, resultado del análisis de código estático de la herramienta de Sonarqube.

Se muestra un total de 2800 líneas de código backend y 26000 líneas de código frontend.

Tabla 44. Resultado de las mediciones del caso de estudio Gestion VR.

Propiedad	Backend	Frontend
Duplicated lines	1228	2823
Coverage lines	865	1534
Cyclomatic complexity	277	1021
Code smells	6	846
Cognitive complexity	0	30
Comment lines	20%	0,40%

- **Aplicar criterios de decisión para las métricas**

A continuación, se aplicaron los criterios de decision en cada una de las métricas de las subcaracterísticas de calidad de los casos de estudio **SDLC** y **Gestion VR** para obtener el grado de calidad respectivamente.

- **Caso de estudio: SDLC**

Se inició con la subcaracterística de modularidad, la cual consta de 3 métricas (Ver **Tabla 38**) y se aplicaron para el backend y frontend del caso de estudio. Para obtener el valor de cada métrica se aplicó fórmulas matemáticas (Ver **Anexo 4**) utilizando los datos de las propiedades de calidad resultado del análisis de código estático de Sonarqube:

Backend

Métrica 1:	Diseño
Propiedad de calidad:	Cyclomatic complexity
Formula	$D = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$
Valor propiedad de calidad	525
Aplicación de la formula	$D = \frac{Complejidad\ ciclomatica}{lineas\ de\ codigo}$ $D = \frac{525}{5300}$ $D = 9.91\%$
Métrica 2:	Duplicaciones
Propiedad de calidad:	Duplicated lines
Formula	$D = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$
Valor propiedad de calidad	707
Aplicación de la formula	$LD = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$ $LD = \frac{707}{5300}$ $LD = 13.34\%$
Métrica 3:	Cobertura
Propiedad de calidad:	Coverage lines
Formula	$CB = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$
Valor propiedad de calidad	1388
Aplicación de la formula	$CB = \frac{1388}{5300}$ $CB = 73.81\%$

Frontend

Métrica 1:	Diseño
Propiedad de calidad:	Cyclomatic complexity
Formula	$D = \frac{Complejidad\ ciclomatica}{lineas\ de\ codigo}$
Valor propiedad de calidad	1613

Aplicación de la formula	$D = \frac{1613}{17000}$
	$D = 9.49\%$

Métrica 2:	Duplicaciones
Propiedad de calidad:	Duplicated Lines
Formula	$LDP = \left(\frac{\text{Duplicated Lines}}{Total\ de\ lineas}\right) x100$
Valor propiedad de calidad	689
Aplicación de la formula	$LD = \frac{689}{17000}$
	$LD = 4.05\%$

Métrica 3:	Cobertura
Propiedad de calidad:	Coverage lines
Formula	$CB = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$
Valor propiedad de calidad	2668
Aplicación de la formula	$CB = \frac{2668}{17000}$
	$CB = 84.31\%$

Para obtener la medición de cada una de las métricas de la subcaracterística de modularidad del producto de software, se aplicó promedio de las métricas del backend y del frontend, como se muestra a continuación.

Métricas	**Backend**	**Frontend**	**Promedio**
Diseño	991%	9.49%	9.70%
Duplicaciones	13.34%	4.05%	8.70%
Cobertura	26.19%	15.69%	20.94%

Para obtener el grado de calidad de la subcaracterística de la modularidad se aplicó la formula **GS** respectivamente**,** conforme se muestra a continuación.

$$M = \frac{D\ +\ DP\ +\ CB}{3}$$

$$M = \frac{9.70\%\ +\ 8.70\%\ +\ 20.94\%}{3}$$

$$M = 86.89\%$$

En el **Anexo 11** se aplicaron los criterios de decisión para las siguientes subcaracterísticas, utilizando sus respectivas métricas y en la **Tabla 45** se muestra un resumen del grado de calidad respectivamente.

Tabla 45. Grado de calidad de las subcaracterísticas del caso de estudio SDLC.

Subcaracterística	**Mediciones**			**GC**
	Métricas	**Backend**	**Frontend**	**Promedio**
Modularidad	Diseño	9.91%	9.49%	9.70%	86.89%
	Duplicaciones	13.34%	4.05%	8.70%	
	Cobertura	26.19%	15.69%	20.94%	
Reusabilidad	Mala práctica	1.13%	0.67%	0.901%	94.95%
	Obsoleto	1.13%	0.67%	0.901%	
	Diseño	9.91%	9.49%	9.697%	
	Duplicaciones	13.34%	4.05%	8.696%	
Analizabilidad	Mala práctica	1.13%	0.67%	0.901%	83.07%
	Redundante	13.34%	4.05%	8.696%	
	Cognitive Complexity	0.08%	0.01%	0.041%	
	Confuso	1.13%	0.67%	0.90%	
	Diseño	9.91%	9.49%	9.697%	
	Dificultad encontrada	0%	0.01%	0.041%	
	Densidad comentarios	98.30%	98.10%	98.20%	
Capacidad modificada	Mala práctica	1.13%	0.67%	0.901%	93.00%
	Redundante	1.13%	4.05%	8.696%	
	Diseño	9.91%	9.49%	9.697%	
	Duplicaciones	13.34%	4.05%	8.696%	
Capacidad de ser probado	Redundante	13.34%	4.05%	8.696%	90.16%
	Diseño	9.91%	9.49%	9.697%	
	Dificultad encontrada	0%	0.01%	0.041%	
	Cobertura	26.19%	15.69%	13.112%	

Grado de calidad (**GC**).

Nota: Para conocer el grado de calidad de la subcaracterística, se aplicó promedio de los valores del backend y el frontend para obtener las métricas generales del producto de software y posterior aplicar la función de calidad **GS**.

Considerando la evaluación y mediciones aplicadas a cada una de las subcaracterísticas descritos en la **Tabla 45**, se procedió a aplicar los criterios de decisión en función de las métricas establecidas, según la **Tabla 39**.

Grado de calidad	**Porcentaje de cumplimiento**	**Representación**
Modularidad	86.89%	Aceptable
Reusabilidad	94.95%	Excelente
Analizabilidad	83.07%	Aceptable

Capacidad de ser modificado	93.00%	Excelente
Capacidad de ser probado	90.16%	Aceptable

- **Caso de estudio: Gestion VR**

Se inició con la subcaracterística de la **modularidad**, la cual consta de 3 métricas (Ver **Tabla 38**) y se aplican para el backend y frontend del caso de estudio. Para obtener el valor de cada métrica se aplicó fórmulas matemáticas (Ver **Anexo 4**) utilizando los datos de las propiedades de calidad resultado del análisis de código estático de Sonarqube:

Backend	
Métrica 1:	Diseño
Propiedad de calidad:	Cyclomatic complexity
Formula	$D = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$
Valor propiedad de calidad	277
Aplicación de la formula	$D = \frac{Complejidad\ ciclomatica}{lineas\ de\ codigo}$ $D = \frac{277}{2161}$ $D = 12.82\%$
Métrica 2:	Duplicaciones
Propiedad de calidad:	Duplicated lines
Formula	$D = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$
Valor propiedad de calidad	1228
Aplicación de la formula	$LD = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$ $LD = \frac{1228}{2161}$ $LD = 56.83\%$
Métrica 3:	Cobertura
Propiedad de calidad:	Coverage lines
Formula	$CB = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$
Valor propiedad de calidad	865

Aplicación de la formula	$CB = \frac{865}{2161}$ $CB = 40.03\%$

Frontend

Métrica 1:	Diseño
Propiedad de calidad:	Cyclomatic complexity
Formula	$D = \frac{Complejidad\ ciclomatica}{lineas\ de\ codigo}$
Valor propiedad de calidad	1021
Aplicación de la formula	$D = \frac{1021}{26000}$ $D = 3.93\%$

Métrica 2:	Duplicaciones
Propiedad de calidad:	Duplicated lines
Formula	$LDP = \left(\frac{\text{Duplicated Lines}}{Total\ de\ lineas}\right) x100$
Valor propiedad de calidad	1228
Aplicación de la formula	$LD = \frac{1228}{26000}$ $LD = 10.86\%$

Métrica 3:	Cobertura
Propiedad de calidad:	Coverage lines
Formula	$CB = \frac{\text{Duplicated Lines}}{Total\ de\ lineas}$
Valor propiedad de calidad	846
Aplicación de la formula	$CB = \frac{1534}{26000}$ $CB = 5.90\%$

Para obtener la medición de cada una de las métricas de la subcaracterística de modularidad del producto de software, se aplicó promedio de las métricas del backend y del frontend.

Métricas	Backend	Frontend	Promedio
Diseño	12.82%	3.93%	8.37%
Duplicaciones	56.83%	10.86%	33.84%
Cobertura	40.03%	5.90%	22.96%

Para obtener el grado de calidad de la subcaracterística de modularidad se aplicó la formula **GS**, conforme se muestra a continuación.

$$G = \frac{D + DP + CB}{3}$$

$$G = \frac{8{,}37\% + 33{,}84\% + 22{,}96\%}{3}$$

$$G = 78.27\%$$

En el **Anexo 12** se aplicaron los criterios de decisión para las siguientes subcaracterísticas, utilizando sus respectivas métricas y en la **Tabla 46** se muestra un resumen del grado de calidad respectivamente.

Tabla 46. Grado de calidad de las subcaracterísticas del caso de estudio Gestion VR.

Subcaracterística	Mediciones			GC
	Métricas	**Backend**	**Frontend**	**Promedio**	
Modularidad	Diseño	12.82%	3.93%	8.37%	78.27%
	Duplicaciones	56.83%	10.86%	33.84%	
	Cobertura	40.03%	5.90%	22.96%	
Reusabilidad	Mala práctica	0.28%	3.25%	1.77%	88.56%
	Obsoleto	0.28%	3.25%	1.77%	
	Diseño	12.82%	3.93%	8.37%	
	Duplicaciones	56.83%	10.86%	33.84%	
Analizabilidad	Mala práctica	0.28%	3.25%	1.77%	80.62%
	Redundante	56.83%	10.86%	33.84%	
	Cognitive Complexity	0.00%	0.12%	0.06%	
	Confuso	0.28%	3.25%	1.77%	
	Diseño	12.82%	3.93%	8.37%	
	Dificultad encontrada	0%	0.12%	0.06%	
	Densidad de comentarios	80%	99.60%	89.80%	
Capacidad de ser modificada	Mala práctica	0.28%	3.25%	1.77%	80.54%
	Redundante	56.83%	10.86%	33.84%	
	Diseño	12.82%	3.93%	8.37%	
	Duplicaciones	56.83%	10.86%	33.84%	
Capacidad de ser probado	Redundante	56.83%	10.86%	33.84%	. 83.69%
	Diseño	12.82%	3.93%	8.37%	
	Dificultad encontrada	0%	0.12%	0.06%	
	Cobertura	40.03%	5.90%	22.96%	

Grado de calidad (**GC**).

Considerando la evaluación y mediciones aplicadas a cada una de las subcaracterísticas descritos en la **Tabla 46,** se procedió a aplicar los criterios de decisión en función de las métricas establecidas, según la **Tabla 39**.

Grado de calidad	Porcentaje de cumplimiento	Representación
Modularidad	78.27%	Aceptable
Reusabilidad	88.56%	Aceptable
Analizabilidad	80.62%	Aceptable
Capacidad de ser modificado	80.54%	Aceptable
Capacidad de ser probado	83.69%	Aceptable

- **Aplicar criterios de decisión para la evaluación**

A continuación, se calculó el grado de calidad de los casos de estudio SDLC y Gestion VR.

Nota: Para conocer el grado de calidad de los casos de estudio, se aplicó la función de calidad **GM**, que consiste en sumar los porcentajes del grado de calidad de las subcaracterísticas y dividir por la cantidad de subcaracterísticas.

- **Caso de estudio: SDLC**

En la **Tabla 47** se muestra un resumen del grado de calidad de cada una de las subcaracterísticas del producto de software, según las métricas aplicadas durante la evaluación del código de la aplicación web SDLC.

Tabla 47. Grado de calidad de las subcaracterísticas de la mantenibilidad.

Subcaracterística	Grado de calidad
Modularidad	86.89%
Reusabilidad	94.95%
Analizabilidad	83.075%
Capacidad para ser modificado	93.002%
Capacidad para ser probado	90.16%

Grado de calidad del producto de software.

$$GM = \frac{M + R + A + CM + CP}{5}$$

$$GM = \frac{86.89\% + 94.95\% + 83.075\% + 93.002\% + 90.16\%}{5}$$

$$GM = 89.61\%$$

Según los resultados obtenidos, el grado de mantenibilidad del caso de estudio SDLC se ha establecido en un GM=89.61%, utilizando la escala de valoración correspondiente (Ver **Tabla 42**), el porcentaje se clasifica como "Altamente mantenible".

- **Caso de estudio: Gestion VR**

En la **Tabla 48** se muestra un resumen del grado de calidad de cada una de las subcaracterísticas del producto de software, según las métricas aplicadas durante la evaluación del código de la aplicación web SDLC.

Tabla 48. Grado de calidad de las subcaracterísticas de la mantenibilidad.

Subcaracterística	Grado de calidad
Modularidad	78.27%
Reusabilidad	88.56%
Analizabilidad	80.62%
Capacidad para ser modificado	80.54%
Capacidad para ser probado	83.69%

Grado de calidad del producto de software.

$$GM = \frac{M + R + A + CM + CP}{5}$$

$$GM = \frac{78.27\% + 88.56\% + 80.62\% + 80.54\% + 83.69\%}{5}$$

$$GM = 82.34\%$$

Según los resultados obtenidos, el grado de mantenibilidad del caso de estudio SDLC se ha establecido en un GM=82.34%, utilizando la escala de valoración correspondiente (ver **Tabla 42**), el porcentaje se clasifica como **Moderadamente mantenible**.

e. Concluir la evaluación.

El propósito de la siguiente actividad es analizar y sintetizar los datos recolectados durante la evaluación del código fuente para su respectivo tratamiento.

Terminado el proceso de evaluación, se obtuvieron los siguientes resultados:

- **Resultados grado de mantenibilidad de los casos de estudio**

En la **Figura 8** se muestra el grado de mantenibilidad de los casos de estudio: SDLC y Gestion VR, en una escala de valoración de 1 al 100, considerados como:

GM-SDLC = 89.61% (Altamente mantenible).

GM-GestionVR = 82.34% (Moderadamente mantenible).

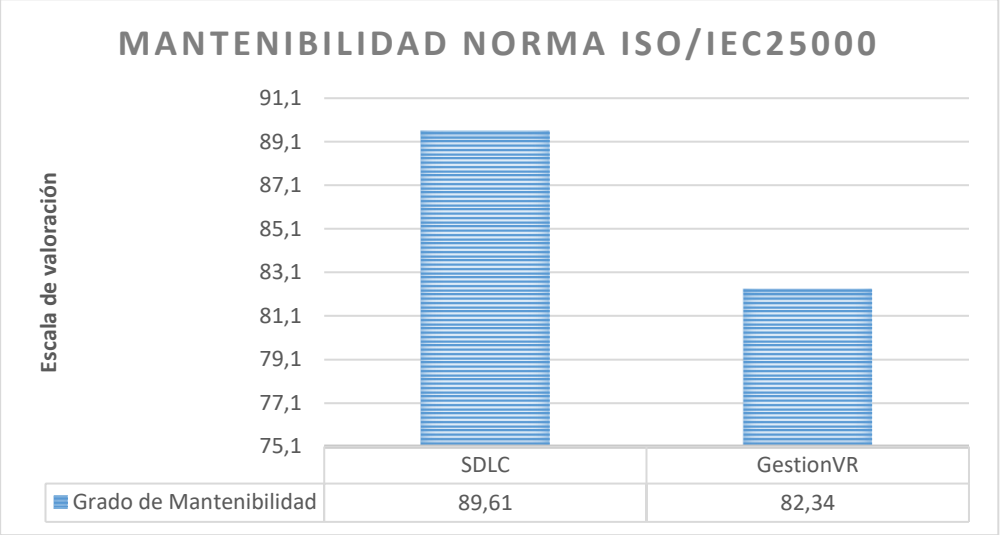


Figura 8. Grado de mantenibilidad de los casos de estudio.

- **Análisis del grado de calidad de las subcaracterísticas de la mantenibilidad**

  - **Caso de estudio SDLC**

En la **Figura 9** se analiza el grado de calidad de cada una de las subcaracterísticas respecto a la mantenibilidad del caso de estudio SDLC en una escala de valoración de 1 al 100.

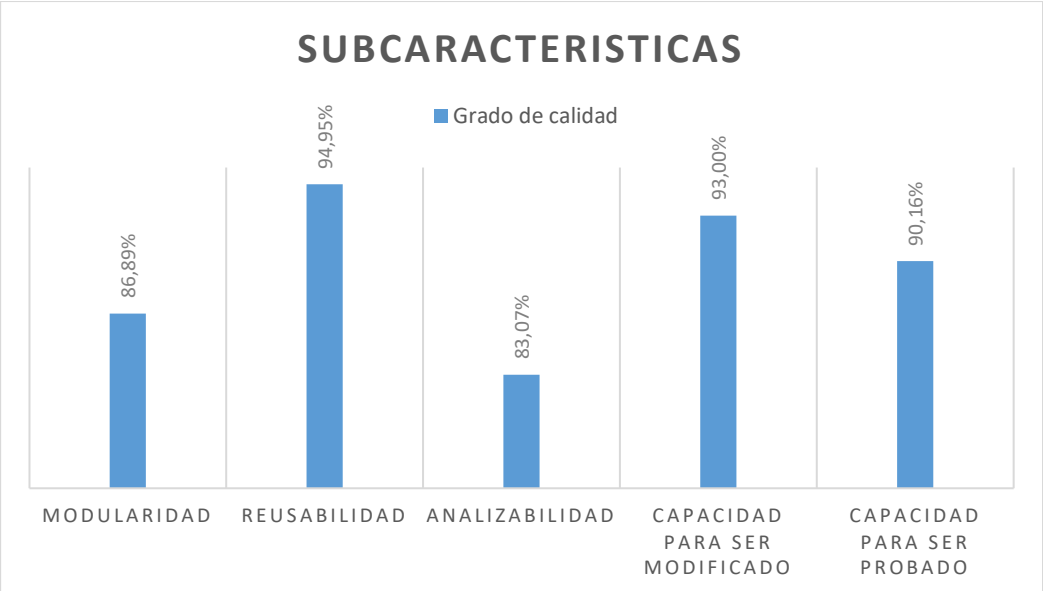


Figura 9. Grado de mantenibilidad de los casos de estudio.

Análisis. - Los resultados de la evaluación de la mantenibilidad del código revelan una perspectiva mayormente favorable, la reusabilidad y la capacidad para ser modificado se sitúan en niveles **Excelente**, lo que indica que el código es altamente flexible y fácil de integrar en otros proyectos, esto es un aspecto crucial para el desarrollo ágil y la sostenibilidad a largo plazo del software. Sin embargo, la analizabilidad y la modularidad se encuentran en niveles **Aceptable** (83.07% y 86.89%, respectivamente), esto sugiere que, aunque el código es funcional, su estructura podría ser optimizada para mejorar su comprensión y manejo (mayor claridad en la organización del código facilitaría su mantenimiento y la colaboración entre desarrolladores). Así mismo, la capacidad para ser probado alcanza un nivel **Aceptable** (90.16%), lo que implica que el código es robusto en términos de pruebas, no obstante, se recomienda incrementar la cobertura de pruebas para asegurar la calidad del software a largo plazo.

- **Caso de estudio Gestion VR**

En la **Figura 10** se muestra el análisis del grado de calidad de cada una de las subcaracterísticas respecto a la mantenibilidad.

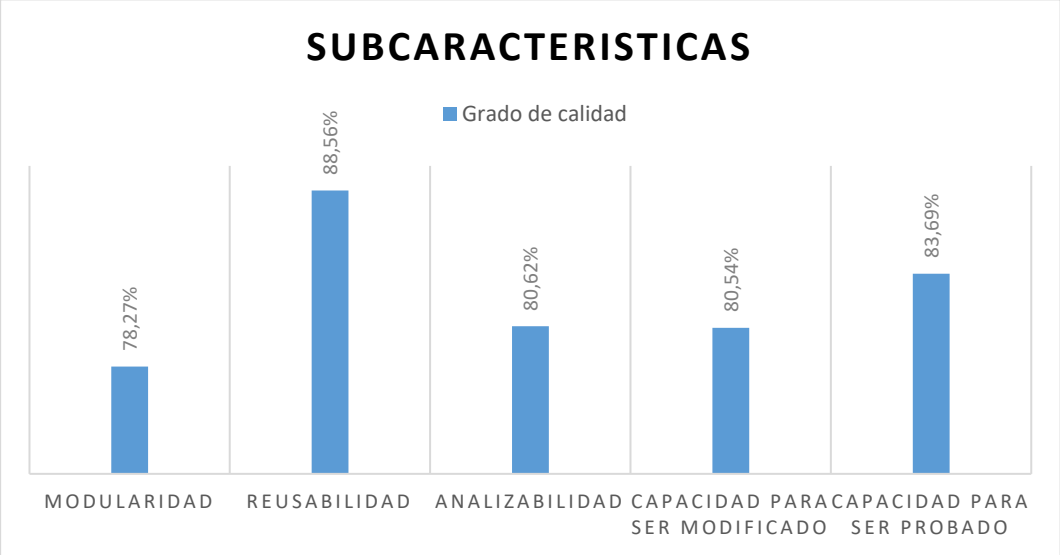


Figura 10. Grado de mantenibilidad de los casos de estudio.

Análisis. - Los resultados de la evaluación de la mantenibilidad del código muestran un panorama variado con un nivel **Aceptable**, la reusabilidad (88.56%) y la capacidad para ser modificado (80.54%), muestra que el código es flexible y fácil de adaptar a nuevos requisitos. Así mismo, la modularidad (78.27%) y la capacidad para ser probado (83.69%) indicando la

necesidad de mejorar la organización y estructura del código para facilitar su comprensión y mantenimiento, así como incrementar los esfuerzos en la cobertura de pruebas. Por otro lado, la analizabilidad (80.62%) lo que indica que el código es relativamente fácil de entender para los desarrolladores. En conjunto, estos resultados destacan las fortalezas del código, al tiempo que señalan áreas clave que requieren atención para optimizar su mantenibilidad a largo plazo.

- **Análisis de métricas relacionadas con las subcaracterísticas de la mantenibilidad**

En esta sección se muestra el análisis de cada una de las métricas de las subcaracterísticas de la mantenibilidad, de cada uno de los casos de estudio:

- **Caso de estudio SDLC**

Para tener un resultado final de cada métrica se aplicó el promedio entre el backend y frontend, lo que da a conocer el valor final de cada métrica.

a. Modularidad.

En la **Tabla 49** se muestra los resultados de cada métrica respecto al backend y el frontend.

Tabla 49. Resultado de las métricas en la característica del modularidad**.**

Métricas	Resultado
Diseño	9.70%
Duplicaciones	8.70%
Cobertura	20.94%

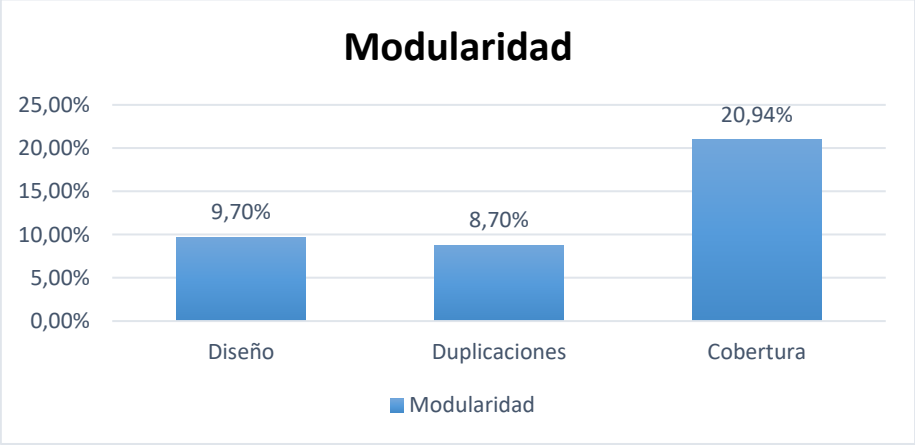


Figura 11. Análisis de subcaracterística de la modularidad.

Análisis. - En la **Figura 11** se presentan los valores de las métricas que influyen en la modularidad del código. La cobertura muestra un valor de 20.94%, lo que indica que las pruebas unitarias son insuficientes para garantizar la calidad del proyecto, en cuanto a las duplicaciones, se observa un valor de 8.70%, lo que sugiere una presencia moderada de código repetido, aunque este porcentaje no es extremadamente alto, puede afectar negativamente la modularidad, ya que cualquier mantenimiento requerirá localizar y modificar todas las instancias de código duplicado, lo que podría generar confusión y aumentar los costos. Por otro lado, la propiedad de diseño presenta un valor de 9.70%, lo que indica que el código es relativamente comprensible, contribuyendo positivamente a la modularidad y facilitando su mantenimiento. Finalmente, estos resultados destacan la necesidad de mejorar la cobertura de pruebas y reducir las duplicaciones para optimizar la modularidad del código y su mantenibilidad a largo plazo.

b. Reusabilidad.

En la **Tabla 50** se muestra los resultados finales de cada métrica respecto al backend y el frontend.

Tabla 50. Resultado de las métricas en la característica de la reusabilidad**.**

Métricas	Medición
Mala práctica	0.90%
Obsoleto	0.90%
Diseño	9.70%
Duplicaciones	8.70%

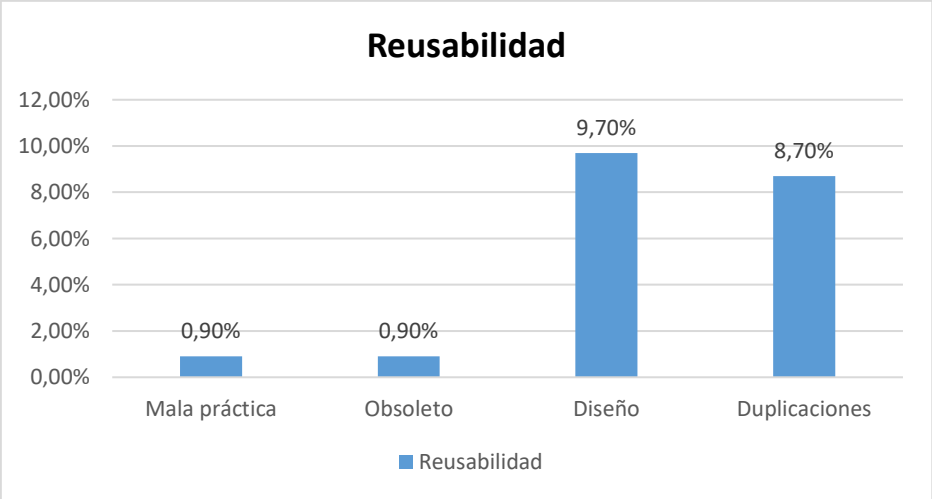


Figura 12. Análisis de subcaracterística de la reusabilidad.

Análisis. - En la **Figura 12** se presentan los valores de las métricas que influyen en la reusabilidad del código. La métrica de mala práctica y obsoleto muestra un valor de 0.90%, lo que indica que hay una mínima cantidad de código en desuso y aspectos de codificación que dificultan la lectura y comprensión del código, la métrica de diseño se sitúa en 9.70%, sugiriendo que la estructura del código es relativamente comprensible, lo que favorece su reutilización. Sin embargo, la duplicación se encuentra en 8.70%, lo que refleja una presencia moderada de código repetido, aunque este porcentaje no es excesivo, puede impactar negativamente en la reusabilidad y el mantenimiento del software. Finalmente, estos resultados destacan la necesidad de abordar prácticas de codificación y reducir las duplicaciones para mejorar la reusabilidad del código y facilitar su integración en otros proyectos.

c. Analizabilidad.

En la **Tabla 51**, se muestra los resultados finales de cada métrica respecto al backend y el frontend.

Tabla 51. Resultado de las métricas en la característica de la analizabilidad**.**

Métricas	Medición
Mala práctica	0.90%
Redundante	8.70%
Complejidad cognitiva	0.04%
Confuso	0.90%
Diseño	9.70%
Dificultad encontrada	0.04%
Densidad de comentarios	98.20%

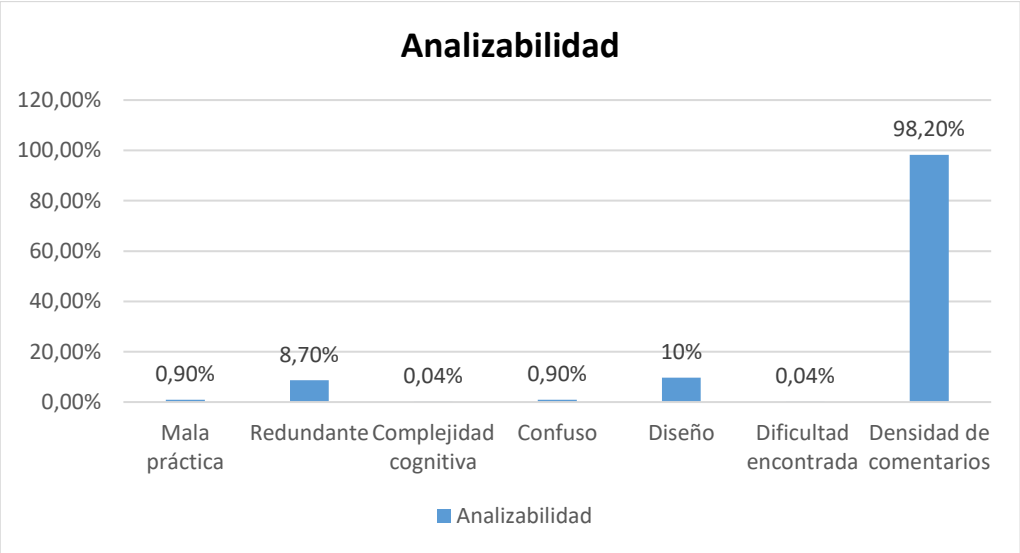


Figura 13. Análisis de subcaracterística de la analizabilidad.

Análisis. - En la **Figura 13** se examinan las métricas que influyen en la analizabilidad del software. La métrica complejidad cognitiva y la dificultad encontrada presentan valores muy bajos de 0.04%, lo que indica que el código es sencillo y comprensible, de la misma forma que la métrica de mala práctica y confuso muestra un valor de 0.90%, mostrando facilidad en lectura del código. La métrica el diseño se sitúa en 6.93%, definiendo que la estructura del código es aceptable, lo que contribuye a su comprensión y reutilización, Sin embargo, la limitación de comentarios es alta, alcanzando el 98.20%, lo que dificulta la comprensión del código, por otro lado, la métrica de redundante es de 8.70%, indicando que la duplicación de código, aunque mínima, debe ser abordada para evitar complicaciones en el mantenimiento. Finalmente, estos resultados sugieren que, aunque la identificación de fallos es fácil, la falta de comentarios puede dificultar las modificaciones en el sistema, lo que requiere atención para mejorar la analizabilidad.

d. Capacidad para ser modificado.

En la **Tabla 52**, se muestra los resultados finales de cada métrica respecto al backend y el frontend.

Tabla 52. Resultado de las métricas en la característica capacidad para ser modificado**.**

Métricas	Medición
Mala práctica	0.90%
Redundante	8.70%
Diseño	9.70%
Duplicaciones	8.70%

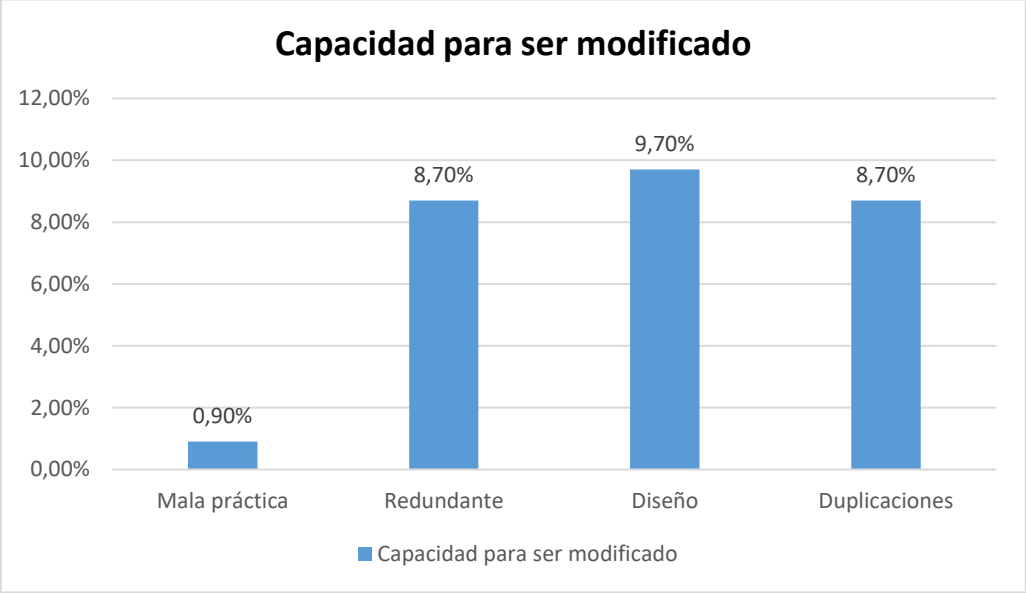


Figura 14 Análisis de subcaracterística de capacidad de ser modificado.

Análisis. – La **Figura 14** muestra los valores de las propiedades de calidad que influyen en la capacidad para ser modificado. Las duplicaciones y la redundancia tienen un valor de 8.70%, lo que, aunque no es excesivo, puede complicar el mantenimiento y las modificaciones, así como la métrica de mala práctica se sitúa en 0.90%, indicando que ciertos aspectos de codificación mínimos pueden dificultar la comprensión del código. Por otro lado, el diseño muestra un valor de 9.70%, lo que sugiere que la estructura del código es relativamente comprensible y facilita las modificaciones. Finalmente, reducir las duplicaciones y mejorar las prácticas de codificación para optimizar la capacidad de modificación del software, asegura que los cambios se realicen de manera efectiva y sin introducir defectos.

e. Capacidad para ser probado.

En la **Tabla 53** se muestra los resultados finales de cada métrica respecto al backend y el frontend.

Tabla 53. Resultado de las métricas en la característica capacidad para ser probado**.**

Métricas	Medición
Redundante	8.70%
Diseño	9.70%
Dificultad encontrada	0.04%
Cobertura	20.94%

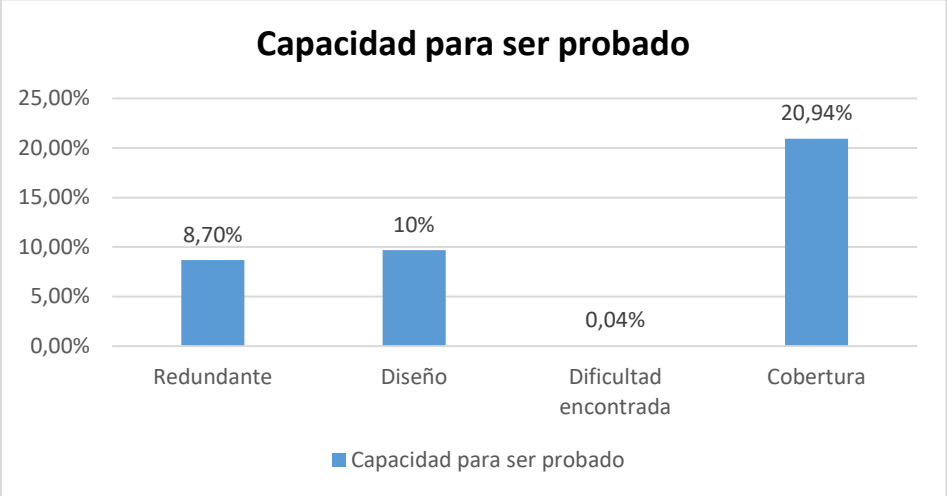


Figura 15. Análisis de subcaracterística de capacidad de ser probado.

Análisis. - En la **Figura 15** se examinan las métricas que influyen en la capacidad del proyecto para ser aprobado. La cobertura de pruebas registra un 20.94%, lo que indica una insuficiente implementación de pruebas unitarias en el código, la métrica de redundante presenta un valor de 8.70%, mostrando un nivel mínimo de duplicación de código y podría complicar su comprensión. Por otro lado, el diseño muestra un valor de 9.70%, lo que indica que la estructura del código es relativamente comprensible, y la dificultad encontrada es muy baja, con un valor de 0.04%, lo que sugiere que no hay problemas significativos en la implementación. Finalmente, estos resultados subrayan la importancia de aumentar la cobertura de pruebas y reducir la redundancia para optimizar la capacidad de prueba del software.

- **Caso de estudio Gestion VR**

En esta sección de muestra el análisis de cada métrica de las subcaracterísticas de la mantenibilidad del caso de estudio **Gestion VR**.

Para tener un resultado final de cada métrica se aplicó promedio entre el backend y frontend, lo que da a conocer el valor final de cada métrica.

a. Modularidad.

En la **Tabla 54** se muestra los resultados finales de cada métrica respecto al backend y el frontend.

Tabla 54. Resultado de las métricas en la característica de modularidad**.**

Métricas	Medición
Diseño	8.37%
Duplicaciones	33.84%
Cobertura	22.96%

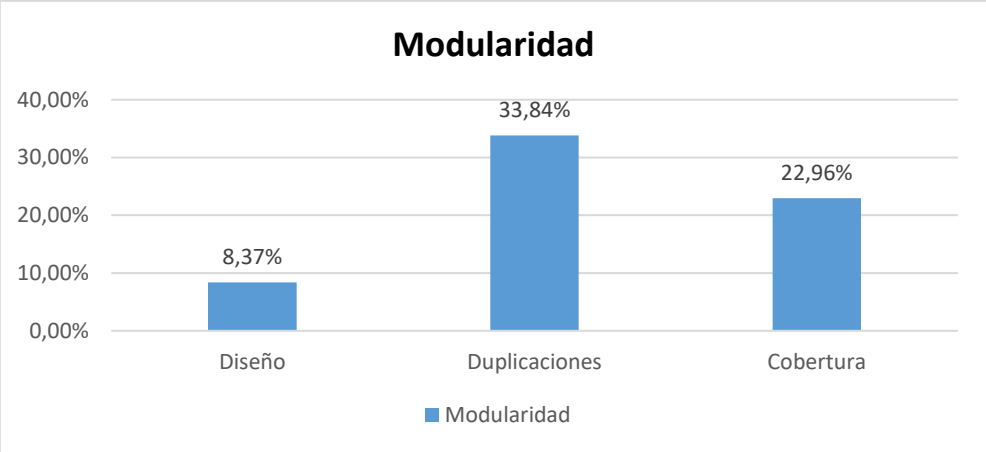


Figura 16. Análisis de subcaracterística de la modularidad.

Análisis. - En la **Figura 16**, se presentan los valores de las métricas que influyen en la modularidad del código. La cobertura muestra un valor de 22.96%, lo que indica que las pruebas unitarias son insuficientes para garantizar la calidad del proyecto, en cuanto a las duplicaciones, se observa un valor de 33.84%, lo que sugiere una presencia significativa de código repetido, aunque este porcentaje puede no ser tan crítico, puede impactar negativamente en la modularidad, ya que cualquier mantenimiento requerirá localizar y modificar todas las instancias de código duplicado, lo que podría generar confusión y aumentar los costos. Por otro lado, la propiedad de diseño presenta un valor de 6.93%, lo que indica que el código es relativamente comprensible, contribuyendo positivamente a la modularidad y facilitando su mantenimiento. Finalmente, estos resultados destacan la necesidad de mejorar la cobertura de pruebas y reducir las duplicaciones para optimizar la modularidad y mantenibilidad del código a largo plazo.

b. Reusabilidad.

En la **Tabla 55** se muestra los resultados finales de cada métrica respecto al backend y el frontend.

Tabla 55. Resultado de las métricas en la característica de la reusabilidad**.**

Métricas	Medición
Mala práctica	1.77%
Obsoleto	1.77%
Diseño	8.37%
Duplicaciones	33.84%

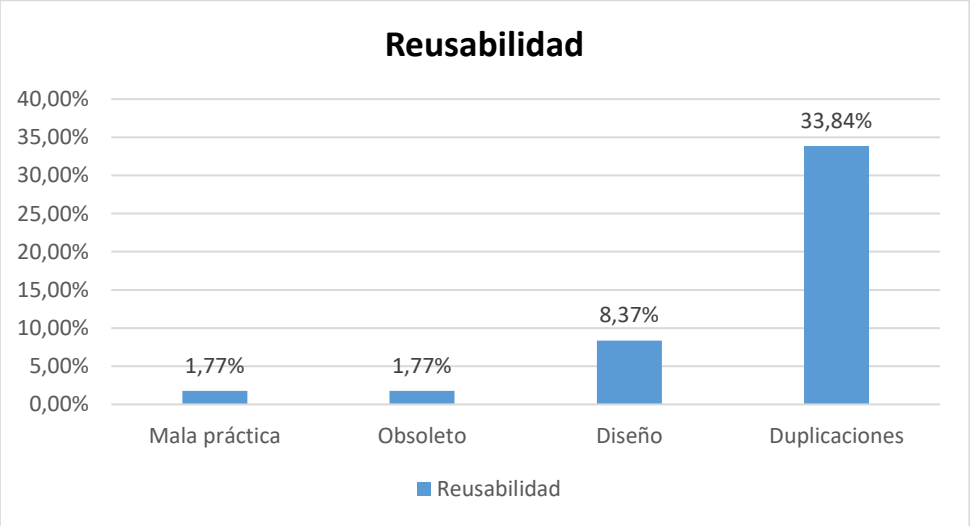


Figura 17. Análisis de subcaracterística de la reusabilidad.

Análisis. - En la **Figura 17** se representan los valores de cada una de las métricas que influyen en la reusabilidad. La métrica de mala práctica y obsoleto muestra un valor de 1.77%, lo que indica que hay una cantidad mínima de aspectos de codificación que dificultan la lectura, comprensión y el desuso del código, lo que es favorable para la reusabilidad. La métrica de diseño se sitúa en 8.37%, definiendo que la estructura del código es aceptable, lo que contribuye a su comprensión y reutilización. Por último, la duplicación se encuentra en 33.84%, lo que refleja una presencia significativa de código repetido, este porcentaje puede impactar negativamente la reusabilidad del software, complicando la reutilización de ciertas porciones de código. Finalmente, estos resultados subrayan la necesidad de mejorar la calidad del código, especialmente en lo que respecta a la reducción de duplicaciones y la eliminación de malas prácticas, para optimizar la reusabilidad del software.

c. Analizabilidad.

En la **Tabla 56** se muestra los resultados finales de cada métrica respecto al backend y el frontend.

Tabla 56. Resultado de las métricas en la característica de la analizabilidad**.**

Métricas	Medición
Mala práctica	1.77%
Redundante	33.84%
Cognitive Complexity	0.06%
Confuso	1.77%
Diseño	8.37%
Dificultad encontrada	0.06%
Densidad de comentarios	89.80%

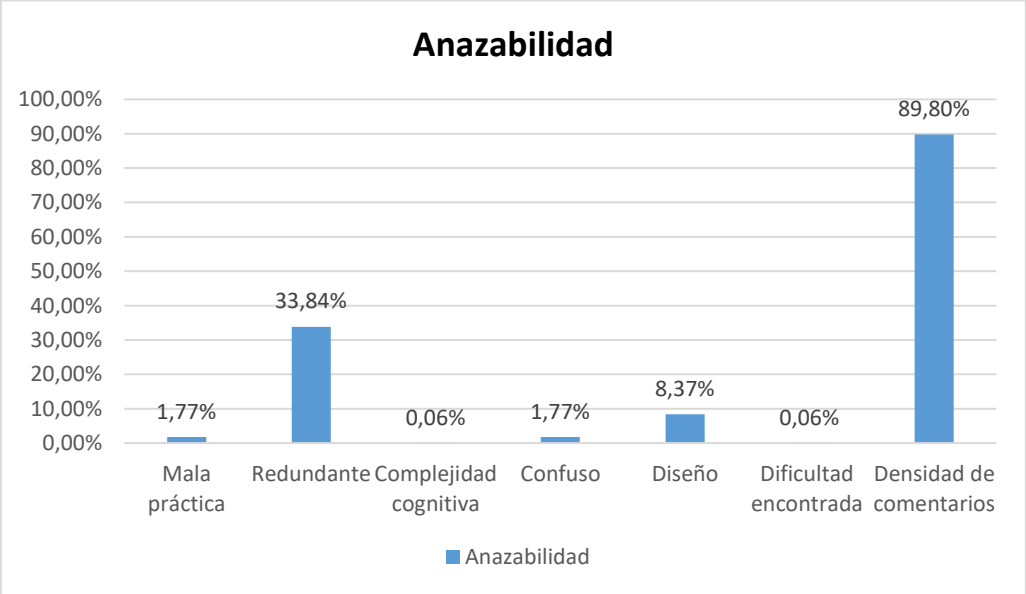


Figura 18. Análisis de subcaracterística de la analizabilidad.

Análisis. - En la **Figura 18** se examinan las métricas que influyen en la analizabilidad del software. La métrica complejidad cognitiva y dificultad encontrada registra un valor bajo de 0.06%, lo que sugiere que el código es relativamente sencillo, comprensible, así mismo, la métrica de mala práctica y confuso muestra un valor de 1.77%, lo que implica que existen aspectos de codificación mínimo que no dificultan la lectura del código fuente y la reutilización. La métrica el diseño se sitúa en 8.37%, definiendo que la estructura del código es aceptable, lo que contribuye a su comprensión y reutilización. Sin embargo, la falta de comentarios es alta, alcanzando el 89.80%, lo que dificulta la comprensión del código, así como la métrica

redundante presenta un valor de 33.84%, sugiriendo una presencia significativa de código duplicado, lo que podría complicar el mantenimiento del software. Finalmente, estos resultados indican que, aunque la identificación de posibles fallos es fácil, las malas prácticas y la duplicación de código pueden dificultar las modificaciones en el sistema, lo que requiere atención para mejorar la analizabilidad del software.

d. Capacidad para ser modificado.

En la **Tabla 57** se muestra los resultados finales de cada métrica respecto al backend y el frontend.

Tabla 57. Resultado de las métricas en la característica capacidad para ser modificado**.**

Métricas	Medición
Mala práctica	1.77%
Redundante	33.84%
Diseño	8.37%
Duplicaciones	33.84%

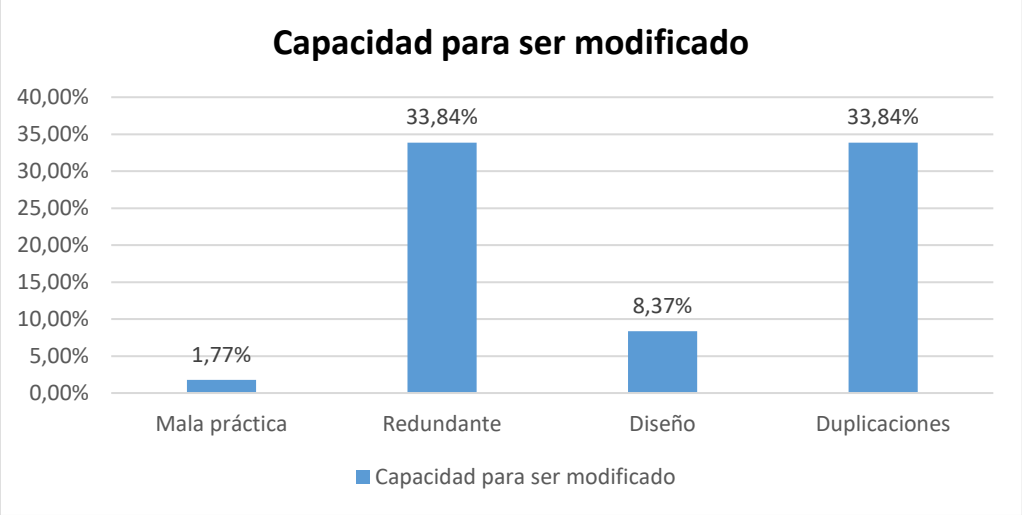


Figura 19. Análisis de subcaracterística de capacidad de ser modificado.

Análisis. - El la **Figura 19** muestra los valores de las propiedades de calidad que influyen en la capacidad para ser modificado. Las duplicaciones y la redundancia tienen un valor elevado de 33.84%, lo que puede generar complicaciones significativas durante el mantenimiento y las modificaciones, la métrica mala práctica se sitúa en 1.77%, mostrando ciertos aspectos de codificación que pueden dificultar la comprensión del código. Por otro lado, el diseño muestra un valor de 8.37%, indicando que la estructura del código es relativamente

comprensible y facilita las modificaciones. Finalmente, estos resultados destacan la necesidad de reducir las duplicaciones y mejorar las prácticas de codificación para optimizar la capacidad de modificación del software, asegurando que los cambios se realicen de manera efectiva y sin introducir defectos.

e. Capacidad para ser probado.

En la **Tabla 58**, se muestra los resultados finales de cada métrica respecto al backend y el frontend.

Tabla 58. Resultado de las métricas en la característica capacidad para ser probado**.**

Métricas	Medición
Redundante	33.84%
Diseño	8.37%
Dificultad encontrada	0.06%
Cobertura	22.96%

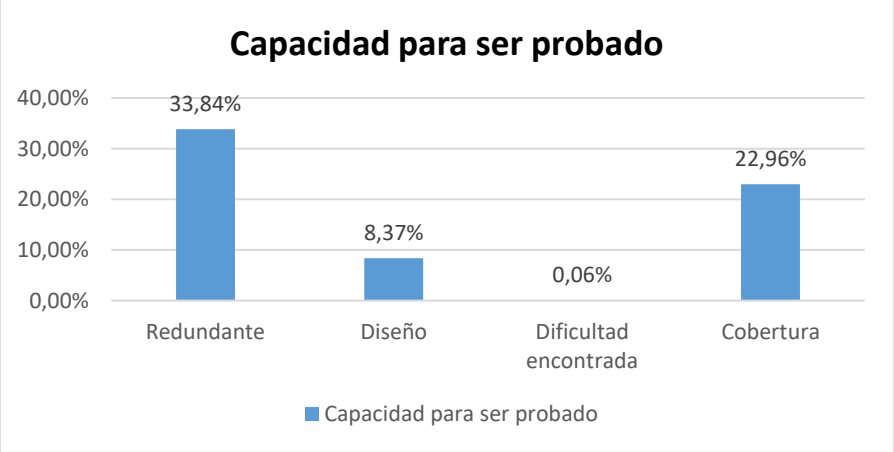


Figura 20. Análisis de subcaracterística de capacidad de ser probado.

Análisis. - En la **Figura 20** se examinan las métricas que influyen en la capacidad del proyecto para ser aprobado. La cobertura de pruebas registra un 22.96%, lo que indica una baja implementación de pruebas unitarias en el código, la métrica de redundante presenta un valor elevado de 33.84%, mostrando duplicación de código lo que podría complicar significativamente la comprensión y capacidad de prueba. Por otro lado, el diseño muestra un valor de 8.37%, lo que indica que la estructura del código es relativamente comprensible, y la dificultad encontrada es baja, con un valor de 0.06%, lo que se identifica que no hay problemas significativos en la implementación. Finalmente, los resultados subrayan la importancia crítica

de aumentar la cobertura de pruebas y reducir drásticamente la redundancia para optimizar la capacidad de prueba del software.

- **Identificación de defectos relacionados con la mantenibilidad**

Para determinar defectos, las propiedades de calidad deben estar por debajo de los porcentajes de aceptación, definidas en el modelo de medición según se detalla en la **Tabla 41**.

Para esto se tomó en consideración las mediciones del backend debido que es la encargada de la lógica de funcionamiento y la gestión de datos de los casos de estudio, para lo cual si una medición no cumple con los porcentajes aceptables es considerado como un error lo que desencadena un defecto en el software.

- **Caso de estudio: SDLC**

En la **Tabla 59** se muestra mediciones de las propiedades aplicadas al caso de estudio.

Tabla 59. Mediciones de las propiedades del caso de estudio SDLC**.**

Propiedad	Medición
Duplicated lines	707
Coverage lines	1388
Cyclomatic complexity	525
Code smells	60
Cognitive complexity	4
Comment lines	1,70%

- **Caso de estudio: Gestion VR**

En la **Tabla 60** se muestra el resultado de las mediciones aplicadas al caso de estudio.

Tabla 60. Mediciones de las propiedades del caso de estudio Gestion VR.

Propiedad	Medición
Duplicated lines	1228
Coverage lines	865
Cyclomatic complexity	277
Code smells	6
Cognitive complexity	0
Comment lines	20%

En la **Tabla 61** y **Tabla 62**, se muestra las propiedades que no cumplen con los porcentajes aceptables del modelo de medición, una **X** representa que la propiedad no cumple con el nivel de aceptación por lo cual es considerado como un defecto.

- **Caso de estudio: SDLC**

Líneas de código backend: 5300.

Tabla 61. Propiedades caso de estudio SDLC.

Propiedad	Medición	Porcentaje	Error
Duplicated lines	707	13.40%	✕
Coverage lines	1388	26.17%	✕
Cyclomatic complexity	525	9.91%	--
Code smells	60	60	✕
Cognitive complexity	4	0.07%	--
Comment lines	1.70%	1.70%	✕

- **Caso de estudio: Gestion VR**

Lines de código frontend: 2161.

Tabla 62. Propiedades caso de estudio Gestion VR.

Propiedad	Medición	Porcentaje	Error
Duplicated lines	1228	56,.77%	✕
Coverage lines	865	40.05%	✕
Cyclomatic complexity	277	12.83%	--
Code smells	6	6	✕
Cognitive complexity	0	0%	--
Comment lines	20%	20%	✕

A continuación, se muestra las propiedades que no cumplieron con los porcentajes aceptables del modelo de medición.

- Duplicated Lines.
- Coverage lines.
- Code smells.
- Comment lines.

Después de identificar las propiedades, se procedió con el reconocimiento de los defectos, esto basado en los análisis de cada una de las métricas que influyeron en el grado de calidad de las subcaracterísticas (Ver **Análisis**).

En la **Tabla 63** Se muestra los defectos derivados de cada propiedad.

Tabla 63. Propiedades caso de estudio Gestion VR.

Propiedad	Defecto
Duplicated Lines	Código duplicado
Coverage lines	Falta de pruebas unitarias
Code smells	Malas prácticas de codificación
Comment lines	Comentarios desactualizados o innecesarios

Nota: Los defectos de mantenibilidad mencionados están relacionados con la falta de aplicación de buenas prácticas de programación.

- **Tratamiento de los datos de evaluación**

La información que se obtuvo del resultado del diagnóstico de aplicaciones web, se documentó con el propósito que sirva como base para aplicar el **modelo de medición de mantenibilidad** para aportar control de calidad en el desarrollo del código de los estudiantes de la CIC de la UNL.

Terminado el diagnóstico del código de los casos de estudio, se pudo identificar que de las 6 propiedades que influyen en la medición de las métricas de las subcaracterísticas de la mantenibilidad, 4 no cumplen con el porcentaje aceptable del modelo de medición, los cuales son considerados como defectos respecto a la mantenibilidad. Además, se ha evidenciado que cada uno de los defectos encontrados están relacionados con la falta de aplicación de buenas prácticas de programación.

En la **Tabla 64** se muestra la matriz de defectos encontrados acorde a la norma ISO/IEC 25000 respecto a la mantenibilidad.

Tabla 64. Matriz de defectos encontrados acorde a la norma ISO/IEC 25000.

	Subcaracterística				
Defecto	**Modularidad**	**Reusabilidad**	**Analizabilidad**	**CM**	**CP**
Código duplicado	**X**	**X**		**X**	
Falta de pruebas unitarias					**X**
Malas prácticas de codificación			**X**		
Comentarios desactualizados o innecesarios			**X**		

Capacidad de ser modificado (**CM**); Capacidad de ser probado (**CP**).

Luego de terminar el proceso de evaluación, se encontraron la cantidad de 4 defectos relacionados a cada una de las subcaracterísticas de la mantenibilidad descritos en la **Tabla 64**, de los cuales se tomará los defectos de mayor prioridad para aplicar las respectivas pautas de mantenibilidad para aplicar la solución mediante el plan de estrategias.

6.2. Objetivo 2: Desarrollar un software web tomando como referencia XP como metodología de desarrollo, para solucionar el grado de mantenibilidad de una de las aplicaciones web diagnosticadas en el objetivo uno

Para iniciar con el objetivo 2, se utilizaron los defectos encontrados en el objetivo 1 (Ver **Tabla 64**), estos defectos se derivan de las propiedades de calidad que influyen directamente en la medición de las métricas que en consecuencia permiten obtener el grado de calidad cada subcaracterísticas para finalmente contribuir en la ponderación del grado de calidad del producto.

Con base en lo expuesto anteriormente, el propósito del presente objetivo es abordar y solucionar los defectos identificados en el caso de estudio Gestion VR, que ha sido seleccionado como primer caso de estudio del presente proyecto (Ver **Anexo 3**). Para lograrlo, se busca alcanzar un grado de mantenibilidad en una escala de valoración que oscile entre el 86% y el 100% (Ver **Tabla 42**), lo cual se considera un nivel óptimo de mantenibilidad (altamente mantenible).

De los cuatro defectos identificados, se priorizó la solución de los dos más críticos: el defecto de "código duplicado" y las "malas prácticas de codificación", esta selección se basó en un análisis técnico que consideró factores como el impacto en la calidad del código, la facilidad de implementación de soluciones y los beneficios a largo plazo para la mantenibilidad del software.

- La duplicación de código dificulta la comprensión de la lógica del programa, aumenta la probabilidad de errores y complica las tareas de modificación y actualización.
- Las malas prácticas de codificación abarcan una amplia gama de problemas, desde el uso inadecuado de convenciones de nomenclatura hasta violaciones en el diseño del software, estas prácticas pueden generar código confuso, propenso a errores y difícil de mantener.

Para cumplir con el objetivo propuesto, se planteó desarrollar una aplicación web utilizando ciertas prácticas de la metodología XP, conjuntamente, se llevó a cabo una revisión bibliográfica de buenas prácticas de programación utilizando algunas tareas de la metodología de Kitchenham, en la recolección de pautas de mantenibilidad para la solución de defectos.

6.2.1. Fase 1: Planeación

6.2.1.1. Revisión bibliográfica para recopilar pautas de mantenibilidad.

Para la investigación y selección de pautas de mantenibilidad se realizó una revisión bibliográfica con el fin de determinar que buenas prácticas de programación que influyen positivamente en la mantenibilidad del software, contribuyendo así a la reducción de defectos.

a. Planificación de la revisión bibliográfica.

Se optó por una metodología adaptada basada en algunas prácticas propuestas por Kitchenham para realizar revisiones sistemáticas de literatura, donde se consideraron los siguientes puntos: Justificación de la revisión, formulación de las preguntas de investigación y diseño de protocolo de búsqueda.

- **Justificación de la revisión**

La presente revisión bibliográfica tiene como objetivo investigar y seleccionar las buenas prácticas de programación para verificar la calidad del código a través de pautas de mantenibilidad.

- **Formulación de las preguntas de la investigación**

Para la formulación de la pregunta de investigación se utilizó el método PICOC, que consiste en definir las variables: Población, Intervención, Comparación, Resultado, Contexto.

En la **Tabla 65** se describen las variables utilizadas para el planteamiento de la pregunta.

Tabla 65. PICOC utilizados para la investigación.

Criterio	Descripción
Población	Buenas prácticas de programación
Intervención	Mantenibilidad del software basado en la norma ISO/IEC 25000
Comparación	Buenas prácticas de programación para asegurar la calidad del código
Resultado	Pautas de mantenibilidad para verificar la calidad del código
Contexto	Verificar la mantenibilidad de aplicaciones web en calidad interna

La pregunta de investigación que se plantea es: **¿Cuáles son las buenas prácticas de programación que aseguran la mantenibilidad del código respecto a la norma ISO/IEC 25000?**

- **Diseño del protocolo de búsqueda**

En este apartado se encuentra descrito cómo se realizó la búsqueda de los estudios primarios que se escogieron como resultado para la revisión bibliográfica.

- **Selección de las fuentes de búsqueda**

Luego de analizar preliminarmente diferentes bases de datos sugeridas en la actualidad como estrategia de búsqueda, se consideró las siguiente IEEE Xplorer, ScienceDirect, Scopus, ACM, Kluwer, Google Scholar, SciELo, Dialnet, entre otras.

En la **Tabla 66** se muestra las bases de datos científicas que se tomaron en cuenta como estrategia de búsqueda.

Tabla 66. Base de datos científicas tomadas en cuenta.

Fuente	Dirección
IEEE	https://ieeexplore.ieee.org/Xplore/home.jsp
ACM	https://dl.acm.org/
Scopus	https://www.scopus.com/home.uri
SciELo	https://www.scielo.org/es/

- **Palabras claves y cadenas de búsqueda**

Para cada motor de búsqueda de las respectivas bases de datos, dada su particular forma de operación, se utilizó la opción avanzada con el fin de formular scripts de búsqueda, probando combinaciones de las palabras clave que se derivaron de la pregunta de investigación:

En la **Tabla 67** se presenta las palabras claves derivadas de la pregunta de investigación.

Tabla 67. Palabras claves para la investigación.

Palabras Claves
Code maintainability
Good programming practices
Software quality
Coding standards
Static code analysis

En la **Tabla 68** se presenta los scripts de búsqueda para la investigación avanzada.

Tabla 68. Scripts de búsqueda para la investigación.

Base de datos científica	Script de búsqueda
ACM	(("code maintainability") AND ("good programming practices" OR "coding standards") AND ("software quality" OR "static code analysis"))
Scopus	((TITLE-ABS-KEY("code maintainability") OR TITLE-ABS-KEY("good programming practices" OR "coding standards")) AND (TITLE-ABS-KEY("software quality") OR TITLE-ABS-KEY("static code analysis"))) AND PUBYEAR > 2012
SciELo	(("code maintainability") AND ("good programming practices" OR "coding standards") AND ("software quality" OR "static code analysis"))
Dialnet	No permite realizar búsquedas basada en scripts.

- **Criterios de inclusión y exclusión**

Se definió criterios de selección para obtener estudios primarios, el propósito es excluir estudios ajenos a la pregunta de investigación, los criterios de inclusión y exclusión se basan en la pregunta de investigación.

En la **Tabla 69** se muestra los criterios de inclusión y exclusión.

Tabla 69. Criterios de inclusión y exclusión.

Criterios de inclusión	Criterios de exclusión
• Artículos científicos revistas o congresos. • Documentos a partir del 2015. • Tipo de documentos como artículos, libros, tesis doctorales o trabajos relacionados. • Documentos publicados en español o en inglés. • Documentos que el título o resumen contengan las palabras claves • Documentos cuyo título tenga relación con el tema de investigación.	• Documentos que no contribuyan a la pregunta de investigación. • Documentos poco claros. • Documentos duplicados. • Documentos que no cumplen con los criterios de inclusión.

- **Selección de estudios primarios**

Para la selección de los estudios primarios se tomó en cuenta los criterios de inclusión y exclusión descritos en la **Tabla 69**, en base al título, resumen y palabras claves de los documentos obtenidos.

En la **Tabla 70** se muestra el resultado de la aplicación de los criterios de inclusión y exclusión, como resultados los documentos primarios.

Tabla 70. Resultado de la aplicación de los criterios de inclusión y exclusión.

BD Científicas	Documentos incluidos	Documentos excluidos	Estudios primarios
IEEE Xplorer ScienceDirect Scopus ACM Google Scholar SciELo Dialnet	43 documentos	15 documentos	28 documentos

Nota: En el siguiente enlace se encuentra los documentos incluidos en la revisión bibliográfica: https://drive.google.com/drive/folders/13Oq5i7BIyvDwUi0_EYEDw5gefgAP_aY1?usp=sharing.

- **Evaluación de calidad**

Se aplicaron los criterios de evaluación de calidad de Kitchenham a los estudios primarios, un total de 28 documentos los cuales fueron renombrados para su respectiva descripción, mismo que se pueden consultar en el siguiente enlace. https://drive.google.com/drive/folders/1TySGU0tRZyklbN-1u2a5xbla5-n_oJGo.

En la **Tabla 71** se muestra las preguntas y parámetros utilizados como puntuación: Y=1, P=0.5, N=0.

Tabla 71. Preguntas y parámetros utilizados para la evaluación de calidad.

Criterio	Pregunta	Evaluación
QA1	¿El documento cumple con los criterios de inclusión y exclusión?	Y= si, N= no, P= parcialmente
QA2	¿El autor o autores sustentan el problema de investigación?	Y= en su totalidad, N= no se sustenta, P=se sustenta parcialmente
QA3	¿El autor o autores sustentan el uso de buenas prácticas de programación en el problema de investigación?	Y= si, N= no, P= parcialmente
QA4	¿Los hallazgos abordan la pregunta de investigación original?	Y= si, N= no, P= parcialmente

En la **Tabla 72** se muestra los resultados obtenidos de la aplicación de la evaluación de calidad.

Tabla 72. Resultados de evaluación de calidad.

Documento	QA1	QA2	QA3	QA4	Puntuación
Investigación 1	Y	Y	Y	Y	4
Investigación 2	Y	Y	Y	Y	4
Investigación 3	Y	Y	Y	Y	4
Investigación 4	Y	Y	Y	Y	4
Investigación 5	Y	Y	Y	Y	4
Investigación 6	Y	Y	Y	Y	4
Investigación 7	Y	Y	Y	Y	4
Investigación 8	Y	Y	Y	Y	4
Investigación 9	Y	Y	Y	Y	4
Investigación 10	Y	Y	Y	Y	4
Investigación 11	Y	Y	Y	Y	4
Investigación 12	Y	Y	Y	Y	4
Investigación 13	Y	Y	Y	Y	4
Investigación 14	Y	Y	Y	Y	4
Investigación 15	Y	Y	Y	Y	4
Investigación 16	Y	Y	Y	Y	4
Investigación 17	Y	Y	Y	Y	4
Investigación 18	Y	Y	Y	Y	4
Investigación 19	Y	Y	Y	Y	4
Investigación 20	Y	Y	Y	Y	4
Investigación 21	Y	Y	Y	Y	4
Investigación 22	Y	Y	Y	Y	4
Investigación 23	Y	Y	Y	Y	4
Investigación 24	Y	Y	Y	Y	4
Investigación 25	Y	Y	Y	Y	4
Investigación 26	Y	Y	Y	Y	4
Investigación 27	Y	Y	Y	Y	4
Investigación 28	Y	Y	Y	Y	4

b. Extracción de datos.

Una vez ejecutado el protocolo de la revisión bibliográfica, con un total de veintiocho documentos primarios, se procedió con la extracción de datos, para lo cual se adaptó una matriz bibliográfica según las necesidades del presente TIC, tomando como modelo tablas de extracción de datos de la metodología aplicada de barbara kitchenham en el desarrollo de revisiones sistemáticas.

En la **Tabla 73** se muestra los atributos de la matriz bibliográfica adaptada.

Tabla 73. Modelo matriz bibliográfica.

Descripción	Detalle
Título del documento	Nombre de la investigación
Autor	Nombre del autor o autores
Referencia/Documento	Ubicación del documento
Año	Año de publicación
Resumen	Descripción de la investigación
Propósito	Objetivo de la investigación
Conclusión destacada	Destaca lo importante del documento
Términos claves	Conceptos importantes a la investigación

La extracción de información del primer documento (Ver **Tabla 74**) y de los siguientes documentos que se detallan en el **Anexo 13**, se realizaron con el propósito de recopilar datos relevantes en la investigación de buenas prácticas de programación.

Tabla 74. Extracción de datos documento 1**.**

Descripción	Detalle
Título del documento	Mantenibilidad del Software - Consideraciones para su especificación y validación
Autor	Jenny Adones Farfán, Vianca Vega Zepeda
Referencia/Documento	https://drive.google.com/file/d/1oN7xzNz3xPvWOtlR6zQeLw_OWwZwCx9K/view?usp=sharing
Año	2020
Propósito	El propósito del presente documento es presentar algunas consideraciones importantes sobre el proceso de mantenimiento de software, que se deben tener en cuenta a la hora de especificar el atributo de mantenibilidad, de tal forma que posteriormente se pueda validar que efectivamente el producto es mantenible. Para esto, el documento hace una revisión bibliográfica y presenta una serie de desafíos del proceso de mantenimiento, métricas asociadas a la mantenibilidad de los productos y algunas recomendaciones para garantizar la mantenibilidad.
Conclusión destacada	Destaca la importancia de la especificación precisa de los requisitos de mantenibilidad del software y del uso de buenas técnicas de ingeniería de software, como el desarrollo orientado a objetos y la administración de la configuración.
Términos claves	Arquitectura de software, estándar de codificación, lista de verificación para la validación de requisitos

c. Matriz de resultados de la revisión bibliográfica

Luego de haber implementado la extracción de datos de los veintiocho documentos que responden la pregunta de investigación, los cuales se encuentran en la nube de Google drive https://drive.google.com/drive/folders/1TySGU0tRZyklbN-1u2a5xbla5-n_oJGo?usp=sharing,

se procedió con el análisis para recopilar las buenas prácticas de programación que se utilizara como pautas de mantenibilidad para la solución de defectos del presente TIC.

A continuación, en la **Figura 21** se resume la recopilación de buenas prácticas de programación de la métrica de mala práctica, relacionada con la subcaracterística de la Analizabilidad.

Mala practica		
Asigna nombres claros y descriptivos a variables, funciones y clases para mejorar la legibilidad y mantenibilidad del código.	**Cumple totalmente**: Todos los nombres son claros y descriptivos, no se necesitan comentarios adicionales.	**Descripcion**: La pauta ayuda a determinar si los nombres están bien elegidos, lo que hace que el código sea más legible y comprensible, facilitando su mantenimiento y evolución. Al seguir esta práctica, los desarrolladores podrán entender rápidamente la finalidad y el uso de cada elemento del código, reduciendo el tiempo necesario para realizar cambios o corregir errores.
	Cumple parcialmente: La mayoría de los nombres son claros y descriptivos, algunos podrían mejorar.	
	Cumple minimamente: Muchos nombres no son claros o descriptivos, se necesitan comentarios adicionales.	
	No Cumple: La mayoría de los nombres son ambiguos o genéricos, es difícil entender el código incluso con comentarios.	
Sigue Convenciones de Nomenclatura CamelCase (JavaScript): Escribir palabras compuestas sin espacios y con la primera letra de cada palabra en mayúscula, excepto la primera palabra. Por ejemplo, "nombreDeVariable", **snake_case (Python)**: Escribir palabras compuestas en minúsculas y separarlas con guiones bajos. Por ejemplo, "nombre_de_variable"	**Excelente**: Todo el código sigue consistentemente las convenciones de nomenclatura del lenguaje.	**Descripcion**: Esta pauta ayuda a determinar si se adhieren a las convenciones de nomenclatura establecidas, lo cual es esencial para mantener analizabilidad, reusabilidad y capacidad de ser modificado del software.
	Cumple parcialmente: La mayoría del código sigue las convenciones, con pocas excepciones menores.	
	Cumple minimamente: Muchas partes del código no siguen las convenciones, causando confusión.	
	No cumple: El código rara vez sigue las convenciones, dificultando la lectura y mantenimiento.	
Escribe comentarios en el código que expliquen el "por qué" de las decisiones de programación, y no solo el "qué" hace el código.	**Cumple totalmente**: Los comentarios en el código explican claramente el propósito y la lógica detrás de las decisiones de programación.	**Descripcion**. - Esta pauta ayudaa evealuar si detrás de ciertas implementaciones. Esto es crucial para mantener la claridad y facilitar el mantenimiento del código a largo plazo.
	Cumple parcialmente: Los comentarios explican en parte el propósito del código, pero no siempre abordan el "por qué" de las decisiones.	
	Cumple mínimamente: Los comentarios están presentes, pero en su mayoría solo describen lo que hace el código sin explicar las razones detrás de las decisiones.	
	No cumple: Los comentarios son inexistentes o extremadamente escasos.	

Figura 21. Buenas prácticas de programación, métrica analizabilidad.

En el siguiente enlace se documenta las buenas prácticas de programación recopiladas, relacionadas con las métricas: Mala práctica, redundante, complejidad cognitiva, obsoleto,

confuso, diseño, dificultad encontrada, densidad de comentarios, duplicaciones y cobertura. Cada una relacionada a las subcaracterísticas de la norma ISO/IEC 25010.

Enlace:
https://docs.google.com/spreadsheets/d/1iUC_GQZKAUWcIXF69jQSfCbip1xQS9H5/edit?usp=sharing&ouid=115484875849752259254&rtpof=true&sd=true.

6.2.1.2. Preparación de entrevista y encuesta para recopilar los requisitos de sistema.

Las preguntas de la encuesta tienen como propósito la recopilación de información asociada con los requisitos de la aplicación web y entrevistas en base a especificaciones técnicas por parte de docentes y gestor académico de la carrera de computación, acerca de la aplicación de control de calidad en el desarrollo del código, las cuales se detallan en el **Anexo 14**.

En la **Tabla 75** se muestra la síntesis de las preguntas formuladas para la entrevista del gestor académico de la carrera como responsable del centro de datos de los laboratorios.

Tabla 75. Preguntas de la entrevista dirigida al gestor académico, para la recopilación de requisitos.

Código	NP	CP
PEG1	Pregunta 1	¿Qué funcionalidades considera esenciales para la gestión de usuarios en la aplicación?
PEG2	Pregunta 2	¿Qué tipo de autenticación considera más adecuada para el registro de usuarios
PEG3	Pregunta 3	¿En qué formato prefiere recibir los informes de mantenibilidad?
PEG4	Pregunta 4	¿Considera importante que el sistema mantenga un historial de todas las evaluaciones de mantenibilidad realizadas?
PEG5	Pregunta 5	¿Qué características considera esenciales para la interfaz de usuario de la aplicación?
PEG6	Pregunta 6	¿Qué otra función considera esencial en la aplicación respecto a lo anterior?

Número de Pregunta (NP); Contenido de Pregunta (CP); Pregunta de Entrevista Gestor (PEG).

En la **Tabla 76** se muestra la síntesis de las preguntas formuladas para la entrevista a los docentes de la carrera de computación con una muestra de 4 participantes.

Tabla 76. Preguntas de la entrevista dirigidas a docentes de la carrera, para la recopilación de requisitos.

Código	NP	CP
PED1	Pregunta 1	¿Cree que sería útil que el software presente escalas de valoración de referencia para las métricas?
PED2	Pregunta 2	¿Cree que sería útil que el software presente escalas aceptación de referencia para las métricas?

Código	NP	CP
PED3	Pregunta 3	¿Cómo considera que debería estructurarse la lista de verificación para las pruebas manuales de mantenibilidad?
PED4	Pregunta 4	¿Cómo le gustaría que el software permita registrar y documentar los resultados de la verificación de la mantenibilidad?
PED5	Pregunta 5	¿Qué tipo de información le gustaría que se registre?
PED6	Pregunta 6	¿Considera importante que la lista incluya instrucciones o guías de uso?
PED7	Pregunta 7	¿Cómo le gustaría que el software presente los resultados de la verificación de mantenibilidad?
PED8	Pregunta 8	¿Preferiría visualizaciones gráficas y sugerencias?
PED9	Pregunta 9	¿Qué otro tipo de información le gustaría que se incluya en los resultados?
PED10	Pregunta 10	¿Qué otras funcionalidades o características le gustarían que tuviera el software para la mejora de la mantenibilidad del código?

Número de Pregunta (NP); Contenido de Pregunta (CP); Pregunta de Entrevista Docentes (PED).

En la **Tabla 77** se muestra la síntesis de las preguntas formuladas para la encuesta a estudiantes del itinerario de ingeniería en software de la carrera, con una muestra de 31 participantes.

Tabla 77. Preguntas de la encuesta dirigida a estudiantes del itinerario de IS, para la recopilación de requisitos.

Código	NP	CP
PEE1	Pregunta 1	¿Considera útil aplicar pruebas manuales que incluya una lista de verificación para evaluar el cumplimiento de las pautas de mantenibilidad del software?
PEE2	Pregunta 2	¿Es importante para usted que las pruebas manuales de calidad relacionadas con el código cubran las subcaracterísticas de la mantenibilidad de la norma ISO/IEC 25000?
PEE3	Pregunta 3	¿Considera útil el uso de APIs de análisis de código estático para la evaluación de mantenibilidad?
PEE4	Pregunta 4	¿Considera importante la utilización de las propiedades del de análisis de código estático para calcular el grado de mantenibilidad basado en la norma ISO/IEC 25000?
PEE5	Pregunta 5	¿Le gustaría que los resultados de la verificación de mantenibilidad generen un informe detallado con recomendaciones de mejora?

Número de Pregunta (NP); Contenido de Pregunta (CP); Pregunta de Encuesta a Estudiantes (PEE).

6.2.1.3. Ejecución de las entrevistas y encuesta para recopilar los requisitos de sistema.

El **Anexo 15** presenta la transcripción de la entrevista que se aplicó al Ing. Pablo Ordoñez, como administrador del centro de datos del laboratorio de la carrera y a los docentes Ing. Edison Coronel, Ing. Valeria Herrera, Ing. Oscar Cumbicus y Ing. Pablo Ordoñez en calidad de asesores técnicos para el proyecto **Software para verificar la mantenibilidad de**

aplicaciones web, además, de la encuesta (Ver **Anexo 16**) que se aplicó a los estudiantes del itinerario de ingeniería de software como usuarios, con el propósito de recopilar los requisitos y necesidades necesarias para verificar la mantenibilidad de aplicaciones web en los laboratorios de la carrera de computación de la UNL.

En la **Tabla 78** muestra las respuestas obtenidas de la entrevista aplicada al gestor académico, organizadas de acuerdo con cada una de las preguntas planteadas en la **Tabla 75**.

Tabla 78. Respuestas de la entrevista aplicada al gestor académico para la recopilación de requisitos.

Código	Respuesta de Pregunta
PEG1	Actualmente para aplicaciones desarrolladas en la carrera en la parte de gestión de usuarios se está implementando Aerobase IAM.
PEG2	La autenticación de usuarios debería llevarse a cabo mediante Aerobase IAM
PEG3	Es conveniente que se exporten en formato PDF.
PEG4	Si es pertinente, puesto que permitirá mantener un registro de evaluaciones de nuestro software lo que facilitará un análisis continuo y mejorará el control de la calidad
PEG5	Facilidad de uso e intuitiva
PEG6	Seria pertinente, que al finaliza una evaluación un docente de la carrera la firme electrónicamente

Pregunta de Entrevista Gestor (PEG).

En la **Tabla 79** muestra la síntesis de las respuestas obtenidas de las entrevistas realizadas a los docentes, organizadas de acuerdo con cada una de las preguntas planteadas en la **Tabla 76**.

Tabla 79. Respuestas de las entrevistas aplicada a los docentes para la recopilación de requisitos.

Código	Respuesta de Pregunta
PEE1	Si es necesario.
PEE2	Si es necesario.
PEE3	Debería estructurarse por subcaracterísticas.
PEE4	Las pruebas manuales necesario en un formato digital como checklist y las pruebas basadas en análisis de código estático semejante a una calculadora.
PEE5	Es conveniente registrar comentarios, además de ponderar y sacar un promedio como calificación.
PEE6	Si, siempre es necesario comprender cada test, además de una descripción de las métricas.
PEE7	Se podría estructurar o dividir entre las métricas con calificaciones bajas y altas, para enviar una alerta respecto a la misma.
PEE8	Es atractivo y contribuye a una mejor comprensión.
PEE9	Se podría agregar sugerencias de acuerdo a las métricas.
PEE10	El informe de resultados permite ser validado mediante una firma electrónica.

Pregunta de Entrevista Estudiante (PEE).

En la **Tabla 80** muestra la síntesis de las respuestas obtenidas de las encuestas realizadas a los estudiantes, organizadas de acuerdo con cada una de las preguntas planteadas en la **Tabla 77**.

Tabla 80. Respuestas de las entrevistas aplicada a los estudiantes para la recopilación de requisitos.

Código	Respuesta de Pregunta
PEE1	El 93.5% considera útil aplicar pruebas manuales que incluyan una lista de verificación para evaluar el cumplimiento de las pautas de mantenibilidad.
PEE2	E 96.8% indica una clara preferencia que las pruebas manuales de calidad del código abarquen las subcaracterísticas de mantenibilidad de la norma ISO/IEC 25000.
PEE3	El 96.8% opinan que sí es útil.
PEE4	El 96.8% considera importante la utilización de APIs de análisis de código estático para calcular el grado de mantenibilidad basado en la norma ISO/IEC 25000
PEE5	El 100% manifestó que considera importante que el informe incluya recomendaciones específicas.

Pregunta de Entrevista Gestor (PEG).

6.2.1.4. Desarrollo de las historias de usuario.

La implementación de las historias de usuario permitió establecer las características de uso que el software debe cumplir, a través de criterios de aceptación que abarcan tanto los requisitos funcionales y no funcionales.

A continuación, se muestra un resumen de las historias de usuario de mayor relevancia para la presente investigación, las cuales están documentadas en el **Anexo 17**.

En la **Figura 22** se muestra el escenario para el inició de sesión de los usuarios de la aplicación web.

Historia de usuario			
ID: HU01	Inicio de sesión		
Usuario/Rol	Administrador		
Prioridad en negocio	Alta	Riesgo en desarrollo	Alta
Descripción	Como administrador, deseo iniciar sesión utilizando el servicio de la carrera de autenticación de Aerobase IAM, de la misma forma para el perfil usuario.		
Nro.	Escenario	Criterio de aceptación	
1	Acceder al software con las credenciales de usuario	• El servicio debe mostrar un formulario para ingresar los datos de inicio de sesión: correo y contraseña. • El servicio permite dar paso al inicio de sesión en el software una vez se hayan ingresado las credenciales. • Si las credenciales están vacías o no existen, se mostrará una alerta o mensaje de error. • El servicio realizara la verificación de las credenciales como estudiantes de la carrera de computación para determinar el acceso al sistema.	
2	Redirección según el rol del administrador y usuario	• El sistema para el perfil administrador lo redirige a la página de administración de pautas, lista de verificación, métricas y subcaracterísticas. • El sistema redirige al usuario a la página principal para verificar la mantenibilidad.	

Figura 22. Historia de usuario, inicio de sesión.

En la **Figura 23** se muestra el escenario administración de pauta de mantenibilidad de las listas de verificación.

Historia de usuario			
ID: HU02	Administrar pauta		
Usuario/Rol	Administrador		
Prioridad en negocio	Media	Riesgo en desarrollo	Media
Descripción	Como administrador, quiero agregar, modificar y eliminar pautas de mantenibilidad en el software, pertenecientes a las listas de verificación.		

Nro.	Escenario	Criterio de aceptación
1	Registrar nueva pauta de mantenibilidad	• Como administrador, accedo a la página de administración de pautas de mantenibilidad. • El sistema muestra la opción para agregar una nueva pauta de mantenibilidad. • Ingreso el texto de la nueva pauta de mantenibilidad. • Si existen parámetros por completar se debe mostrar la alerta respectiva. • Guardo la nueva pauta de mantenibilidad. • Se muestra un mensaje temporal indicando que la pauta de mantenibilidad se ha creado exitosamente. • La nueva pauta de mantenibilidad se agrega a la lista de pauta de mantenibilidad disponibles.
2	Modificar pauta de mantenibilidad existente	• Como administrador, accedo a la página de administración de pautas de mantenibilidad. • Selecciono la pauta de mantenibilidad que quiero actualizar/modificar. • Modifico el texto de la pauta de mantenibilidad o su opción de respuesta. • Guardo los cambios. • Se muestra un mensaje temporal indicando que la pauta de mantenibilidad se ha modificado exitosamente. • La pauta de mantenibilidad modificada se actualiza en la lista de preguntas.
3	Eliminar pauta de mantenibilidad existente	• Como administrador, accedo a la página de pautas de mantenibilidad. • Selecciono la pauta de mantenibilidad que quiero eliminar. • El sistema muestra el modal de confirmación de eliminación. • Confirmo la eliminación. • Se muestra un mensaje temporal indicando que la pauta de mantenibilidad se ha eliminado exitosamente. • La pauta de mantenibilidad se elimina de la lista de pautas de mantenibilidad disponibles.

Figura 23. Historia de usuario, administrar pauta de mantenibilidad.

En la **Figura 24** se muestra el escenario administrar lista de verificación, relacionada a cada una de las métricas.

Historia de usuario			
ID: HU03	Administrar lista de verificación		
Usuario/Rol	Administrador		
Prioridad en negocio	Media	Riesgo en desarrollo	Media
Descripción	Como administrador, quiero poder agregar, modificar y eliminar las listas de verificación en el sistema, pertenecientes a las metricas. También, cada lista de verificación tendrá preguntas únicas asociadas.		

Nro.	Escenario	Criterio de aceptación
1	Agregar nueva lista de verificación	• Como administrador, accedo a la página de administración de listas de verificación. • El sistema muestra la opción para agregar una nueva lista de verificación. • Ingreso el texto de la nueva lista de verificación y las pautas de mantenibilidad disponibles/no perteneciente a otra lista de verificación. • Guardo la nueva lista de verificación. • Se muestra un mensaje temporal indicando que la lista de verificación se ha creado exitosamente. • La nueva lista de verificación se agrega a la lista de prácticas disponibles.
2	Modificar lista de verificación existente	• Como administrador, accedo a la página de administración de listas de verificación. • Selecciono la lista de verificación que quiero actualizar/modificar. • Modifico el texto de la lista de verificación o sus pautas de mantenibilidad/ pautas de mantenibilidad no seleccionadas. • Guardo los cambios. • Se muestra un mensaje temporal indicando que la lista de verificación se ha modificado exitosamente. • La lista de verificación se ha modificada se actualiza del registro de lista de verificación.
3	Eliminar lista de verificación existente	• Como administrador, accedo a la página de administración de listas de verificación. • Selecciono la lista de verificación que quiero eliminar. • El sistema muestra el modal de confirmación de eliminación. • Confirmo la eliminación. • Se muestra un mensaje temporal indicando que la lista de verificación se ha eliminado exitosamente. • La lista de verificación se elimina del registro de lista de verificación disponibles.

Figura 24. Historia de usuario, administrar lista de verificación.

En la **Figura 25** se muestra el escenario de administrar proyecto, como caso de prueba para aplicar la prueba de mantenibilidad.

Historia de usuario			
ID: HU06	Administrar proyecto		
Usuario/Rol	Administrador/usuario		
Prioridad en negocio	Media	Riesgo en desarrollo	Media
Descripción	Como administrador o usuario quiero poder agregar, modificar y eliminar proyectos.		

Nro.	Escenario	Criterio de aceptación
1	Agregar nuevo proyecto	• Como usuario/administrador, presiono en el botón de nuevo proyecto. • Ingreso el nombre y descripción del proyecto. • Y presiono el botón de guardar. • Se muestra un mensaje temporal indicando que se creó el proyecto exitosamente. • El sistema redirecciona a mis proyectos. • El nuevo proyecto se agrega a la lista de mis proyectos.
2	Modificar proyecto existente	• Como usuario/administrador, accedo a la página de mis proyectos. • Selecciono el proyecto que quiero actualizar/modificar. • Modifico los datos que deseo. • Guardo los cambios. • Se muestra un mensaje temporal indicando que el proyecto se ha modificado exitosamente. • El proyecto se actualiza en la lista de mis proyectos.
3	Eliminar proyecto existente	• Como usuario/administrador, accedo a la página de mis proyectos. • Selecciono el proyecto que quiero eliminar. • El sistema muestra el modal de confirmación de eliminación. • Confirmo la eliminación. • Se muestra un mensaje temporal indicando que el proyecto se ha eliminado exitosamente. • El proyecto se elimina de mis proyectos.
4	Lista de proyectos creados	• Como usuario/administrador, accedo a la página de mis proyectos. • Visualizo la lista completa de mis proyectos creados en el sistema con las opciones de ver, editar y eliminar.

Figura 25. Historia de usuario, administrar proyecto.

En la **Figura 26** se muestra el escenario administrar prueba de mantenibilidad, para el establecimiento de la estructura de la lista de verificación.

Historia de usuario			
ID: HU07	Administrar prueba de mantenibilidad		
Usuario/Rol	Administrador/Usuario		
Prioridad en negocio	Media	Riesgo en desarrollo	Media
Descripción	Como administrador o usuario, quiero poder configurar el plan de prueba para verificar la mantenibilidad de aplicaciones web mediante el cumplimiento de pautas de mantenibilidad.		

Nro.	Escenario	Criterio de aceptación
1	Acceder a administración de pruebas de mantenibilidad	• Como administrador/usuario, accedo a la lista de proyectos. • El sistema muestra la opción para direccionar a prueba de mantenibilidad. • Selecciono pruebas y el sistema direcciona la página de administración de pruebas de mantenibilidad.
	Configurar estructura de la lista de verificación para implementar prueba de mantenibilidad	• El software muestra en una pestaña configuración lista de verificación. - Fase de planificación. - Fase de ejecución • En la fase de planificación de muestra cuatro opciones para estructurar la prueba de mantenibilidad. - Subcaracterísticas. - Subcaracterística especifica - Métricas - Métrica especifica. • En la fase de ejecución muestra dos opciones para ejecutar la lista de verificación. - Resolver prueba - Ver pruebas. • Selecciono la estructura de la prueba.
2	Asignar un parámetro de evaluación a una prueba de mantenibilidad existente	• Como administrador, accedo a la página de administración de pruebas de mantenibilidad. • El sistema muestra la opción para direccionar a ver respuestas de las pruebas de mantenibilidad. • Selecciono ver respuestas y el sistema direcciona la página de administración de resultados o respuestas de las pruebas de mantenibilidad. • Selecciono la prueba que se desea asignar un parámetro de evaluación. • Se muestra un mensaje temporal indicando que si desea agregar otra métrica subcaracterística. • Seleccionamos la métrica o subcaracterística y el sistema estructura la prueba.

Figura 26. Historia de usuario, administrar prueba de mantenibilidad.

En la **Figura 27** se muestra el escenario ejecutar prueba de mantenibilidad, que contempla la en responder la lista de verificación.

Historia de usuario			
ID: HU08	Ejecutar prueba de mantenibilidad		
Usuario/Rol	Administrador/usuario,		
Prioridad en negocio	Media	Riesgo en desarrollo	Media
Descripción	Como administrador o usuario, quiero ejecutar la prueba de mantenibilidad en cada uno de los proyectos.		

Nro.	Escenario	Criterio de aceptación
1	Acceder a administración de pruebas de mantenibilidad	• Como usuario/administrador, accedo a la página de administración de pruebas de mantenibilidad. • Selecciono el proyecto que para evaluar. • El software permite administrar la lista de verificación (ID: HU07). • Selecciono ejecutar prueba de mantenibilidad.
2	Responder lista de verificación de mantenibilidad	• Agrego un título a prueba aplicada. • Respondo las pautas de mantenibilidad según las opciones proporcionadas. • Opcional: Registrar comentarios por cada pauta de mantenibilidad • Operacional del software: promedio del cumplimiento de las pautas de mantenibilidad y almacena los resultados. • Selecciono guarda resultados de evaluación. • Se muestra un mensaje temporal indicando que las respuestas se han guardado exitosamente.

Figura 27. Historia de usuario, ejecutar prueba de mantenibilidad.

6.2.1.5. Levantamiento de requerimientos basado el estándar IEEE 830.

En el **Anexo 18** se muestran los requerimientos funciones de la aplicación web a detalle, esto permitió identificar o comprender las necesidades de los estudiantes como parte interesada y docentes como técnicos en control de calidad. La documentación de los requisitos funcionales y no funcionales se desarrolló mediante el estándar IEEE 830, después de la determinación de las historias de usuario.

En la **Tabla 81** se muestra los requisitos funcionales de la aplicación web, esenciales para guiar el desarrollo y asegurar funcionalidades requeridos.

Tabla 81. Requerimientos funcionales de la aplicación web.

Id.	N.	Nombre	Prioridad
01	RF01	Inicio de sesión	Alta
02	RF02	Administrar pauta de mantenibilidad	Media
03	RF03	Administrar lista de verificación	Media
04	RF04	Administrar métrica	Media
05	RF05	Administrar subcaracterística	Media
06	RF06	Administrar proyecto	Media
07	RF07	Administrar prueba de mantenibilidad	Media
08	RF08	Ejecutar prueba de mantenibilidad	Media
09	RF09	Informe lista de verificación	Media
10	RF10	Generar PDF	Media
11	RF11	Calcular grado de mantenibilidad	Media

La **Tabla 82** se muestra los requisitos no funcionales de la aplicación web, esenciales para asegurar la operabilidad de la aplicación web.

Tabla 82. Requerimientos no funcionales de la aplicación web.

Id.	N.	Nombre	Prioridad
01	RNF01	Interfaz Gráfica	Alta
02	RNF02	Accesibilidad	Alta
03	RNF03	Operacional	Alta
04	RNF04	Confidencialidad	Alta

6.2.2. Fase 2: Diseño

6.2.2.1. Diseño de arquitectura de aplicación web.

A continuación, se presentan los diagramas elaborados que tienen como propósito ilustrar el diseño de la arquitectura del software. Estos diagramas detallan la estructura y los componentes clave de la aplicación.

En la **Tabla 83** se muestra la descripción de los modelos implementados.

Tabla 83. Arquitectura de la aplicación web.

Id.	Diseño	Nombre	Prioridad
AS1	Vista de escenarios	Diagrama de casos de uso	Describe la interacción entre los actores y los escenarios donde se va a desenvolver el software.
AS2	Vista lógica	Diagrama de clase	Representa las funcionalidades y el servicio que proporciona a los usuarios.
AS3	Vista despliegue	Diagrama de componentes	Describe los componentes del sistema para que el desarrollador pueda comprender las interacciones que existen entre ellos.
AS4	Vista física	Diagrama de despliegue	Presenta los componentes físicos del sistema.

Arquitectura de software (AS).

a. Diagrama de casos de uso.

El diagrama de casos de uso se compone de dos usuarios: Usuario y Administrador, los cuales realizan diferentes acciones de acuerdo a la actividad que desempeñen.

En la **Figura 28** se muestra gráficamente el diagrama de casos de uso para la autenticación de usuarios en el software, utilizando el servicio de Aerobase IAM, por lo tanto, únicamente se permite el inicio de sesión.

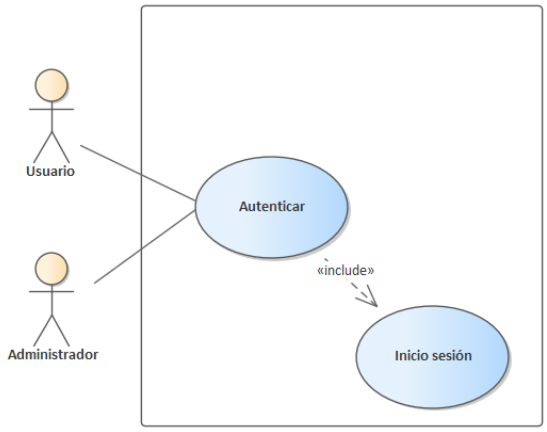


Figura 28. Diagrama de caso de uso, autenticar usuarios.

En la **Figura 29** se presenta gráficamente el diagrama de casos de uso general, en donde se muestra los diferentes tipos de escenarios de operación de la aplicación web y su asignación a cada tipo de usuario. Se percibe un total de 10 escenarios para dar cumplimiento a las necesidades del usuario final.

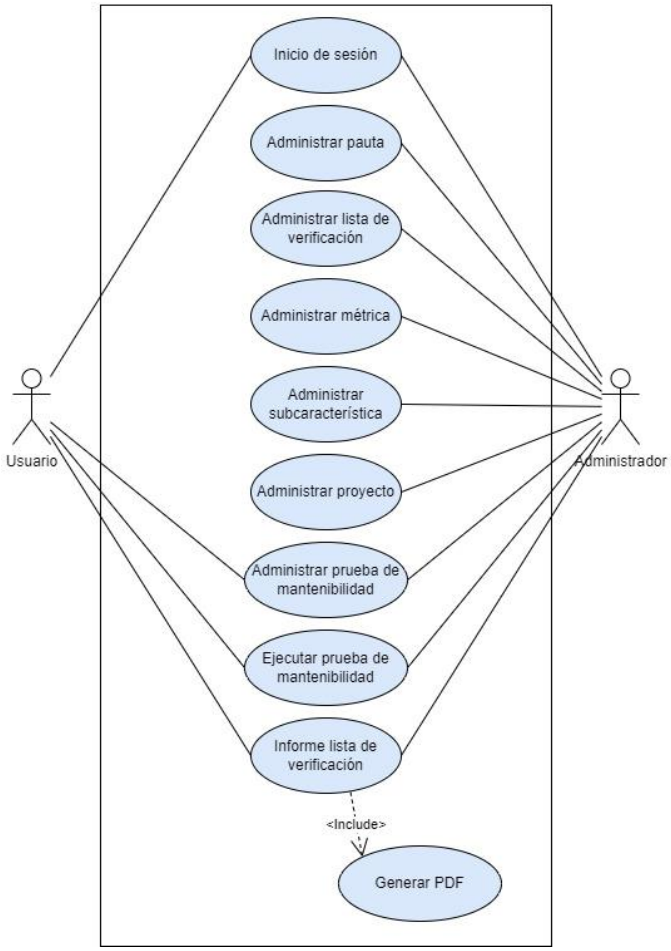


Figura 29. Diagrama de caso de uso general.

b. Diagrama de clases.

La **Figura 30** muestra el diagrama de clases de la aplicación web, representando la estructura del software mediante las clases y sus respectivas relaciones.

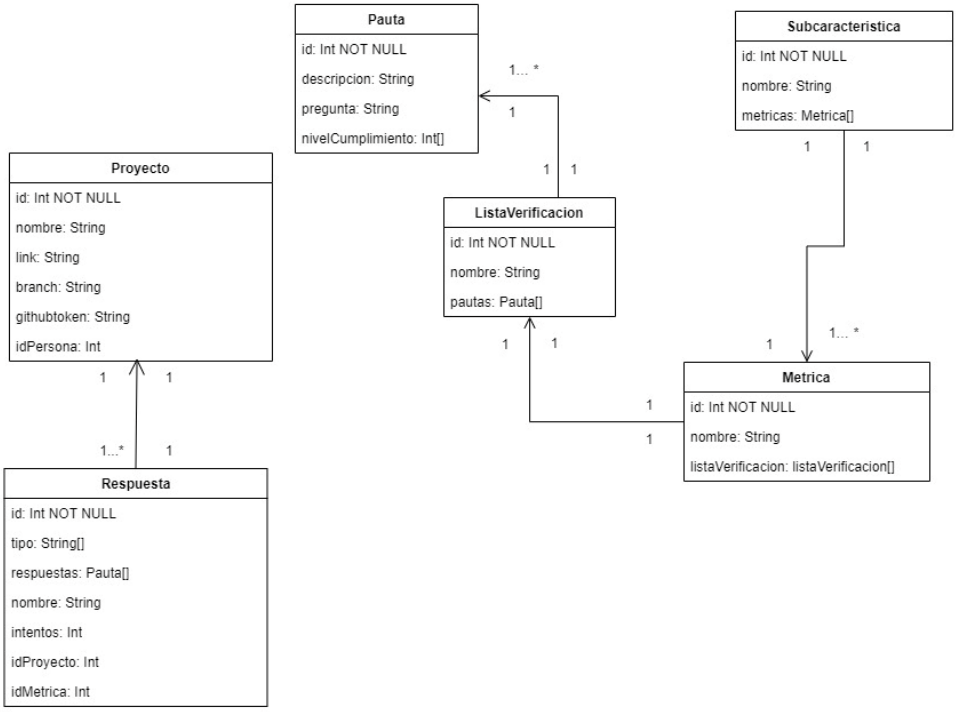


Figura 30. Diagrama de clases.

c. Diagrama de componentes.

La **Figura 31** presenta el diagrama de componentes de la aplicación web, destaca de manera clara y sencilla los elementos clave involucrados en la construcción del sistema. Este diagrama no solo identifica los componentes relevantes, sino que también ilustra las relaciones y dependencias entre ellos, facilitando así una comprensión integral de la arquitectura del sistema.

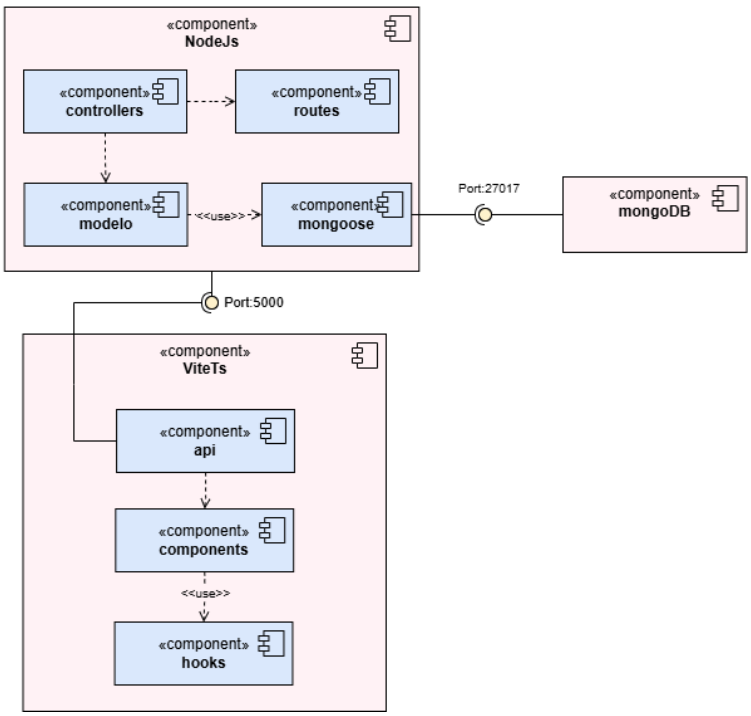


Figura 31. Diagrama de componentes.

d. Diagrama de despliegue.

La **Figura 32** muestra el diagrama de despliegue, que ilustra los componentes clave del sistema y sus relaciones. El diagrama es fundamental para la planificación del despliegue de la aplicación, ya que muestra cómo se organizan y conectan los diferentes elementos en el entorno de producción.

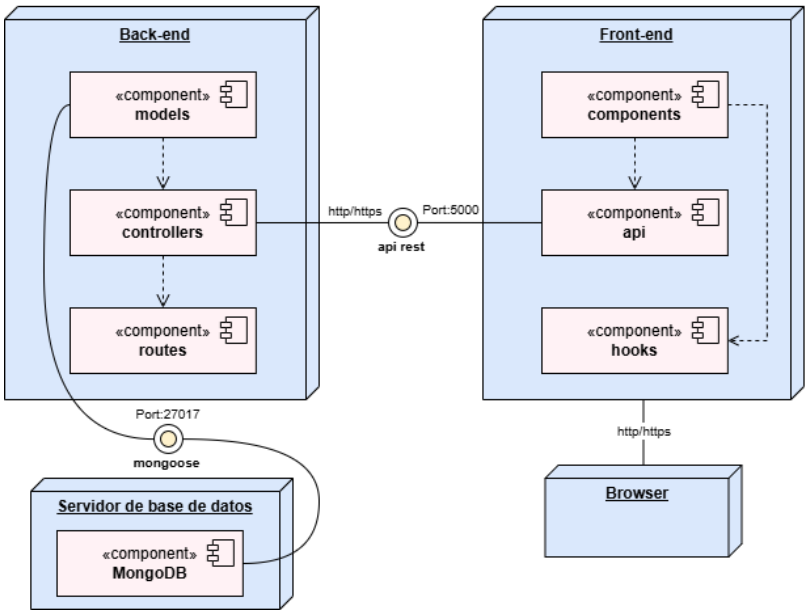


Figura 32. Diagrama de despliegue.

6.2.3. Fase 3: Codificación.

Para el proceso de codificación de la aplicación web, se realizó una planificación detallada de iteraciones, las cuales se denominan **tareas**. Esta planificación permitió organizar el trabajo de manera eficiente y asegurar que cada aspecto del desarrollo fuera abordado adecuadamente. En los siguientes puntos se encuentra los enlaces de los repositorios de GitHub, en donde se encuentra el código fuente del sistema:

- **Backend**: https://github.com/gilsonOrlando/backend-qa.
- **Frontend**: https://github.com/gilsonOrlando/frontend-qa.

6.2.3.1. Planificación de tareas previo al desarrollo.

Se consideraron las historias de usuarios descritas en el **Anexo 17** para definir las funcionalidades que se codificarán durante el proceso de desarrollo de la aplicación web, este enfoque asegura que las características implementadas estén alineadas con las necesidades y expectativas de los usuarios finales.

La **Tabla 84** se muestra la lista de tareas descritas, incluyendo la referencia a la historia de usuario.

Tabla 84. Planificación de tareas.

Tarea	Identificador	RHU	FIC	FFC
1	Inicio de sesión	HU01	18/06/2024	21/06/2024
2	Administrar pauta	HU02	18/06/2024	21/06/2024
3	Administrar lista de verificación	HU03	18/06/2024	21/06/2024
4	Administrar métrica	HU04	18/06/2024	21/06/2024
5	Administrar subcaracterística	HU05	25/06/2024	28/06/2024
6	Administrar proyecto	HU06	25/06/2024	28/06/2024
7	Administrar prueba de mantenibilidad	HU07	01/07/2024	05/07/2024
8	Ejecutar prueba de mantenibilidad	HU08	08/07/2024	12/07/2024
9	Informe lista de verificación	HU09	08/07/2024	12/07/2024
10	Generar PDF	HU010	15/07/2024	19/07/2024
11	Calcular grado de mantenibilidad	HU011	15/07/2024	19/07/2024

Referencia de Historia de Usuario (RHU); Fecha de inicio del desarrollo (FID); Fecha fin del desarrollo (FFD).

6.2.3.2. Codificación de tareas y pruebas unitarias de la aplicación web.

En esta sección se presenta aspectos relevantes para la fase de codificación de la aplicación web, en la cual se utilizó Node Js y Express para la implementación de backend,

Vite Js y Tailwind CSS para el frontend y la configuración para la conexión de la base de datos MongoDB Atlas. En este contexto, se presentan secciones importantes del desarrollo del código.

En la **Figura 33** se destaca la estructura general del frontend de la aplicación web, destacando la separación por archivos para la configuración, componentes, estilos, páginas y otras configuraciones de importancia.

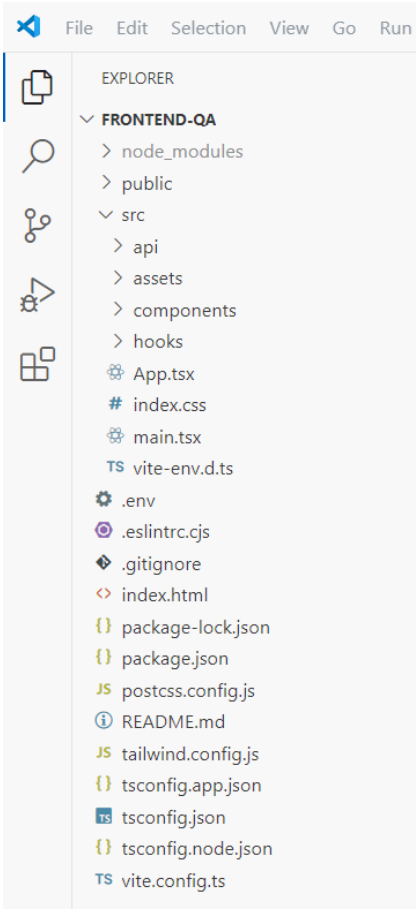


Figura 33. Estructura del frontend de la aplicación web.

En la **Figura 34** se destaca la estructura general del backend de la aplicación web, mostrando la carpeta src que contiene la configuración de los modelos de clase, la carpeta routes con los archivos de configuración de rutas, validaciones etc. y además de otros archivos de importancia.

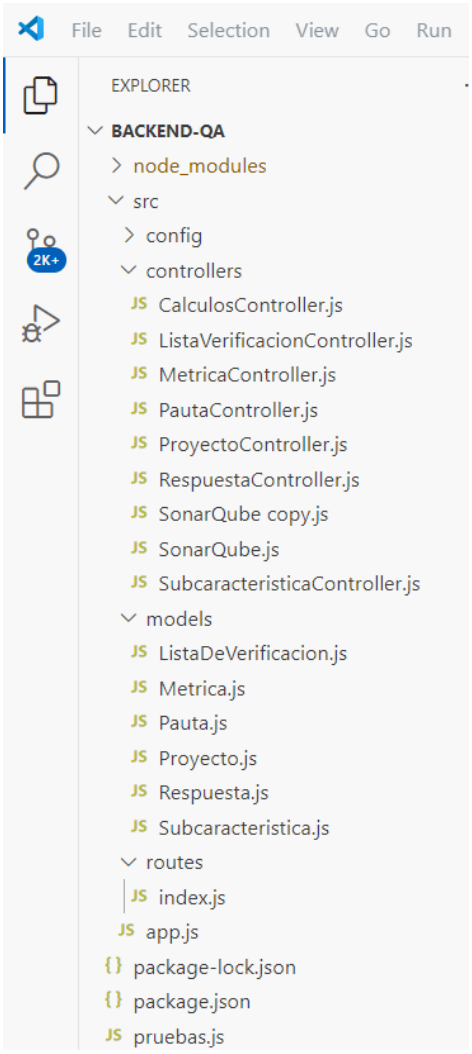


Figura 34. Estructura del backend de la aplicación web.

La **Figura 35** se presenta la especificación de las rutas implementadas en el backend, cada ruta cumple una función que es accedida desde el frontend de la aplicación web a través de peticiones POST, GET, PUT o DELETE.

index.js

src > routes > index.js > ...

```
1  const express = require('express');
2  const router = express.Router();
3
4  const PautaController = require('../controllers/PautaController');
5  const ListaVerificacionController = require('../controllers/ListaVerificacionController');
6  const MetricaController = require('../controllers/MetricaController');
7  const SubcaracteristicaController = require('../controllers/SubcaracteristicaController');
8  const ProyectoController = require('../controllers/ProyectoController');
9  const RespuestaController = require('../controllers/RespuestaController');
10 const calculosController = require('../controllers/CalculosController')
11 const SonarQubeController = require('../controllers/SonarQube');
12
13 router.use('/pautas', PautaController);
14 router.use('/listasVerificacion', ListaVerificacionController);
15 router.use('/metricas', MetricaController);
16 router.use('/subcaracteristicas', SubcaracteristicaController);
17 router.use('/proyectos', ProyectoController);
18 router.use('/respuestas', RespuestaController);
19 router.use('/calculos', calculosController)
20 router.use('/sonarqube', SonarQubeController);
21
22 module.exports = router;
```

Figura 35. Rutas de acceso desde frontend al backend.

A continuación, se presente un resumen de la implementación de las tareas planificadas previamente para el proceso de codificación de la aplicación web, las cuales están documentadas en el **Anexo 23**. Las tareas se realizaron con el propósito de servir como base para la ejecución de pruebas unitarias de las funciones desarrolladas, asegurando que cada componente del sistema funcione correctamente y cumpla con los requisitos establecidos.

En el cumplimiento de las tareas se planificaron las siguientes actividades:

- **Diseño**: Se planifico el modelamiento de interfaces.
- **Codificación**: Se llevó a cabo el desarrollo de modelos y controladores.
- **Prueba**, Se aplicó pruebas unitarias de acuerdo a los escenarios de las historias de usuario, para ello se utilizó la herramienta Postman.

- **Tarea 1: Inicio de sesión**

  - **Diseño**

En la **Figura 36** se muestra el diseño para el inicio de sesión, que brinda el autenticador centralizado de la Carrera de Computación.

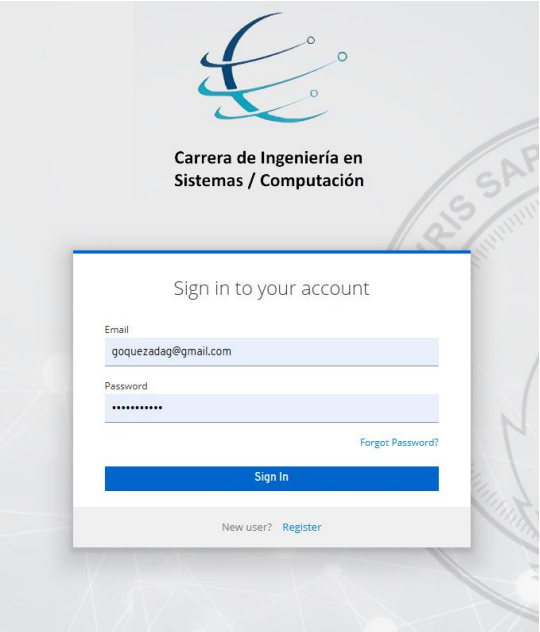


Figura 36. Diseño de inicio de sesión.

- **Codificación**

En la **Figura 37** se muestra la iniciación de las funciones para administrar el inicio de sesión utilizando keycloak, la codificación del inicio de sesión no se muestra por cuestiones de privacidad. En este caso de muestra configuración de variables del cliente y el estado de autenticación del usuario mediante el hook llamado useAuth y también la función de logout para cerra la sesión y restablecer el estado.

useAuth.tsx

src > hooks > useAuth.tsx > ...

```tsx
import { useState, useEffect, useRef, useCallback } from "react";
import KeycloakConfig from "keycloak-js";

// Configuración del cliente de Keycloak
const client = new KeycloakConfig({
    url: import.meta.env.VITE_KEYCLOAK_URL,
    realm: import.meta.env.VITE_KEYCLOAK_REALM,
    clientId: import.meta.env.VITE_KEYCLOAK_CLIENT,
});

// Tipos para el estado de autenticación
interface AuthData {
    isLogin: boolean;
    userId: string | null;
    roles: string[];
}

const useAuth = () => {
    const isGo = useRef(false);
    const [authData, setAuthData] = useState<AuthData>({
        isLogin: false,
        userId: null,
        roles: []
    });
```

Figura 37. Iniciación de las funciones para administrar el inicio de sesión utilizando keycloak.

- **Pruebas**

Para el inicio de sesión se registró un caso de prueba unitaria, como se detalla en la **Tabla 85**, la misma que tiene su respectiva la ejecución.

Tabla 85. Prueba unitaria inicio de sesión PU-01**.**

Prueba Unitaria – PU-01			
Número: PU-01		**Versión**: 1.0	
Componente evaluado: Caso de uso: Inicio de sesión			
Identificador: Inicio de sesión		**Historia de usuario**: HU01	
Caso de prueba: Verificación del flujo de inicio de sesión mediante Keycloak			
Datos de entrada	**Descripción**	**Salida esperada**	**Resultado**
Body(x-www-form-urlencoded) grant_type: password client_id: myappclient username: {} password: {]	Verificar si el usuario puede iniciar sesión correctamente.	El servidor debe devolver un código de estado 200 con un Access_token en la respuesta.	Aceptado

Referencia de historia de usuario (RHU).

En la **Figura 38** se muestra el objeto json de prueba y la API para verificar el inicio de sesión.

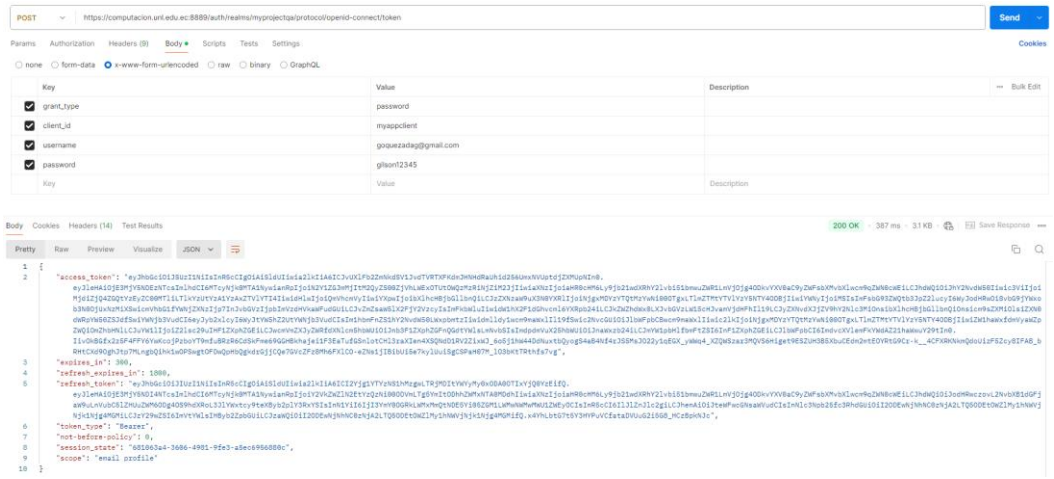


Figura 38. Objeto json de prueba y la API para verificar el inicio de sesión.

- **Tarea 2: Administrar pauta**

  - **Diseño**

En la **Figura 39** se muestra el diseño para registrar una nueva pauta de mantenibilidad, la codificación del mismo se encuentra en el repositorio git mencionado al inicio de la presente **sección**.

Figura 39. Diseño, crear pauta de mantenibilidad.

En la **Figura 40** se muestra el diseño para actualizar y eliminar una nueva pauta de mantenibilidad.

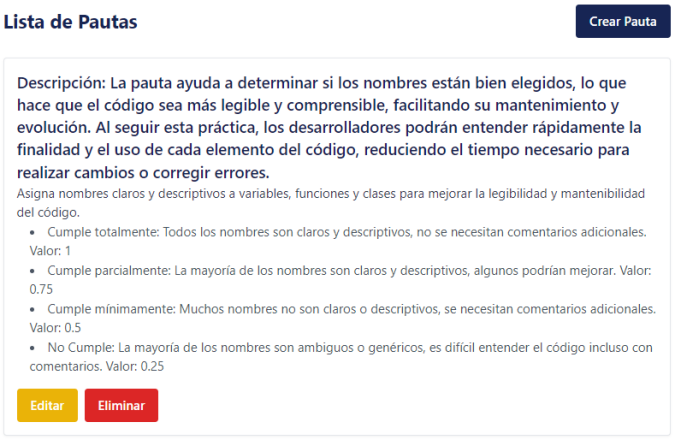


Figura 40. Diseño, actualizar y eliminar una pauta de mantenibilidad.

Los casos de pruebas están relacionados a cada uno de los diseños **Figura 39** y **40**, cuyo objetivo es verificar el modelo para crear y llevar a cabo modificación, eliminación y obtener datos.

- **Codificación**

En la **Figura 41** se destaca las funciones para registrar una nueva pauta, obtener los registros de todas las pautas y en la **Figura 42** las funciones para permitir la actualización de los datos existentes de una pauta, así como su eliminación.

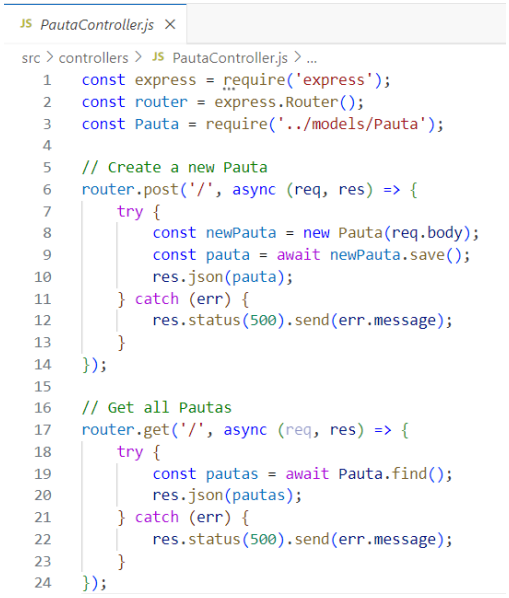


Figura 41. Código para crear y obtener las pautas de mantenibilidad.

Figura 42. Código para actualizar y elimina una pauta de mantenibilidad.

- **Pruebas**

Para las pruebas unitarias se registró dos casos de prueba, presentando la ejecución de manera más detallada.

La **Tabla 86** se detalla el caso de prueba para verificar el modelo pauta, mediante el escenario crear pauta.

Tabla 86. Prueba unitaria administrar pauta de mantenibilidad- PU-02.

Prueba Unitaria – PU-02			
Número: PU-02		**Versión**: 1.0	
Componente evaluado: Caso de uso: Administrar pauta			
Identificador: Administrar pauta		**Historia de usuario**: HU02	
Caso de prueba: Verificación del modelo Pauta			
Datos de entrada	**Descripción**	**Salida esperada**	**Resultado**
Json pauta	Verificar si el modelo Pauta se crea correctamente	Las instancias del modelo Pauta deben crearse correctamente y verificar el estado del registro.	Aceptado

Referencia de historia de usuario (RHU).

En la **Figura 43** se muestra la API y el objeto json de prueba para crear el modelo pauta.

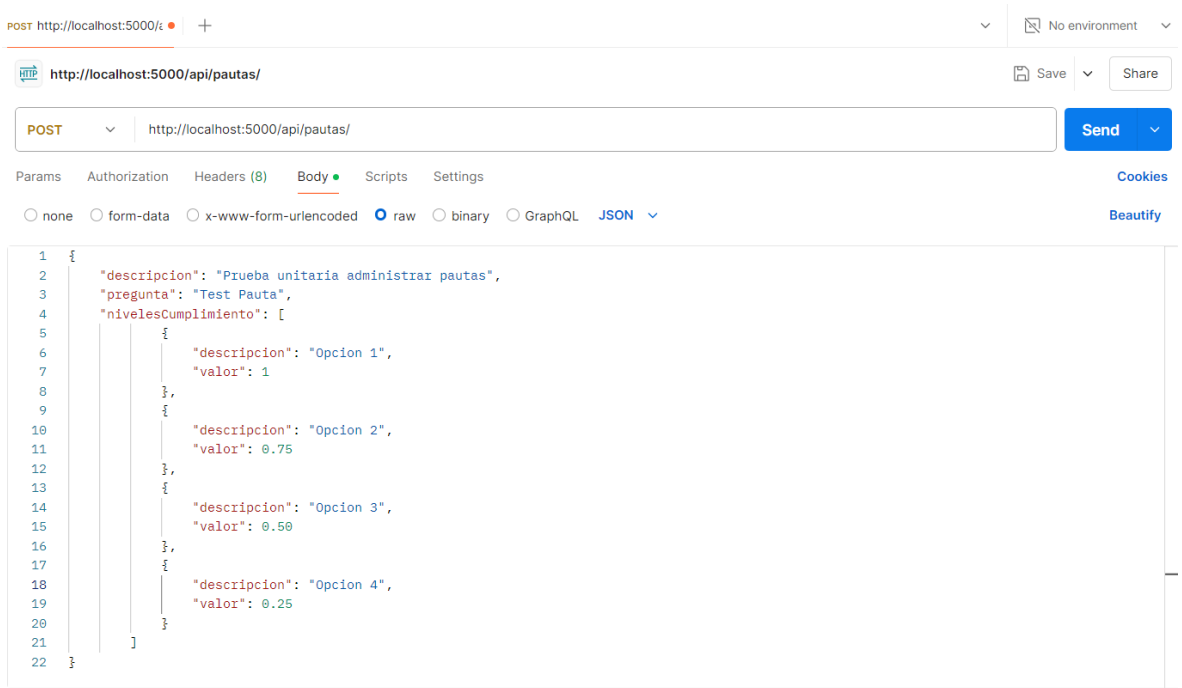


Figura 43. Api y objeto json Pauta para crear un modelo pauta - PU-02.

En la **Figura 44** se muestra la verificación del modelo pauta, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

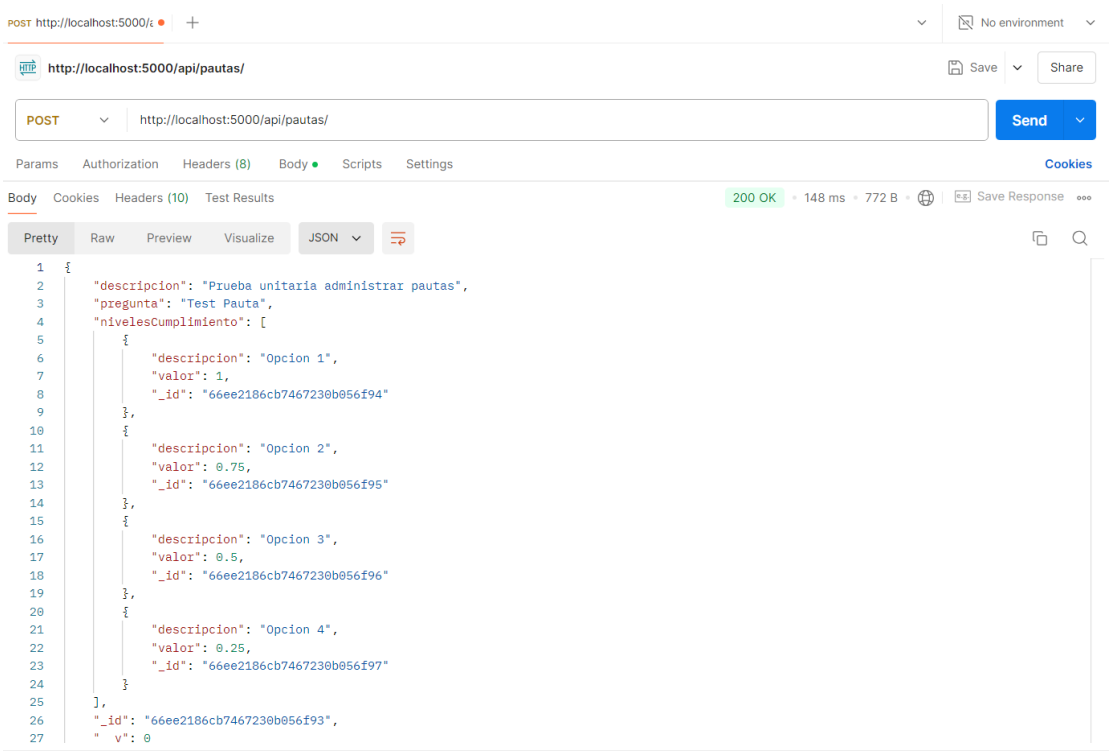


Figura 44. Verificación del modelo pauta - PU-02.

En la **Figura 45** se muestra la pauta de mantenibilidad registrada, desde la ejecución del diseño del frontend.

Figura 45. Verificación del registro pauta - PU-02.

La **Tabla 87** se detalla el caso de prueba del modelo para llevar a cabo modificaciones, eliminaciones y obtener datos.

Tabla 87. Prueba unitaria administrar pauta de mantenibilidad- PU-03.

Prueba Unitaria – PU-03			
Número: PU-03		**Versión**: 1.0	
Componente evaluado: Caso de uso: Administrar pauta			
Identificador: Administrar pauta		**Historia de usuario**: HU02	
Caso de prueba: Modificación, eliminación y obtención de datos			
Datos de entrada	**Descripción**	**Salida esperada**	**Resultado**
Modelo pauta	Verificar si se efectúan los métodos de modificación, eliminación y listar los datos de los registros del modelo de pauta	Obtener los datos de cada uno de los registros para tener la opción de eliminarlos o modificarlos	Aceptado

Referencia de historia de usuario (RHU).

En la **Figura 46** se muestra la API para listar los datos de los registros del modelo pauta.

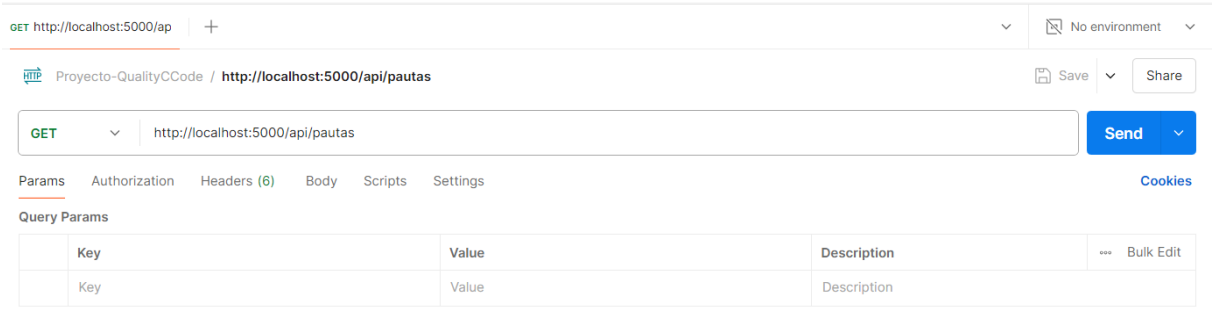


Figura 46. Api para listar los datos de los registros del modelo pauta - PU-03.

En la **Figura 47** se muestra la verificación de listar los registros de pautas, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

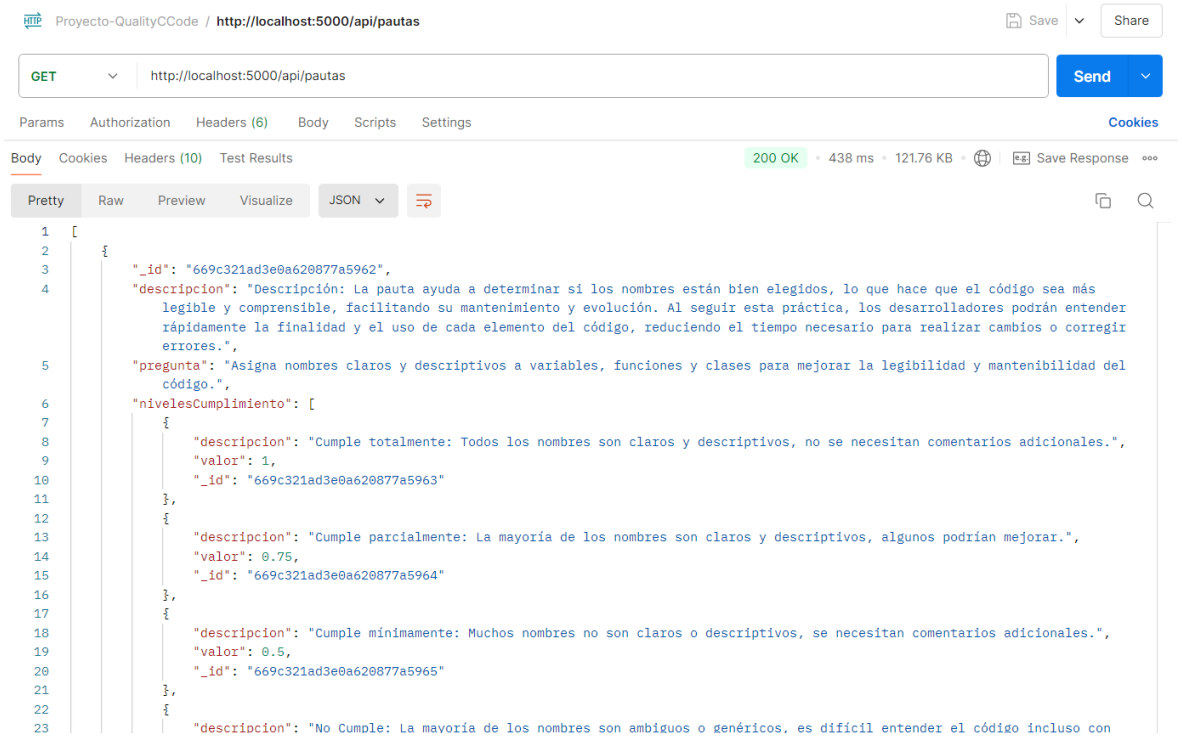


Figura 47. Verificación de listar los registros de pautas - PU-03.

En la **Figura 48** se muestra la API en la cual establece la variable Id para modificar los datos y eliminar un registro del modelo pauta.

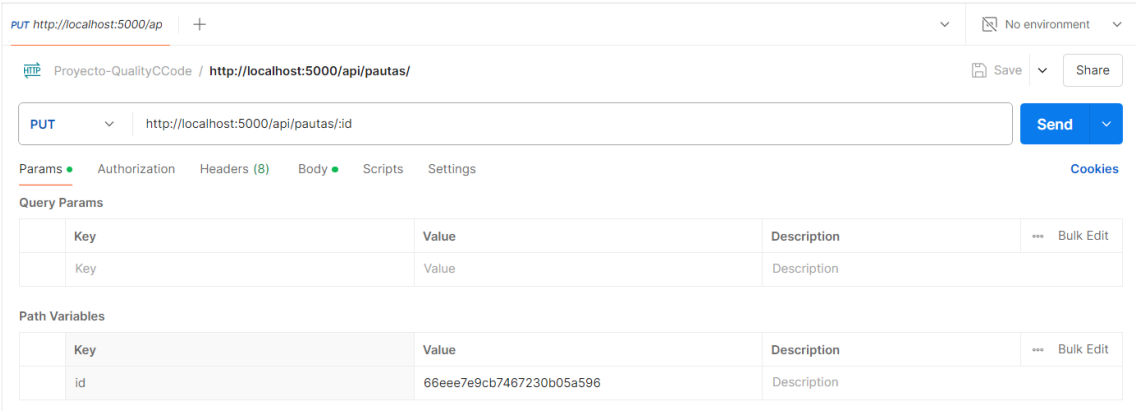


Figura 48. Variable Id para modificar y eliminar un registro del modelo pauta - PU-03.

En la **Figura 49** se muestra la API y el objeto Json para modificar los datos de un registro modelo pauta.

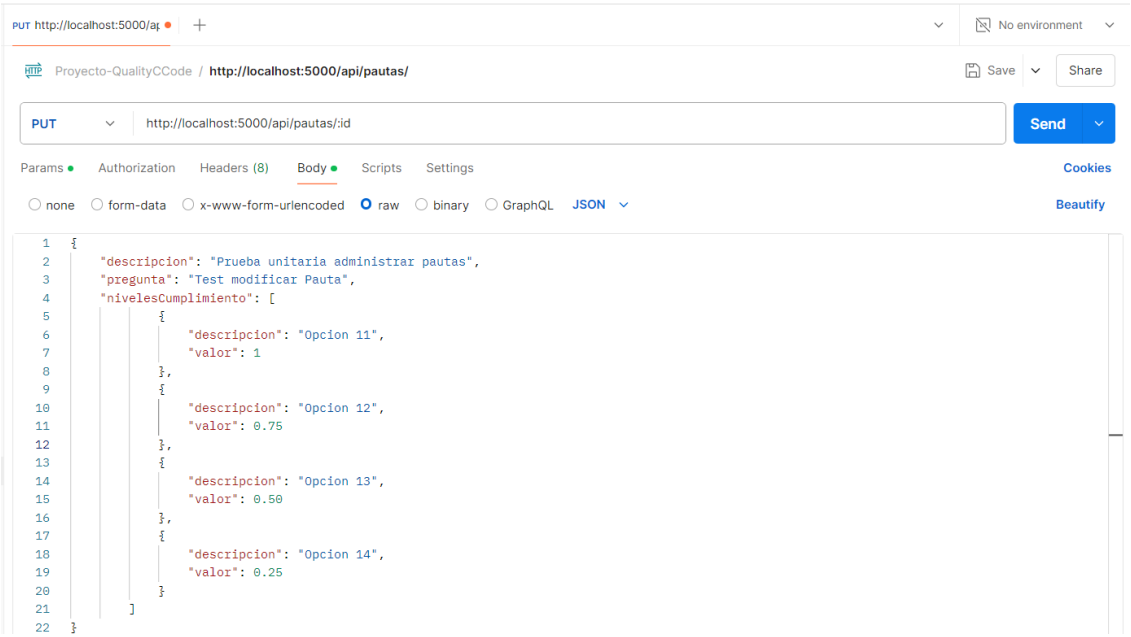


Figura 49. API y el objeto Json para modificar los datos de un registro del modelo pauta - PU-03.

En la **Figura 50** se muestra la verificación de modificar un registro pauta, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

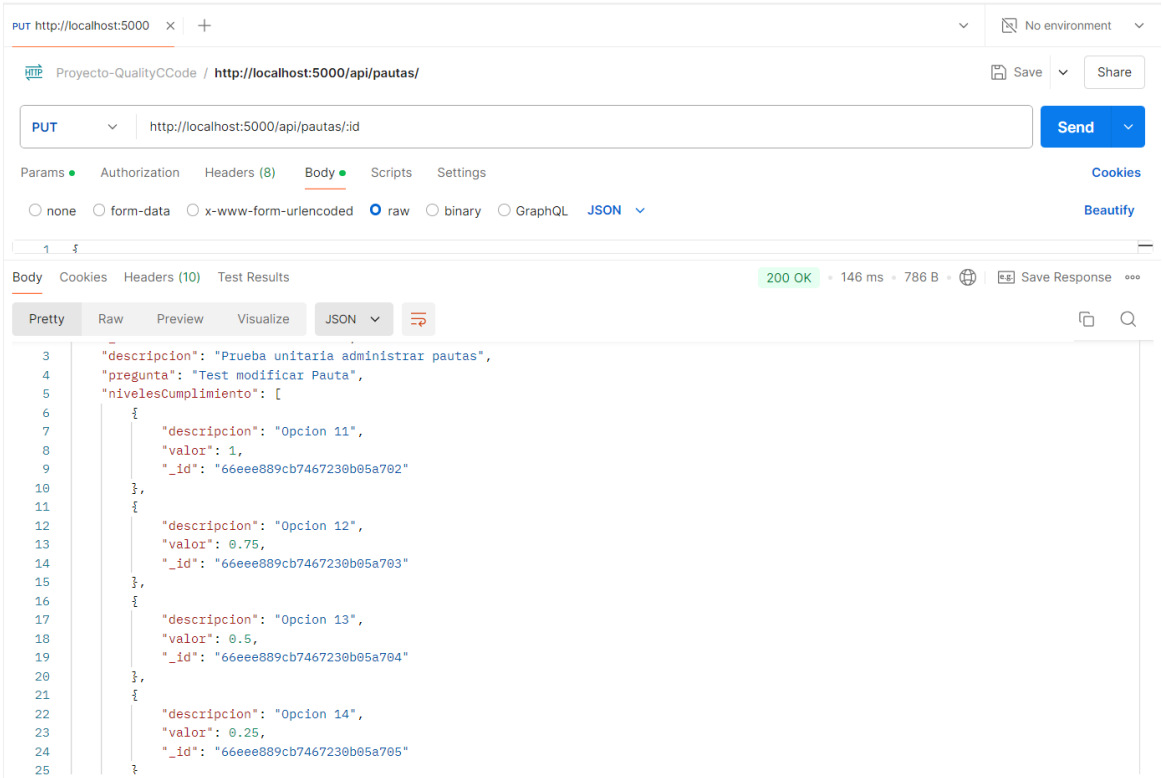


Figura 50. Verificación de modificar un registro del modelo pauta - PU-03.

En la **Figura 51** se muestra la pauta de mantenibilidad registrada en la PU-02, en ejecución del diseño del frontend se ha actualizado correctamente.

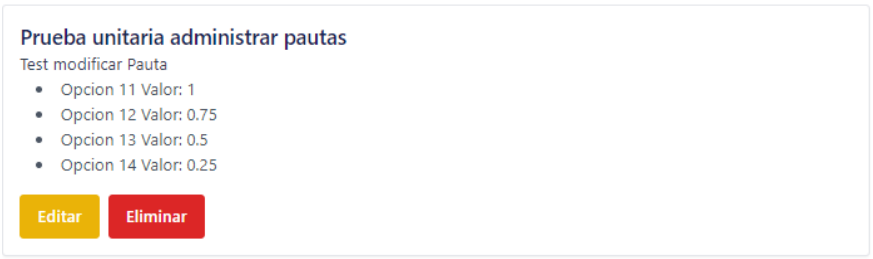


Figura 51. Verificación de la modificación del registro pauta - PU-03.

En la **Figura 52** se muestra la API para eliminar un registro modelo pauta.

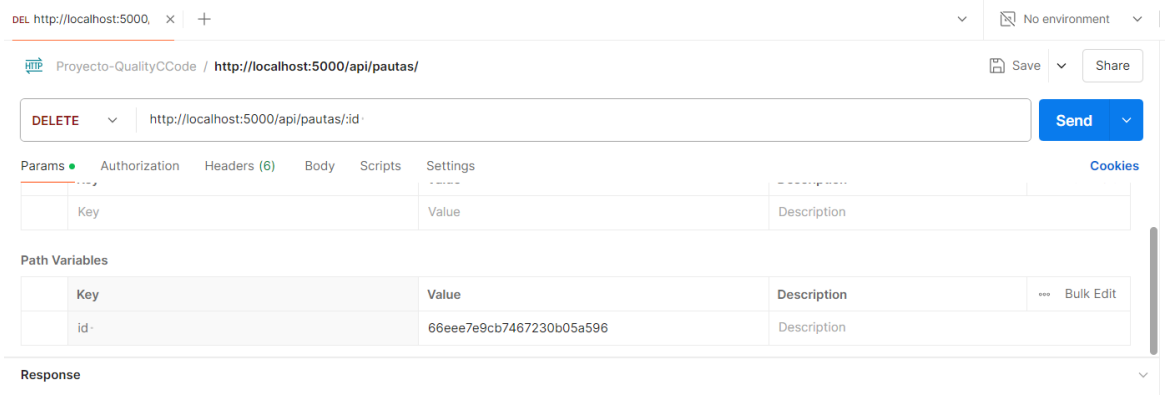


Figura 52. API para eliminar un registro del modelo pauta - PU-03.

En la **Figura 53** se muestra la verificación de eliminar un registro pauta, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

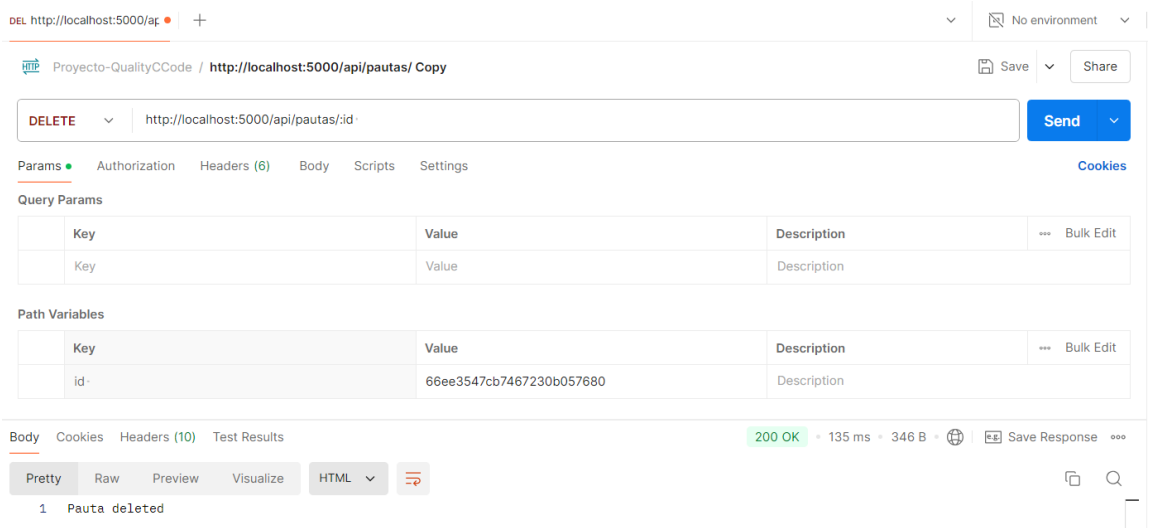


Figura 53. Verificación de eliminar un registro del modelo pauta - PU-03.

- **Tarea 3: Administrar lista de verificación**

  - **Diseño**

En la **Figura 54** se muestra el diseño para registrar una nueva lista de verificación, la codificación del mismo se encuentra en el repositorio git mencionado al inicio de la presente **sección.**

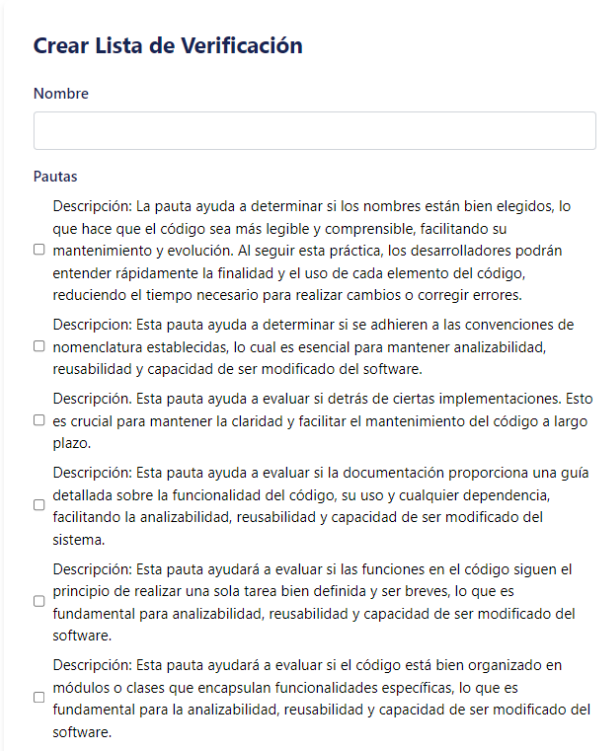


Figura 54. Diseño, crear lista de verificación.

En la **Figura 55** se muestra el diseño para actualizar y eliminar una nueva lista de verificación.

Figura 55. Diseño, actualizar y eliminar una lista de verificación.

Los casos de prueba están relacionados a cada uno de los diseños **Figura 54 y 55**, cuyo objetivo es verificar el modelo para crear y llevar a cabo modificación, eliminación y obtener datos.

- **Codificación**

En la **Figura 56** se destaca las funciones para registrar una nueva lista de verificación, obtener los registros de todas las listas de verificación y en la **Figura 57** las funciones para permitir la actualización de los datos existentes de una lista, así como su eliminación.

```
JS ListaVerificacionController.js ×
src > controllers > JS ListaVerificacionController.js > router.post('/') callback
1  const express = require('express');
2  const router = express.Router();
3  const ListaVerificacion = require('../models/ListaDeVerificacion');
4  const mongoose = require('mongoose');
5
6  // Create a new ListaVerificacion
7  router.post('/', async (req, res) => {
8      try {
9          const newLista = new ListaVerificacion(req.body);
10         const lista = await newLista.save();
11         res.json(lista);
12     } catch (err) {
13         res.status(500).send(err.message);
14     }
15 });
16
17 // Get all ListasVerificacion
18 router.get('/', async (req, res) => {
19     try {
20         const listas = await ListaVerificacion.find().populate('pautas');
21         res.json(listas);
22     } catch (err) {
23         res.status(500).send(err.message);
24     }
25 });
```

Figura 56. Código para crear y obtener las listas de verificación.

```
73 // Update a ListaVerificacion by ID
74 router.put('/:id', async (req, res) => {
75     try {
76         const lista = await ListaVerificacion.findByIdAndUpdate(req.params.id, req.body, { new: true }).populate('pautas');
77         if (!lista) {
78             return res.status(404).send('ListaVerificacion not found');
79         }
80         res.json(lista);
81     } catch (err) {
82         res.status(500).send(err.message);
83     }
84 });
85
86 // Delete a ListaVerificacion by ID
87 router.delete('/:id', async (req, res) => {
88     try {
89         const lista = await ListaVerificacion.findByIdAndDelete(req.params.id);
90         if (!lista) {
91             return res.status(404).send('ListaVerificacion not found');
92         }
93         res.send('ListaVerificacion deleted');
94     } catch (err) {
95         res.status(500).send(err.message);
96     }
97 });
```

Figura 57. Código para actualizar y elimina una lista de verificación.

- **Pruebas**

Para las pruebas unitarias se registró dos casos de prueba con la respectiva ejecución de manera detallada.

La **Tabla 88** se detalla el caso de prueba para verificar el modelo lista de verificación, mediante el escenario crear lista de verificación.

Tabla 88. Prueba unitaria administrar lista de verificación- PU-04.

Prueba Unitaria – PU-04			
Número: PU-04		**Versión**: 1.0	
Componente evaluado: Caso de uso: Administrar lista de verificación			
Identificador: Administrar lista de verificación		**Historia de usuario**: HU03	
Caso de prueba: Verificación del modelo Lista de verificación			
Datos de entrada	**Descripción**	**Salida esperada**	**Resultado**
Json lista de verificación	Verificar si el modelo Lista de verificación se crea correctamente	Las instancias del modelo lista de verificación deben crearse correctamente y verificar el estado del registro.	Aceptado

Referencia de historia de usuario (RHU).

En la **Figura 58** se muestra la API y el objeto json de prueba para crear el modelo lista de verificación, al registrar una lista es necesario agregar una pauta de mantenibilidad para ello se hace uso el modelo pauta implementado en la PU-02.

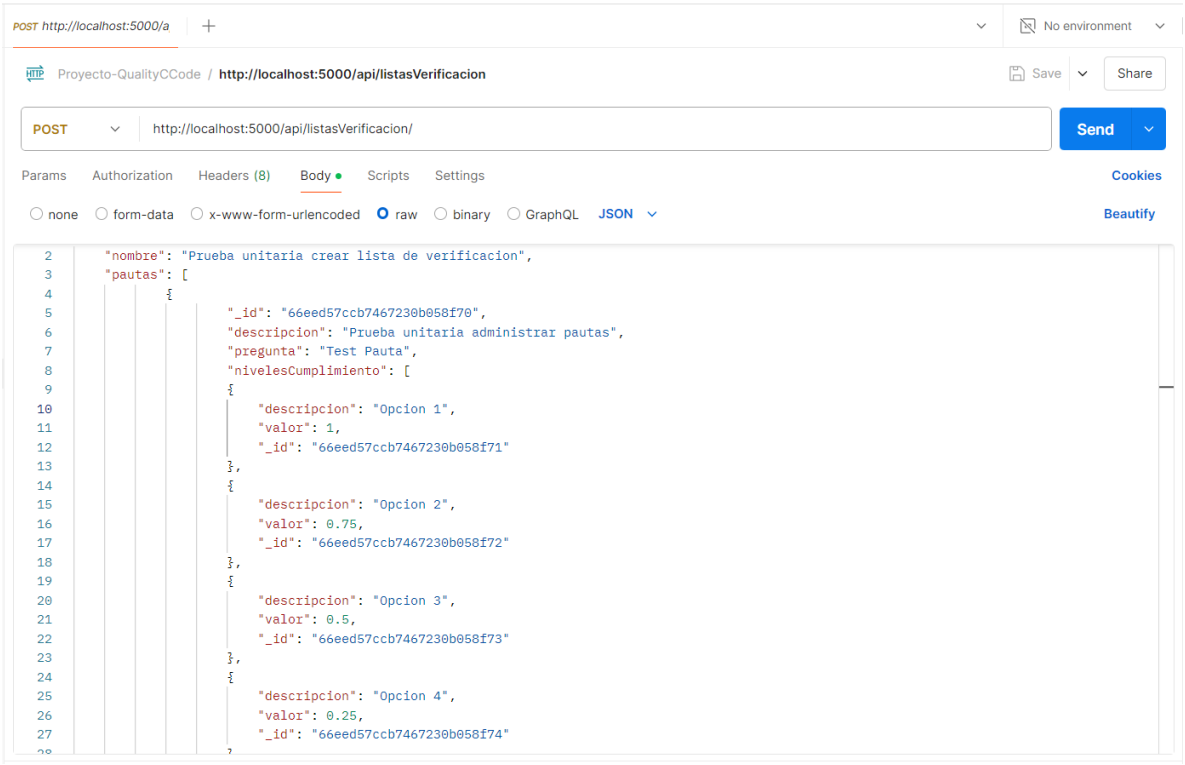


Figura 58. Api y objeto json Pauta para crear un modelo lista de verificación - PU-04.

En la **Figura 59** se muestra la verificación del modelo lista de verificación, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

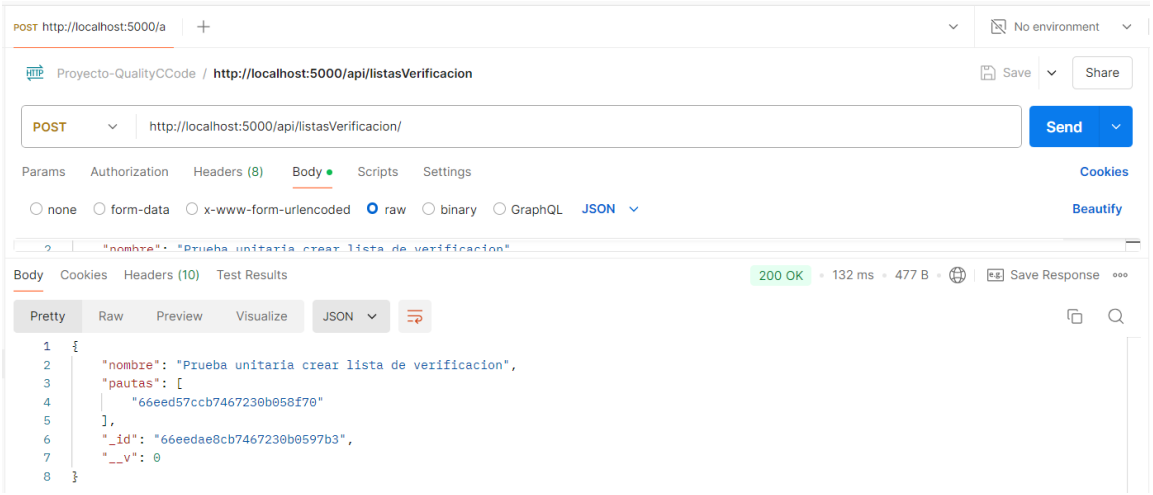


Figura 59. Verificación del modelo lista de verificación - PU-04.

En la **Figura 60** se muestra la lista de verificación, mediante la ejecución del diseño del frontend.

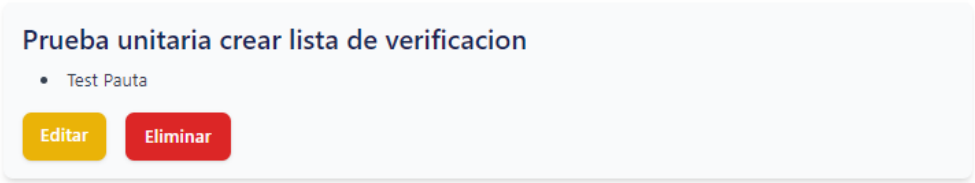


Figura 60. Verificación del registro lista de verificación - PU-04.

La **Tabla 89** se detalla el caso de prueba del modelo para llevar a cabo modificaciones, eliminaciones y obtener datos.

Tabla 89. Prueba unitaria administración lista de verificación- PU-05.

Prueba Unitaria – PU-05			
Número: PU-05		**Versión**: 1.0	
Componente evaluado: Caso de uso: Administrar lista de verificación			
Identificador: Administrar lista de verificación		**Historia de usuario**: HU03	
Caso de prueba: Modificación, eliminación y obtención de datos			
Datos de entrada	**Descripción**	**Salida esperada**	**Resultado**
Modelo pauta. Modelo lista de verificación.	Verificar si se efectúan los métodos de modificación, eliminación y listar los datos de los registros del modelo de pauta	Obtener los datos de cada uno de los registros para tener la opción de eliminarlos o modificarlos	Aceptado

Referencia de historia de usuario (RHU).

En la **Figura 61** se muestra la API para listar los datos de los registros del modelo lista de verificación.

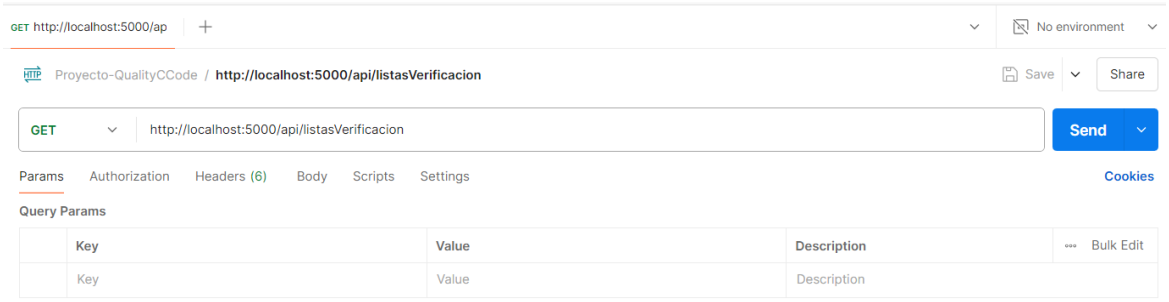


Figura 61. Api para listar los datos de los registros del modelo pauta - PU-05.

En la **Figura 62** se muestra la verificación de listar los registros de las listas de verificación, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

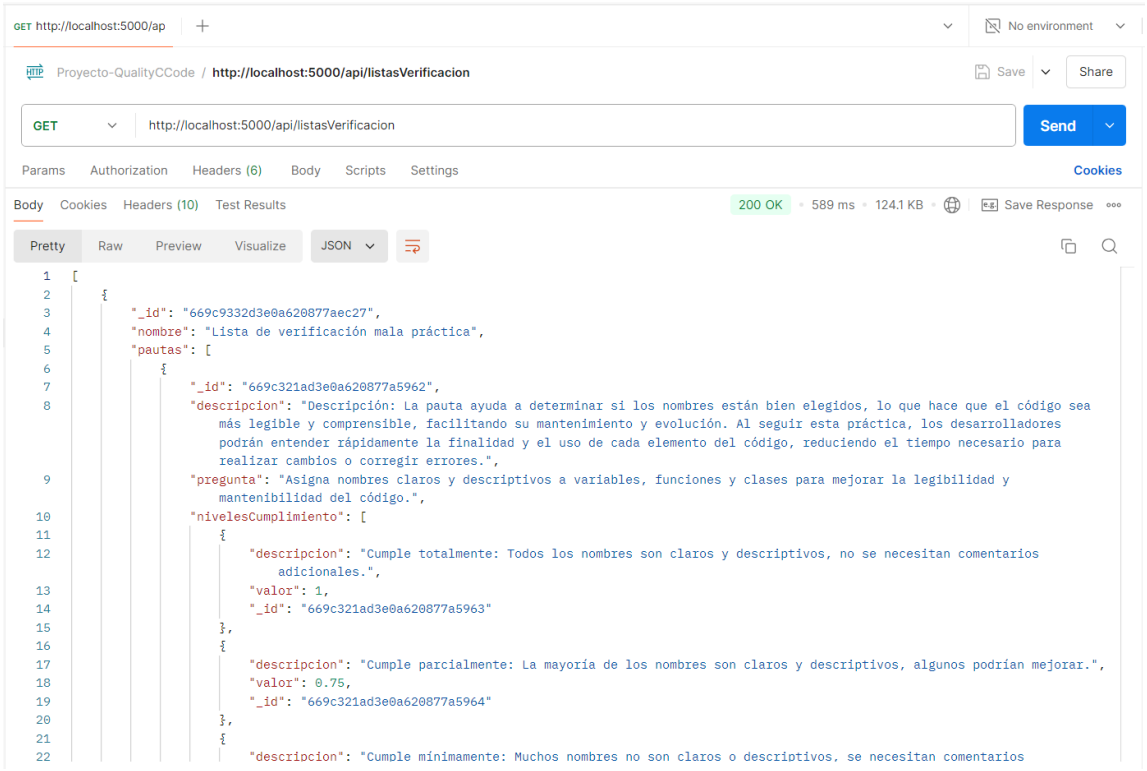


Figura 62. Verificación de listar los registros de pautas - PU-05.

En la **Figura 63** se muestra la API en la cual establece la variable Id para modificar los datos y eliminar un registro del modelo lista de verificación.

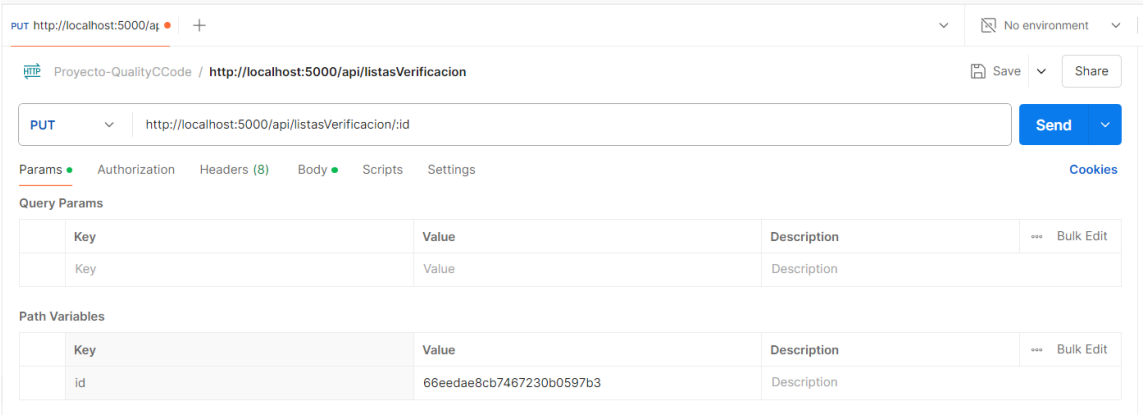


Figura 63. Variable Id para modificar y eliminar un registro del modelo lista de verificación - PU-05.

En la **Figura 64** se muestra la API y el objeto Json para modificar los datos de un registro modelo lista de verificación.

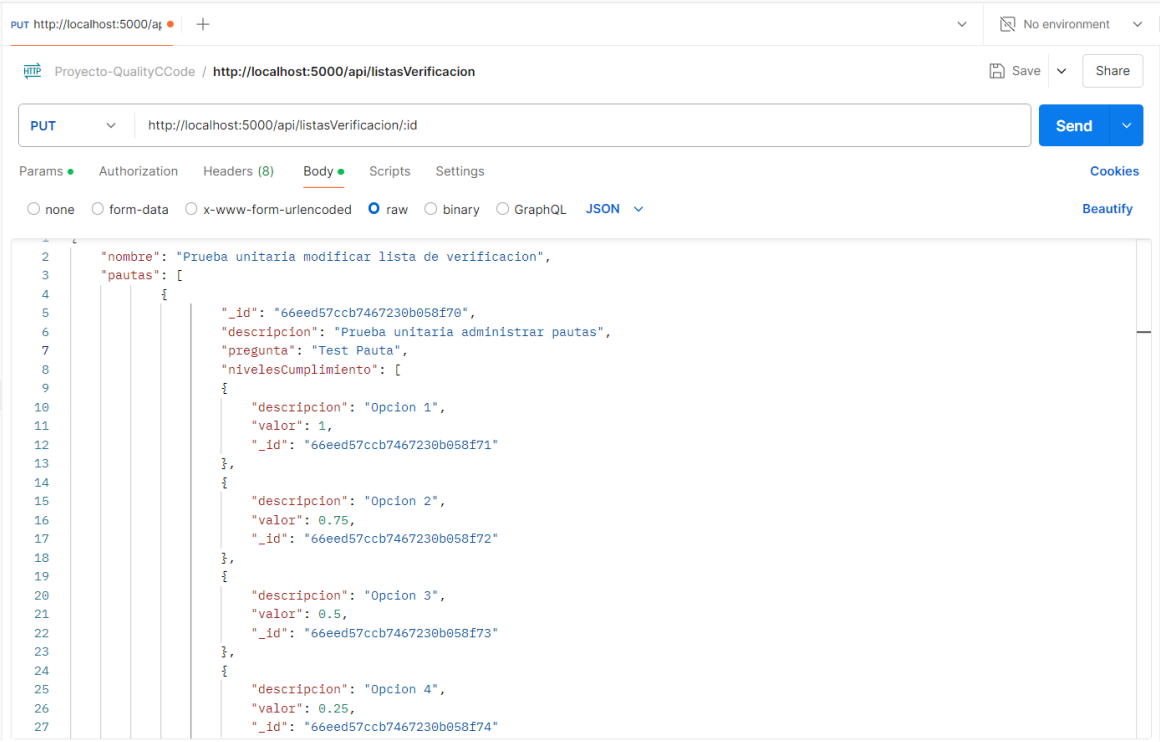


Figura 64. API y el objeto Json para modificar la modelo lista de verificación - PU-05.

En la **Figura 65** se muestra la verificación de modificar un registro lista de verificación, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

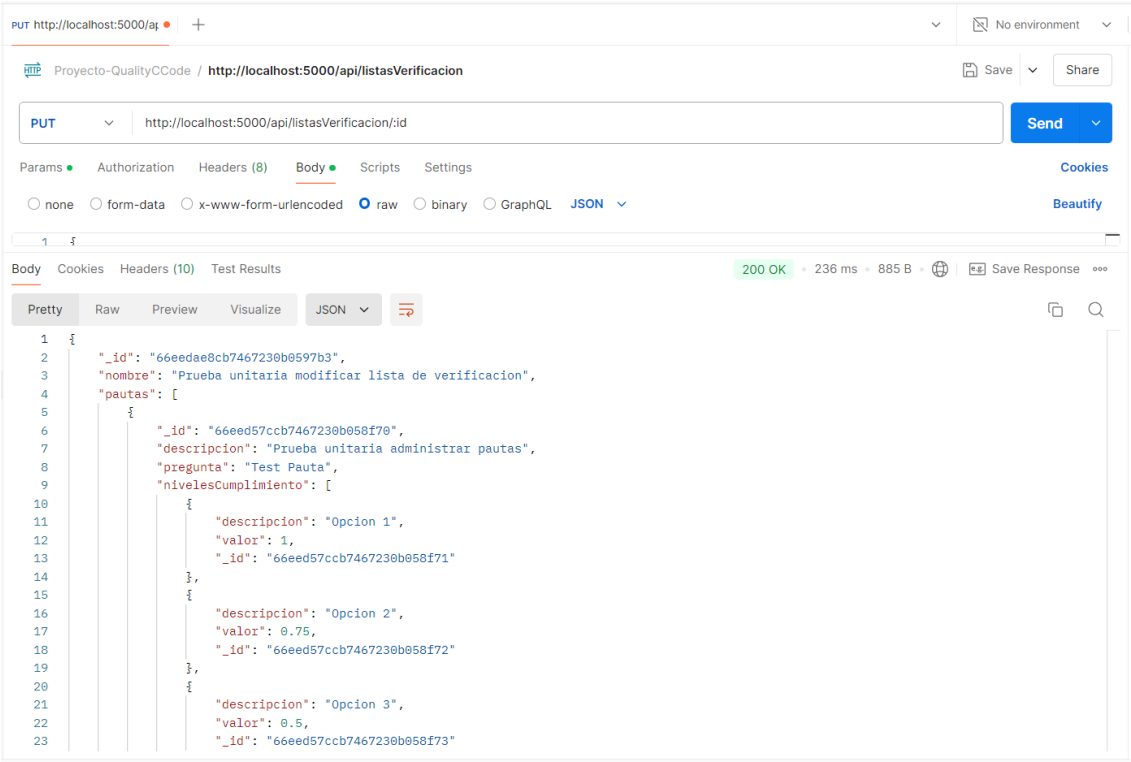


Figura 65. Verificación de modificar un registro del modelo lista de verificación - PU-05.

En la **Figura 66** se muestra la lista de verificación registrada en la PU-04, mediante la ejecución del diseño del frontend.

Figura 66. Verificación de la modificación del registro lista de verificación - PU-05.

En la **Figura 67** se muestra la API para eliminar un registro modelo lista de verificación.

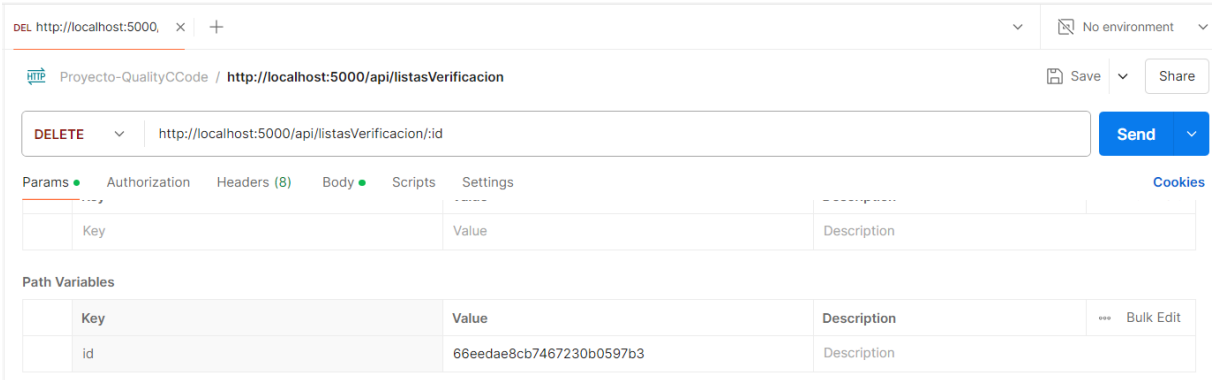


Figura 67. API para eliminar un registro del modelo lista de verificación - PU-05.

En la **Figura 68** se muestra la verificación de eliminar un registro lista de verificación, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

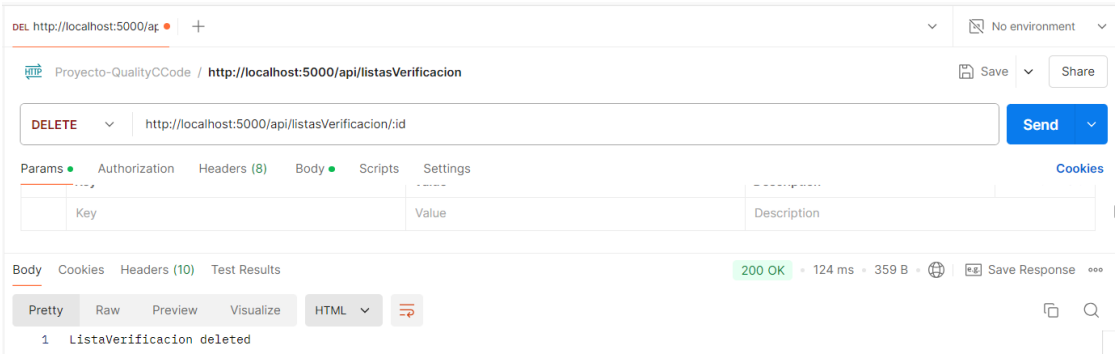


Figura 68. Verificación de eliminar un registro del modelo lista de verificación - PU-05.

6.2.4. Fase 4: Pruebas

6.2.4.1. Evaluación de la aplicación web mediante pruebas de aceptación – director del proyecto.

Se llevó a cabo la evaluación de aceptación de la aplicación web, revisada y ejecutada por el Ing. Francisco Alvarez Pineda, en su calidad de director y especialista. En **Tabla 90** presenta las pruebas realizadas respecto a las historias de usuario, las cuales se detallan en el **Anexo 20.** Se detalla, tras aplicar la primera revisión no se obtuvo el nivel de cumplimiento adecuado, sin embargo, con las observaciones y recomendaciones del director, se alcanzó el cumplimiento total de las funcionalidades requeridas, esto se evidencia en la fecha de aceptación de cada una de las historias de usuario.

Tabla 90. Pruebas de aceptación basadas en las historias de usuario**.**

HU	Caso de prueba	Nivel	FR	FA
HU01	Inicio de sesión	Completado	22/07/2024	10/09/2024
HU02	Administrar pauta	Completado	22/07/2024	10/09/2024
HU03	Administrar lista de verificación	Completado	22/07/2024	10/09/2024
HU04	Administrar métrica	Completado	22/07/2024	10/09/2024
HU05	Administrar subcaracterística	Completado	22/07/2024	10/09/2024
HU06	Administrar proyecto	Completado	22/07/2024	10/09/2024
HU07	Administrar prueba de mantenibilidad	Completado	22/07/2024	10/09/2024
HU08	Ejecutar prueba de mantenibilidad	Completado	22/07/2024	10/09/2024
HU09	Informe lista de verificación	Completado	22/07/2024	10/09/2024
HU010	Generar PDF	Completado	22/07/2024	10/09/2024
HU011	Calcular grado de mantenibilidad	Completado	22/07/2024	10/09/2024

Historia de Usuario (HU); Fecha de revisión (FR); Fecha de aceptación (FA).

6.2.4.2. Evaluación de la aplicación web mediante pruebas de aceptación – usuario final.

Para evaluar la aceptación de la aplicación web, fue esencial recopilar datos durante la sesión de pruebas del software, para lo cual se utilizó el cuestionario diseñado específicamente para medir la aceptación de la aplicación (Ver **Anexo 21**), mismo que se detalla en la **Tabla 91**.

Tabla 91. Pruebas de aceptación basadas en las historias de usuario**.**

N.	Caso de prueba
1	¿Considera que el software es sencillo y fácil de usar?
2	¿Encuentra intuitiva la interfaz de pruebas para verificar la mantenibilidad de aplicaciones web?
3	¿Cree que el software proporciona instrucciones claras para realizar el análisis de código estático?
4	¿Encuentra intuitiva la interfaz que resuelve el cálculo del grado de mantenibilidad de aplicaciones web?
5	¿La navegación entre cada sección del software es sencilla?
6	¿El software proporciona datos relevantes sobre los resultados de las pruebas destinadas para verificar la mantenibilidad de aplicaciones web?
7	¿El PDF generado proporciona información relevante sobre los resultados de las pruebas destinadas para verificar la mantenibilidad de aplicaciones web?
8	¿Considera que el software permite conocer pautas de mantenibilidad y propiedades de calidad para la solución de defectos de aplicaciones web?
9	¿Considera el software una herramienta útil para verificar la mantenibilidad de las aplicaciones web?
10	¿El software posee tiempos de respuesta buenos?

El objetivo es conocer la facilidad de uso y la utilidad percibida del software para verificar la mantenibilidad en aplicaciones web, mediante el nivel de aceptación desde un punto del usuario final, para lo cual se utilizó la escala de Likert que permita obtener una medida detallada y cuantificable de la percepción de los usuarios con respecto al uso del software. Por tal razón, en la **Tabla 92** exhibe la escala de valoración utilizada que comprende valores del 1 al 3 con su respectivo criterio de decisión, cada uno de los criterios se estableció como respuesta a las interrogantes. Estos datos son los que permitieron determinar el nivel de aceptación del software.

Tabla 92. Puntuaciones en base a la escala de Likert.

Escala	Criterio de aceptación	Puntos
1	Si	2
2	En parte	1
3	No	0

Para la determinación de la muestra, se seleccionó una muestra de 25 usuarios, utilizando un muestreo no probabilístico por conveniencia. La elección se basó en la accesibilidad de los participantes y su relevancia como representantes del grupo objetivo.

A continuación, en la **Tabla 93** se presenta un resumen de los resultados obtenidos de las encuestas aplicadas, mismo que se detalla en el (Ver **Anexo 22**).

Tabla 93. Pruebas de aceptación basadas en las historias de usuario**.**

US	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
1	1	1	2	2	2	2	2	2	2	2
2	2	2	2	1	2	1	1	2	2	2
3	2	2	2	2	2	2	2	2	2	2
4	2	1	2	2	1	2	2	2	2	2
5	2	2	2	1	2	2	2	2	2	2
6	2	2	1	1	2	2	1	1	1	1
7	2	2	1	2	1	2	2	2	1	2
8	2	2	2	1	2	2	2	2	2	1
9	2	2	2	2	2	2	2	2	2	2
10	2	2	2	2	2	2	2	2	2	2
11	2	2	2	2	2	2	2	2	2	1
12	2	1	1	2	2	1	2	1	2	1
13	2	2	2	2	2	2	2	2	2	2
14	1	2	2	1	2	2	1	1	1	1
15	2	2	2	2	2	2	2	2	2	2
16	2	2	1	1	2	2	2	1	2	2
17	2	2	2	2	2	2	2	2	2	2
18	2	2	2	2	2	2	2	2	2	2
19	2	2	2	2	2	2	2	2	2	2
20	2	2	2	2	2	2	2	1	2	2
21	2	2	2	2	2	2	2	2	2	2
22	2	2	2	2	2	2	2	2	2	2
23	1	1	2	1	1	2	2	1	2	2
24	2	2	2	2	2	2	2	2	2	2
25	2	2	1	1	2	2	2	2	2	2

Usuario (US); Pregunta (P)

En la **Tabla 94** se muestra el cálculo de los puntajes en relación con los criterios de decisión de cada pregunta, se realizó la suma de los puntos obtenidos por cada pregunta en función del total de encuestados.

Tabla 94. Pruebas de aceptación basadas en las historias de usuario**.**

	Criterio	**Criterio**	**Usuarios**	**Valoración**	**Puntos**
Pregunta 1	Sí	2	22	44	47
	En parte	1	3	3	
	No	0	0	0	
Pregunta 2	Sí	2	21	42	46
	En parte	1	4	4	
	No	0	0	0	
Pregunta 3	Sí	2	20	40	45
	En parte	1	5	5	
	No	0	0	0	
Pregunta 4	Sí	2	17	34	42
	En parte	1	8	8	
	No	0	0	0	
Pregunta 5	Sí	2	22	44	47
	En parte	1	3	3	
	No	0	0	0	
Pregunta 6	Sí	2	23	46	48
	En parte	1	2	2	
	No	0	0	0	
Pregunta 7	Sí	2	22	44	47
	En parte	1	3	3	
	No	0	0	0	
Pregunta 8	Sí	2	19	38	44
	En parte	1	6	6	
	No	0	0	0	
Pregunta 9	Sí	2	22	44	47
	En parte	1	3	3	
	No	0	0	0	
Pregunta 10	Sí	2	20	40	45
	En parte	1	5	5	
	No	0	0	0	

Para el análisis de resultados globales se sumo los puntos de todas las preguntas para obtener un total global de puntos posibles.

$$47 + 46 + 45 + 42 + 47 + 48 + 47 + 44 + 47 + 45 = 458$$

A partir de una medida detallada y cuantificable de la percepción de los usuarios finales sobre el uso del software, se estableció criterios de evaluación para definir nivel de aceptación.

El porcentaje de aceptación es de acuerdo al número de encuestados.

Rango	**Resultado**
0% - 40%	No aceptado (hay problemas importantes).
41% - 70%	Aceptado con reservas (hay puntos que mejorar).
71% - 100%	Aceptado sin cambios (buen desempeño).

Calculamos el porcentaje total de **458** en relación con los **500** puntos.

$$\text{Porcentaje} = \frac{458}{500} * 100$$

$$\text{Porcentaje} = 0.916 \times 100 = 91.60\%$$

- **Análisis de resultados**

El propósito de la siguiente prueba fue evaluar la facilidad de uso y la utilidad percibida del software para verificar la mantenibilidad de aplicaciones web, desde la perspectiva de los usuarios finales. Para ello, se utilizó un cuestionario basado en la escala de Likert, que permitió obtener una medida cuantificable y detallada de la percepción de los usuarios con respecto al uso del software, el cuestionario incluyó 10 preguntas clave, donde los usuarios podían elegir entre tres opciones de respuesta: **No**, **En parte**, y **Sí**. A cada respuesta se le asignó un puntaje: 0 para **No**, 1 para **En parte**, y 2 para **Sí**, el puntaje máximo posible era de 500 puntos, calculado a partir de 25 usuarios que completaron el cuestionario.

De los resultados obtenidos, el puntaje total fue de 458 puntos, lo que representa un 91.60% del puntaje máximo, el resultado refleja que la mayoría de los usuarios tiene una opinión favorable sobre el software, tanto en términos de facilidad de uso como en la utilidad de sus funcionalidades para verificar la mantenibilidad de aplicaciones web.

- **Interpretación de los Resultados**

El puntaje obtenido de 458 puntos indica un nivel de aceptación que se encuentra en el rango es alto según los criterios establecidos (71% - 100%: aceptado sin cambios), esto sugiere que, en general, los usuarios consideran que el software cumple adecuadamente con las expectativas y objetivos planteados.

Los puntos clave que se destacan en este análisis incluyen:

- **Facilidad de Uso**: Una gran parte de los usuarios considera que el software es fácil de usar y que la navegación entre las distintas secciones es sencilla e intuitiva. Esto es crucial, ya que la simplicidad en la interacción mejora la experiencia del usuario final y reduce la curva de aprendizaje.
- **Utilidad percibida:** Los resultados muestran que los usuarios perciben que el software ofrece instrucciones claras y proporciona resultados útiles para verificar la mantenibilidad de aplicaciones web. El hecho de que los usuarios valoren positivamente la capacidad del software para generar datos relevantes y reportes en PDF refuerza su valor como herramienta en este ámbito.
- **Mejoras Potenciales:** Aunque el nivel general de aceptación es alto, algunos usuarios sugirieron mejoras puntuales, principalmente relacionadas con la interfaz de usuario y los tiempos de respuesta del software. Si bien estos aspectos no impiden el uso eficiente del sistema, podrían considerarse áreas de optimización para futuras versiones.

- **Conclusión**

Con un puntaje total de 458 puntos, se concluye que el software ha alcanzado un alto nivel de aceptación por parte de los usuarios finales, el resultado evidencia que el software no solo es percibido como una herramienta fácil de usar, sino también como una solución efectiva para analizar la mantenibilidad de aplicaciones web.

Para mantener este nivel de satisfacción y mejorar aún más la experiencia del usuario, sería recomendable explorar posibles optimizaciones en la interfaz y los tiempos de respuesta. Sin embargo, en términos generales, el software está bien posicionado para cumplir con el propósito principal y continuar siendo una herramienta útil en el ámbito del análisis de mantenibilidad del software.

- **Registro Fotográfico**

En la presente sección se muestra las fotografías durante la implementación de las pruebas de aceptación del software desarrollado **QualityCCode**.

Figura 69. Fotografía implementación de pruebas de aceptación 1.

Figura 70. Fotografía **i**mplementación de pruebas de aceptación 2.

6.2.5. Fase 5: Solución de defectos relacionado con la mantenibilidad del software

La solución de los defectos de mantenibilidad identificados en el caso de estudio **Gestión VR** se aborda a través de un plan de estrategias enfocado específicamente en el backend, como área encargada de la lógica de la aplicación web. Para esto se hace uso de la aplicación web desarrollada QualityCCode, la cual se utilizó con objetivo de recopilar soluciones efectivas como pautas y calcular el grado respecto a la mantenibilidad del software, SonarQube para identificar áreas específicas en donde se presenta el defecto y Excel para registrar y documentar el grado de mantenibilidad del caso de estudio.

En esta sección, se lleva a cabo un análisis exhaustivo de los resultados obtenidos tras la implementación del plan de estrategias. El **Anexo 19** ofrece un desglose detallado de la ejecución de dicho plan, abarcando cada caso de prueba aplicado a los defectos priorizados y aceptados que tienen mayor relevancia.

6.2.5.1. Análisis de Resultados.

Para la medición de calidad del caso de estudio, se diseñó una matriz para evidenciar el cálculo del grado de mantenibilidad para el backend y frontend, la cual se ilustra en el **Anexo 24** y puede ser accedida a través de un enlace a un documento de Excel en Google Drive. (Ir https://docs.google.com/spreadsheets/d/1MpQICqJyBhKjncv_MjdPYDzYmtBwIh_d/edit?usp=sharing&ouid=103537596169053493808&rtpof=true&sd=true), La matriz permitió realizar el seguimiento del progreso a medida que se aplicaron las modificaciones para solucionar los defectos, en sí, el documento se basa en cálculos matemáticos utilizando fórmulas para evaluar las métricas y subcaracterísticas de mantenibilidad, recopiladas durante la investigación del presente TIC.

El documento excel permitió cuantificar el grado de mantenibilidad del software de manera objetiva y sistemática, utilizando una escala de valoración del 0% al 100%. Según los resultados después de aplicar el plan de estrategias para la solución de los defectos de mayor relevancia se ha logrado alcanzar un grado de mantenibilidad del caso de estudio “Gestion VR” del 82.34% al 89.62%, el cual se ubica con una escala de valoración de altamente mantenible, este rango representa un nivel de calidad en el que el software es fácil de entender y modificar.

Para la medición de calidad del backend mediante del plan de estrategias, de la misma forma se diseñó una matriz accesible a través de un enlace a un documento de Excel en Google

Drive. En la **Tabla 95** se encuentra los enlaces para direccionar a la matriz e ilustrar en la sección de anexos, la matriz permitió evaluar el grado de mantenibilidad en dos estados: antes y después de la aplicación del plan.

Tabla 95. Enlaces para direccionar a la matriz de evaluación.

Estado	Ilustración	Enlace
Sin solución de defectos	Anexo 25	Documento Excel
Con solución de defectos	Anexo 26	Documento Excel

a. Defecto 1: Código duplicado.

Las modificaciones realizadas, ha permitido evidenciar mejoras significativas en la calidad del código, especialmente en lo que respecta al **código duplicado** en las clases del módulo **controllers**, uno de los más críticos del proyecto. La implementación de las sugerencias basadas en pautas de mantenibilidad ha sido fundamental para lograr estos avances, esto se evidencia en la reducción del porcentaje de código duplicado del 56.77% al 3.1%, lo que incrementa el grado de mantenibilidad del backend del 73.07% al 87.56%.

Las sugerencias incluyen principios como DRY (Don't Repeat Yourself), SRP (Single Responsibility Principle), encapsulación y modularización, ha permitido crear un código más limpio, legible y fácil de mantener. La aplicación de estas pautas ha tenido un impacto positivo en la calidad del software, facilitando la identificación y corrección de errores, y sentando las bases para una mayor escalabilidad y flexibilidad en el futuro. Estos resultados demuestran la efectividad de las pautas de mantenibilidad y la importancia de adoptar buenas prácticas de programación.

b. Defecto 3: Malas prácticas de codificación.

La aplicación de buenas prácticas de programación ha permitido evidenciar mejoras significativas en la calidad del código, especialmente en lo que respecta al defecto de **malas prácticas de codificación**, en las clases del módulo **controllers**, **middlewares** y **config**, unos de los más críticos del proyecto. La implementación de las sugerencias basadas en pautas de mantenibilidad ha sido fundamental para lograr estos avances, esto se evidencia en la eliminación por completo de estas apariciones, reduciendo la cantidad de las malas prácticas, lo que incrementa el grado de mantenibilidad del backend del 87.56% al 87.63%.

La implementación de técnicas como el manejo de errores, a través de err.message y registros de error ha permitido gestionar de manera efectiva las situaciones excepcionales que pueden surgir durante la ejecución del programa. Además, la construcción de rutas de archivos mediante path.join garantiza que las rutas se generen de forma segura y eficiente, evitando problemas relacionados con la portabilidad entre diferentes sistemas operativos. La utilización de excepciones proporciona un marco para controlar y gestionar errores, lo que mejora la robustez del software, asimismo, el manejo de variables y control de alcance asegura que las variables se declaren y utilicen de manera adecuada, minimizando conflictos y errores.

Así mismo, como en la solución del defecto 1, se aplicó el principio DRY (Don't Repeat Yourself), el cual contribuyo a la reducción de código duplicado mediante la creación de funciones auxiliares, mientras que el Single Responsibility Principle (SRP) aseguro que cada función se encargue de una única tarea específica. En conjunto, estas prácticas no solo mejoran la calidad del software, sino que también facilitan la identificación y corrección de errores, estableciendo una base sólida para la escalabilidad y flexibilidad futura del sistema.

6.2.5.2. Impacto de las soluciones en la mantenibilidad del caso de estudio.

Los resultados de la aplicación de pautas de mantenibilidad para abordar los defectos de código duplicado y malas prácticas de codificación son altamente positivos. Esto refleja en una significativa disminución de los defectos en el código: el porcentaje de líneas duplicadas ha caído del 56.77% al 3.1%, mientras que las malas prácticas de codificación se han reducido de 6 a 0. Estos avances demuestran la efectividad de las estrategias aplicadas y su impacto en la calidad general del código. Un ejemplo notable es el aumento en la calidad de la documentación del código, que pasó del 20% al 21.60%, como resultado de la efectividad de la corrección de los dos defectos, este incremento resalta la importancia de seguir prácticas de mantenibilidad para mejorar no solo el código en sí, sino también la documentación que lo acompaña.

La corrección de los defectos como **código duplicado** y **malas prácticas de codificación** ha tenido un impacto significativo en la mejora del grado de mantenibilidad del backend. Antes de la aplicación de las pautas, el grado se situaba en el 73.07%, sin embargo, después de implementar las soluciones y corregir estos defectos, el índice de mantenibilidad aumentó notablemente hasta alcanzar el 87.63%.

En efecto, el grado de mantenibilidad del caso de estudio incremento un 14,56%, debido al plan de estrategias aplicado backend, pasando del 82.34% al 89.62%. Los resultados reflejan de manera contundente la efectividad de las pautas de mantenibilidad implementadas, que incluyen principios como la modularización, la encapsulación y la eliminación de redundancias, demostrando su capacidad para optimizar la estructura y legibilidad del código.

En la **Figura 71** se presenta el análisis de código estático correspondiente al caso de estudio sin la implementación del plan de estrategias destinado a la solución de defectos.

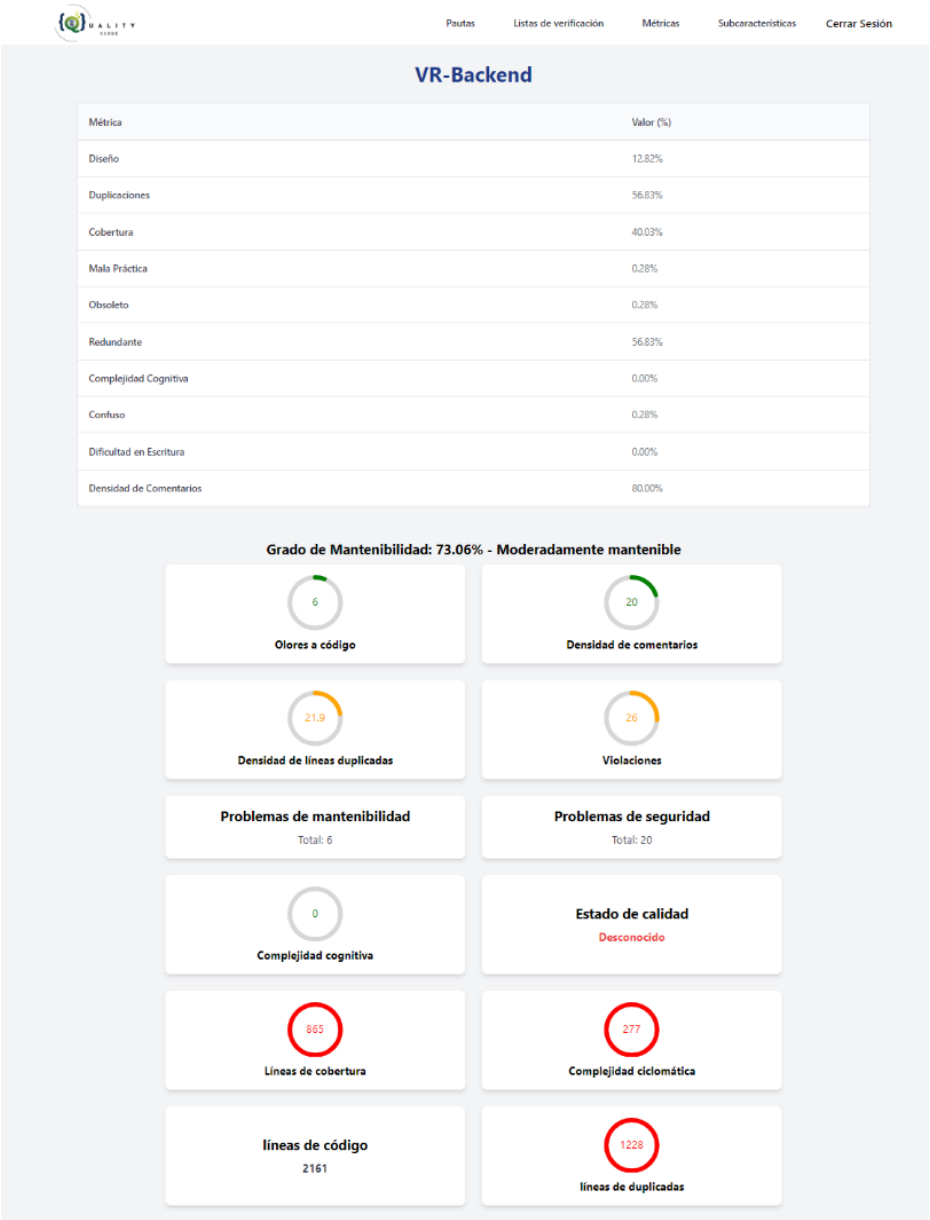


Figura 71. Análisis de código estático sin plan de estrategias.

En la **Figura 72** se presenta el análisis de código estático correspondiente al caso de estudio con la implementación del plan de estrategias destinado a la solución de defectos.

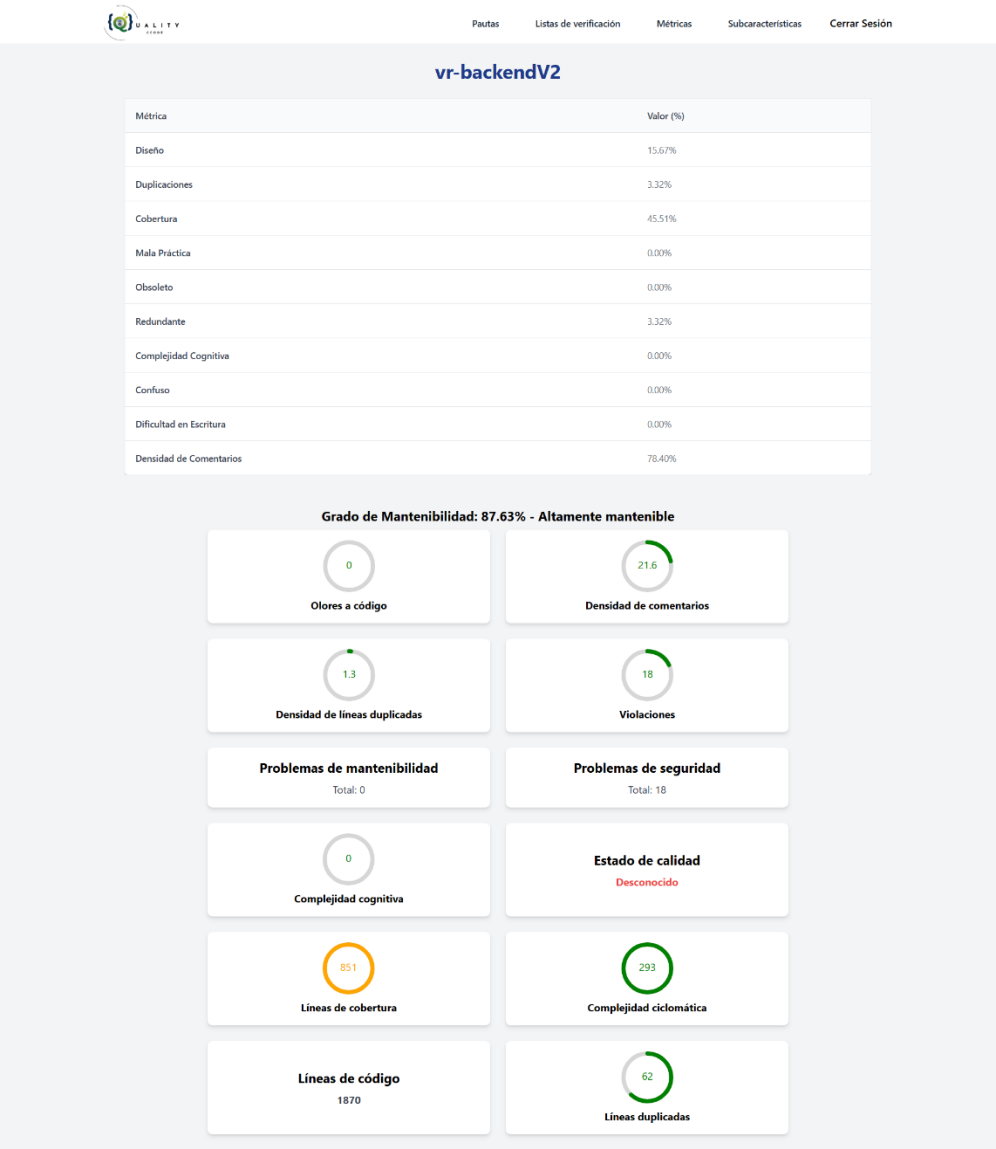


Figura 72. Análisis de código estático con plan de estrategias.

En definitiva, la implementación del plan de estrategias para el backend del caso de estudio "Gestión VR" ha demostrado ser efectivo en la mejora del grado de mantenibilidad, a través de un enfoque sistemático que incluyó la identificación y corrección de defectos como código duplicado y malas prácticas de codificación, se logró un incremento notable en la calidad del código, elevando el grado del 73.07% al 87.63%, lo que resalta la importancia de las pautas aplicadas en el plan. En este sentido, la adopción de buenas prácticas de programación contribuye que un software sea más robusto, eficiente y mantenible.

7. Discusión

El desarrollo del presente TIC surge como respuesta a la falta de aplicación de buenas prácticas de programación, situación evidenciada por encuestas y entrevistas realizadas a docentes y estudiantes de la carrera de Computación. Esta deficiencia ha resultado en un alto número de fallas relacionadas con la mantenibilidad del software, con el objetivo de abordar estos desafíos, se aplicó un proceso de investigación para seleccionar y adaptar un modelo de medición de calidad para evaluar la mantenibilidad de dos aplicaciones web, lo que permitió identificar los defectos. A partir de esto, se desarrolló un software diseñado para verificar la mantenibilidad y apoyar con pautas resultado de una revisión bibliográfica de buenas prácticas de programación que faciliten la mejora continua en la codificación de aplicaciones web.

Objetivo 1: Diagnosticar las aplicaciones web en producción de los laboratorios de la carrera de Computación de la UNL, mediante la evaluación del grado de mantenibilidad basada en la Norma ISO/IEC 25000

Los estudios previos [98], [99], [100], [101] permitieron determinar la viabilidad de la implementación de la ISO/IEC 25000, considerando las normas: ISO/IEC 25010 y ISO/IEC 25040. Además, como se menciona en [52], se destaca la conveniencia de esta norma para la evaluación de productos de software, aspecto que se evidencia en el presente TIC, debido a que la norma ayudó con la estructura para la evaluación gracias al conjunto de características y subcaracterísticas y el modelo referencial para la evaluación.

La selección de un modelo de evaluación basado en ISO/IEC 25000 se respaldó en los estudios [102],[105], los cuales especificaron las directrices para garantizar una evaluación efectiva y alineada con los estándares de calidad del software. Este método de procesos permitió identificar de forma clara los requisitos para seleccionar un modelo de evaluación (Ver **Fase 1**), facilitando un ecosistema automatizado para evaluar la calidad. Se destaca que, durante la selección, surgieron dificultades como incluir y excluir modelos; sin embargo, estas limitaciones se superaron con la incorporación de criterios de selección (Ver **Criterios**) de los siguientes estudios [103], [104].

La adaptación del Modelo propuesto por Galán a la norma ISO/IEC 25040 (Ver **Adaptación**), resulto ser un método apropiado para evaluar la mantenibilidad del software, enfoque similar a los estudios [106], [8], los cuales obtuvieron resultados favorables al implementarlo, ya que el propósito es alinear a las necesidades específicas del proyecto o al

caso de estudio. En este contexto, se corrobora el enfoque planteado por las investigaciones [52], [107], [39], como un método efectivo, que facilita la realización ordenada de las actividades para el diagnóstico y la detección de defectos en el software.

Los estudios previos [108], [68], permiten determinar la viabilidad de la utilización de la herramienta de Sonarqube para la medición de las propiedades de calidad de un producto de software, la herramienta no solo facilitó la evaluación de la calidad del software, como menciona en [69], sino que también proporciona métricas precisas que permiten identificar defectos y áreas de mejora. Este aspecto se evidencia en el presente TIC, puesto que en conjunto con el modelo de medición adaptado ayudó a identificar cuatro defectos relacionados con la mantenibilidad (código duplicado, falta de pruebas unitarias, malas prácticas de codificación y comentarios desactualizados o innecesarios) los cuales son críticos y esenciales para asegurar la calidad del software.

Se corrobora la problemática expuesta, acerca del **déficit en la aplicación de buenas prácticas de programación durante el desarrollo de aplicaciones web en la carrera** (Ver **PIC**), dado que al finalizar el diagnóstico del código de los casos de estudio, se identificó que de las 6 propiedades de calidad que influyen en la medición de las métricas de la mantenibilidad, cuatro no cumplen con el porcentaje aceptable establecido por el modelo de medición, lo que se considera como un defecto, mismos que están directamente relacionados con la falta de aplicación de buenas prácticas de programación. Al aplicar el modelo de medición la aplicación SDLC alcanzó el 89.61% clasificado como altamente mantenible, mientras que GestionVR obtuvo un 82.34%, considerado moderadamente mantenible (**Ver Fase 2**).

Como resultado de la implementación del primer objetivo, se logró desarrollar una matriz de defectos (Ver **Tabla 64**) alineada con la norma ISO/IEC 25000, específicamente enfocada en la característica de mantenibilidad. Esta matriz permitió clasificar y analizar los defectos encontrados de manera sistemática, proporcionando una visión detallada de las áreas críticas que afectan la calidad del software.

Objetivo 2: Desarrollar un software web tomando como referencia XP como metodología de desarrollo, para solucionar el grado de mantenibilidad de una de las aplicaciones web diagnosticadas en el objetivo uno

La investigación previa de los estudios [73], [74], [75], permitieron calificar positivamente la implementación de la metodología XP en todas sus fases: planificación, diseño, codificación y pruebas. La metodología se destaca por resultados favorables en el desarrollo de productos de software, como se menciona en [109] y en la presente implementación XP simplificó significativamente los esfuerzos en el desarrollo del proyecto, debido a sus lineamientos organizativos, capacidad para adaptarse a cambios, facilidad de implementación en equipos reducidos y su estrecha relación con los casos de uso e historias de usuario.

En relación con los documentos incluidos sobre buenas prácticas de programación descubiertos en la revisión bibliográfica, se encontraron la cantidad de 28 documentos, donde estudios como [110], [111], [112], las describen como pautas esenciales que orientan a los desarrolladores en la creación de software de alta calidad, demostrando ser versátiles y efectivos en la corrección de defectos, siendo reconocidos en investigaciones como la de [71], [113]. Este aspecto se evidencia, debido a que la utilización de las pautas ayudó optimizar el proceso de desarrollo y en efecto mejora la calidad del producto.

La extracción de pautas se respaldó en el estudio [69], el cual especifica la estructura de las métricas de acuerdo a las subcaracterísticas de la mantenibilidad, este modelo permitió asociar cada pauta a una métrica específica, encargada de evaluar el impacto de dichas subcaracterísticas. Como resultado, se facilitó la implementación de prácticas de desarrollo orientadas a mejorar las propiedades de calidad del código. Esta vinculación entre pautas y métricas no solo optimizó el proceso de evaluación, sino que también promovió un enfoque más sistemático en la mejora continua del software como plantea [8] y el presente enfoque.

En la técnica de la entrevista (Ver **Anexo 15**) y la encuesta (Ver **Anexo 16**) se obtuvieron resultados positivos que sentaron las bases para un desarrollo centrado en el usuario, similar a los estudios previos [101], [13], observados en su implementación, ya que la información obtenida no solo enriqueció el análisis, sino que también proporcionó una perspectiva más profunda sobre las necesidades y expectativas de los usuarios, lo que es fundamental en el desarrollo de un producto de software.

En la fase de codificación de la aplicación web, se obtuvieron resultados favorables gracias a la planificación de interacciones, conforme a lo indicado en la investigación previa[109]. Esta planificación facilitó la identificación de los módulos programados y simplificó significativamente el proceso de construcción del software, como resultado, se lograron evitar errores de comunicación entre el backend y el frontend, así como problemas relacionados con el consumo de servicios de terceros y la implementación de archivos PDF. Para verificar el funcionamiento de cada uno de los módulos programados, se tomó como referencia lo señalado por [79], [93], [76], que destaca la importancia de verificar el correcto funcionamiento del código. Para lo cual, en el presente proyecto las pruebas iniciales arrojaron resultados desfavorables, ante esta situación, se decidió planificar fases específicas para el desarrollo del código, denominadas tareas (diseño, codificación y pruebas). Esta estrategia facilitó un enfoque más estructurado para abordar los problemas, lo que condujo a resultados finales positivos.

Basado en el estudio [92], se comprueba que los lineamientos del muestreo no probabilístico por conveniencia son útiles para determinar la muestra en contextos específicos. En el presente TIC, este método permitió seleccionar un total de 25 usuarios de la carrera de computación, en donde la selección es representativa del grupo objetivo y proporciona una base sólida para el análisis de los resultados obtenidos.

Para establecer el nivel de aceptación del software, se optó por un modelo hibrido respaldado por estudios previos [114], [115], que permite determinar la aceptación de un sistema en una población especifica. La evaluación se llevó a cabo considerando dos factores clave: la facilidad de uso y la utilidad percibida. En este sentido, se aplicaron encuestas a una muestra de 25 usuarios, quienes completaron un cuestionario con un puntaje máximo de 20 puntos, en base al total de encuestados que se tiene un puntaje máximo para el software de 500 puntos. Se obtuvo un total de 458 puntos, lo que indica que la respuesta predominante para el criterio de decisión fue "Sí”, en representación de la escala de Likert, este resultado muestra un 91.60% del puntaje máximo, lo que refleja que los usuarios perciben la aplicación como útil y fácil de usar, reforzando el potencial del software desarrollado para ser implementado y utilizado en la práctica.

Los resultados obtenidos permitieron dar respuesta la pregunta de investigación planteada: ¿Qué grado de mantenibilidad tienen las aplicaciones web en producción de los

laboratorios de la carrera de computación de la UNL, tomando como referencia la familia de la norma ISO/IEC 25000?, el presente trabajo reveló que, utilizando el modelo de medición de mantenibilidad adaptado, la aplicación SDLC alcanzó un grado de mantenibilidad del 89,61%, mientras que la aplicación GestionVR obtuvo un 82.34%. Al aplicar el plan de estrategias utilizando como caso de estudio GestionVR, se incrementa el grado de mantenibilidad del 82.34% al 89.62%, esto confirma la implementación de herramientas de control de calidad tiene un impacto positivo en el proceso de desarrollo, similar a los estudios [3], [4], [5], ya que la automatización de listas de verificación y el cálculo del grado de mantenibilidad facilita la verificación de la mantenibilidad del software.

Como resultado de la implementación del segundo objetivo, se desarrolló la aplicación **QualityCCode**, para verificar la mantenibilidad de aplicaciones web, la implementación de esta herramienta corroboró la problemática expuesta sobre la **utilidad de una herramienta de control de calidad propia de la carrera** (ver **PIC**), esto en base a la implementación de un plan de estrategias a través de QualityCCode, que ha demostrado ser un método efectivo para abordar los defectos de mantenibilidad, utilizando como caso de estudio al software Gestión VR. Los resultados fueron contundentes al mejorar el grado de mantenibilidad de la aplicación clasificándose como altamente mantenible (ver **Anexo 19**), lo que evidencia que el software ahora requiere menos esfuerzo para su comprensión y modificación, validando la utilidad de la herramienta en la mejora continua de la calidad del software.

8. Conclusiones

Al finalizar el presente TIC se concluye lo siguiente en base los resultados obtenidos:

- En el presente TIC se pudo determinar que, la adaptación del modelo de medición de mantenibilidad que combina el Modelo Galán y la norma ISO/IEC25040 resultó ser un modelo apropiado para la evaluación de calidad del software, ya que permitió organizar de manera efectiva las actividades necesarias para evaluar el grado de mantenibilidad y facilitar la identificación de defectos.
- Para garantizar un control de calidad efectivo del código, se utilizó como guía las normas de la familia ISO/IEC 25000, estas normas proporcionan un marco estructurado y bien definido para evaluar las diversas áreas de calidad del software, facilitando la identificación y medición de los productos desarrollados. En este contexto, la investigación se centró en la evaluación del código generado durante el proceso de desarrollo, permitiendo no solo detectar defectos y áreas de mejora, sino también asegurar que el producto final cumpla con los estándares de calidad requeridos.
- El uso del modelo de medición de mantenibilidad propuesto muestra resultados positivos gracias a su flexibilidad y capacidad para adaptarse a las necesidades del presente proyecto, ya que se pudo determinar el grado de mantenibilidad de los casos de estudio, en lo esencial, permitió identificar las propiedades de calidad que no cumplen con los criterios de aceptación establecidos en el modelo, lo que facilito la detección de defectos.
- La implementación de la metodología XP para el desarrollo de la aplicación QualityCCode permite realizar un desarrollo ágil y flexible, debido que facilita la planificación de actividades como entrevista y encuesta, creación de historias de usuario, definición de requisitos, elaboración de diagramas y codificación, a través de sus cuatro fases: planeación, diseño, codificación y pruebas. Esto permitió que la aplicación se desarrollara exitosamente cumpliendo las especificaciones para la que fue diseñada.
- El uso del modelo hibrido junto con la escala de Likert permitió establecer los criterios de decisión claros, facilitando la evaluación del nivel de aceptación de la aplicación web por parte de los usuarios finales en aspectos como la facilidad de uso y la utilidad percibida. Esto se evidencia en el resultado obtenido, con un alto nivel

de aceptación del 91.60% de los encuestados lo que demuestra estar convencidos de las funcionalidades ofrecidas por el software.

- Se logró responder satisfactoriamente la pregunta de investigación **¿Qué grado de mantenibilidad tienen las aplicaciones web en producción de los laboratorios de la carrera de computación de la UNL, tomando como referencia la familia de la norma ISO/IEC 25000?**, esto se alcanzó mediante la adaptación de un modelo de medición de mantenibilidad, desarrollo de una aplicación web y la implementación de un plan de estrategias. Se realizó una medición inicial en el cual la aplicación SDLC alcanzó un grado de mantenibilidad del 89,61%, mientras que la aplicación GestionVR obtuvo un 82,34%, posteriormente tras aplicar el plan de estrategias para solucionar los defectos, el grado del caso de estudio Gestion VR se incrementó al 89,62%, este avance indica una reducción significativa en el esfuerzo requerido para comprender y modificar el código fuente. En definitiva, este enfoque se consolida como un método eficaz para la identificación y resolución de defectos de mantenibilidad.

9. Recomendaciones

Al finalizar el presente TIC se recomienda lo siguiente:

- Del estudio realizado es recomendable que los estudiantes de la carrera durante el desarrollo de software adopten modelos de calidad similares al propuesto por Galán Pausic para la evaluación de la mantenibilidad del código, puesto que al integrar procesos, técnicas y herramientas basadas en las normas ISO/IEC 25000, y específicamente las ISO/IEC 25010 e ISO/IEC 25040, se puede asegurar un control de calidad más robusto y sistemático.
- La adopción de la norma ISO/IEC 25040 para establecer un proceso estructurado de evaluación de la calidad del software, esta norma permite adaptarse a las necesidades específicas, ajustando el proceso de evaluación a sus objetivos, recursos y restricciones particulares. Para maximizar los beneficios de la norma, se sugiere seleccionar cuidadosamente las métricas adecuadas para medir la calidad del software, asegurando así que los resultados obtenidos sean tanto relevantes como significativos.
- La adopción de las normas de la familia ISO/IEC 25000 para el control de calidad puede sistematizar el proceso de evaluación y medición de la calidad del código, garantizando productos de software más robustos y mantenibles.
- Utilizar el modelo de calidad propuesto y la estructura del proceso de evaluación del código referente al modelo Galán y la Norma ISO/IEC25040, para que sirva de referencia en el desarrollo de otros proyectos.

9.1. Trabajos futuros

La implementación de la primera versión de la aplicación web QualityCCode para verificar la mantenibilidad arrojó resultados favorables respecto a la problemática expuesta, no obstante, para versiones futuras se recomienda explorar la aplicación de técnicas de inteligencia artificial y aprendizaje automático para predecir áreas del código que son más propensas a fallos, optimizando así el proceso de mantenimiento.

10. Bibliografía

[1] J. Reyes Juárez Ramírez, G. L. Sandoval, A. Pardo Zurita, and J. A. García Sánchez, "Introducción: Administración y Medición de la Mantenibilidad del Software, usando Mapas de Mantenimiento y Grafos Dirigidos."

[2] E. Irrazábal, "Mejora de la mantenibilidad con un modelo de medición de la calidad: resultados en una gran empresa," 2020. Accessed: Jul. 29, 2024. [Online]. Available: https://core.ac.uk/download/pdf/301067116.pdf

[3] J. Chávez, "ESTANDARIZACIÓN DE LOS PROCESOS DE DESARROLLO DE SOFTWARE UTILIZANDO BUENAS PRÁCTICAS DE PROGRAMACIÓN Y SCRUM COMO MARCO DE TRABAJO ÁGIL EN DEPARTAMENTOS DE TI"," Ambato, 2019. Accessed: Mar. 12, 2024. [Online]. Available: https://repositorio.uta.edu.ec/bitstream/123456789/29604/1/Tesis_t1561msi.pdf

[4] J. Bautista, "EVALUACIÓN DEL SISTEMA VIÁTICOS DEL GOBIERNO AUTÓNOMO DESCENTRALIZADO DE LA PROVINCIA DE ESMERALDAS (GADPE) BASADO EN LA NORMA ISO 25000," Esmeraldas, 2019. Accessed: Mar. 12, 2024. [Online]. Available: https://repositorio.pucese.edu.ec/bitstream/123456789/1904/1/BAUTISTA%20ANGULO%20JENNIFFER%20VER%C3%93NICA.pdf

[5] S. Pardo, "MANTENIBILIDAD DE PRODUCTOS DE SOFTWARE SEGÚN EL MODELO SQUARE ISO/IEC 25000," Tingo María- Perú, 2019. Accessed: Mar. 12, 2024. [Online]. Available: https://repositorio.unas.edu.pe/bitstream/handle/20.500.14292/1360/SRPM_2018.pdf?sequence=1&isAllowed=y

[6] J. L. Hernando Bahamon, "CONTROL DE CALIDAD ·EN EL SOFTWARE." Accessed: May 12, 2024. [Online]. Available: https://repository.icesi.edu.co/biblioteca_digital/bitstream/10906/4008/1/Control_calidad_software.pdf

[7] E. Ingrid, "Métricas de legibilidad del software: Una revisión sistemática de literatura," 2020. Accessed: Apr. 15, 2024. [Online]. Available:

https://repositorio.unal.edu.co/bitstream/handle/unal/77441/1121829674.2020.pdf?sequence=1&isAllowed=y

[8] J. Adones Farfán and V. Vega-Zepeda, “Mantenibilidad del Software- Consideraciones para su especificación y validación,” 2020. Accessed: Mar. 12, 2024. [Online]. Available: https://www.scielo.cl/pdf/ingeniare/v28n4/0718-3305-ingeniare-28-04-654.pdf

[9] J. Martínez-Erazo, A. Florez-Gómez, and F. Pino, “Generando productos software mantenibles desde el proceso de desarrollo: El modelo de referencia MANTuS Creating maintainable software products from the development process: The reference model MANTuS,” 2016. Accessed: Apr. 27, 2024. [Online]. Available: https://www.scielo.cl/pdf/ingeniare/v24n3/art07.pdf

[10] “Home | Computación UNL.” Accessed: Apr. 27, 2024. [Online]. Available: https://computacion.unl.edu.ec/

[11] “Computación | Universidad Nacional de Loja.” Accessed: Apr. 27, 2024. [Online]. Available: https://unl.edu.ec/oferta_academica/facultad-de-la-energia-las-industrias-y-los-recursos-naturales-no-renovables/computacion-2019

[12] P. Roa, C. Morales, and P. Gutiérrez, “Norma ISO/IEC 25000 ISO/IEC 25000 Standard,” 2015. [Online]. Available: http://revistas.udistrital.edu.co/ojs/index.php/tia

[13] E. Reina and S. Patiño, “Evaluación de la calidad en uso de un sistema web/ móvil de control de asistencia a clases de docentes y estudiantes aplicando la norma ISO/IEC 25000 SQuaRe,” 2019. Accessed: Apr. 16, 2024. [Online]. Available: https://www.researchgate.net/publication/335754151_Evaluacion_de_la_calidad_en_uso_de_un_sistema_web_movil_de_control_de_asistencia_a_clases_de_docentes_y_estudiantes_aplicando_la_norma_ISOIEC_25000_SQuaRe

[14] “Modelo de medición y evaluación de calidad del software basado en la norma ISO/IEC 25000 para medir la usabilidad en productos de software académicos universitarios”.

[15] P. V. Mario, “Calidad de Sistemas de Información.” Accessed: Mar. 13, 2024. [Online]. Available: https://www.ebsco.com/find-my-organization?returnUrl=https%3a%2f%2fsearch.ebscohost.com%2fwebauth%2fLogin.aspx%3fdirect%3dtrue%26db%3dnlebk%26AN%3d

[16] M. Callejas, A. Alarcón, and A. Álvarez, “Modelos de calidad del software, un estado del arte,” 2020, Accessed: Mar. 13, 2024. [Online]. Available: https://www.redalyc.org/journal/2654/265452747018/html/

[17] D. V. Paucar Bernardo, P. T. Acho Santillan, and C. S. Peralta Delgado, “Relación de la gestión de riesgos y calidad de software realizados por los profesionales del Colegio de Ingenieros del Perú del Consejo Departamental de Lima,” *Interfases*, no. 014, 2021, doi: 10.26439/interfases2021.n014.5111.

[18] ISO 25000, “La familia de normas ISO/iec 25000.” Accessed: Mar. 13, 2024. [Online]. Available: https://iso25000.com/index.php/normas-iso-25000

[19] Y. Sifuentes and J. Peralta, “Modelo de medición y evaluación de calidad del software basado en la norma ISO/IEC 25000 para medir la usabilidad en productos de software académicos universitarios,” 2022, Accessed: Mar. 13, 2024. [Online]. Available: https://dialnet.unirioja.es/descarga/articulo/8510614.pdf

[20] C. Calero, Á. Moraga, and M. G. Piattini, “Calidad Del Producto Y Proceso Software.” Accessed: Mar. 18, 2024. [Online]. Available: https://www.google.com.ec/books/edition/Calidad_Del_Producto_Y_Proceso_Software/MY0zoXYFVd8C?hl=es-419&gbpv=1&dq=evaluacion+de+software+o+calidad+libros&printsec=frontcover

[21] J. Guillén, “Tendencias de Calidad de Software 2024.” Accessed: Apr. 21, 2024. [Online]. Available: https://www.linkedin.com/pulse/tendencias-de-calidad-software-2024-jessica-guill%C3%A9n-c--ttppc/

[22] ISTQB, “Probador Certificado del ISTQB ®,” 2018.

[23] L. Castillo, “Evaluación, calidad y gestión de calidad total en Documentación”, Accessed: Apr. 21, 2024. [Online]. Available: https://www.uv.es/macas/T10.pdf

[24] J. Luque, “Métricas de Productividad Del Software para La Gestión de Proyectos.” Accessed: Mar. 18, 2024. [Online]. Available: https://es.scribd.com/document/473430073/3-Metricas-de-productividad-del-software-para-la-gestion-de-proyectos-Claveras-2015

[25] D. Galin, “Aseguramiento de la calidad del software,” 2019, Accessed: Mar. 19, 2024. [Online]. Available: https://sedici.unlp.edu.ar/bitstream/handle/10915/3956/3_-_Aseguramiento_de_la_calidad_del_software.pdf?sequence=11&isAllowed=y

[26] A. Antonio, “GESTIÓN, CONTROL Y GARANTÍA DE LA CALIDAD DEL SOFTWARE,” 2018. Accessed: Mar. 19, 2024. [Online]. Available: https://www.academia.edu/10940209/GESTI%C3%93N_CONTROL_Y_GARANT%C3%8DA_DE_LA_CALIDAD_DEL_SOFTWARE

[27] R. Pressman, “Libro Capitulo 19- MetricasTecnicas del Software,” 2012, Accessed: Apr. 16, 2024. [Online]. Available: https://www.uv.mx/personal/asumano/files/2012/08/MetricasTecnicas.pdf

[28] E. Castaño and W. Castillo, “Métricas en la evaluación de la calidad del software,” *Computer and Electronic Sciences: Theory and Applications*, vol. 2, no. 2, pp. 21–26, Dec. 2021, doi: 10.17981/cesta.02.02.2021.03.

[29] M. Callejas, A. Alarcón, and A. Álvarez, “Vista de Modelos de calidad del software.” Accessed: Mar. 18, 2024. [Online]. Available: https://revistas.unilibre.edu.co/index.php/entramado/article/view/428/339

[30] Instituto Antioqueño de Investigación, “La Prueba del Software como Ciencia,” 2013. Accessed: Apr. 21, 2024. [Online]. Available: https://www.studocu.com/es-ar/document/universidad-siglo-21/ingenieria-en-sistemas/la-prueba-del-software-como-ciencia/55175781/download/la-prueba-del-software-como-ciencia.pdf

[31] I. Sommerville, *Libro- Software engineering*. Pearson, 2011. Accessed: Apr. 21, 2024. [Online]. Available: https://engineering.futureuniversity.com/BOOKS%20FOR%20IT/Software-Engineering-9th-Edition-by-Ian-Sommerville.pdf

[32] L. Baquero, “P R I M E R A E D I C I Ó N G U Í A R Á P I D A D E FUNDAMENTOS DEL TESTING ™,” 2023.

[33] Thegeek, “La importancia de las pruebas estaticas en el desarrollo de Software – Geek QA.” Accessed: May 12, 2024. [Online]. Available: https://geekqa.net/la-importancia-de-las-pruebas-estaticas-en-el-desarrollo-de-software/

[34] M. Capobianco, S. Gottifredi, and G. Simari, “Profesora titular de la cátedra,” 2019. [Online]. Available: http://www.gnu.org/copyleft/fdl.html

[35] F. Ruiz, “Mantenimiento del Software,” 2001. [Online]. Available: http://alarcos.inf-cr.uclm.es/doc/mso/

[36] Mantenibilidad- ISO/IEC 25000, “Mantenibilidad- ISO/IEC 25000.” Accessed: Apr. 05, 2024. [Online]. Available: https://iso25000.com/index.php/normas-iso-25000/iso-25010/26-mantenibilidad

[37] L. Tamayo and N. Silega, “Gestión de la mantenibilidad desde etapas tempranas en el desarrollo de software,” 2020, Accessed: Apr. 05, 2024. [Online]. Available: http://scielo.sld.cu/pdf/rcci/v15n1/2227-1899-rcci-15-01-52.pdf

[38] J. M. Ruiz, C. D. Pacifico, and M. M. Pérez, “Clasificación y Evaluación de Métricas de Mantenibilidad Aplicables a Productos de Software Libre,” 2019. Accessed: Mar. 25, 2024. [Online]. Available: https://sedici.unlp.edu.ar/bitstream/handle/10915/61928/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y

[39] M. Callejas-Cuervo, A. C. Alarcón-Aldana, and A. M. Álvarez-Carreño, “Modelos de calidad del software, un estado del arte,” *ENTRAMADO*, vol. 13, no. 1, pp. 236–250, 2017, doi: 10.18041/entramado.2017v13n1.25125.

[40] S. Rugel and J. Chacón, “Artículo de Revisión. Teorías, Modelos y Sistemas de Gestión de Calidad and Systems of Quality Management,” 2018. [Online]. Available: https://www.researchgate.net/publication/331544414

[41] M. Julia Blas, S. Gonnet, and H. Leone, “Especificación de la Calidad en Software-as-a-Service: Definición de un Esquema de Calidad basado en el Estándar ISO/IEC 25010,” 2016. Accessed: Apr. 22, 2024. [Online]. Available: https://core.ac.uk/download/pdf/301073189.pdf

[42] T. Vaca and A. Jácome, “Calidad de software del módulo de talento humano del sistema informático de la Universidad Técnica del Norte bajo la norma ISO/IEC 25000,” 2018. Accessed: Apr. 22, 2024. [Online]. Available: https://www.researchgate.net/publication/325022337_Calidad_de_software_del_modul

o_de_talento_humano_del_sistema_informatico_de_la_Universidad_Tecnica_del_Norte_bajo_la_norma_ISOIEC_25000

[43] M. Constanzo, “Comparacion de Modelos de Calidad, Factores y Metricas en el Ambito de la Ingenieria de Software,” 2014, Accessed: Apr. 22, 2024. [Online]. Available: https://publicaciones.unpa.edu.ar/index.php/ICTUNPA/article/view/395/374

[44] A. López, C. Cabrera, and L. Valencia, “INTRODUCCIÓN A LA CALIDAD DE SOFTWARE,” 2008, [Online]. Available: http://buscon.rae.es/draeI/html/cabecera.htm,

[45] C. Alma, “Calidad del software, el camino al éxito,” 2009, doi: 10.13140/2.1.2684.0322.

[46] M. Callejas-Cuervo, A. C. Alarcón-Aldana, and A. M. Álvarez-Carreño, “Modelos de calidad del software, un estado del arte,” *ENTRAMADO*, vol. 13, no. 1, pp. 236–250, 2017, doi: 10.18041/entramado.2017v13n1.25125.

[47] N. Bevan, “Calidad Del Producto Y Proceso Software - Google Libros.” Accessed: Apr. 23, 2024. [Online]. Available: https://books.google.com.ec/books?id=MY0zoXYFVd8C&pg=PA55&lpg=PA55&dq=Los+nuevos+modelos+de+ISO+para+la+calidad+y+la+calidad+en+uso+del+software,+Calidad+del+producto+y+proceso+software,+BEVAN+y+Nigel&source=bl&ots=VijlipMBQK&sig=ACfU3U1KLmEZei5pIa0MAcaf3NGCZ31nzA&hl=es-419&sa=X&ved=2ahUKEwjm-easkduFAxWtezABHUn7AvsQ6AF6BAgnEAM#v=onepage&q=Los%20nuevos%20modelos%20de%20ISO%20para%20la%20calidad%20y%20la%20calidad%20en%20uso%20del%20software%2C%20Calidad%20del%20producto%20y%20proceso%20software%2C%20BEVAN%20y%20Nigel&f=false

[48] A.-A. Catherine-Andrea, G.-S. Juan-Sebastián, and R.-T. Sandra-Lucía, “ISO/IEC 15504-Guide for Small and Medium-Sized Enterprises of Software Development Guide d’après le standard ISO/IEC 15504 pour Petites et Moyennes Entreprises qui développent des logiciels,” 2011. [Online]. Available: http://revistavirtual.ucn.edu.co/],

[49] J. Baldeón, “MÉTODO PARA LA EVALUACIÓN DE CALIDAD DE SOFTWARE BASADO EN ISO/IEC 25000,” 2015. Accessed: Apr. 23, 2024. [Online]. Available: https://repositorio.usmp.edu.pe/bitstream/handle/20.500.12727/1480/baldeon_vej.pdf?isAllowed=y&sequence=1

[50] ISO/IEC 25010: 2011, “ISO/IEC 25010:2011(en), Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.” Accessed: Apr. 24, 2024. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en

[51] ISO/IEC 25023, “ISO/IEC 25023:2016(en), Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Measurement of system and software product quality.” Accessed: Apr. 24, 2024. [Online]. Available: https://www.iso.org/obp/ui/es/#iso:std:iso-iec:25023:ed-1:v1:en

[52] J. Calabrese, S. Esponda, A. Pasini, M. Boracchia, and P. Pesado, “Guía para evaluar calidad de datos basada en ISO/IEC 25012,” 2019. Accessed: Apr. 04, 2024. [Online]. Available: https://core.ac.uk/download/pdf/301104068.pdf

[53] G. Ruiz, A. Peña, C. Castro, A. Alaguna, L. Areiza, and R. Rincón, “Modelo de Evaluación de Calidad de Software Basado en Lógica Difusa, Aplicada a Métricas de Usabilidad de Acuerdo con la Norma ISO/IEC 9126,” *2006,* Accessed: Apr. 04, 2024. [Online]. Available: https://www.redalyc.org/pdf/1331/133114988005.pdf

[54] S. Yenny and P. José, “Modelo de medición y evaluación de calidad del software basado en la norma ISO/IEC 25000,” *TecnoHumanismo,* vol. 1, no. 5, pp. 34–58, 2021, doi: 10.53673/th.v1i5.25.

[55] ISO/IEC 25040:, “ISO/IEC 25040:2011(en), Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Evaluation process.” Accessed: Apr. 28, 2024. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:25040:ed-1:v1:en

[56] Universidad del Valle Fraser, “LA FAMILIA DE NORMAS ISO 25000 - SQuaRE - College Sidekick.” Accessed: Mar. 29, 2024. [Online]. Available: https://www.collegesidekick.com/study-docs/1784464

[57] E. Lluna, “Análisis estático de código en el ciclo de desarrollo de software de seguridad crítica,” 2011. Accessed: Apr. 07, 2024. [Online]. Available: https://www.redalyc.org/pdf/922/92222551004.pdf

[58] M. Ganuza, “INGENIERÍA DE APLICACIONES Verificación y Validación de Software,” 2019. Accessed: May 13, 2024. [Online]. Available:

http://www.cs.uns.edu.ar/~mlg/iap/downloads/Clases%20Teoricas/08_TestingDeSw_IAP_2019.pdf

[59] L. Pauta Ayabaca and S. M. Bernal, “Verificación y Validación de Software Software Verification and Validation,” *Revista Killkana Técnica*, vol. 1, no. 3, pp. 25–32, 2017, Accessed: May 13, 2024. [Online]. Available: https://killkana.ucacue.edu.ec/index.php/killkana_tecnico/article/download/112/148/369

[60] L. Aguirre and N. Serna, “SiStemaS e informática.” Accessed: May 13, 2024. [Online]. Available: https://www.redalyc.org/pdf/816/81643819019.pdf

[61] M. L. Larrea, “El Análisis Estático como Herramienta de Evaluación en Cátedras con Proyectos de Programación,” 2021. Accessed: Apr. 07, 2024. [Online]. Available: https://sedici.unlp.edu.ar/bitstream/handle/10915/122678/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y

[62] L. Rodríguez, “Herramientas para análisis estático de seguridad: estado del arte,” 2021. Accessed: Apr. 07, 2024. [Online]. Available: https://owasp.org/www-pdf-archive/OWASP_Spain_20080314_Herramientas_de_an%C3%A1lisis_est%C3%A1tico_de_seguridad_del_c%C3%B3digo_estado_del_arte.pdf

[63] SonarQube, “Herramienta de análisis estático, seguridad y calidad de código | SonarQube.” Accessed: May 14, 2024. [Online]. Available: https://www.sonarsource.com/products/sonarqube/

[64] O. Juan-Pablo, “Análisis de seguridad y calidad de aplicaciones (Sonarqube),” 2015. Accessed: May 14, 2024. [Online]. Available: https://openaccess.uoc.edu/bitstream/10609/43263/3/gnupabloTFM0715memoria.pdf

[65] Panel SQA and Testing, “El manual de la calidad software - Panel Ssistemas - 2ªedicion2015 - 20240515_090852,” pp. 41–50, 2024, Accessed: May 14, 2024. [Online]. Available: https://es.slideshare.net/MktPanel/el-manual-de-la-calidad-software-panel-sistemas-2edicion2015

[66] B. Pajuelo, “Análisis estático utilizando la herramienta Sonarqube según el modelo de calidad ISO/IEC 25010,” 2018. Accessed: May 14, 2024. [Online]. Available: https://ilide.info/doc-viewer-v2#google_vignette

[67] L. Albea, “Normativa de Análisis del Código con SONAR OAC-Oficina de Calidad,” 2022. Accessed: May 14, 2024. [Online]. Available: https://www.juntadeandalucia.es/medioambiente/portal/documents/13391456/31168455/Normativa+Calidad+C%C3%B3digo+con+SonarQube.pdf/acad4aa9-1eb7-e34e-72e8-f817f11d7706?t=1641813532571

[68] H.-A. Vera-Celis, “Estudio Funcional de la plataforma ‘SONARQUBE’ para la evaluación de código fuente con respecto a la calidad de producto de software software,” 2019. Accessed: May 14, 2024. [Online]. Available: http://repositoriodspace.unipamplona.edu.co/jspui/bitstream/20.500.12744/5601/1/VERA_2019_TG.pdf

[69] M. V. Galán Pausic, “Estudio de la plataforma SonarQube para la implantación de ISO/IEC 25000.” Accessed: May 14, 2024. [Online]. Available: https://books.google.com.ec/books/about/Estudio_de_la_plataforma_SonarQube_para.html?id=VNqUtAEACAAJ&redir_esc=y

[70] I. Figueroa, “Breve Guía Buenas Prácticas de Programación en C,” 2019. Accessed: Apr. 07, 2024. [Online]. Available: https://www.academia.edu/42119980/Breve_Gu%C3%ADa_Buenas_Pr%C3%A1cticas_de_Programaci%C3%B3n_en_C

[71] R. Dumitru Boboia, “Herramientas y buenas prácticas para el desarrollo, mantenimiento y evolución de Software en Java,” 2019. Accessed: Apr. 07, 2024. [Online]. Available: https://oa.upm.es/55663/1/TFG_RADU_DUMITRU_BOBOIA.pdf

[72] S. McConnell, “Resumen de ‘ Cap 1 Código Completo,’” 2024.

[73] Q. R. Nora, “Estudio sobre Metodologías Ágiles en los Proyectos Software. Propuesta de Plan de Implantación para PYMES.,” 2020. Accessed: Apr. 07, 2024. [Online]. Available: https://biblus.us.es/bibing/proyectos/abreproy/71934/descargar_fichero/TFM-1934+QUESADA+REYES%2C+NORA.pdf

[74] C. Calderín, I. Yanislaidi, E. Blanco, and A. López, “Seminario de Ingeniería de Software,” 2017. Accessed: Apr. 08, 2024. [Online]. Available:

https://www.researchgate.net/publication/321398240_Metodologias_de_desarrollo_de_software_XP

[75] E. G. Maida and J. Pacienzia, “Metodologías de desarrollo de software,” 2015. Accessed: Apr. 09, 2024. [Online]. Available: https://repositorio.uca.edu.ar/bitstream/123456789/522/1/metodologias-desarrollo-software.pdf

[76] R. Pressman, “Software Engineering,” 2005. Accessed: Jun. 18, 2024. [Online]. Available: https://repository.dinus.ac.id/docs/ajar/Software_Engineering_-_Pressman.pdf

[77] S. Kurkovsky, S. Ludi, and L. Clark, “Active learning with LEGO for software requirements,” in *SIGCSE 2019 - Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, Inc, Feb. 2019, pp. 218–224. doi: 10.1145/3287324.3287444.

[78] R. Pressman, “Software Engineering,” 2005. Accessed: Jun. 18, 2024. [Online]. Available: https://repository.dinus.ac.id/docs/ajar/Software_Engineering_-_Pressman.pdf

[79] R. Pressman, “Software Engineering: A Practitioner’s Approach,” 2005. [Online]. Available: www.mhhe.com/pressman.

[80] Y. Molina, A. Granda, and A. Velázquez, “Los requisitos no funcionales de software. Una estrategia para su desarrollo en el Centro de Informática Médica,” 2019, Accessed: Jun. 18, 2024. [Online]. Available: http://scielo.sld.cu/pdf/rcci/v13n2/2227-1899-rcci-13-02-77.pdf

[81] H. Cervantes, P. Velasco, and L. Castro, “2015_ArquitecturaDeSoftware-C1-3,” 2016, Accessed: Jun. 18, 2024. [Online]. Available: https://www.researchgate.net/profile/Perla-Velasco-Elizondo/publication/281137715_Arquitectura_de_Software_Conceptos_y_Ciclo_de_Desarrollo/links/57144e1408aeebe07c0641ab/Arquitectura-de-Software-Conceptos-y-Ciclo-de-Desarrollo.pdf

[82] M. Blas and S. Gonnet, “Using Model-to-Model Transformations for Web Software Architecture Simulation,” 2022, Accessed: Jun. 18, 2024. [Online]. Available: https://latamt.ieeer9.org/index.php/transactions/article/view/5222/1243

[83] A. Clementi, E. Natale, and I. Ziccardi, “Parallel Load Balancing on Constrained Client-Server Topologies,” in *Annual ACM Symposium on Parallelism in Algorithms and Architectures*, Association for Computing Machinery, Jul. 2020, pp. 163–173. doi: 10.1145/3350755.3400232.

[84] M. (W. M. Richards, *Software architecture patterns : understanding common architecture patterns and when to use them*. 2015. Accessed: Jun. 18, 2024. [Online]. Available: https://51world.win/pic_flow/2017/2017072801-software-architecture-patterns.pdf

[85] R. Celi, M. Boné, and A. Mora, “Programación Web-Frontend Backend,” 2023, Accessed: Jun. 18, 2024. [Online]. Available: https://dialnet.unirioja.es/descarga/libro/933116.pdf

[86] “Tecnologías web frontend y backend en sitios de redes sociales: Facebook como ejemplo.” Accessed: Jun. 18, 2024. [Online]. Available: https://www.infona.pl/resource/bwmeta1.element.ieee-art-000007076874

[87] I. Ahmad, E. Suwarni, R. I. Borman, Asmawati, F. Rossi, and Y. Jusman, “Implementation of RESTful API Web Services Architecture in Takeaway Application Development,” in *2021 1st International Conference on Electronic and Electrical Engineering and Intelligent System, ICE3IS 2021*, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 132–137. doi: 10.1109/ICE3IS54102.2021.9649679.

[88] Node JS, “Node.js: JavaScript.” Accessed: Apr. 07, 2024. [Online]. Available: https://nodejs.org/en

[89] Vite, “Introducción | Vite.” Accessed: Jul. 21, 2024. [Online]. Available: https://es.vitejs.dev/guide/

[90] L. Valencia Cabrera lvalencia and V. Ramos González, “Introducción a MongoDB,” 2022.

[91] A. S. Acharya, A. Prakash, P. Saxena, and A. Nigam, “Sampling: why and how of it?,” *Indian Journal of Medical Specialities*, vol. 4, no. 2, Jul. 2013, doi: 10.7713/ijms.2013.0032.

[92] O. Hernández González, “Aproximación a los distintos tipos de muestreo no probabilístico que existen,” 2021. [Online]. Available: http://www.revmgi.sld.cu/index.php/mgi/article/view/907

[93] H. Felbinger, F. Wotawa, and M. Nica, “Adapting unit tests by generating combinatorial test data,” in *Proceedings - 2018 IEEE 11th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2018*, Institute of Electrical and Electronics Engineers Inc., Jul. 2018, pp. 352–355. doi: 10.1109/ICSTW.2018.00072.

[94] S. D. Garomssa, R. Kannan, I. Chai, and D. Riehle, “How Software Quality Mediates the Impact of Intellectual Capital on Commercial Open-Source Software Company Success,” *IEEE Access*, vol. 10, pp. 46490–46503, 2022, doi: 10.1109/ACCESS.2022.3170058.

[95] J. S. Rudas, L. M. Gómez, and A. O. Toro, “Revisión sistemática de literatura. Caso de estudio: Modelamiento de un par deslizante con fines de predecir desgaste Systematic literature reviews. Case of study: Modeling of a rubbing pair for wear purpose,” 2013.

[96] J. Higgins and S. Green, “Manual Cochrane de revisiones sistemáticas de intervenciones,” 2011. [Online]. Available: www.cochrane-handbook.org.

[97] B. Kitchenham and P. Brereton, “A systematic review of systematic review process research in software engineering,” 2013, *Elsevier B.V.* doi: 10.1016/j.infsof.2013.07.010.

[98] E. Balseca, “Evaluación de Calidad de un Producto de Software,” 2016. Accessed: Apr. 03, 2024. [Online]. Available: https://sedici.unlp.edu.ar/bitstream/handle/10915/58934/Documento_completo.pdf?sequence=3

[99] S. Pardo, “Mantenibilidad de productos de software según el modelo Square ISO/IEC 25000,” 2018.

[100] D. Mena, “Análisis de mantenibilidad y portabilidad del Framework react native aplicando la norma iso/iec 25010 mediante un caso de estudio en la aplicación de Gestión de eventos oaq.,” 2020. Accessed: Jun. 17, 2024. [Online]. Available: https://bibdigital.epn.edu.ec/bitstream/15000/21372/1/CD%2010421.pdf

[101] J. Leiva and W. Sánchez, “Investigación y selección de norma IEEE e ISO para aportar control de calidad en la fase de análisis de desarrollo de Software, en aplicaciones web desarrolladas por el área de TI de CoopMego,” Loja, Universidad Nacional de Loja, 2023. Accessed: Jul. 01, 2023. [Online]. Available: https://dspace.unl.edu.ec/jspui/bitstream/123456789/26979/1/JorgeDaniel_LeivaPaladinez%20_%20WilsonAntonio_SanchezCarrion.pdf

[102] M. Rodríguez and M. Piattini, “Experiencias en la Industria del Software: Certificación del Producto con ISO/IEC 25000,” 2015. Accessed: May 15, 2024. [Online]. Available: https://eventos.spc.org.pe/cibse2015/pdfs/01_IT15.pdf

[103] M. Rodríguez, Ó. Pedreira, and C. M. Fernández, “Certificación de la Mantenibilidad del Producto Software: Un Caso Práctico,” 2015, Accessed: May 15, 2024. [Online]. Available: https://www.researchgate.net/publication/283699208_Certificacion_de_la_Mantenibilidad_del_Producto_Software_Un_Caso_Practico

[104] E. Irrazábal, “Construcción de un Entorno para la Medición Automatizada de la Calidad de los Productos Software,” 2012. Accessed: May 16, 2024. [Online]. Available: https://burjcdigital.urjc.es/bitstream/handle/10115/11880/Tesis_Emanuel_Irrazabal.pdf?sequence=1&isAllowed=y

[105] Y. Sifuentes and J. Peralta, “Modelo de medición y evaluación de calidad del software basado en la norma ISOIEC 25000 para medir la usabilidad en productos de software académicos universitarios,” 2022, Accessed: Nov. 19, 2024. [Online]. Available: https://tecnohumanismo.online/index.php/tecnohumanismo/article/view/125

[106] J. Leiva and W. Sánchez, “Investigación y selección de norma IEEE e ISO para aportar control de calidad en la fase de análisis de desarrollo de Software, en aplicaciones web desarrolladas por el área de TI de CoopMego,” Loja, Universidad Nacional de Loja, 2023. Accessed: Jul. 01, 2023. [Online]. Available:

https://dspace.unl.edu.ec/jspui/bitstream/123456789/26979/1/JorgeDaniel_LeivaPaladinez%20_%20WilsonAntonio_SanchezCarrion.pdf

[107] Y. Sifuentes and J. Peralta, “Modelo de medición y evaluación de calidad del software basado en la norma ISOIEC 25000 para medir la usabilidad en productos de software académicos universitarios,” 2022, Accessed: Nov. 19, 2024. [Online]. Available: https://tecnohumanismo.online/index.php/tecnohumanismo/article/view/125

[108] P. Becker, L. Olsina, and F. Papa, “Especificación de Métricas de Mantenibilidad para Mejorar Código Java: Caso Aplicado en un Curso Avanzado de Ingeniería en Sistemas,” 2023, Accessed: Nov. 22, 2024. [Online]. Available: https://www.researchgate.net/publication/374504878_Especificacion_de_Metricas_de_Mantenibilidad_para_Mejorar_Codigo_Java_Caso_Aplicado_en_un_Curso_Avanzado_de_Ingenieria_en_Sistemas

[109] C. J. Ríos Simancas, G. Suing-Albito, and B. L. Iñiguez Auquilla, “Prototipo de herramienta web para la intervención y estimulación psicopedagógica de niños con TDAH,” *Revista Tecnológica - ESPOL*, vol. 36, no. 1, pp. 98–122, Jun. 2024, doi: 10.37815/rte.v36n1.1162.

[110] A. E. Soraluz Soraluz, M. Á. Valles Coral, and D. Levano Rodríguez, “Desarrollo guiado por comportamiento: buenas prácticas para la calidad de software,” *Ingeniería y Desarrollo*, vol. 39, no. 01, pp. 190–204, Apr. 2023, doi: 10.14482/inde.39.1.005.3.

[111] Y. Niño, M. del R. Morales Salgado, S. Vazquez Reyes, and B. E. Sánchez Rinza, “Best practices and quality criteria in the coding development process in higher education programming courses,” 2020. Accessed: Nov. 24, 2024. [Online]. Available: https://www.researchgate.net/publication/350418323_Mejores_practicas_y_criterios_de_calidad_en_el_proceso_de_desarrollo_de_codigo_en_los_cursos_de_programacion_en_la_ensenanza_superior

[112] G. del R. Veloz Remache, J. A. Menéndez Verdecia, and L. N. Aguilar Moncayo, “Ciencias Técnicas y Aplicadas Artículo de investigación,” vol. 6, pp. 656–668, 2021, doi: 10.23857/pc.v6i1.2170.

[113] L. del C. Ramírez and A. Smith Flórez, “BUENAS PRÁCTICAS, UNA SOLUCIÓN PARA UN MEJOR DESARROLLO DE SOFTWARE GOOD PRACTICE, A

SOLUTION FOR BETTER SOFTWARE DEVELOPMENT," 2014. Accessed: Nov. 24, 2024. [Online]. Available: https://dialnet.unirioja.es/descarga/articulo/5109243.pdf

[114] D. Campo, "Technological acceptance model (TAM) in the Physics Laboratory III based on the Internet of Things in the Systems Engineering Program of the University of Cartagena," *Educación • Education • Educação •*, vol. 41, [Online]. Available: https://www.revistaespacios.com

[115] A. Samouti and M. Fathy, "Field Study of the Acceptability of NFC Near-Field Communication Technology in Iran Based on TAM Methodology," in *Proceeding of 4th International Conference on Smart Cities, Internet of Things and Applications, SCIoT 2020*, Institute of Electrical and Electronics Engineers Inc., Sep. 2020, pp. 12–17. doi: 10.1109/SCIOT50840.2020.9250201.

11. Anexos

Anexo 1. Extracción de datos utilizando matrices bibliográficas

1. Datos recopilados

Tabla 1-A1. Matriz biográfica 1.

Atributo	Descripción
Tipo documento	Artículo
Referencia	https://www.scielo.cl/pdf/ingeniare/v28n4/0718-3305-ingeniare-28-04-654.pdf.
Resumen	"Las organizaciones actuales utilizan productos de software para sus procesos de negocio, los cuales requieren mantenimiento durante su vida útil. Este artículo revisa bibliografía para resaltar la importancia de especificar claramente la mantenibilidad en los requisitos del software, abordando desafíos, métricas y recomendaciones para su correcta especificación. Se concluye que las organizaciones deben recopilar datos históricos sobre las modificaciones del software para decidir si mantener o reemplazar el producto, y que los requisitos deben incluir aspectos funcionales y de proceso para validar objetivamente su mantenibilidad.".
Tema	Mantenibilidad del Software. Consideraciones para su especificación y validación.
Propósito	Presentar algunas consideraciones importantes sobre el proceso de mantenimiento de software, que se deben tener en cuenta a la hora de especificar el atributo de mantenibilidad de tal forma que se pueda validar que efectivamente el producto es mantenible.
Aspectos de interés del documento	Métricas internas relacionadas con las subcaracterísticas de la mantenibilidad.
Año	2020

Tabla 2-A1. Matriz biográfica 2.

Atributo	Descripción
Tipo documento	Artículo
Referencia	https://eventos.spc.org.pe/cibse2015/pdfs/01_IT15.pdf
Resumen	"En los últimos años, la calidad del software ha ganado gran relevancia debido a su omnipresencia y la necesidad de asegurar su correcto funcionamiento. A pesar de los estudios existentes, la mayoría se han enfocado en la calidad de los procesos de desarrollo y no en el producto de software en sí. Actualmente, no existe una propuesta integral para la evaluación y certificación de la calidad del software basada en la nueva familia de normas ISO/IEC 25000. Este artículo presenta un caso de estudio sobre la evaluación, mejora y certificación de la mantenibilidad de un software, involucrando a una empresa desarrolladora, un laboratorio de evaluación acreditado y una entidad de certificación. Se describen las características de estas evaluaciones, los roles de las entidades participantes y el proceso desde la evaluación inicial hasta la certificación final."
Tema	Experiencias en la Industria del Software: Certificación del Producto con ISO/IEC 25000.
Propósito	Presentar un conjunto de experiencias en la evaluación y certificación de productos software en la industria, junto con el ecosistema de entidades que han participado en el proceso. También se describe el proceso de evaluación.
Aspectos de interés del documento	Proceso de evaluación basado en la norma ISO/IEC 25000.
Año	2015

Tabla 3-A1. Matriz biográfica 3.

Atributo	Descripción
Tipo documento	Tesis doctoral
Referencia	https://burjcdigital.urjc.es/bitstream/handle/10115/11880/Tesis_Emanuel_Irrazabal.pdf?sequence=1&isAllowed=y
Resumen	"Esta Tesis Doctoral presenta un entorno basado en software libre para medir la mantenibilidad del software en niveles operativo, táctico y estratégico, y un soporte metodológico para evaluar dicha mantenibilidad, calculando el valor y el retorno de inversión de las refactorizaciones necesarias. El entorno KEMIS, desarrollado como parte de esta tesis, ofrece una solución automatizada y panificable para empresas, especialmente aquellas que han externalizado el desarrollo de software y necesitan controlar la calidad de sus productos de manera efectiva. La implementación de KEMIS en entornos reales ha demostrado su valor, obteniendo una amplia aceptación y destacando las dificultades que enfrentan las empresas al mejorar manualmente el código fuente. Esta herramienta ayuda a las empresas a reducir el riesgo asociado con la priorización informal de mejoras de software, proporcionando un modelo sistemático para la toma de decisiones informadas. Futuras investigaciones se centrarán en los beneficios específicos de cada mejora y en la automatización de los indicadores de valor."
Tema	Construcción de un Entorno para la Medición Automatizada de la Calidad de los Productos Software.
Propósito	Desarrollar y proponer un entorno basado en software libre, denominado KEMIS, para medir y evaluar la mantenibilidad del software en niveles operativo, táctico y estratégico. Además, busca ofrecer un soporte metodológico que permita calcular el valor y el retorno de inversión de las refactorizaciones necesarias, ayudando a las empresas a controlar la calidad de sus productos de software de manera automatizada y panificable, especialmente en contextos de desarrollo externalizado.
Aspectos de interés del documento	Proceso de medición de mantenibilidad basado en la norma ISO/IEC 25000.
Año	2012

Tabla 4-A1. Matriz biográfica 4.

Atributo	Descripción
Tipo documento	Artículo
Referencia	https://books.google.com.ec/books/about/Estudio_de_la_plataforma_SonarQube_para.html?id=VNqUtAEACAAJ&redir_esc=y
Resumen	"La calidad de un sistema se define por su capacidad para satisfacer las necesidades explícitas e implícitas de sus interesados, proporcionando valor. Estas necesidades se estructuran en la serie SQuaRE mediante modelos que clasifican la calidad del producto en características y subcaracterísticas. Para medir estas características, es necesario identificar y medir una colección de propiedades específicas y combinarlas computacionalmente. Este trabajo examina el modelo y las métricas de la familia de normas ISO/IEC 25000, y utiliza la herramienta de software libre SonarQube para medir propiedades de calidad de forma estática a partir del código fuente. Se analiza la cobertura de SonarQube respecto a las características del modelo ISO/IEC 25000, y se proponen fórmulas para calcular la Mantenibilidad, Fiabilidad, Vulnerabilidad y Rendimiento, según la norma ISO/IEC 25010. La metodología se ilustra con un ejemplo práctico."
Tema	Estudio de la plataforma SonarQube para la implantación de ISO/IEC 25000.
Propósito	El propósito del trabajo es evaluar la calidad de un sistema en función de su capacidad para satisfacer las necesidades de los interesados, utilizando el modelo SQuaRE de la familia de normas ISO/IEC 25000. Se analizan las métricas de calidad y se emplea la herramienta SonarQube para medir propiedades del código fuente estáticamente. El trabajo propone fórmulas para calcular la mantenibilidad, fiabilidad, vulnerabilidad y rendimiento del software, basándose en la norma ISO/IEC 25010, y se ilustra la metodología con un ejemplo práctico.
Aspectos de interés del documento	Proceso o modelo de evaluación basado en la norma ISO/IEC 25000.
Año	2017

Anexo 2. Análisis y selección de lenguaje de programación como enfoque de estudio

En la siguiente sección se muestra el procedimiento para seleccionar los dos lenguajes de programación Predominantes en el desarrollo de aplicaciones web de la Carrera de Ingeniería Sistemas/Computación de la UNL.

1. Introducción

En el contexto del desarrollo de aplicaciones web en la carrera, se ha realizado un meticuloso análisis para determinar los lenguajes de programación más frecuentemente empleados. Este informe se propone seleccionar dos de estos lenguajes con el fin de abordar de manera efectiva los defectos de mantenibilidad que puedan surgir en proyectos futuros, mediante pautas de programación y evaluación del grado de mantenibilidad según los dos lenguajes predominantes.

2. Recopilación de Datos

Se revisaron diversas memorias de TT y TIC en el DSPACE de la Universidad Nacional de Loja, de aplicaciones web desarrollados por estudiantes de la carrera ingeniería en Sistemas/Computación.

En la **Tabla 1-A2** se presenta un resumen de las memorias que tiene como objetivo desarrollar aplicaciones web o módulos, aplicando un filtro del año 2020 en adelante:

Tabla 1-A2. Memorias de TT y TIC que implementan aplicaciones web.

N.	Título del TT o TIC	LP	Link dspace
1	Gestión del Ciclo de Vida del Desarrollo del Software (SDLC) en la Carrera de Ingeniería en Sistemas/Computación de la UNL.	JavaScript	https://dspace.unl.edu.ec/jspui/handle/123456789/25343
2	Desarrollo de una aplicación web que permita al departamento de talento humano el registro de permisos y consulta de vacaciones disponibles de los funcionarios de la Dirección Distrital 05D03 Pangua - La Maná – Salud.	Java	https://dspace.unl.edu.ec/jspui/handle/123456789/26959
3	Aplicación Web para la automatización del cobro de consumo de agua potable en el Gobierno Autónomo Descentralizado Rural de la Parroquia Milagro del Cantón Atahualpa.	Python	https://dspace.unl.edu.ec/jspui/bitstream/123456789/26632/1/EdhissonAlexis_SanmartInFreire-.pdf
4	Aplicación web para la gestión del portafolio docente en Institutos Tecnológicos de Educación Superior de la ciudad de Loja.	PHP	https://dspace.unl.edu.ec/jspui/bitstream/123456789/25727/1/JoseAngel_LojaJimenez.pdf

N.	Título del TT o TIC	LP	Link dspace
5	Aplicación web para la capacitación empresarial en el coworking Innova-T de la Universidad Nacional de Loja.	Javascript	https://dspace.unl.edu.ec/jspui/bitstream/123456789/28967/1/VictoriaAdelaida_CoronelUlloa_WilsonLizandro_ValverdeJadan.pdf
6	Desarrollo un aplicativo web para desplegar los resultados del trabajo investigativo de la Universidad Nacional de Loja denominado "Análisis contable-financiero y de rentabilidad en el proceso de producción de maíz en la Provincia de Loja".	Python	https://dspace.unl.edu.ec/jspui/handle/123456789/26680
7	Desarrollo de una aplicación web multiplataforma de gestión y seguimiento de pedidos para la empresa textil IKERANY.	Javascript	https://dspace.unl.edu.ec/jspui/bitstream/123456789/26955/1/DiegoJhonatan_ChambaSaca.pdf
8	Diseño de aplicación web mediante el patrón de diseño Factory para Micro-Learning del lenguaje de señas ecuatoriano.	JavaScript	https://dspace.unl.edu.ec/jspui/handle/123456789/25946
9	Desarrollo de una aplicación informática para la búsqueda de oferta de alquiler de inmuebles en la ciudad de Loja.	Dart	https://dspace.unl.edu.ec/jspui/bitstream/123456789/24650/1/SteevenMichael%20_ArmijosBravo.pdf
10	Desarrollar una Aplicación Web para la difusión y venta de obras de arte para la carrera de Artes Plásticas / Visuales de la Universidad Nacional de Loja.	JavaScript	https://dspace.unl.edu.ec/jspui/handle/123456789/28682
11	Aplicación web para el agendamiento de citas médicas en el consultorio médico San Benito de la ciudad de Catamayo	PHP	https://dspace.unl.edu.ec/jspui/handle/123456789/26703
12	Aplicación web para la gestión de la representatividad de árboles urbanos del proyecto de investigación Dinámica de crecimiento y servicios ecosistémicos del arbolado urbano de la ciudad de Loja.	PHP	https://dspace.unl.edu.ec/jspui/handle/123456789/26629
13	Desarrollo de una aplicación web progresiva para la gestión y comercialización de productos y servicios para la empresa "Compumars" de la ciudad de Loja.	JavaScript	https://dspace.unl.edu.ec/jspui/handle/123456789/26413
14	Desarrollo de una aplicación web para la gestión de materia prima y control de ventas en la empresa Breldy del cantón Yantzaza.	JavaScript	https://dspace.unl.edu.ec/jspui/bitstream/123456789/28240/3/NayoFrancisco_SalinasMinga.pdf
15	Aplicación web para determinar el perfil lipídico y su relación con el riesgo cardiovascular en pacientes diabéticos tipo 2 e hipertensos de la consulta externa del hospital ceibos.	Python	https://dspace.unl.edu.ec/jspui/bitstream/123456789/25246/1/RogerAlexander_%20TorresYaguana.pdf

N.	Título del TT o TIC	LP	Link dspace
16	Sistema informático para la gestión de pedidos en los localesfísicos de cafeterías de la ciudad de Loja aplicando arquitectura distribuidas (microservicios y/o serverless).	JavaScript	https://dspace.unl.edu.ec/jspui/handle/123456789/28932
17	Prototipo de aplicación web para seguimiento de procesos de selección de personal en la empresa LojaSoft Solutions	Python	https://dspace.unl.edu.ec/jspui/handle/123456789/26616
18	Sistema web para la gestión y monitoreo de las denuncias de acoso escolar en la Unidad Educativa Lauro Damerval Ayora	Python	https://dspace.unl.edu.ec/jspui/handle/123456789/29173
19	Desarrollo de aplicativo web para la gestión de cobro de mensualidades a estudiantes del preuniversitario CENES	Python	https://dspace.unl.edu.ec/jspui/bitstream/123456789/26679/1/CarlaIsabel_TroyaCapa.pdf

Trabajo de Titulación (TT); Trabajo de Integración Curricular (TIC); Lenguaje de Programación (LP).

3. Análisis de Datos

Se ha recopiló 19 Memorias en desarrollo de Aplicaciones Web, se identifica que los lenguajes de programación más utilizados en el desarrollo de aplicaciones web en nuestra carrera son: JavaScript con una cantidad de 8, Python con una cantidad de 6 y PHP como lenguajes secundarios con una cantidad de 3.

4. Criterios de Selección

Para seleccionar los dos lenguajes de programación en los que se enfocara para la solución de problemas de mantenibilidad, consideramos los siguientes criterios.

- **Criterio 1:** Popularidad y utilización en la carrera de computación.
- **Criterio 2:** Disponibilidad de recursos de aprendizaje.
- **Criterio 3:** Experiencia previa en los lenguajes por parte de los estudiantes y profesores.
- **Criterio 4:** Facilidad de uso y flexibilidad para resolver una amplia gama de problemas.

Para responder los siguientes criterios, se dividió cada una de las memorias recopiladas con su respectivo lenguaje de programación para su posterior análisis.

5. Desarrollo

División de las memorias según el tipo de lenguajes de programación.

- **JavaScript:** 8
- **Python:** 6
- **PHP:** 3

Para el proceso de selección, se tomará en cuenta el número de memorias que cumplen con el criterio de selección, dando con ello un porcentaje según el número de memorias que hayan aplicado por el mismo lenguaje de programación.

Formula: (A/B) * 100

A: Numero de memorias que contemplan un criterio.

B: Numero de memorias que existe en cada grupo del lenguaje de programación.

Aplicación de la fórmula:

- **JavaScript**: De las 8 memorias, se identifica que de las 8 memorias todas se dirigen por los 4 criterios de selección.

Criterio 1: (8 / 8) * 100 = 1 * 100 = 100%

Criterio 2: (8 / 8) * 100 = 1 * 100 = 100%

Criterio 3: (8 / 8) * 100 = 1 * 100 = 100%

Criterio 4: (8 / 8) * 100 = 1 * 100 = 100%

Resultado: (100% + 100% + 100% + 100%) / 4 = 400% / 4 = 100%

- **Python**: De las 6 memorias, En el criterio 1 se identifica que 5 memorias seleccionan por su popularidad y utilización en la carrera de computación, Criterio 2, que 6 por su disponibilidad de recursos de aprendizaje, Criterio 3, que 5 por experiencia previa en los lenguajes por parte de los estudiantes y profesores y Criterio 4, que 8 por la facilidad de uso y flexibilidad para resolver una amplia gama de problemas.

Criterio 1: (5 / 6) * 100 = 83.33%

Criterio 2: (6 / 6) * 100 = 1 * 100 = 100%

Criterio 3: (5 / 6) * 100 = 83.33%

Criterio 4: (6 / 6) * 100 = 1 * 100 = 100%

Resultado: (83,33% + 100% + 83,33% + 100%) / 4 = 366,66% / 4 = 91,67%

- **PHP**: De las 3 memorias, se identifica en el Criterio 1, que 0 memorias seleccionan por su popularidad y utilización en la carrera de computación, Criterio 2, que 3 por su disponibilidad de recursos de aprendizaje, Criterio 3, que 1 por experiencia previa en los lenguajes por parte de los estudiantes y profesores y Criterio 4, que 1 por la facilidad de uso y flexibilidad para resolver una amplia gama de problemas.

Criterio 1: (0 / 3) * 100 = 0%.

Criterio 2: (3 / 3) * 100 = 1 * 100 = 100%

Criterio 3: (1 / 3) * 100 = 33,33%

Criterio 4: (1 / 3) * 100 = 33,33%

Resultado: (0% + 100% + 33,33% + 33,33%) / 4 = 166,66% / 4 = 41,67%

En la **Tabla 2-A2** se muestra el porcentaje de cumplimiento de cada criterio de selección, según la división de memorias (JavaScript, Python y PHP).

Tabla 2-A2. Porcentaje de cumplimiento de cada criterio de selección

N.	Criterio de Selección	MJS (8)	MP (6)	MPH (3)
1	Popularidad y utilización en carrera de computación	8	5	0
2	Disponibilidad de recursos aprendizaje	8	6	3
3	Experiencia previa en los lenguajes por parte de los estudiantes y profesores	8	5	1
4	Facilidad de uso y flexibilidad para resolver una amplia gama de problemas	8	6	1
Porcentaje de cumplimiento		100%	91.67%	41.67

Memorias según JavaScript (MJS); Memorias según Python (MP); Memorias según PHP (MPH).

6. Resultados

Después de determinar el porcentaje de aceptación de los criterios de selección de cada grupo de memorias, se ha decidido seleccionar los siguientes lenguajes de programación según a los resultados de la Tabla 31:

- Principal JavaScript: Basado en el análisis, además como lenguaje ampliamente utilizado en el desarrollo web tanto en el Front-end como en el Back-end. Cuenta con una gran comunidad de desarrolladores, una amplia variedad de bibliotecas y frameworks, y es altamente demandado en el mercado laboral.
- Secundario Python: Basado en el análisis, además por su conocida facilidad de uso, legibilidad y versatilidad. Es ampliamente utilizado en el desarrollo web, la ciencia de datos, la inteligencia artificial y muchas otras áreas, además, tiene una gran cantidad de recursos de aprendizaje disponibles y es muy popular entre estudiantes y profesionales.

7. Conclusiones

Después de un exhaustivo análisis, se ha establecido que los lenguajes de programación preeminentes en la carrera de Ingeniería en Sistemas/Computación para la creación de aplicaciones web son JavaScript y Python. Esta conclusión orienta el enfoque de nuestro estudio.

Anexo 3. Análisis y selección de casos de estudio

1. Introducción

En el siguiente apartado, se realizará un análisis y selección de dos casos de estudio o productos de software, específicamente aplicaciones web en producción desarrolladas por estudiantes de la carrera de Ingeniería en Sistemas/Computación. Este proceso se llevará a cabo mediante un muestreo no probabilístico por conveniencia, entre criterios de inclusión y exclusión.

2. Desarrollo

2.1. Población

Se ha considerado las aplicaciones web y los módulos en producción creados por los estudiantes de la CIC de la UNL para los laboratorios de la carrera, teniendo en cuenta los lenguajes de programación predominantes, como JavaScript y Python (Ver **Anexo 2**).

2.2. Objeto de análisis

Cada una de las aplicaciones web recopiladas ha sido desarrollada por los estudiantes de la CIC de la UNL con el objetivo de abordar una problemática específica.

En la **Tabla 1-A3** se detallan las aplicaciones web desarrolladas para la CIC cuyo objetivo es contribuir en la misma.

Tabla 1-A3. Porcentaje de cumplimiento de cada criterio de selección.

Aplicación web	Servidor
Dislalia	Laboratorios de la CIC
Gestion VR	Laboratorios de la CIC
SDLC	Laboratorios de la CIC
GCPC	AWS
MSMPA	Laboratorios de la CIC

Nota: Los nombres de las aplicaciones web es de acuerdo a cada memoria del TT/TIC.

2.3. Tamaño de muestra

Debido que la población es cuantitativa, se optó por un muestreo no probabilístico o por conveniencia según Alaminos y Castejón, es decir, seleccionar casos que proporcionen una mayor cantidad de información.

Para determinar el tamaño de la muestra se toma en cuenta atributos por conveniencia de cada aplicación web, por ejemplo, si el producto fue desarrollado para CIC, contribuye en la carrera, acceso al código fuente y documentación. Para abarcar el cumplimiento de estos atributos se aplicará criterios de exclusión e inclusión.

Estos criterios permiten garantizar que la muestra recopilada sea relevante para los objetivos de la investigación y que se mantenga la coherencia en la selección de las aplicaciones web a evaluar.

En la **Tabla 2-A3** se muestra los criterios de inclusión para seleccionar las aplicaciones web.

Tabla 2-A3. Criterios de inclusión para seleccionar aplicaciones web.

CI	Descripción
Propósito	La aplicación web fue desarrollada específicamente para satisfacer los requerimientos de la carrera.
Contribución	La aplicación web contribuye al proceso académico de los estudiantes.
Código Fuente	Autores que compartan el código fuente para los fines del presente trabajo.
Productos en Producción	Aplicaciones que se encuentren en producción, de acuerdo con el objetivo del proyecto.
Lenguaje de programación	Aplicaciones que ha utilizado los lenguajes de JavaScript y Python como los más predominantes en la carrera según el **Anexo 2**.
Año de publicación	El año publicación mayor al 2020.

Criterio de Inclusión (CI).

Para cumplir con los criterios de exclusión se aplicará el no cumplimiento de los criterios de inclusión. Es decir, la aplicación web no cumple con el criterio de inclusión será omitida o excluida.

2.4. Aplicación del método

Para la aplicación del método de muestreo no probabilístico por conveniencia, se procedió a evaluar cada una de las aplicaciones web recopiladas según los criterios de inclusión previamente establecidos.

En la **Tabla 3-A3** se muestra la aplicación de los criterios de inclusión para seleccionar las aplicaciones web.

Tabla 3-A3. Aplicación de los criterios de inclusión.

Aplicación	PS	CB	CF	PP	LP	AP
Dislalia	Sí	Sí	No	Sí	JavaScript	2021
Gestion VR	Sí	Sí	Sí	Sí	JavaScript	2023
SDLC	Sí	Sí	Sí	Sí	JavaScript	2022
GCPC	No	No	No	No	ODO	2021
MSMPA	Sí	Sí	No	Sí	JavaScript	2022

Propósito (PS); Contribución (CB); Código Fuente (CF); Productos en Producción (PP); Lenguaje de programación (LP); Año de publicación (AP).

Nota: La selección se basa principalmente en la observación y en criterios predefinidos (no en operaciones matemáticas). La elección de los casos de estudio se realiza de manera subjetiva, enfocándose en aquellos que mejor se ajusten a los objetivos de la investigación y a los criterios de inclusión establecidos.

Como se puede observar en la tabla, las aplicaciones Gestion VR y SDLC cumplen con todos los criterios de inclusión establecidos, lo que las hace elegibles como casos de estudio para el presente proyecto.

3. Conclusión

Mediante el muestreo no probabilístico por conveniencia y la aplicación de criterios de inclusión, se ha seleccionado a la aplicación web **Gestion VR** como primer caso de estudio seleccionado y la aplicación web **SDLC** como segundo caso de estudio seleccionado para el diagnóstico de mantenibilidad. Estas aplicaciones cumplen con todos los criterios de inclusión establecidos, lo que garantiza su relevancia y coherencia con los objetivos de la investigación.

Anexo 4. Fórmulas para el cálculo de métricas

En la siguiente sección de muestra las fórmulas aplicadas para obtener los valores de las métricas según Sonarqube [69].

a. Mala práctica.

Tabla 1-A4. Formula métrica mala práctica.

Métrica	Mala practica
Función de medición	R= Vpropiedad / LC
Elementos de medición	Resultado =R Líneas de código=LC Vpropiedad = Valor de Propiedad

b. Redundante.

Tabla 2-A4. Formula métrica redundante.

Métrica	Redundante
Función de medición	R= Vpropiedad / LC
Elementos de medición	Resultado =R Líneas de código=LC Valor de Propiedad = Vpropiedad

c. Complejidad cognitiva.

Tabla 3-A4. Formula métrica complejidad cognitiva.

Métrica	Complejidad cognitiva
Función de medición	R= Vpropiedad / LC
Elementos de medición	Resultado =R Líneas de código=LC Valor de Propiedad = Vpropiedad

d. Obsoleto.

Tabla 4-A4. Formula métrica obsoleta.

Métrica	Obsoleto
Función de medición	R= Vpropiedad / LC
Elementos de medición	Resultado =R Líneas de código=LC Valor de Propiedad = Vpropiedad

e. Confuso.

Tabla 5-A4. Formula métrica confusa.

Métrica	Confuso
Función de medición	R= Vpropiedad / LC
Elementos de medición	Resultado =R Líneas de código=LC Valor de Propiedad = Vpropiedad

f. Diseño.

Tabla 6-A4. Formula métrica diseño.

Métrica	Diseño
Función de medición	R= Vpropiedad / LC
Elementos de medición	Resultado =R Líneas de código=LC Valor de Propiedad = Vpropiedad

g. Dificultad encontrada.

Tabla 7-A4. Formula métrica dificultad encontrada.

Métrica	Dificultad encontrada
Función de medición	R=LC/Vpropiedad
Elementos de medición	Resultado =R Líneas de código=LC Valor de Propiedad = Vpropiedad

h. Densidad de comentarios.

Tabla 8-A4. Formula métrica densidad de comentarios.

Métrica	Densidad de comentarios
Función de medición	R=100%- LCC%
Elementos de medición	Resultado =R Líneas de código comentadas=LCC%

i. Duplicaciones.

Tabla 9-A4. Formula métrica duplicaciones.

Métrica	Duplicaciones
Función de medición	R=Vpropiedad/LC
Elementos de medición	Resultado =R Líneas de código=LC Vpropiedad = Valor de Propiedad

j. Cobertura.

Tabla 10-A4. Formula métrica cobertura.

Métrica	Cobertura
Función de medición	R=Vpropiedad/LC
Elementos de medición	Resultado =R Líneas de código=LC Vpropiedad = Valor de Propiedad

Anexo 5. Descripción Casos de estudio o Productos de Software

En esta sección, se presentan los detalles de los casos de estudio o productos de software para la evaluación de mantenibilidad, se incluyen diagramas de caso de uso, secuencia y clases correspondiente a las aplicaciones web **SDLC** y **Gestión VR**. Estos elementos proporcionarán una visión detallada de la estructura y funcionalidad de los productos, facilitando así la evaluación de su mantenibilidad.

1. Producto de Software SDLC

La aplicación web denominada SDLC de los laboratorios de la Carrera de Computación a diagnosticar, fue desarrollado utilizando un enfoque de backend y frontend. En el backend, mediante ApiRest creado con Node.js y en el frontend mediante Angular, tiene como objetivo gestionar el ciclo de vida del desarrollo del software en los proyectos académicos que se llevan a cabo dentro de la misma carrera.

La aplicación en promedio es útil como un caso de estudio, puesto que fue desarrollado con el propósito de resolver una problemática detectada en la Carrera de Ingeniería en Sistemas/Computación, específicamente la falta de herramientas para la gestión de proyectos de software en el ámbito académico. Esto lo convierte en un caso de estudio relevante para los estudiantes de la carrera, ya que ofrece una solución concreta a un problema real que ellos mismos han enfrentado.

- **Arquitectura de software**

En la **Figura 1-A5** se puede observar la arquitectura del software de producto SDLC mediante el diagrama de despliegue, la cual permite identificar topología de los componentes del sistema.

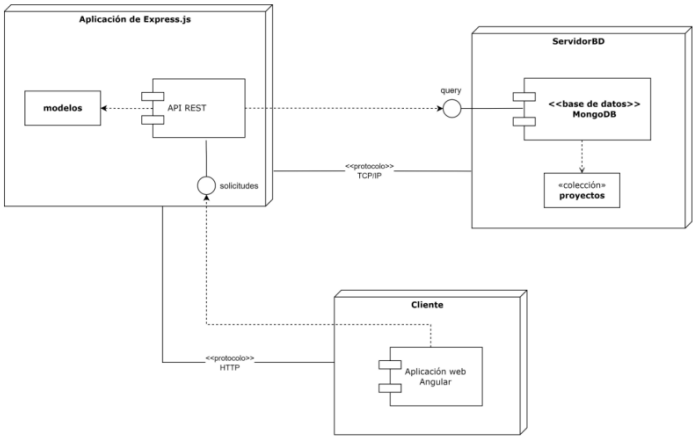


Figura 1-A4. Arquitectura de software mediante el diagrama de despliegue.

- **Diagramas UML**

El software SDLC utiliza diagramas de UML para llevar a cabo el diseño arquitectónico del sistema, en conjunto con la metodología XP, les ha permitido gestionar de manera eficiente y organizada el proyecto de software.

- **Diagrama de caso de uso**

En la **Figura 2-A5** se muestra el diagrama de casos de uso en el cual se puede observar interacciones entre los objetos y procesos del producto de software SDLC.

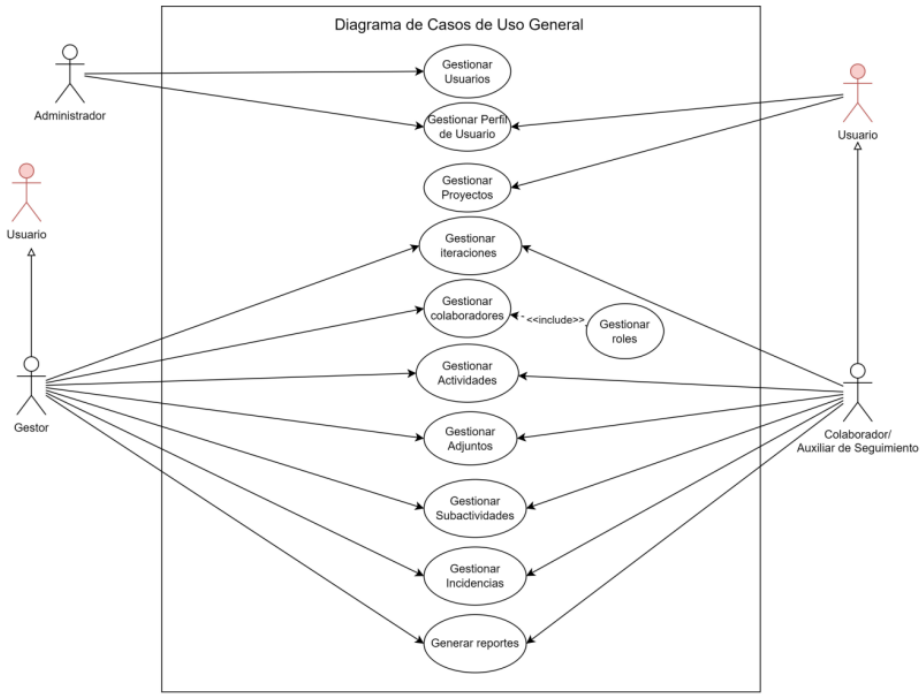


Figura 2-A5. Diagrama de casos de uso general del producto de software SDLC

- **Diagrama del proceso general del producto de software**

En la **Figura 3-A4**, se muestra proceso general del producto de software SDLC, el cual especifica el flujo normal en el que trabajarían los estudiantes y docentes al llevar a cabo el proceso SDLC a través de la Metodología XP dentro del software.

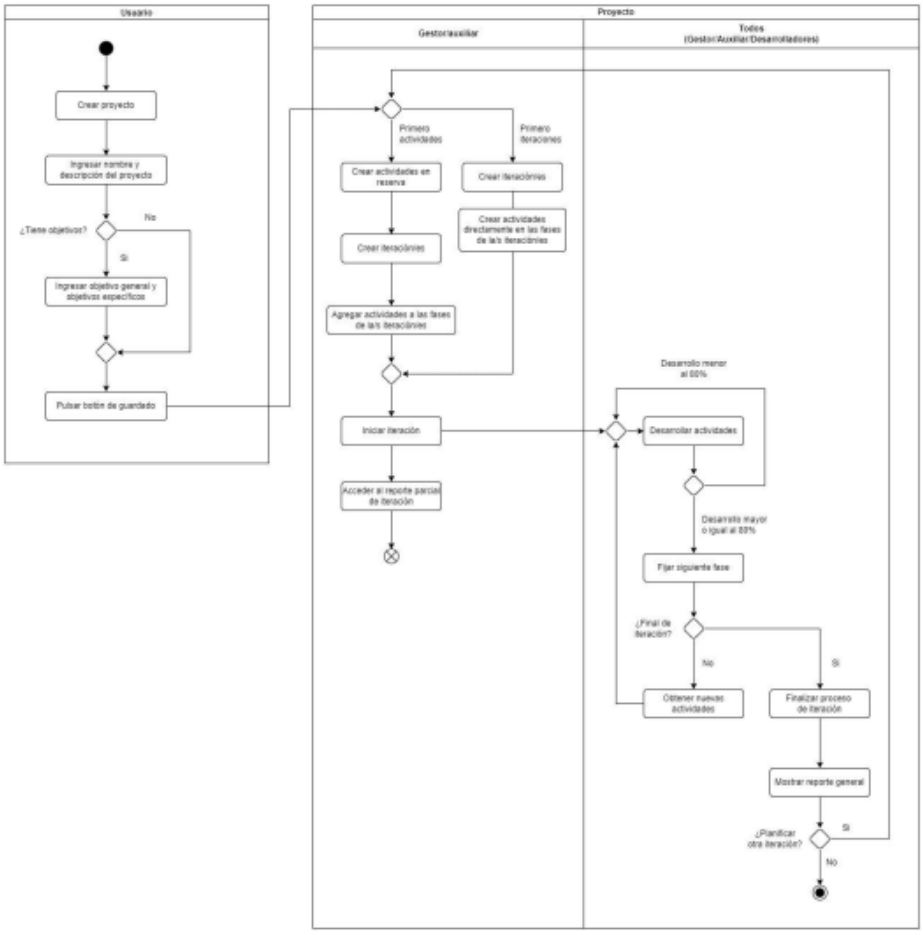


Figura 3-A5. Diagrama del proceso general del producto de software SDLC

- **Diagrama de clases**

En la **Figura 3-A5** se muestra el diagrama de clases general del producto de software SDLC, en el cual se ha identificado la estructura y las relaciones entre las clases del software.

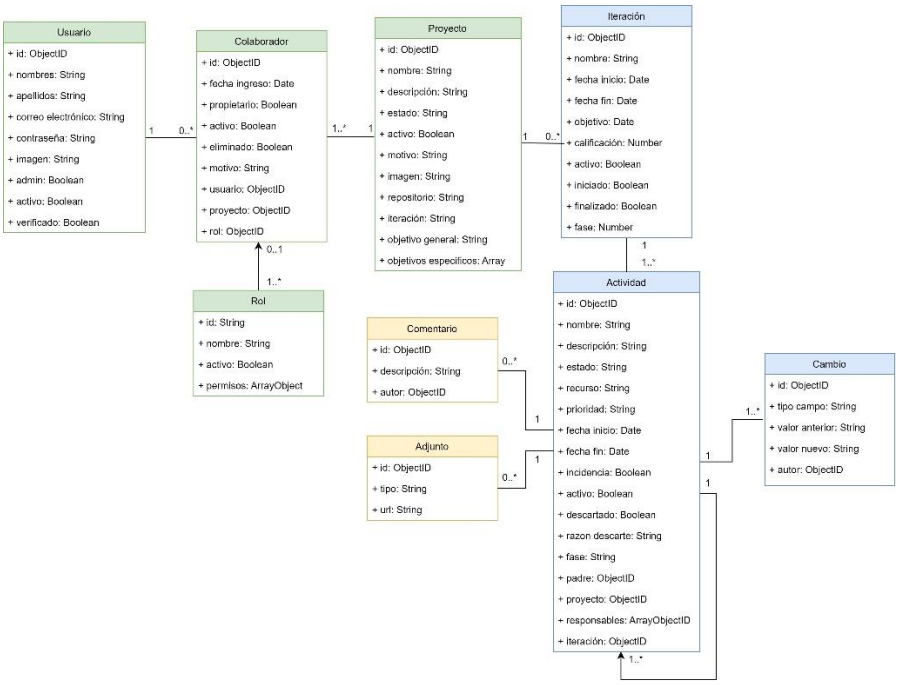


Figura 4-A5. Diagrama de clase del producto de software SDLC

2. Producto de Software Gestión VR

La aplicación web denominada Gestión VR de los laboratorios de la Carrera de Computación a diagnosticar, fue desarrollado utilizando un enfoque de backend y frontend. En el backend, mediante ApiRest creado con Node.js y en el frontend mediante React.js, tiene como objetivo actualizar la Información que se presenta en la aplicación de realidad virtual "Energía VR" de la misma carrera.

La aplicación en promedio es útil como un caso de estudio, puesto que facilita al gestor ingresar, registrar información, actualizar datos y visualizar archivos que se presenta en el aplicativo Energía VR, además de permitir a los estudiantes navegar e interactuar de una manera interactiva, acerca de los procesos académicos más solicitados de la carrera de Computación de la UNL, sin tener que estar aparentemente presentes en la universidad.

- **Arquitectura de software**

En la **Figura 5-A5** se puede apreciar la arquitectura de software del producto de software Gestión VR.

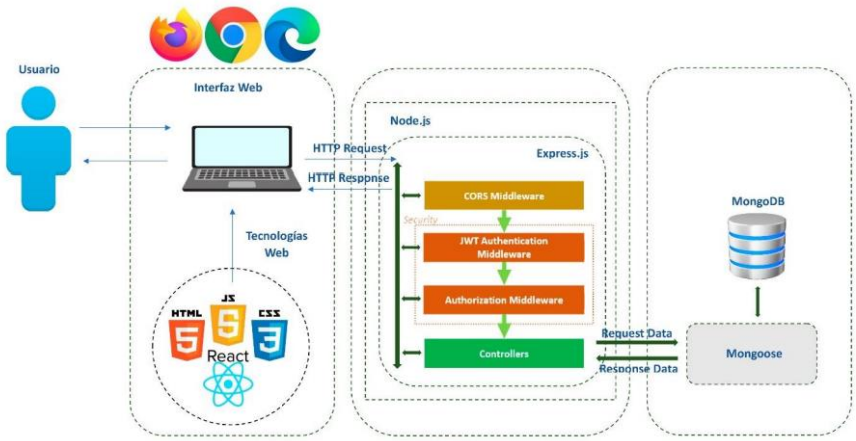


Figura 5-A5. Arquitectura del producto de software Gestión VR.

- **Diagramas UML**

El software Gestión VR utiliza diagramas de UML para representar visualmente la estructura y relaciones entre las clases o tablas que gestionan diversos procesos. Estas clases incluyen: administrativo, audio, directivo, docente, documento, función, imagen, links, maestría, patio, pdfs, personal, solicitud, televisor, user, vídeo y words.

Los diagramas le han permitido una comprensión clara de cómo interactúan estas entidades dentro del sistema.

- **Diagrama de casos de uso**

En la **Figura 6-A5** se muestra el diagrama de casos de uso, en la cual se ha podido identificar las interacciones que existe entre los objetos y procesos del producto de software denominado Gestión VR. Según las especificaciones de la memoria, cuenta con los siguientes tipos de usuarios: Usuario y Administrador.

Figura 6-A5. Diagrama de casos de uso general de Gestión VR

- **Diagrama de clases**

En la **Figura 7-A5** se muestra el diagrama de clases, en donde se ha podido identificar la estructura y funcionamiento del producto de software denominado Gestión Vr. Este diagrama incluye las clases relacionadas con las historias de usuario.

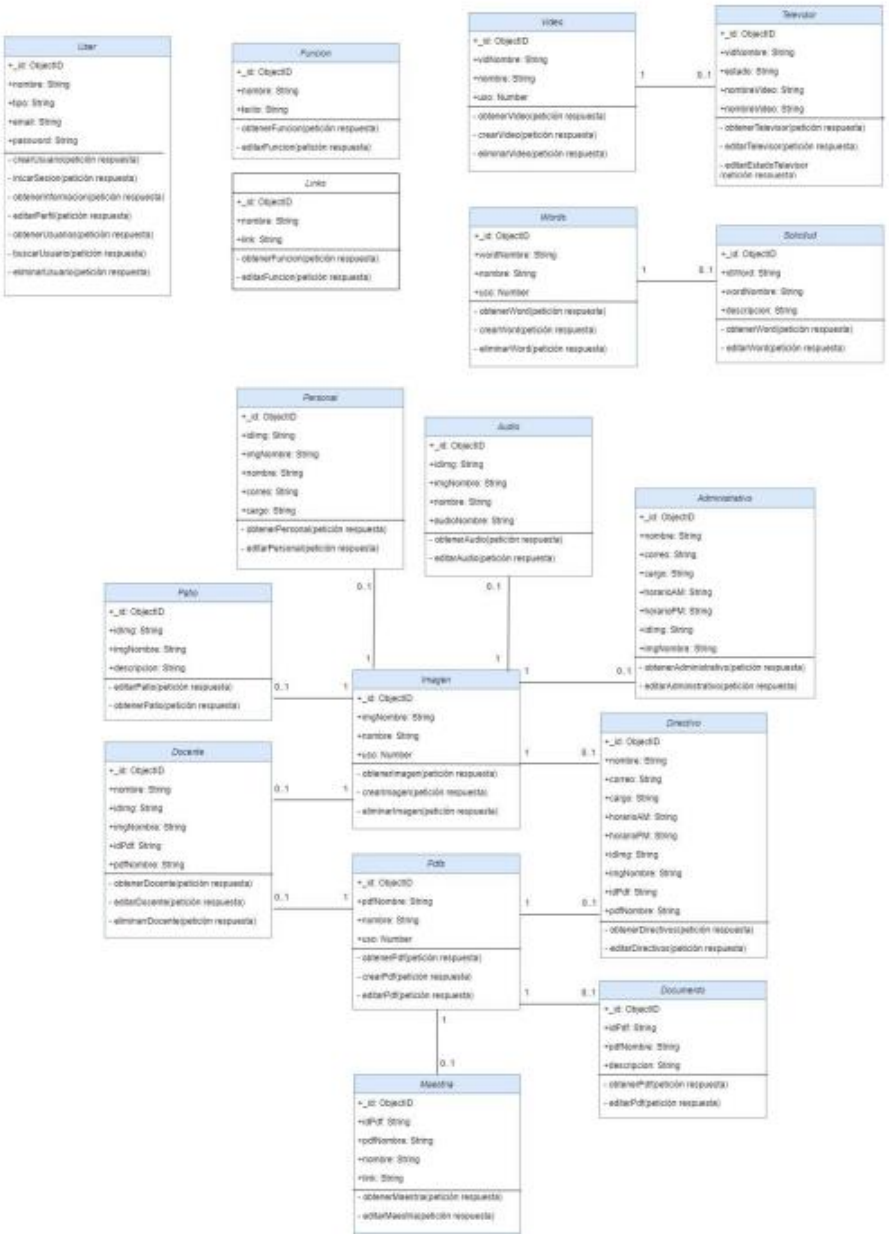


Figura 7-A5. Diagrama de casos de clases general de Gestión VR.

- **Diagrama de Componentes**

En la **Figura 8-A5** se ha logrado identificar como está compuesto el producto de software Gestión VR y la comunicación entre el cliente y el servidor de base de datos.

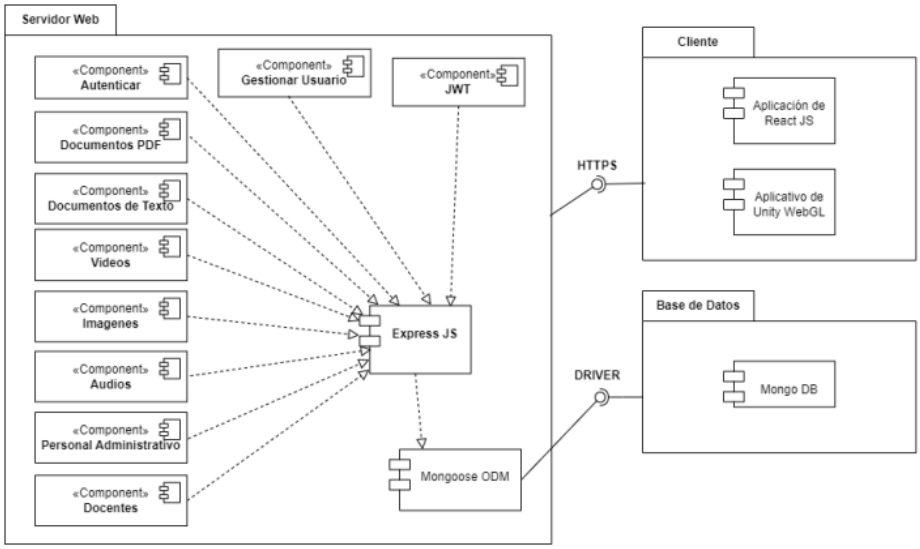


Figura 8-A5. Diagrama de componentes de Gestión VR

Anexo 6: Entrevista responsable del centro de datos de los laboratorios de la CIC Y CIS

En esta sección se muestra los resultados de la entrevista al responsable del centro de datos de los laboratorios de la CIC Y CIS, de los cuales se puede definir defectos de mantenibilidad.

- **Rol**: Gestor Académico de la Carrera Ingeniería en Computación.
- **Entrevista Firmada:** https://drive.google.com/file/d/1jZXwJEfFtqBlXO1h3kaAZgvnFlDc74kk/view?usp=sharing
- **Grabación de la entrevista:** https://drive.google.com/file/d/1F48UmDcaLu8lDtCH3r1i5gWfJRGKE15n/view?usp=sharing
- **Participantes:**
  - **Hablante 1:** Ing. Pablo Fernando Ordoñez Mg.Sc.
  - **Hablante 2:** Estudiante Gilson Orlando Quezada
- **Análisis:**
  - La información recopilada permitió platear la problemática del PIC y para el presente TIC contrastar en base los resultados obtenidos.

Anexo 7: Entrevista 1 aplicada

En esta sección se muestra los resultados de la entrevista aplicada a docentes que participan como tutores en proyectos de desarrollo de software, de los cuales se puede definir defectos de mantenibilidad.

- **Rol**: Tutor/Miembro Tribunal de grado de la Carrera Ingeniería en Computación.
- **Entrevista Firmada:** https://drive.google.com/file/d/1W3Q5XKTEAHpCgvQZGuev_pe4ikP1QPZ0/view?usp=drive_link.
- **Grabación de la entrevista:** https://drive.google.com/file/d/18CwIeoDgTgknAD4bhgdBXLHbgO8qmmMy/view?usp=drive_link.
- **Participantes:**
  - **Hablante 1:** Ing. Andrés Roberto Navas Mg. Sc.
  - **Hablante 2:** Estudiante Gilson Orlando Quezada.
- **Análisis:**
  - La información recopilada permitió platear la problemática del PIC y para el presente TIC contrastar en base los resultados obtenidos.

Anexo 8: Entrevista 2 aplicada

En esta sección se muestra los resultados de la entrevista aplicada a docentes que participan como tutores en proyectos de desarrollo de software, de los cuales se puede definir defectos de mantenibilidad.

- **Rol**: Tutor/Miembro Tribunal de grado de la Carrera Ingeniería en Computación.
- **Entrevista Firmada:**
  https://drive.google.com/file/d/1HrIpguzRqgcO5T_WOcjEBLpI8mlLL-v1/view?usp=sharing.
- **Grabación de la entrevista:**
  https://drive.google.com/file/d/1phcigRzB4NGFaEs6EB2bJ2YReRyJwq6-/view?usp=sharing.
- **Participantes:**
  - **Hablante 1:** Ing. Edison Leonardo Coronel Mg. Sc
  - **Hablante 2:** Estudiante Gilson Orlando Quezada
- **Análisis:**
  - La información recopilada permitió platear la problemática del PIC y para el presente TIC contrastar en base los resultados obtenidos.

Anexo 9: Entrevista 3 aplicada

En esta sección se muestra los resultados de la entrevista aplicada a docentes que participan como tutores en proyectos de desarrollo de software, de los cuales se puede definir defectos de mantenibilidad.

- **Rol**: Tutor/Miembro Tribunal de grado de la Carrera Ingeniería en Computación.
- **Entrevista Firmada:** https://drive.google.com/file/d/1lEhq9WqfBP2pnmSnHBdroyY2ETs1hlJ0/view?usp=sharing.
- **Grabación de la entrevista:** https://drive.google.com/file/d/1VP2Ujbe6PTDmtP4e4dyr76gLKGp9bysr/view?usp=sharing.
- **Participantes:**
  - **Hablante 1:** Ing. Roberth Gustavo Figueroa Mg. Sc
  - **Hablante 2:** Estudiante Gilson Orlando Quezada
- **Análisis:**
  - La información recopilada permitió platear la problemática del PIC y para el presente TIC contrastar en base los resultados obtenidos.

Anexo 10: Encuesta aplicada

En esta sección se muestra los resultados de la encuesta aplicada a los estudiantes del itinerario de ingeniería de software que participan como encargados en la implementación de proyectos de desarrollo de software, de los cuales se puede definir defectos de mantenibilidad.

- **Recurso encuesta:** https://forms.gle/4cHUUa1N1HqRL2nn7
- **Resultado de encuestas XLS:** https://docs.google.com/spreadsheets/d/16vsatdJXg3GeLh9Ttm_Hxq8HWPeQ2VU0X4VvQoU-80/edit?usp=sharing.
- **Participantes:**
  - **Participantes:** Estudiantes del itinerario de ingeniería de software.
  - **Encuestador:** Estudiante Gilson Orlando Quezada.
- **Análisis:**
  - La información recopilada permitió platear la problemática del PIC y para el presente TIC contrastar en base los resultados obtenidos.

Anexo 11. Aplicación de criterios de decisión caso de estudio SDLC

Para obtener los valores de las métricas, se aplicaron fórmulas de SonarQube (Ver **Anexo 4**) utilizando los datos de las propiedades de calidad del análisis del código estático del caso de estudio.

Se aplicó para el backend y frontend de manera independiente:

- **Backend**

Propiedad	**Backend**
Duplicated Lines	707
Coverage lines	1388
Cyclomatic complexity	525
Code smells	60
Cognitive Complexity	4
Comment lines	1.70%

- **Diseño**

Fórmula.

$$D = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

Se ha utilizado cyclomatic complexity.

Aplicamos la fórmula:

$$D = \frac{Complejidad\ ciclomatica}{lineas\ de\ codigo}$$

$$D = \frac{525}{5300}$$

$$D = 9.91\%$$

- **Duplicaciones**

Se utilizo duplicated lines.

Aplicamos formula de la duplicación.

$$D = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$LD = \frac{707}{5300}$$

$$LD = 13.34\%$$

- **Cobertura**

Se utilizo coverage lines.

Se calculó (Vpropiedad/líneas de código) .

$$CB = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$CB = \frac{1388}{5300}$$

$$CB = 26.19\%$$

- **Mala practica**

Se utilizo code smells:

$$MP = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$MP = \frac{60}{5300}$$

$$MP = 1.13\%$$

- **Obsoleto**

Se utilizo code smells.

$$O = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$O = \frac{60}{5300}$$

$$= 1.13\%$$

- **Redundante**

Se utilizo duplicated lines.

$$R = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$R = \frac{707}{5300}$$

$$R = 13.34\%$$

- **Complejidad cognitiva**

Se utilizo cognitive complexity.

$$CC = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$CC = \frac{4}{5300}$$

$$CC = 0.08\%$$

- **Confuso**

Se utilizo code smells.

$$C = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$C = \frac{60}{5300}$$

$$C = 1.13\%$$

- **Dificultad encontrada**

Se utilizo cognitive complexity.

$$DE = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$DE = \frac{4}{5300}$$

$$DE = 0\%$$

- **Densidad de comentarios**

Se utilizo comment lines.

$$DC = 100\% - valor\ de\ la\ propiedad\%$$

$$DC = 100\% - 1.70\%$$

$$DC = 98.30\%$$

- **Frontend**

Propiedad	**Frontend**
Duplicated Lines	689
Coverage lines	2668
Cyclomatic complexity	1613
Code smells	114
Cognitive Complexity	1

Comment lines	1.90%

- **Diseño**

Fórmula.

$$D = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

Se ha utilizado cyclomatic complexity.

Aplicamos la fórmula:

$$D = \frac{Complejidad\ ciclomatica}{lineas\ de\ codigo}$$

$$D = \frac{1613}{17000}$$

$$D = 9.49\%$$

- **Duplicaciones**

Se utilizo duplicated lines.

Aplicamos formula de la duplicación.

$$D = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$LD = \frac{689}{17000}$$

$$LD = 4.05\%$$

- **Cobertura**

Se utilizo coverage lines.

Se calculó (Vpropiedad/líneas de código).

$$CB = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$CB = \frac{2668}{17000}$$

$$CB = 15.69\%$$

- **Mala practica**

Se utilizo code smells.

$$MP = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$MP = \frac{114}{17000}$$

$$MP = 0.67\%$$

- **Obsoleto**

Se utilizo code smells.

$$O = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$O = \frac{114}{17000}$$

$$O = 0.67\%$$

- **Redundante**

Se utilizo duplicated lines.

$$R = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$R = \frac{689}{17000}$$

$$R = 4.05\%$$

- **Complejidad cognitiva**

Se utilizo cognitive complexity.

$$CC = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$CC = \frac{1}{17000}$$

$$CC = 0.01\%$$

- **Confuso**

Se utilizo code smells.

$$C = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$C = \frac{114}{17000}$$

$$C = 0.67\%$$

- **Dificultad encontrada**

Se utilizo cognitive complexity.

$$DE = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$DE = \frac{1}{17000}$$

$$DE = 0.01\%$$

- **Densidad de comentarios**

Se utilizo comment lines.

$$DC = 100\% - valor\ de\ la\ propiedad\%$$

$$DC = 100\% - 1.90\%$$

$$DC = 98.10\%$$

Para obtener el valor general de cada una de las métricas de las subcaracterísticas del producto de software, se aplicó promedio de las métricas del backend y del frontend, como se muestra a continuación. Esto permitirá obtener el grado de calidad de las subcaracterísticas aplicando la formulas **GS,** conforme se muestra a continuación.

a. Modularidad.

Esta subcaracterística abarca métricas de calidad como: diseño, duplicaciones y cobertura. Para obtener resultados de cada métrica se aplicó:

- **Diseño**: Se dividió el valor de la propiedad para las líneas de código.
- **Duplicaciones**: se dividió el valor de la propiedad para líneas de código.
- **Cobertura**: se calculó (Vpropiedad/líneas de código))

En la **Tabla 1-A11** se muestra los valores de las métricas de calidad respecto al modularidad del backend.

Tabla 1-A11. Criterios de inclusión para seleccionar aplicaciones web.

Métricas	Valor de la propiedad	Resultado%
Diseño	525	9.91%
Duplicaciones	707	13.34%
Cobertura	1388	26.19%

En la **Tabla 2-A11** se muestra los valores de las métricas de calidad respecto al modularidad del frontend.

Tabla 2-A11. Valores de las métricas de calidad, según el modularidad del frontend.

Métricas	Valor de la propiedad	Resultado%
Diseño	1613	9.49%
Duplicaciones	689	4.05%
Cobertura	2668	15.69%

En la **Tabla 3-A11** se muestra los resultados finales de cada métrica respecto al backend y el frontend, para lo cual se ha calculado mediante el promedio, lo que da a conocer el valor final de la subcaracterística modularidad.

Tabla 3-A11. Promedio de las métricas calidad en la característica del modularidad.

Métricas	Backend	Frontend	Promedio
Diseño	9.91%	9.49%	9.70%
Duplicaciones	13.34%	4.05%	8.70%
Cobertura	26.19%	15.69%	20.94%

Grado de la subcaracterística modularidad:

$$M = \frac{D + DP + CB}{3}$$

$$M = \frac{9.70\% + 8.70\% + 20.94\%}{3}$$

$$M = 86.89\%$$

b. Reusabilidad.

Esta subcaracterística abarca métricas de calidad como: mala práctica, obsoleto, diseño y duplicaciones, para obtener resultados de cada métrica se aplicó:

- **Mala práctica, obsoleto y diseño**: Se dividió el valor de la propiedad para las líneas de código.
- **Duplicaciones**: Se dividió el valor de la propiedad para líneas de código.

En la **Tabla 4-A11** se muestra los valores de las métricas de calidad respecto al reusabilidad del backend.

Tabla 4-A11. Valores de las métricas de calidad, según la reusabilidad del backend.

Métricas	Valor de la propiedad	Resultado%
Mala práctica	60	1.13%
Obsoleto	60	1.13%
Diseño	525	9.91%
Duplicaciones	707	13.34%

En la **Tabla 5-A11** se muestra los valores de las métricas de calidad respecto al reusabilidad del frontend.

Tabla 5-A11. Valores de las métricas de calidad, según la reusabilidad del frontend.

Métricas	Valor de la propiedad	Resultado%
Mala práctica	114	0.67%
Obsoleto	114	0.67%
Diseño	1613	9.49%
Duplicaciones	689	4.05%

En la **Tabla 6-A11** se muestra los resultados finales de cada métrica respecto al backend y el frontend, para lo cual se ha calculado mediante el promedio, lo que da a conocer el valor final de la subcaracterística reusabilidad.

Tabla 6-A11. Promedio de las métricas calidad en la subcaracterística de la reusabilidad.

Métricas	Backend	Frontend	Promedio
Mala práctica	1.13%	0.67%	0.901%
Obsoleto	1.13%	0.67%	0.901%
Diseño	9.91%	9.49%	9.697%
Duplicaciones	13.34%	4.05%	8.696%

Grado de la subcaracterística reusabilidad:

$$M = \frac{MP + O\ +\ D\ +\ DP}{4}$$

$$M = \frac{0.90\%\ + 0.90\%\ +\ 9.69 + 8.69\%}{4}$$

$$M = 94.95\%$$

c. Analizabilidad.

Esta subcaracterística abarca métricas de calidad como: mala práctica, redundante, Cognitive Complexity, confuso, diseño, dificultad encontrada y densidad de comentarios, para obtener resultados de cada métrica se aplicó:

- **Mala práctica, redundante, Cognitive Complexity, confuso, diseño y dificultad encontrada**: Se dividió el valor de la propiedad para las líneas de código, además se multiplicó por líneas de código con el fin de obtener un porcentaje de las mismas.
- **Densidad de comentario:** Se resto el 100% para el total de líneas comentadas%.

En la **Tabla 7-A11** se muestra los valores de las métricas de calidad respecto al analizabilidad del backend.

Tabla 7-A11. Valores de las métricas de calidad, según la analizabilidad del backend.

Métricas	Valor de la propiedad	Resultado%
Mala práctica	60	1.13%
Redundante	707	13.34%
Cognitive Complexity	4	0.08%
Confuso	60	1.13%
Diseño	525	9.91%
Dificultad encontrada	4	0%
Densidad de comentarios	1.70%	98.30%

En la **Tabla 8-A11** se muestra los valores de las métricas de calidad respecto al reusabilidad del frontend.

Tabla 8-A11. Valores de las métricas de calidad, según la analizabilidad del frontend.

Métricas	Valor de la propiedad	Resultado%
Mala práctica	114	0.67%
Redundante	689	4.05%
Cognitive Complexity	1	0.01%
Confuso	114	0.67%
Diseño	1613	9.49%
Dificultad encontrada	1	0.01%
Densidad de comentarios	1.90%	98.10%

En la **Tabla 9-A11,** se muestra los resultados finales de cada métrica respecto al backend y el frontend, para lo cual se ha calculado mediante el promedio, lo que da a conocer el valor final de la subcaracterística analizabilidad.

Tabla 9-A11. Promedio de las métricas calidad en la subcaracterística de la analizabilidad.

Métricas	Backend	Frontend	Promedio
Mala práctica	1.13%	0.67%	0.901%
Redundante	13.34%	4.05%	8.696%
Cognitive Complexity	0.08%	0.01%	0.041%
Confuso	1.13%	0.67%	0.90%
Diseño	9.91%	9.49%	9.697%
Dificultad encontrada	0%	0.01%	0.041%
Densidad de comentarios	98.30%	98.10%	98.20%

Grado de la subcaracterística analizabilidad:

$$M = \frac{MP + R + CC + C + D\ +\ DE\ +\ DC}{7}$$

$$M = \frac{0.90\%\ +\ 8.69\%\ +\ 0.04 + 0.90\% + 9.67\% + 0.041\% + 98.20\%}{4}$$

$$M = 83.075\%$$

d. Capacidad para ser modificado.

Esta subcaracterística abarca métricas de calidad como: mala práctica, redundante, diseño, y duplicaciones, para obtener resultados de cada métrica se aplicó:

- **Mala práctica, redundante y diseño:** Se dividió el valor de la propiedad para las líneas de código.
- **Duplicaciones:** Se dividió el valor de la propiedad para líneas de código, para obtener un porcentaje.

En la **Tabla 10-A11** se muestra los valores de las métricas de calidad respecto a la capacidad para ser modificado del backend.

Tabla 10-A11. Valores de las métricas de calidad, según la capacidad para ser modificado del backend.

Métricas	Valor de la propiedad	Resultado%
Mala práctica	60	1.13%
Redundante	707	1.13%
Diseño	525	9.91%
Duplicaciones	707	13.34%

En la **Tabla 11-A11** se muestra los valores de las métricas de calidad respecto a la capacidad para ser modificado del frontend.

Tabla 11-A11. Valores de las métricas de calidad, según la capacidad para ser modificado del frontend.

Métricas	Valor de la propiedad	Resultado%
Mala práctica	114	0.67%
Redundante	689	4.05%
Diseño	1613	9.49%
Duplicaciones	689	4.05%

En la **Tabla 12-A11** se muestra los resultados finales de cada métrica respecto al backend y el frontend, para lo cual se ha calculado mediante el promedio, lo que da a conocer el valor final de la subcaracterística capacidad para ser modificado.

Tabla 12-A11. Promedio de las métricas calidad en la subcaracterística de la capacidad para ser modificado.

Métricas	Backend	Frontend	Promedio
Mala práctica	1.13%	0.67%	0.901%
Redundante	1.13%	4.05%	8.696%
Diseño	9.91%	9.49%	9.697%
Duplicaciones	13.34%	4.05%	8.696%

Grado de capacidad de ser modificado:

$$M = \frac{MP + R + D\ +\ DP}{4}$$

$$M = \frac{0.91\% + 8.69\% + 9.69\%\ +\ 8.69\%}{4}$$

$$M = 93.002\%$$

e. Capacidad para ser probado.

Esta subcaracterística abarca métricas de calidad como: redundante, diseño, dificultad encontrada y cobertura, para obtener resultados de cada métrica se aplicó:

- **Redundante, diseño y dificultad encontrada**: se dividió el valor de la propiedad para las líneas de código.
- **Cobertura**: se calculó (Vpropiedad/líneas de código)).

En la **Tabla 13-A11** se muestra los valores de las métricas de calidad respecto a la capacidad para ser probado del backend.

Tabla 13-A11. Valores de las métricas de calidad, según la capacidad para ser probado del backend.

Métricas	Valor de la propiedad	Resultado%
Redundante	707	13.34%
Diseño	525	9.91%
Dificultad encontrada	4	0%
Cobertura	1388	26.19%

En la **Tabla 14-A11** se muestra los valores de las métricas de calidad respecto a la capacidad para ser probado del frontend.

Tabla 14-A11. Valores de las métricas de calidad, según la capacidad para ser probado del frontend.

Métricas	Valor de la propiedad	Resultado%
Redundante	689	4.05%
Diseño	1613	9.49%
Dificultad encontrada	1	0.01%
Cobertura	2668	15.69%

En la **Tabla 15-A11** se muestra los resultados finales de cada métrica respecto al backend y el frontend, para lo cual se ha calculado mediante el promedio, lo que da a conocer el valor final de la subcaracterística capacidad para ser probado.

Tabla 15-A11. Promedio de las métricas calidad en la subcaracterística de la capacidad para ser probado.

Métricas	**Backend**	**Frontend**	**Promedio**
Redundante	13.34%	4.05%	8.696%
Diseño	9.91%	9.49%	9.697%
Dificultad encontrada	0%	0.01%	0.041%
Cobertura	26.19%	15.69%	20.94%

Grado de capacidad de ser probado:

$$M = \frac{R + D + DE\ +\ CB}{4}$$

$$M = \frac{8.69\% + 9.69\% + 0.04\%\ + 79.05\%}{4}$$

$$M = 90.16\%$$

Anexo 12. Aplicación de criterios de decisión caso de estudio Gestion VR

Para Para obtener los valores de las métricas, se aplicaron fórmulas de SonarQube (Ver **Anexo 4**) utilizando los datos de las propiedades de calidad del análisis del código estático del caso de estudio.

Se aplicó para el backend y frontend de manera independiente:

- **Backend**

Líneas de código: 2161

Tabla 1-A12. Valores de las propiedades de calidad.

Propiedad	**Backend**
Duplicated Lines	1228
Coverage lines	865
Cyclomatic complexity	277
Code smells	6
Cognitive complexity	0
Comment lines	20%

- **Diseño**

Fórmula.

$$D = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

Se ha utilizado cyclomatic complexity.

Aplicamos la fórmula:

$$D = \frac{Complejidad\ ciclomatica}{lineas\ de\ codigo}$$

$$D = \frac{277}{2161}$$

$$D = 12.82\%$$

- **Duplicaciones**

Se utilizo duplicated lines.

Aplicamos formula de la duplicación

$$D = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$LD = \frac{1228}{2161}$$

$$LD = 56.83\%$$

- **Cobertura**

Se utilizo coverage lines.

Se calculó (Vpropiedad/líneas de código)

$$CB = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$CB = \frac{865}{2161}$$

$$CB = 40.03\%$$

- **Mala practica**

Se utilizo code smells.

$$MP = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$MP = \frac{6}{2161}$$

$$MP = 0.28\%$$

- **Obsoleto**

Se utilizo code smells.

$$O = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$O = \frac{6}{2161}$$

$$= 0.28\%$$

- **Redundante**

Se utilizo duplicated lines.

$$R = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$R = \frac{1228}{2161}$$

$$R = 56.83\%$$

- **Complejidad cognitiva**

Se utilizo cognitive complexity.

$$CC = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$CC = \frac{0}{2161}$$

$$CC = 0$$

$$CC = 0\%$$

- **Confuso**

Se utilizo code smells.

$$C = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$C = \frac{6}{2161}$$

$$C = 0.28\%$$

- **Dificultad encontrada**

Se utilizo cognitive complexity.

$$DE = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$DE = \frac{0}{2161}$$

$$DE = 0\%$$

- **Densidad de comentarios**

Se utilizo comment lines.

$$DC = 100\% - valor\ de\ la\ propiedad\%$$

$$DC = 100\% - 20\%$$

$$DC = 80\%$$

- **Frontend**

Líneas de código: 26000

Tabla 1-A12. Valores de las propiedades de calidad.

Propiedad	Backend
Duplicated Lines	2823
Coverage lines	1534
Cyclomatic complexity	1021
Code smells	846
Cognitive Complexity	30,00
Comment lines	0.40%

- **Diseño**

Fórmula.

$$D = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

Se ha utilizado cyclomatic complexity.

Aplicamos la fórmula:

$$D = \frac{Complejidad\ ciclomatica}{lineas\ de\ codigo}$$

$$D = \frac{1021}{26000}$$

$$D = 3.93\%$$

- **Duplicaciones**

Se utilizo duplicated lines.

Aplicamos formula de la duplicación

$$D = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$LD = \frac{1228}{26000}$$

$$LD = 10.86\%$$

- **Cobertura**

Se utilizo coverage lines.

Se calculó (Vpropiedad/líneas de código)

$$CB = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$CB = \frac{1534}{2600}$$

$$CB = 5.90\%$$

- **Mala practica**

Se utilizo code smells.

$$MP = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$MP = \frac{846}{26000}$$

$$MP = 3.25\%$$

- **Obsoleto**

Se utilizo code smells.

$$O = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$O = \frac{11}{26000}$$

$$= 3.25\%$$

- **Redundante**

Se utilizo duplicated lines.

$$R = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$R = \frac{1228}{26000}$$

$$R = 10.86\%$$

- **Complejidad cognitiva**

Se utilizo cognitive complexity.

$$CC = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$

$$CC = \frac{846}{26000}$$
$$CC = 0.12\%$$

- **Confuso**

Se utilizo code smells.

$$C = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$
$$C = \frac{11}{26000}$$
$$C = 3.25\%$$

- **Dificultad encontrada**

Se utilizo cognitive complexity.

$$DE = \frac{valor\ de\ la\ propiedad}{lineas\ de\ codigo}$$
$$DE = \frac{0}{26000}$$
$$DE = 0.12\%$$

- **Densidad de comentarios**

Se utilizo comment lines.

$$DC = 100\% - valor\ de\ la\ propiedad\%$$
$$DC = 100\% - 0.40\%$$
$$DC = 99.60\%$$

Para obtener el valor general de cada una de las métricas de las subcaracterísticas del producto de software, se aplicó promedio de las métricas del backend y del frontend, como se muestra a continuación. Esto permitirá obtener el grado de calidad de las subcaracterísticas aplicando la formulas **GS**, conforme se muestra a continuación.

a. Modularidad

Esta subcaracterística abarca métricas de calidad como: diseño, duplicaciones y cobertura. Para obtener resultados de cada métrica se aplicó:

- **Diseño**: Se dividió el valor de la propiedad para las líneas de código.
- **Duplicaciones**: se dividió el valor de la propiedad para líneas de código.

- **Cobertura**: se calculó (Vpropiedad/líneas de código))

En la **Tabla 2-A12** se muestra los valores de las métricas de calidad respecto al modularidad del backend.

Tabla 2-A12. Valores de las métricas de calidad, según el modularidad del backend.

Métricas	Valor de la propiedad	Resultado%
Diseño	277	12.82%
Duplicaciones	1228	56.83%
Cobertura	865	40.03%

En la **Tabla 3-A12** se muestra los valores de las métricas de calidad respecto al modularidad del frontend.

Tabla 3-A12. Valores de las métricas de calidad, según el modularidad del frontend.

Métricas	Valor de la propiedad	Resultado%
Diseño	1021	3.93%
Duplicaciones	2823	10.86%
Cobertura	1534	5.90%

En la **Tabla 4-A12** se muestra los resultados finales de cada métrica respecto al backend y el frontend, para lo cual se ha calculado mediante el promedio, lo que da a conocer el valor final de la subcaracterística modularidad.

Tabla 4-A12. Promedio de las métricas calidad en la característica del modularidad.

Métricas	Backend	Frontend	Promedio
Diseño	12.82%	3.93%	8.37%
Duplicaciones	56.83%	10.86%	33.84%
Cobertura	40.03%	5.90%	22.96%

Grado de la subcaracterística modularidad:

$$G = \frac{D + DP + CB}{3}$$

$$G = \frac{8.37\% + 33.84\% + 22.96\%}{3}$$

$$G = 78.27\%$$

b. Reusabilidad

Esta subcaracterística abarca métricas de calidad como: mala práctica, obsoleto, diseño y duplicaciones, para obtener resultados de cada métrica se aplicó:

- **Mala práctica, obsoleto y diseño:** Se dividió el valor de la propiedad para las líneas de código.
- **Duplicaciones:** Se dividió el valor de la propiedad para líneas de código.

En la **Tabla 5-A12** se muestra los valores de las métricas de calidad respecto al reusabilidad del backend.

Tabla 5-A12. Valores de las métricas de calidad, según la reusabilidad del backend.

Métricas	Valor de la propiedad	Resultado%
Mala práctica	6	0.28%
Obsoleto	6	0.28%
Diseño	277	12.82%
Duplicaciones	1228	56.83%

En la **Tabla 6-A12** se muestra los valores de las métricas de calidad respecto al reusabilidad del frontend.

Tabla 6-A12. Valores de las métricas de calidad, según la reusabilidad del frontend.

Métricas	Valor de la propiedad	Resultado%
Mala práctica	846	3.25%
Obsoleto	846	3.25%
Diseño	1021	3.93%
Duplicaciones	2823	10.86%

En la **Tabla 7-A172** se muestra los resultados finales de cada métrica respecto al backend y el frontend, para lo cual se ha calculado mediante el promedio, lo que da a conocer el valor final de la subcaracterística reusabilidad.

Tabla 7-A12. Promedio de las métricas calidad en la subcaracterística de la reusabilidad.

Métricas	Backend	Frontend	Promedio
Mala práctica	0.28%	3.25%	1.77%
Obsoleto	0.28%	3.25%	1.77%
Diseño	12.82%	3.93%	8.37%
Duplicaciones	56.83%	10.86%	33.84%

Grado de la subcaracterística reusabilidad:

$$G = \frac{MP + O + D + DP}{4}$$

$$G = \frac{1.77\% + 1.77\% + 8.37\% + 33.84\%}{4}$$

$$G = 88.56\%$$

c. Analizabilidad

Esta subcaracterística abarca métricas de calidad como: mala práctica, redundante, Cognitive Complexity, confuso, diseño, dificultad encontrada y densidad de comentarios, para obtener resultados de cada métrica se aplicó:

- **Mala práctica, redundante, Cognitive Complexity, confuso, diseño y dificultad encontrada**: Se dividió el valor de la propiedad para las líneas de código, además se multiplicó por 100 con el fin de obtener un porcentaje de las mismas.
- **Densidad de comentario:** Se resto el 100% para el total de líneas comentadas%.

En la **Tabla 8-A12** se muestra los valores de las métricas de calidad respecto al analizabilidad del backend.

Tabla 8-A12. Valores de las métricas de calidad, según la analizabilidad del backend.

Métricas	Valor de la propiedad	Resultado%
Mala práctica	6	0.28%
Redundante	1228	56.83%
Cognitive Complexity	0	0.00%
Confuso	6	0.28%
Diseño	277	12.82%
Dificultad encontrada	0	0%
Densidad de comentarios	20	80%

En la **Tabla 9-A12** se muestra los valores de las métricas de calidad respecto al reusabilidad del frontend.

Tabla 9-A12. Valores de las métricas de calidad, según la analizabilidad del frontend.

Métricas	Valor de la propiedad	Resultado%
Mala práctica	846	3.25%
Redundante	2823	10.86%
Cognitive Complexity	30	0.12%
Confuso	846	3.25%
Diseño	1021	3.93%
Dificultad encontrada	30	0.12%
Densidad de comentarios	0.40%	99.60%

En la **Tabla 10-A12** se muestra los resultados finales de cada métrica respecto al backend y el frontend, para lo cual se ha calculado mediante el promedio, lo que da a conocer el valor final de la subcaracterística analizabilidad.

Tabla 10-A12. Promedio de las métricas calidad en la subcaracterística de la analizabilidad.

Métricas	Backend	Frontend	Promedio
Mala práctica	0.28%	3.25%	1.77%
Redundante	56.83%	10.86%	33.84%
Cognitive Complexity	0.00%	0.12%	0.06%
Confuso	0.28%	3.25%	1.77%
Diseño	12.82%	3.93%	8.37%
Dificultad encontrada	0%	0.12%	0.06%
Densidad de comentarios	80%	99.60%	89.80%

Grado de la subcaracterística analizabilidad:

$$G = \frac{MP + R + CC + C + D + DE + DC}{7}$$

$$G = \frac{1.77\% + 33.84\% + 0.06 + 1.77\% + 8.37\% + 0.06\% + 89.80\%}{7}$$

$$G = 80.62\%$$

d. Capacidad para ser modificado

Esta subcaracterística abarca métricas de calidad como: mala práctica, redundante, diseño, y duplicaciones, para obtener resultados de cada métrica se aplicó:

- **Mala práctica, redundante y diseño:** Se dividió el valor de la propiedad para las líneas de código.
- **Duplicaciones:** Se dividió el valor de la propiedad para las líneas de código.

En la **Tabla 11-A12** se muestra los valores de las métricas de calidad respecto a la capacidad para ser modificado del backend.

Tabla 11-A12. Valores de las métricas de calidad, según la capacidad para ser modificado del backend.

Métricas	Valor de la propiedad	Resultado%
Mala práctica	6	0.28%
Redundante	1228	56.83%
Diseño	277	12.82%
Duplicaciones	1228	56.83%

En la **Tabla 12-A12** se muestra los valores de las métricas de calidad respecto a la capacidad para ser modificado del frontend.

Tabla 12-A12. Valores de las métricas de calidad, según la capacidad para ser modificado del frontend.

Métricas	Valor de la propiedad	Resultado%
Mala práctica	846	3.25%
Redundante	2823	10.86%
Diseño	1021	3.93%
Duplicaciones	2823	10.86%

En la **Tabla 13-A12**, se muestra los resultados finales de cada métrica respecto al backend y el frontend, para lo cual se ha calculado mediante el promedio, lo que da a conocer el valor final de la subcaracterística capacidad para ser modificado.

Tabla 13-A12. Promedio de las métricas calidad en la subcaracterística de la capacidad para ser modificado.

Métricas	Backend	Frontend	Promedio
Mala práctica	0.28%	3.25%	1.77%
Redundante	56.83%	10.86%	33.84%
Diseño	12.82%	3.93%	8.37%
Duplicaciones	56.83%	10.86%	33.84%

Grado de capacidad de ser modificado:

$$G = \frac{MP + R + D + DP}{4}$$

$$G = \frac{1.77\% + 33.84 + 8.37\% + 33.84\%}{4}$$

$$G = 80.54\%$$

e. Capacidad para ser probado

Esta subcaracterística abarca métricas de calidad como: redundante, diseño, dificultad encontrada y cobertura, para obtener resultados de cada métrica se aplicó:

- **Redundante, diseño y dificultad encontrada**: se dividió el valor de la propiedad para las líneas de código.
- **Cobertura**: se calculó (Vpropiedad/líneas de código)).

En la **Tabla 14-A12** se muestra los valores de las métricas de calidad respecto a la capacidad para ser probado del backend.

Backend: líneas de código.

Tabla 14-A12. Valores de las métricas de calidad, según la capacidad para ser probado del backend.

Métricas	**Valor de la propiedad**	**Resultado%**
Redundante	1228	56.83%
Diseño	277	12.82%
Dificultad encontrada	0	0%
Cobertura	865	40.03%

En la **Tabla 15-A12,** se muestra los valores de las métricas de calidad respecto a la capacidad para ser probado del frontend.

Tabla 15-A12. Valores de las métricas de calidad, según la capacidad para ser probado del frontend.

Métricas	**Valor de la propiedad**	**Resultado%**
Redundante	2823	10.86%
Diseño	1021	3.93%
Dificultad encontrada	30	0.12%
Cobertura	1534	5.90%

En la **Tabla 16-A12** se muestra los resultados finales de cada métrica respecto al backend y el frontend, para lo cual se ha calculado mediante el promedio, lo que da a conocer el valor final de la subcaracterística capacidad para ser probado.

Tabla 16-A12. Promedio de las métricas calidad en la subcaracterística de la capacidad para ser probado.

Métricas	**Backend**	**Frontend**	**Promedio**
Redundante	56.83%	10.86%	33.84%
Diseño	12.82%	3.93%	8.37%
Dificultad encontrada	0%	0.12%	0.06%
Cobertura	40.03%	5.90%	22.96%

Grado de capacidad de ser probado:

$$G = \frac{R + D + DE\ +\ CB}{4}$$

$$G = \frac{33.85\% + 8.37\% + 0.06\%\ + 22.96\%}{4}$$

$$G = 83.69\%$$

Anexo 13. Extracción de datos utilizando matrices bibliográficas

1. Extracción de Datos

En la **Tabla 1-A13** se muestra el resultado de la extracción de datos del documento 2.

Tabla 1-A13. Extracción de datos documento 2.

Descripción	Detalle
Título del documento	Mantenibilidad del código
Autor	Peter Hoey
Referencia/Documento	https://drive.google.com/file/d/15P850fVcARLbd3rTlwIgA5oesFuGx0uH/view?usp=sharing
Año	2018
Propósito	El propósito del presente documento es presentar los resultados de un estudio que examina la mantenibilidad del código fuente entregado por proyectos de código abierto, y discutir la importancia que tiene la mantenibilidad en el desarrollo de código abierto. Hace mención a métricas de calidad del código, y se enfoca en la aplicación de buenas prácticas de programación.
Conclusión destacada	Enfatiza la importancia de la mantenibilidad en el desarrollo de código abierto y se sugiere que los proyectos de código abierto pueden requerir reingeniería y aplicar prácticas de desarrollo para evitar la creación de sistemas obsoletos en la medida que envejecen.
Términos claves	Calidad de software, métricas de la mantenibilidad

En la **Tabla 2-A13** se muestra el resultado de la extracción de datos del documento 3.

Tabla 2-A13. Extracción de datos documento 3.

Descripción	Detalle
Título del documento	Generador de valores interesantes para casos de pruebas unitarias
Autor	Dania Mailen Rojas-Robert, Zeyla Pérez-Morales, Martha Dunia Delgado Dapena
Referencia/Documento	https://drive.google.com/file/d/1v0ojFDS1dQ1tE8bERsKd11GlCoIoLGNv/view?usp=sharing
Año	2019
Propósito	El propósito del presente documento es presentar una propuesta para la generación automática de valores interesantes para pruebas unitarias, utilizando técnicas de diseño de Bucles y Condiciones, con el objetivo de hacer las pruebas lo más abarcadoras posibles al tener en cuenta un subconjunto de los posibles casos y valores de prueba con mayor probabilidad de detectar errores en el software.
Conclusión destacada	El documento propone un modelo de optimización para la reducción de combinaciones de valores y destaca la importancia de la combinación adecuada de las diferentes técnicas de caja blanca para lograr una mayor detección de los errores de software.
Términos claves	Técnicas de diseño de Bucles y Condiciones, pruebas unitarias

En la **Tabla 3-A13** se muestra el resultado de la extracción de datos del documento 4.

Tabla 3-A13. Extracción de datos documento 4.

Descripción	Detalle
Título del documento	Inyección de Dependencias en el Lenguaje de Programación
Autor	Gutiérrez Morales, Carlos Eduardo; Verduzco Ramírez, Jesús Alberto; Farías Mendoza, Nicandro
Referencia/Documento	https://drive.google.com/file/d/1tQrq_b5IF-QhqAQ4cao1ZnUH6J0eQTa8/view?usp=sharing
Año	2015
Propósito	El presente documento muestra una librería de código abierto para la inyección de dependencias, con el fin de crear un código más flexible y fácil de mantener. Además, se mencionan conceptos y principios de diseño de software como la inyección de dependencias, el principio de inversión de control y la cohesión y acoplamiento entre otros, que pueden ser considerados buenas prácticas en programación.
Conclusión destacada	El uso de la inyección de dependencias puede mejorar la flexibilidad y la facilidad de mantenimiento del código, lo que a su vez puede ayudar a un proyecto de software a ser más adaptable a los cambios y más fácil. de mantener.
Términos claves	Inyección de dependencias, dependencias.

En la **Tabla 4-A13** se muestra el resultado de la extracción de datos del documento 5.

Tabla 4-A13. Extracción de datos documento 5.

Descripción	Detalle
Título del documento	Código limpio
Autor	Robert Martin
Referencia/Documento	https://drive.google.com/file/d/1iD61nbhkz08-QiZzQhv1v5gELchZXTDN/view?usp=sharing
Año	2018
Propósito	El propósito del presente libro es proporcionar orientación y enseñanzas sobre buenas prácticas de programación, con el objetivo de ayudar a los programadores a mejorar la calidad de su código y a desarrollar software bien diseñado. Se enfoca en principios como la ejecución de pruebas, la eliminación de duplicados, la expresión clara de la intención del programador y la minimización de clases y métodos. Además, se destaca la importancia de la legibilidad, la robustez y la expresividad del código, así como la necesidad de considerar el control de errores como una preocupación separada para facilitar el mantenimiento del código a largo plazo.
Conclusión	La calidad del código es fundamental para el éxito de un proyecto de software. Se destaca la importancia de mantener un código limpio, legible, expresivo y robusto, ya que esto no solo facilita la comprensión y mantenimiento del software, sino que también mejora su flexibilidad, capacidad de reutilización y capacidad de adaptación a cambios futuros. Se enfatiza la necesidad de ejecutar pruebas de manera oportuna, expresar claramente la intención del código a través de nombres adecuados y estructuras bien diseñadas, y considerar el control de errores como una preocupación separada.
Términos claves	Pruebas, eliminación de duplicados, control de errores, mantenibilidad.

En la **Tabla 5-A13** se muestra el resultado de la extracción de datos del documento 6.

Tabla 5-A13. Extracción de datos documento 6.

Descripción	Detalle
Título del documento	Patrones de diseño
Autor	Shvets Alexander
Referencia/Documento	https://drive.google.com/file/d/1mh9IYAALoFt7F9_Ot5aHhIf_ARNomGlS/view?usp=sharing
Año	2021
Propósito	El propósito del presente documento es proporcionar información sobre los patrones de diseño y los principios de diseño de software que pueden ayudar a los desarrolladores a crear arquitecturas de software flexibles, estables y fáciles de comprender, y así mejorar sus prácticas de programación.
Conclusión	El uso adecuado de patrones de diseño y principios de diseño de software puede mejorar significativamente la calidad de las prácticas de programación al crear arquitecturas de software flexibles, estables y fáciles de comprender. Además, estos principios pueden ayudar a los desarrolladores a reducir los costos y tiempos de desarrollo y aumentar la reutilización de código, lo que resulta en una mejor eficiencia y calidad del desarrollo de software en general.
Términos claves	Programación orientada a objetos (POO), patrones de diseño, principios de diseño de software, encapsulación, cohesión, acoplamiento, reutilización de código, flexibilidad, estabilidad, facilidad de comprensión y cambio constante.

En la **Tabla 6-A13** se muestra el resultado de la extracción de datos del documento 7.

Tabla 6-A13. Extracción de datos documento 7.

Descripción	Detalle
Título del documento	Personalización del proceso de pruebas unitarias empleando la herramienta NUnit
Autor	Susana González Espinosa, Rafael Bello Lara
Referencia/Documento	https://drive.google.com/file/d/136JeyMlgllk0hivwxnf611CYClqW7u24/view?usp=sharing
Año	2016
Propósito	El propósito del presente documento es proponer una estrategia para realizar pruebas unitarias de forma automatizada utilizando la herramienta NUnit. El objetivo de esta estrategia es facilitar la labor de los desarrolladores durante el proceso de construcción de un sistema informático, permitiéndoles detectar fallos en el código desde la etapa inicial del desarrollo del software y garantizar que el programa no contiene errores de diseño o codificación.
Conclusión	La conclusión importante del presente documento es que la realización de pruebas unitarias automatizadas es una práctica efectiva que puede mejorar significativamente la calidad del software, reducir los costos y el tiempo dedicado a las pruebas manuales y asegurar que el software desarrollado cumpla con las expectativas del cliente.
Términos claves	Pruebas unitarias, Desarrollo orientado a objetos.

En la **Tabla 7-A13** se muestra el resultado de la extracción de datos del documento 8.

Tabla 7-A13. Extracción de datos documento 8.

Descripción	Detalle
Título del documento	Implementación de pruebas de software para el módulo para el diseño de mapas de impresión para la plataforma ULTRON.
Autor	Yelena Vento Diaz, Gerdys Ernesto Jiménez Moya, Reinier Suarez Estevez.
Referencia/Documento	https://drive.google.com/file/d/1nUiG0Ix_PfzE5HA0sBB7yvYYbT6tIkGy/view?usp=sharing
Año	2019
Propósito	En cuanto a las buenas prácticas de programación, la propuesta de la presente investigación es hacer uso de los estándares de codificación, los cuales son un conjunto no formal de reglas que han ido surgiendo en las distintas comunidades de desarrolladores con el paso del tiempo y que bien aplicadas, pueden incrementar la calidad del código. Algunos beneficios de la codificación estandarizada es el mejoramiento de la comunicación en los equipos de desarrollo de software, reducir los errores de programación y mejorar la calidad del software.
Conclusión	La presente investigación respecto a las buenas prácticas de programación es que la definición y el correcto uso de los estándares de codificación, así como la realización de pruebas tanto unitarias como de integración y sistema son esenciales para la garantía de calidad del software
Términos claves	Estándares de codificación, codificación estandarizada.

En la **Tabla 8-A13** se muestra el resultado de la extracción de datos del documento 9.

Tabla 8-A13. Extracción de datos documento 9.

Descripción	Detalle
Título del documento	Clasificación y Evaluación de Métricas de Mantenibilidad Aplicables a Productos de Software Libre
Autor	José M. Ruiz, Cristian D. Pacifico, Martín M. Pérez
Referencia/Documento	https://drive.google.com/file/d/1W-qZN2hmmJHUDdRce598ClaLYytZVjYM/view?usp=sharing
Año	2018
Propósito	El propósito de la presente investigación es evaluar y clasificar métricas para la “mantenibilidad” aplicable a software libre y desarrollar herramientas que las implementen, con el objetivo de favorecer la comprensión y modificación del código de los productos de software libre.
Conclusión	La investigación busca establecer métricas cuantitativas y objetivas para evaluar la mantenibilidad de productos de software libre, permitiendo medir su calidad interna y facilitando su comprensión y modificación. Además, se menciona que es fundamental que la mantenibilidad se establezca como objetivo tanto en las fases iniciales de desarrollo del ciclo de vida del software como durante el mantenimiento posterior, para reducir los efectos laterales y garantizar el éxito del proyecto.
Términos claves	Software libre, métricas, mantenibilidad, calidad.

En la **Tabla 9-A13** se muestra el resultado de la extracción de datos del documento 10.

Tabla 9-A13. Extracción de datos documento 10.

Descripción	Detalle
Título del documento	Legibilidad
Autor	Mary C. Dyson
Referencia/Documento	https://drive.google.com/file/d/1H2gLkqhiTFLzasabGhe_j4jZCMtsUPll/view?usp=sharing
Año	2023
Propósito	El propósito de la presente investigación es analizar y contextualizar los resultados obtenidos en relación con la legibilidad en el diseño, con el fin de comprender cómo estos resultados pueden influir en la práctica del diseño. Se destaca la importancia de no simplificar demasiado las directrices y recomendaciones para lograr una óptima legibilidad, ya que cada diseño puede presentar particularidades que influyen en su efectividad.
Conclusión	La investigación destaca que cuando la investigación empírica respalda las prácticas existentes, es positivo y alentador. Sin embargo, si hay discrepancias entre la práctica existente y la investigación, o entre diferentes estudios, puede haber motivos de preocupación. Se sugiere resolver estas discrepancias mediante nuevos estudios pertinentes para mejorar la comprensión y la aplicación de las buenas prácticas de programación.
Términos claves	Mejorar comprensión y aplicación, prácticas existentes.

En la **Tabla 10-A13** se muestra el resultado de la extracción de datos del documento 11.

Tabla 10-A13. Extracción de datos documento 11.

Descripción	Detalle
Título del documento	Fundamentos de programación
Autor	Luis Joyanes Aguilar
Referencia/Documento	https://drive.google.com/file/d/1z1aclLvrJ2sxc-vtt5VuU49L3SZ5KhHZ/view?usp=sharing
Año	2008
Propósito	El propósito principal del libro "Fundamentos de Programación: Algoritmos, estructura de datos y objetos", es introducir los conceptos fundamentales de la programación y las buenas prácticas para escribir código de calidad. Algunos de los objetivos clave del libro respecto a las buenas prácticas de programación.
Conclusión	La investigación destaca la aplicación de buenas prácticas de programación para la solución de defectos.
Términos claves	Duplicación de código, comentarios, documentación, estándares de codificación.

En la **Tabla 11-A13** se muestra el resultado de la extracción de datos del documento 12.

Tabla 11-A13. Extracción de datos documento 12.

Descripción	Detalle
Título del documento	Guía de aprendizaje de programación.
Autor	Joe Llerena Izquierdo
Referencia/Documento	https://drive.google.com/file/d/1J0LhyyX5kPN657thg1UnxRit7P05eOAJ/view?usp=sharing
Año	2023
Propósito	El propósito de la presente investigación es proporcionar recomendaciones y pautas para que los estudiantes universitarios puedan aprender a programar de manera responsable y eficiente. Se enfoca en establecer un tiempo y espacio adecuados para las tareas académicas, promoviendo valores como la responsabilidad, dedicación y constancia. Además, se sugiere realizar prototipos funcionales y reescribirlos para mejorar, siguiendo las fases de desarrollo de un programa.
Conclusión	La conclusión más importante de la presente investigación es que se deben seguir las fases de desarrollo de un programa, realizando prototipos funcionales y reescribiéndolos para mejorarlos. Se destaca la importancia de establecer un tiempo y espacio adecuados para las tareas académicas, promoviendo valores como la responsabilidad, dedicación y constancia en el aprendizaje de la programación.
Términos claves	Comentarios, documentación, estándares de codificación, transparencia.

En la **Tabla 12-A13** se muestra el resultado de la extracción de datos del documento 13.

Tabla 12-A13. Extracción de datos documento 13.

Descripción	Detalle
Título del documento	Herramienta de apoyo en la detección de reutilización de código fuente
Autor	Raymundo Picazo-Alvarez, Esaú Villatoro-Tello, Wulfrano A. Luna-Ramírez, Carlos R. Jaimez González
Referencia/Documento	https://drive.google.com/file/d/1AWtwVlMU-9zOtpdIrDcTmCGZyIKv69WA/view?usp=sharing
Año	2015
Propósito	El propósito de la presente investigación es proponer una herramienta orientada al combate del plagio de código fuente en programas escritos en un mismo lenguaje de programación. Dada la facilidad de acceso y las prácticas de reutilización de contenidos sin citar las fuentes, surge la necesidad de contar con herramientas para combatir el plagio, en especial, de código fuente.
Conclusión	La presente investigación es que se propone una herramienta para la detección de la reutilización de código fuente. Hasta el momento, se ha observado que los comentarios, palabras reservadas y caracteres repetitivos (como puntos, comas y los signos propios de los lenguajes de programación) pueden interferir en la detección, pero el sistema propuesto ha demostrado efectividad en detectar casos de reutilización alta.
Términos claves	Similitud de documentos, reutilización de código fuente y procesamiento del lenguaje natural.

En la **Tabla 13-A13** se muestra el resultado de la extracción de datos del documento 14.

Tabla 13-A13. Extracción de datos documento 14.

Descripción	Detalle
Título del documento	Optimización de Compiladores
Autor	Joseph Raphael Gómez Tzorin
Referencia/Documento	https://drive.google.com/file/d/1GN8i8TlqKpKNQnaaeeZwPwOACtTcKqG-/view?usp=sharing
Año	2023
Propósito	El propósito de esta investigación es exponer técnicas de implementación de compiladores, técnicas de optimización de código que puedan mejorar el desempeño de los programas generados por los compiladores, sin alterar su comportamiento, verificación y validación de compiladores.
Conclusión	La optimización de compiladores y la garantía de la calidad y seguridad del código generado son procesos esenciales que deben realizarse para garantizar que los programas sean eficientes y seguros.
Términos claves	Reordenamiento, Subyacentes, Ejecutable, Optimización

En la **Tabla 14-A13** se muestra el resultado de la extracción de datos del documento 15.

Tabla 14-A13. Extracción de datos documento 15.

Descripción	Detalle
Título del documento	Modelo de Pruebas de Regresión Automatizadas en Procesos de Integración Continua en Sistemas Web.
Autor	Oscar Wile Mamani Alanoca
Referencia/Documento	https://drive.google.com/file/d/1eomwH7rDqTM09GFLzV6RWrLz887dkpB2/view?usp=sharing.
Año	2020
Propósito	El propósito del presente proyecto de investigación es presentar un modelo de pruebas de regresión automatizadas en procesos de integración continua en sistemas web y su implementación como prototipo, con el objetivo de evaluar herramientas y metodologías para automatizar las pruebas de regresión y procedimientos de integración continua. para obtener eficiencia en la detección oportuna de defectos en el software de sistemas web.
Conclusión	La implementación de un modelo de pruebas de regresión automatizadas en procesos de integración continua mejora la eficiencia en la detección oportuna de defectos en el software en un 29,16%. Además, esto permite disminuir el tiempo de pruebas de regresión y procesos manuales con la automatización, así como optimizar los recursos a utilizar.
Términos claves	Pruebas de software, casos de prueba, automatización de pruebas, integración continua.

En la **Tabla 15-A13** se muestra el resultado de la extracción de datos del documento 16.

Tabla 15-A13. Extracción de datos documento 16.

Descripción	Detalle
Título del documento	Patrones de diseño.
Autor	Erich Gama.
Referencia/Documento	https://drive.google.com/file/d/1nBzbIqjT2Jb8Ye2rArnufRqV9b08KeVQ/view?usp=sharing.
Año	2018
Propósito	El libro trata sobre patrones de diseño en programación orientada a objetos, específicamente los patrones estructurales para crear diseños de software flexibles y extensibles, permitiendo adaptar y componer objetos de formas sofisticadas.
Conclusión	Los patrones estructurales permiten crear diseños de software más flexibles y extensibles, facilitando la adaptación y composición de objetos de manera sofisticada.
Términos claves	Composición, encapsulación, abstracción, modularidad.

En la **Tabla 16-A13** se muestra el resultado de la extracción de datos del documento 17.

Tabla 16-A13. Extracción de datos documento 17.

Descripción	Detalle
Título del documento	Depuración Declarativa de Programas Lógico Funcionales
Autor	Francisco Jos´e Correa Zabala
Referencia/Documento	https://drive.google.com/file/d/1vOcc3sdCBXw7cpDBChv_5YK3e8tNXxf8/view?usp=sharing
Año	2022
Propósito	El propósito de la investigación es desarrollar un marco teórico sólido y genérico para la depuración declarativa de programas lógico-funcionales, que permita detectar y diagnosticar errores de manera eficaz, aprovechando las ventajas de los paradigmas declarativos. Esto contribuye a las buenas prácticas de programación al facilitar la verificación y corrección de este tipo de programas.
Conclusión	El enfoque basado en semánticas declarativas ricas y su aproximación abstracta constituye una base sólida para mejorar las prácticas de programación en el paradigma lógico-funcional. Permite verificar formalmente las propiedades de los programas, detectar y corregir errores de manera efectiva, y desarrollar herramientas de análisis prácticas.
Términos claves	Corrección y completitud, interpretación abstracta, documentación.

En la **Tabla 17-A13** se muestra el resultado de la extracción de datos del documento 18.

Tabla 17-A13. Extracción de datos documento 18.

Descripción	Detalle
Título del documento	Principios SOLID
Autor	Antonio Leiva
Referencia/Documento	https://drive.google.com/file/d/18rmMWJaAe0UeT3bbRpcVfWpkLcU9ql3b/view?usp=sharing
Año	2016
Propósito	El propósito de la presente investigación es presentar los principios SOLID de la programación orientada a objetos, proporcionando una guía rápida para aprender qué son y cómo aplicarlos en el día a día. En otras palabras, el propósito es presentar las buenas prácticas de programación para desarrolladores de software.
Conclusión	La implementación de los principios SOLID en el proceso de desarrollo de software, favorece la creación de un código más sostenible, mantenible, escalable y de mayor calidad, además de permitir la reducción de deudas técnicas y de la complejidad del código.
Términos claves	Principios SOLID, programación orientada a objetos, la responsabilidad única, el principio abierto/cerrado, el principio de Liskov, el principio de segregación de interfaces y el principio de inversión de dependencias.

En la **Tabla 18-A13** se muestra el resultado de la extracción de datos del documento 19.

Tabla 18-A13. Extracción de datos documento 19.

Descripción	Detalle
Título del documento	Python Programación
Autor	Luis Rodríguez Ojeda
Referencia/Documento	https://drive.google.com/file/d/1nXuudRzQHPqQwf_V6cJlzUeUcv_sMBDJ/view?usp=sharing
Año	2016
Propósito	El propósito de la presente investigación es desarrollar el aprendizaje de la programación con el lenguaje Python aplicando metodologías de programación estructurada, modular y orientada a objetos, y fomentar el uso de buenas prácticas como la documentación, pruebas y depuración para asegurar precisión y claridad en el desarrollo de programas.
Conclusión	La conclusión más importante de la presente investigación respecto a las buenas prácticas de programación es la necesidad de una documentación cuidadosa y detallada en cada etapa del desarrollo, así como la importancia de realizar pruebas exhaustivas y mantener versiones anteriores de los programas para asegurar la calidad y la facilidad de mantenimiento del código.
Términos claves	Documentación, pruebas, depuración, modularidad, y versiones.

En la **Tabla 19-A13** se muestra el resultado de la extracción de datos del documento 20.

Tabla 19-A13. Extracción de datos documento 20.

Descripción	Detalle
Título del documento	Uso del sistema de control de versiones Git en reemplazo de un sistema de administración de la enseñanza.
Autor	Gunnar Wolf
Referencia/Documento	https://drive.google.com/file/d/1DOAi77MIwDBXMvLAUVtp9vlPJvI3RlRd/view?usp=sharing.
Año	2016
Propósito	El propósito de esta investigación es presentar una experiencia sobre el uso de un sistema de Administración de la Enseñanza (LMS) por completo por el uso del sistema de control de versiones Git.
Conclusión	Muestra la experiencia de utilizar el sistema de control de versiones Git como plataforma de enseñanza en lugar de los sistemas de administración de la enseñanza tradicionales.
Términos claves	Versionamiento, manejo de versiones en GIT.

En la **Tabla 20-A13** se muestra el resultado de la extracción de datos del documento 21.

Tabla 20-A13. Extracción de datos documento 21.

Descripción	Detalle
Título del documento	Uso del sistema de control de versiones Git en reemplazo de un sistema de administración de la enseñanza.
Autor	Gunnar Wolf
Referencia/Documento	https://drive.google.com/file/d/1DOAi77MIwDBXMvLAUVtp9vlPJvI3RlRd/view?usp=sharing.
Año	2016
Propósito	El propósito de esta investigación es presentar una experiencia sobre el uso de un sistema de Administración de la Enseñanza (LMS) por completo por el uso del sistema de control de versiones Git.
Conclusión	Muestra la experiencia de utilizar el sistema de control de versiones Git como plataforma de enseñanza en lugar de los sistemas de administración de la enseñanza tradicionales.
Términos claves	Versionamiento, manejo de versiones en GIT.

En la **Tabla 21-A13** se muestra el resultado de la extracción de datos del documento 22.

Tabla 21-A13. Extracción de datos documento 22.

Descripción	Detalle
Título del documento	Principios y patrones del desarrollo de software.
Autor	Autentia.
Referencia/Documento	https://drive.google.com/file/d/1VpFCCjdkY90ow5LND773TG4WaZfeb5IF/view?usp=sharing.
Año	2020
Propósito	El propósito de la presente investigación respecto a las buenas prácticas de programación es proporcionar una serie de reglas y recomendaciones específicas que los programadores deben seguir para escribir un código limpio, comprensible y fácil de mantener. Además, busca reunir la experiencia de los profesionales del sector en una serie de recetas o buenas prácticas para que el software crezca y evolucione de una forma sostenible.
Conclusión	La conclusión más importante de la presente investigación respecto a las buenas prácticas de programación es que aplicar principios y patrones de diseño, como la regla del Boy Scout, es esencial para mejorar la calidad del código y evitar su deterioro. Esto permite que el software crezca y evolucione de manera sostenible, facilitando futuras entregas de valor rápidas y de calidad sin romper la funcionalidad existente.
Términos claves	Principios SOLID, regla del Boy Scout, deuda técnica, calidad del código, sostenibilidad, y trabajo en equipo.

En la **Tabla 22-A13** se muestra el resultado de la extracción de datos del documento 23.

Tabla 22-A13. Extracción de datos documento 23.

Descripción	Detalle
Título del documento	Buenas prácticas hacia el éxito en el desarrollo de software.
Autor	Juan García Carmona.
Referencia/Documento	https://drive.google.com/file/d/1VpFCCjdkY90ow5LND773TG4WaZfeb5IF/view?usp=sharing.
Año	2015
Propósito	El propósito de la investigación de Buenas prácticas hacia el éxito en el desarrollo de software es analizar los principios SOLID y GRASP, y poner en práctica algunos de los principios básicos del diseño y la programación orientada a objetos con el objetivo de tomar conciencia de diversas buenas prácticas que harán que los sistemas que diseñamos y los productos de software que desarrollamos sean de la más alta calidad.
Conclusión	Las buenas prácticas hacia el éxito en el desarrollo de software es que la aplicación de los principios SOLID y GRASP en la programación orientada a objetos proporciona una estructura sólida y fácilmente utilizable para el desarrollo de software de alta calidad, que cumpla con los objetivos establecidos, tenga una excelente probabilidad de cumplir los requisitos del usuario final, y de ser mantenible a lo largo del tiempo y fácilmente extensible.
Términos claves	SOLID, GRASP, programación orientada a objetos, calidad del software, alta cohesión, bajo acoplamiento, indirección, polimorfismo.

En la **Tabla 23-A13** se muestra el resultado de la extracción de datos del documento 24.

Tabla 23-A13. Extracción de datos documento 24.

Descripción	Detalle
Título del documento	Analyzing Static Analysis Metric.
Autor	Thomas Karanikiotis, Michail D. Papamichail, Andreas L. Symeonidis.
Referencia/Documento	https://drive.google.com/file/d/16xpm875UkZS_MlztwO9kaurXJN1LfIwj/view?usp=sharing.
Año	2021
Propósito	El propósito de esta investigación es la evaluación de mantenibilidad de forma temprana los componentes de software que son no mantenibles. Se argumenta un conjunto de correcciones que eliminan la deuda técnica, sino más bien el resultado de un proceso continuo a lo largo del ciclo de vida del software, en propiedades principales del código fuente, como complejidad, cohesión, herencia y acoplamiento.
Conclusión	El análisis de las tendencias de las métricas de análisis estático de manera semanal a nivel de clase de software puede proporcionar información valiosa sobre el grado de mantenibilidad.
Términos claves	Complejidad, cohesión, herencia, acoplamiento, plataformas de alojamiento de código.

En la **Tabla 24-A13** se muestra el resultado de la extracción de datos del documento 25.

Tabla 24-A13. Extracción de datos documento 25.

Descripción	Detalle
Título del documento	Herramienta para la Gestión de Programming Project.
Autor	Alba Hurtado Redondo.
Referencia/Documento	https://drive.google.com/file/d/1EczLlg1_TZf9PsBHJgsP9m_JT_vdnSrN/view?usp=sharing.
Año	2021
Propósito	El propósito de la presente investigación es desarrollar una aplicación web desde cero, que automatiza la gestión de grupos de alumnos y recopila información relevante utilizando la API de las herramientas GitLab y SonarQube, con el fin de minimizar el tiempo dedicado a la corrección del código.
Conclusión	La integración de SonarQube en la herramienta desarrollada, se ha promovido una mayor calidad del código y se ha fomentado el uso de buenas prácticas de programación.
Términos claves	Cobertura, pruebas de calidad, documentación.

En la **Tabla 25-A13** se muestra el resultado de la extracción de datos del documento 26.

Tabla 25-A13. Extracción de datos documento 26.

Descripción	Detalle
Título del documento	Herramientas y buenas prácticas para el desarrollo, mantenimiento y evolución de Software en Java.
Autor	Radu Dumitru Boboia.
Referencia/Documento	https://drive.google.com/file/d/1K4gSw0HwRc1P8hPSeT68Qij8xKrYdGL1/view?usp=sharing.
Año	2021
Propósito	El propósito del presente trabajo es definir una serie de buenas prácticas en el desarrollo, mantenimiento y evolución de software en Java, que afectarían positivamente tanto al proceso como al producto en cuanto a su estructuración y calidad se refiere.
Conclusión	La conclusión más importante de la presente investigación respecto a las buenas prácticas de programación es que aplicar estas prácticas puede hacer que el proceso de desarrollo, mantenimiento y evolución de un software sea más sencillo, rápido y estructurado.
Términos claves	Código Duplique, reutilización, alta cohesión, bajo acoplamiento, documentación, patrones, excepciones, tratamiento de excepciones.

En la **Tabla 26-A13** se muestra el resultado de la extracción de datos del documento 27.

Tabla 26-A13. Extracción de datos documento 27.

Descripción	Detalle
Título del documento	Técnicas de Refactorización.
Autor	Ángel Cuenca Ortega.
Referencia/Documento	https://drive.google.com/file/d/17VqW6CqucSnepAxxwAZasmokUTH-gxkZ/view?usp=sharing.
Año	2015
Propósito	El propósito de la investigación es estudiar y aplicar la refactorización de programas funcionales para mejorar la calidad del código, centrándose en la legibilidad, mantenimiento y estética del mismo.
Conclusión	La investigación concluye que la refactorización funcional en programas funcionales puede ser altamente beneficiosa, ya que mejora la calidad del código al enfocarse en la legibilidad, mantenimiento y estética. Se destaca que los lenguajes de programación funcional son más adecuados para la refactorización debido a su estilo de programación "primero codificar, luego revisar", lo que puede llevar a mejoras significativas en la programación funcional.
Términos claves	Refactorización funcional, lenguajes de programación funcional.

En la **Tabla 27-A13** se muestra el resultado de la extracción de datos del documento 28.

Tabla 27-A13. Extracción de datos documento 28.

Descripción	Detalle
Título del documento	Análisis y clasificación de atributos de mantenibilidad del software
Autor	J.D. Erazo, A.S. Florez y F.J. Pino..
Referencia/Documento	https://drive.google.com/file/d/1yxUCoExmfHqdZrOTehUd0Erke-sKj2DP/view?usp=sharing.
Año	2016
Propósito	El propósito de la presente investigación es identificar y clasificar los atributos de mantenibilidad del software, centrándose en subcaracterísticas como capacidad para ser analizado, modularidad, capacidad para ser modificado, reusabilidad y capacidad para ser probado.
Conclusión	La investigación destaca la importancia de considerar diversos factores como modularidad, documentación, facilidad de lectura, consistencia, simplicidad, entre otros, para lograr un alto nivel de mantenibilidad del software.
Términos claves	Modularidad, documentación, facilidad de lectura, consistencia, simplicidad, capacidad de expansión, instrumentación, estandarización, nivel de validación y pruebas, complejidad, trazabilidad, propiedades estructurales o de código, habilidades de equipo de mantenimiento, elección de clases y nombre, diseño, patrones de diseño, arquitectura, componente, encapsulamiento, herencia, librerías, comentarios, y reusabilidad del software.

Anexo 14. Planificación de entrevista y encuesta

- **Preguntas de la entrevista**

a. Docentes.

1. ¿Reconoce alguna de las siguientes métricas para la medición de calidad respecto a la mantenibilidad?

Mala practica
Redundante
Complejidad cognitiva
Obsoleto
Confuso
Diseño
Dificultad encontrada
Densidad de comentarios
Duplicaciones
Cobertura

2. ¿Cree que sería útil que el software presente escalas de valoración de referencia para las métricas?

3. ¿Cree que sería útil que el software presente escalas aceptación de referencia para esas métricas?

4. ¿Cómo considera que debería estructurarse la lista de verificación para las pruebas manuales de mantenibilidad (métricas o subcaracterísticas etc.)?

5. ¿Cómo le gustaría que el software permita registrar y documentar los resultados de las pruebas manuales de mantenibilidad (formulario digital, una plantilla editable, o alguna otra forma de captura de información)?

6. ¿Qué tipo de información le gustaría que se registre (comentarios, calificación)?

7. ¿Considera importante que la lista incluya instrucciones o guías de uso?

8. ¿Cómo le gustaría que el software presente los resultados de la verificación de mantenibilidad?

9. ¿Preferiría visualizaciones gráficas y sugerencias?

10. ¿Qué otro tipo de información le gustaría que se incluya en los resultados?

11. ¿Considera importante que el software genere un historial de evaluaciones de mantenibilidad a lo largo del tiempo?

12. ¿Qué otras funcionalidades o características le gustarían que tuviera el software para la mejora de la mantenibilidad del código?

b. Gestor académico.

1. ¿Qué funcionalidades considera esenciales para la gestión de usuarios en la aplicación?
2. ¿Qué tipo de autenticación considera más adecuada para el registro de usuarios (ej. contraseña, autenticación de dos factores)?
3. ¿Considera necesario actualizar los datos de pruebas manuales como listas de verificación para evaluar la mantenibilidad del código (Ejemplo: tablas de aceptación y escala de valoración)?
4. ¿Considera necesario actualizar los datos de las pruebas automáticas para calcular el grado mantenibilidad del código mediante APIs (Ejemplo: tablas de aceptación y escala de valoración)?
5. ¿En qué formato prefiere recibir los informes de mantenibilidad?
6. ¿Considera importante que el sistema mantenga un historial de todas las evaluaciones de mantenibilidad realizadas?
7. ¿Qué características considera esenciales para la interfaz de usuario de la aplicación?

- **Preguntas encuesta**

1. ¿Considera útil aplicar pruebas manuales que incluya una lista de verificación para evaluar el cumplimiento de las pautas de mantenibilidad del software?
   - Si, lo considero.
   - No, lo considero.
2. ¿Le gustaría que las pruebas manuales incluyan la revisión de buenas prácticas de programación del código?
   - Si, lo considero.
   - No, lo considero.
3. ¿Es importante para usted que las pruebas manuales de calidad relacionadas con el código cubran las subcaracterísticas de la mantenibilidad de la norma ISO/IEC 25000?
   - Si, lo considero.
   - No, lo considero.

4. Seleccione cuáles subcaracterísticas consideraría necesarias para verificar mantenibilidad basado en la norma ISO/IEC 25000.
   - Modularidad
   - Reusabilidad
   - Analizabilidad
   - Capacidad de ser probado
   - Capacidad de ser modificado
5. ¿Le gustaría que las pruebas manuales generen un informe detallado con recomendaciones de mejora?
   - Si, lo considero.
   - No, lo considero.
6. ¿Considera útil el uso de APIs de análisis de código estático para la evaluación de mantenibilidad?
   - Si, lo considero.
   - No, lo considero.
7. Actualmente, la utilización de analizadores de código estático mediante servicios (APIs) son críticos para la evaluación de la mantenibilidad en el desarrollo de software.

   ¿Considera importante la utilización de APIs de análisis de código estático para calcular el grado de mantenibilidad basado en la norma ISO/IEC 25000?
   - Si, lo considero.
   - No, lo considero.
8. ¿Qué atributos considera necesarios para representar el grado de mantenibilidad?
   - Visualización gráfica de las métricas de mantenibilidad.
   - Recomendaciones para mejora de la mantenibilidad basada en cada métrica.
   - Informe detallado de la evaluación aplicada.
9. ¿Le gustaría recibir los informes de mantenibilidad en un formato PDF?
   - Si, lo considero.
   - No, lo considero.
10. ¿Es importante que el informe detalle las métricas utilizadas para la evaluación?
    - Si, lo considero.
    - No, lo considero.

11. ¿Le gustaría que el informe proporcione recomendaciones específicas para mejorar la mantenibilidad?

- Si, lo considero.
- No, lo considero.

12. ¿Es importante que el software incluya un historial de evaluaciones para seguimiento a lo largo del tiempo?

- Si, lo considero.
- No, lo considero.

- **Validez de la entrevista y encuesta**

Yo, **Gilson Orlando Quezada Guartizaca**, estudiante de la Universidad Nacional de Loja en calidad de tesista, declaro en forma libre y voluntaria que las interrogantes mostradas para la realización de entrevista y la encuesta a los docentes y estudiantes del itinerario de IS de la carrera de Computación de la Universidad Nacional de Loja, son válidas y auténticas. Con el objetivo de utilizar la información pertinente para el desarrollo del presente Trabajo de Integración Curricular que versa como segundo objetivo sobre "*Desarrollar un software web tomando como referencia XP como metodología de desarrollo, para solucionar el grado de mantenibilidad de una de las aplicaciones web diagnosticadas en el objetivo uno*". Las mismas que ha sido evaluadas y aceptadas por el director/a del presente trabajo de integración curricular.

Se adjunta firma del director de tesis.

Firma: _____

Ing. Francisco J Álvarez-Pineda Mg. Sc

DOCENTE DE LA CARRERA DE COMPUTACIÓN

Anexo 15. Transcripción de entrevista

- **Perfil "Gestor de la carrera"**

La siguiente entrevista se aplicó al gestor de la carrera como responsable del centro de datos de los laboratorios, es decir rol administrador.

- **Planificación de la entrevista**

Entrevistador	Gilson Orlando Quezada Guartizaca
Entrevistado	Ing. Pablo Fernando Ordoñez
Cargo del entrevistado	Gestor de la Carrera de Computación
Fecha de entrevista	05/07/2024
Hora de entrevista	8 h54 am
Canal de Comunicación	Entrevista presencial

- **Transcripción de la entrevista**

1. **¿Qué funcionalidades considera esenciales para la gestión de usuarios en la aplicación?**

   Actualmente para aplicaciones desarrolladas en la carrera en la parte de gestión de usuarios se está implementando Aerobase IAM.

2. **¿Qué tipo de autenticación considera más adecuada para el registro de usuarios (ej. contraseña, autenticación de dos factores)?**

   De acuerdo a lo mencionado anteriormente, la autenticación de usuarios llevaría llevarse a cabo mediante Aerobase IAM lo que permitirá identificar a los usuarios registrados de la carrera de computación y habilitar el registro de usuarios externos.

3. **¿Considera necesario actualizar los datos de pruebas manuales como listas de verificación para evaluar la mantenibilidad del código (Ejemplo: tablas de aceptación y escala de valoración)?**

   Si, se considera necesario actualizar datos de pruebas manuales, dado que estos deberían ser volátiles.

4. **¿Considera necesario actualizar los datos de las pruebas automáticas para calcular el grado mantenibilidad del código mediante APIs (Ejemplo: tablas de aceptación y escala de valoración)?**

   Si se necesario actualizar datos de pruebas automáticos, pues deberían ser volátiles.

5. **¿En qué formato prefiere recibir los informes de mantenibilidad?**
   Es conveniente que se exporten en formato PDF, ya que esto añadirá un valor agregado al software.
6. **¿Considera importante que el sistema mantenga un historial de todas las evaluaciones de mantenibilidad realizadas?**
   Si es pertinente, puesto que permitirá mantener un registro de evaluaciones de nuestro software lo que facilitará un análisis continuo y mejorará el control de la calidad, especialmente la del desarrollo del código.
7. **¿Qué características considera esenciales para la interfaz de usuario de la aplicación?**
   Facilidad de uso e intuitiva.
8. **¿Qué otra función considera esencial en la aplicación respecto a lo anterior?**
   Seria pertinente, que al finaliza una evaluación un docente de la carrera la firme electrónicamente agregando valor y proporcionando un respaldo adicional como anexo de la evaluación.

- **Validez de la entrevista**

Yo, **Gilson Orlando Quezada**, estudiante de la Universidad Nacional de Loja en calidad de tesista, declaro en forma libre y voluntaria que la presente entrevista ha sido realizada al Ing. Pablo Fernando Ordoñez, como gestor de la carrera de Computación de la Universidad Nacional de Loja y responsable del centro de datos del laboratorio.

En el siguiente enlace se puede tomar como evidencia y valides de la aplicación de la entrevista mediante un audio recolectado: https://drive.google.com/drive/folders/1hbkohOzS9zv_6M5Jf1ea0vtWSmndpGEV?usp=sharing.

- **Perfil “Docente de la carrera de computación”**

La siguiente entrevista se aplicó a un docente relacionado con el área desarrollo de software de la carrera de computación como parte interesada, es decir rol usuario.

- **Planificación de la entrevista**

Entrevistador	Gilson Orlando Quezada Guartizaca
Entrevistado	Ing. Edison Coronel Romero
Cargo del entrevistado	Docente de la Carrera de Computación
Fecha de entrevista	04/07/2024
Hora de entrevista	8 h54 am
Canal de Comunicación	Entrevista presencial

- **Transcripción de la entrevista**

1. **¿Reconoce alguna de las siguientes métricas para la medición de calidad respecto a la mantenibilidad?**

Mala practica
Redundante
Complejidad cognitiva
Obsoleto
Confuso
Diseño
Dificultad encontrada
Densidad de comentarios
Duplicaciones
Cobertura

Si reconoce la mayoría de las métricas, y a su vez hace referencia a que muchas de ellas están relacionadas con la aplicación de buenas prácticas de programación, como los principios SOLID. Aunque es posible aplicar todas las métricas, se considera necesario adaptarse a aquellas que se pueden medir efectívamele

2. **¿Cree que sería útil que el software presente escalas de valoración de referencia para las métricas?**

Si es necesario para poder analizar qué medida de calificación tiene la calidad del código.

3. **¿Cree que sería útil que el software presente escalas aceptación de referencia para esas métricas?**

   Si es necesario, la misma nos permitirá identificar que defecto podemos tener en nuestro código y poderlo corregir.

4. **¿Cómo considera que debería estructurarse la lista de verificación para las pruebas manuales de mantenibilidad (métricas o subcaracterísticas etc.)?**

   Es conveniente relacionarlas por subcaracterísticas para identificar que área afecta las métricas.

5. **¿Cómo le gustaría que el software permita registrar y documentar los resultados de las pruebas manuales de mantenibilidad (formulario digital, una plantilla editable, o alguna otra forma de captura de información)?**

   Es necesario en un formato digital como checklist para seleccionar el cumplimiento de las pruebas y para documentar es conveniente dividir como los más críticos y menos críticos.

6. **¿Qué tipo de información le gustaría que se registre (comentarios, calificación)?**

   Es conveniente registrar comentarios, además de ponderar y sacar un promedio como calificación.

7. **¿Considera importante que la lista incluya instrucciones o guías de uso?**

   Si, siempre es necesario comprender cada test, además de una descripción de las métricas.

8. **¿Cómo le gustaría que el software presente los resultados de la verificación de mantenibilidad?**

   Se podría estructurar o dividir entre las métricas con calificaciones bajas y altas, para enviar una alerta respecto a la misma.

9. **¿Preferiría visualizaciones gráficas y sugerencias?**

   Es atractivo y contribuye a una mejor comprensión.

10. **¿Qué otro tipo de información le gustaría que se incluya en los resultados?**

    Se podría agregar sugerencias de acuerdo a las métricas

11. **¿Considera importante que el software genere un historial de evaluaciones de mantenibilidad a lo largo del tiempo?**
    Si es necesario, para mi parecer lo veo necesario.
12. **¿Qué otras funcionalidades o características le gustarían que tuviera el software para la mejora de la mantenibilidad del código?**
    Ninguno, me conformo con mencionado anteriormente.

- **Validez de la entrevista**

Yo, **Gilson Orlando Quezada**, estudiante de la Universidad Nacional de Loja en calidad de tesista, declaro en forma libre y voluntaria que la presente entrevista ha sido realizada al Ing. Edison Coronel Romero, como docente de la carrera de Computación de la Universidad Nacional de Loja.

En el siguiente enlace se puede tomar como evidencia y valides de la aplicación de la entrevista mediante un audio recolectado: https://drive.google.com/drive/folders/1jEeYKSVj9YM6F3CeJDklbkUZhMGJv42z?usp=sharing.

- **Perfil “Docente de la carrera de computación”**

La siguiente entrevista se aplicó a un docente relacionado con el área ingeniería del software de la carrera de computación como parte interesada, es decir rol usuario.

- **Planificación de la entrevista**

Entrevistador	Gilson Orlando Quezada Guartizaca
Entrevistada	Ing. Valeria Herrera Salazar
Cargo del entrevistado	Docente de la Carrera de Computación
Fecha de entrevista	05/07/2024
Hora de entrevista	8 h 24 am
Canal de Comunicación	Entrevista presencial

- **Transcripción de la entrevista**

1. **¿Reconoce alguna de las siguientes métricas para la medición de calidad respecto a la mantenibilidad?**

Mala practica
Redundante
Complejidad cognitiva
Obsoleto
Confuso
Diseño
Dificultad encontrada
Densidad de comentarios
Duplicaciones
Cobertura

Si son reconocidas, en mi criterio si se mejora estas métricas favorecería no solo la mantenibilidad sino también disponibilidad, accesibilidad y adaptabilidad. Sin embargo, tendrías que tener un criterio como medirlas.

2. **¿Cree que sería útil que el software presente escalas de valoración de referencia para las métricas?**

Si es necesario, pero la misma tiene que estar bajo un criterio científico de autores que convaliden la escala de valoración.

3. **¿Cree que sería útil que el software presente escalas aceptación de referencia para esas métricas?**

Si lo considero, en este caso me va permitir identificar defectos y validar lo que espero de mi código.

4. **¿Cómo considera que debería estructurarse la lista de verificación para las pruebas manuales de mantenibilidad (métricas o subcaracterísticas etc.)?**

   En mi criterio debería estructurarse conforme las métricas, para poder reconocer todas las métricas con una pequeña descripción relacionada a la misma.

5. **¿Cómo le gustaría que el software permita registrar y documentar los resultados de las pruebas manuales de mantenibilidad (formulario digital, una plantilla editable, o alguna otra forma de captura de información)?**

   En mi parecer es mejor digital, utilizando una lista de verificación mediante checklist para evaluar y seleccionar el cumplimiento de cada métrica.

6. **¿Qué tipo de información le gustaría que se registre (comentarios, calificación)?**

   Es necesario un comentario e internamente agregar un valor para mostrar el grado de calidad del producto

7. **¿Considera importante que la lista incluya instrucciones o guías de uso?**

   Es conveniente una pequeña descripción a cerca de la métrica que se está aplicando

8. **¿Cómo le gustaría que el software presente los resultados de la verificación de mantenibilidad?**

   Considero que debería arrojar un mensaje por cada métrica para verificar si se está cumpliendo lo que espero.

9. **¿Preferiría visualizaciones gráficas y sugerencias?**

   Es necesario y seria atractivo y para mayor comprensión.

10. **¿Qué otro tipo de información le gustaría que se incluya en los resultados?**

    Se podría agregar sugerencias de acuerdo a las métricas

11. **¿Considera importante que el software genere un historial de evaluaciones de mantenibilidad a lo largo del tiempo?**

    Si es importante, esto para detallar como la calidad del software va mejorando en un estilo como una auditoria.

12. **¿Qué otras funcionalidades o características le gustarían que tuviera el software para la mejora de la mantenibilidad del código?**
    Detallar las sugerencias de cada métrica respecto al test de calidad.

- **Validez de la entrevista**

Yo, **Gilson Orlando Quezada**, estudiante de la Universidad Nacional de Loja en calidad de tesista, declaro en forma libre y voluntaria que la presente entrevista ha sido realizada al Ing. Valeria Herrera Salazar, como docente de la carrera de Computación de la Universidad Nacional de Loja.

En el siguiente enlace se puede tomar como evidencia y valides de la aplicación de la entrevista mediante un audio recolectado: https://drive.google.com/drive/folders/1iDDOvCFIljTRdKK9cYv3QTxear7vPI0o?usp=sharing.

- **Perfil “Docente de la carrera de computación”**

La siguiente entrevista se aplicó a un docente relacionado con el área de desarrollo de software de la carrera de computación como parte interesada, es decir rol usuario.

- **Planificación de la entrevista**

Entrevistador	Gilson Orlando Quezada Guartizaca
Entrevistada	Ing. Oscar Cumbicus Pineda
Cargo del entrevistado	Docente de la Carrera de Computación
Fecha de entrevista	05/07/2024
Hora de entrevista	8 h 24 am
Canal de Comunicación	Entrevista presencial

- **Transcripción de la entrevista**

1. **¿Reconoce alguna de las siguientes métricas para la medición de calidad respecto a la mantenibilidad?**

Mala practica
Redundante
Complejidad cognitiva
Obsoleto
Confuso
Diseño
Dificultad encontrada
Densidad de comentarios
Duplicaciones
Cobertura

Si identifico algunas, además se podría considerar la complejidad computacional la que me permite determinar cual complejo es el código.

2. **¿Cree que sería útil que el software presente escalas de valoración de referencia para las métricas?**

Si importante, la cual nos permitirá determinar cuál mantenibles es el software.

3. **¿Cree que sería útil que el software presente escalas aceptación de referencia para esas métricas?**

Si es una buena alternativa para identificar defectos, este aspecto favorece para dar una conclusión si el software contribuye lo planificado.

4. **¿Cómo considera que debería estructurarse la lista de verificación para las pruebas manuales de mantenibilidad (métricas o subcaracterísticas etc.)?**

   Es necesario estructurar por subcaracterísticas y la relación de cada métrica.

5. **¿Cómo le gustaría que el software permita registrar y documentar los resultados de las pruebas manuales de mantenibilidad (formulario digital, una plantilla editable, o alguna otra forma de captura de información)?**

   En mi parecer es mejor digital.

6. **¿Qué tipo de información le gustaría que se registre (comentarios, calificación)?**

   Es importante una descripción acerca de la métrica por ejemplo que mide y cómo influye en el software.

7. **¿Considera importante que la lista incluya instrucciones o guías de uso?**

   Es conveniente una pequeña descripción a cerca de la métrica que se esta aplicando

8. **¿Cómo le gustaría que el software presente los resultados de la verificación de mantenibilidad?**

   En un formato pdf, además de una descripción de las métricas y cuáles son sus defectos.

9. **¿Preferiría visualizaciones gráficas y sugerencias?**

   Lo prefiero de forma visual, además de una escala que muestre que tan mantenibles es mi código.

10. **¿Qué otro tipo de información le gustaría que se incluya en los resultados?**

    Se podría agregar sugerencias de acuerdo a los defectos de cada métrica.

11. **¿Considera importante que el software genere un historial de evaluaciones de mantenibilidad a lo largo del tiempo?**

    Si es importante, la cual permitirá tener un registro para validar la calidad el software.

12. **¿Qué otras funcionalidades o características le gustarían que tuviera el software para la mejora de la mantenibilidad del código?**

    Que él informa permite ser validad mediante una firma electrónica o un código QR respecto a un docente que aplique la evaluación.

- **Validez de la entrevista**

Yo, **Gilson Orlando Quezada**, estudiante de la Universidad Nacional de Loja en calidad de tesista, declaro en forma libre y voluntaria que la presente entrevista ha sido realizada al Ing. Oscar Cumbicus Pineda, como docente de la carrera de Computación de la Universidad Nacional de Loja.

En el siguiente enlace se puede tomar como evidencia y valides de la aplicación de la entrevista mediante un audio recolectado: https://drive.google.com/drive/folders/1jg7RYwNv9YzVDbKOA-oOjz8jexbvnyi9?usp=sharing.

- **Perfil “Docente de la carrera de computación”**

La siguiente entrevista se aplicó a un docente relacionado con el área de desarrollo de software de la carrera de computación como parte interesada, es decir rol usuario.

- **Planificación de la entrevista**

Entrevistador	Gilson Orlando Quezada Guartizaca
Entrevistada	Ing. Pablo Fernando Ordoñez
Cargo del entrevistado	Docente de la Carrera de Computación
Fecha de entrevista	05/07/2024
Hora de entrevista	8 h 24 am
Canal de Comunicación	Entrevista presencial

- **Transcripción de la entrevista**

1. **¿Reconoce alguna de las siguientes métricas para la medición de calidad respecto a la mantenibilidad?**

Mala practica
Redundante
Complejidad cognitiva
Obsoleto
Confuso
Diseño
Dificultad encontrada
Densidad de comentarios
Duplicaciones
Cobertura

Si identifico la mayoría y están relacionada específicamente en la aplicación de buenas prácticas de programación.

2. **¿Cree que sería útil que el software presente escalas de valoración de referencia para las métricas?**

Sí, es importante, ya que nos permitirá determinar el grado de mantenibilidad del software.

3. **¿Cree que sería útil que el software presente escalas aceptación de referencia para esas métricas?**

Sí, sería una buena alternativa para identificar defectos, ya que esto facilitaría la evaluación y permitiría concluir si el software cumple con lo planificado.

4. **¿Cómo considera que debería estructurarse la lista de verificación para las pruebas manuales de mantenibilidad (métricas o subcaracterísticas etc.)?**

   La lista de verificación debe estructurarse por subcaracterísticas, vinculando cada una con sus métricas correspondientes.

5. **¿Cómo le gustaría que el software permita registrar y documentar los resultados de las pruebas manuales de mantenibilidad (formulario digital, una plantilla editable, o alguna otra forma de captura de información)?**

   Prefiero que sea de forma digital.

6. **¿Qué tipo de información le gustaría que se registre (comentarios, calificación)?**

   Considera importante registrar comentarios y calificaciones. Además, incluya una breve descripción de cada métrica que indique qué mide y cómo influye en el software.

7. **¿Considera importante que la lista incluya instrucciones o guías de uso?**

   Sí, es útil proporcionar una breve descripción de la métrica que se está aplicando.

8. **¿Cómo le gustaría que el software presente los resultados de la verificación de mantenibilidad?**

   En un formato PDF que incluya una descripción detallada de las métricas utilizadas y un análisis de los defectos encontrados.

9. **¿Preferiría visualizaciones gráficas y sugerencias?**

   Sí, prefiero visualizaciones gráficas acompañadas de una escala que indique el nivel de mantenibilidad de mi código.

10. **¿Qué otro tipo de información le gustaría que se incluya en los resultados?**

    Podrían incluirse sugerencias específicas basadas en los defectos identificados por cada métrica evaluada.

    .

11. **¿Considera importante que el software genere un historial de evaluaciones de mantenibilidad a lo largo del tiempo?**

    Sí, es crucial para mantener un registro que valide la calidad del software a lo largo del tiempo.

12. **¿Qué otras funcionalidades o características le gustarían que tuviera el software para la mejora de la mantenibilidad del código?**

    Se podría considerar la implementación de una funcionalidad que permita validar el informe mediante una firma electrónica, facilitando la validación por parte de los docentes encargados de la evaluación.

- **Validez de la entrevista**

Yo, **Gilson Orlando Quezada**, estudiante de la Universidad Nacional de Loja en calidad de tesista, declaro en forma libre y voluntaria que la presente entrevista ha sido realizada al Ing. Pablo Fernando Ordoñez, como docente de la carrera de Computación de la Universidad Nacional de Loja.

En el siguiente enlace se puede tomar como evidencia y valides de la aplicación de la entrevista mediante un audio recolectado: https://drive.google.com/drive/folders/1hbkohOzS9zv_6M5Jf1ea0vtWSmndpGEV?usp=sharing.

.

Anexo 16. Transcripción de encuesta

La siguiente encuesta se aplicó a los estudiantes del itinerario de ingeniería de software de la carrera de computación, es decir rol de usuario.

- **Planificación de la entrevista**

Entrevistador	Gilson Orlando Quezada Guartizaca
Encuestados	Estudiantes del itinerario de ingeniería de software
Cargo del entrevistado	Gestor de la Carrera de Computación
Fecha inicio de entrevista	03/07/2024
Fecha fin de entrevista	06/07/2024
Canal de Comunicación	Encuesta en online

- **Transcripción de la entrevista**

1. **¿Considera útil aplicar pruebas manuales que incluya una lista de verificación para evaluar el cumplimiento de las pautas de mantenibilidad del software?**

La encuesta reveló que la mayoría de los encuestados con un 93,5% considera útil aplicar pruebas manuales que incluyan una lista de verificación para evaluar el cumplimiento de las pautas de mantenibilidad del software y solo el 6,5% de los encuestados no considera útil esta práctica.

2. **¿Le gustaría que las pruebas manuales incluyan la revisión de buenas prácticas de programación del código?**

En base a la encuesta realizada reveló la preferencia por la inclusión de la revisión de buenas prácticas de programación del código en las pruebas manuales y con un 96.8% de los encuestados a favor y solo un 3.2% en contra.

3. **¿Es importante para usted que las pruebas manuales de calidad relacionadas con el código cubran las subcaracterísticas de la mantenibilidad de la norma ISO/IEC 25000?**

El alto porcentaje de respuestas afirmativas (96.8%) indica una clara preferencia y necesidad entre los encuestados para que las pruebas manuales de calidad del código abarquen las subcaracterísticas de mantenibilidad de la norma ISO/IEC 25000 y la baja proporción de respuestas negativas (3.2%) sugiere que la mayoría de los usuarios reconocen los beneficios de aplicar estos criterios.

4. **Seleccione cuáles subcaracterísticas consideraría necesarias para verificar mantenibilidad basado en la norma ISO/IEC 25000.**

A partir de la encuesta realizada, se determinó que la **modularidad** (77.4%) y la **capacidad de ser probado** (64.5%) son las subcaracterísticas más valoradas para la mantenibilidad del software. La **analizabilidad** (45.2%) también es significativa, permitiendo una mejor comprensión y revisión del código. **Reusabilidad** y **capacidad de ser modificado** (ambas con 41.9%) son importantes, pero menos prioritarias.

5. **¿Le gustaría que las pruebas manuales generen un informe detallado con recomendaciones de mejora?**

De los encuestados, el 93.5% expresó interés en esta funcionalidad, mientras que el 6.5% no lo consideró relevante, esta información subraya la alta demanda por herramientas que faciliten análisis detallados y sugerencias de mejora basadas en pruebas manuales.

6. **¿Considera útil el uso de APIs de análisis de código estático para la evaluación de mantenibilidad?**

Según los datos recolectados, el 96.8% de los encuestados opinan que sí es útil y solo el 3.2% de los encuestados opinan que no es útil. Esto sugiere una fuerte preferencia por utilizar APIs de análisis de código estático como herramienta para evaluar la mantenibilidad del código.

7. **Actualmente, la utilización de analizadores de código estático mediante servicios (APIs) son críticos para la evaluación de la mantenibilidad en el desarrollo de software.**
   ¿Considera importante la utilización de APIs de análisis de código estático para calcular el grado de mantenibilidad basado en la norma ISO/IEC 25000?

El 96.8% de los encuestados considera importante la utilización de APIs de análisis de código estático para calcular el grado de mantenibilidad basado en la norma ISO/IEC 25000, mientras que solo el 3.2% no lo considera importante, lo que refleja una fuerte preferencia por implementar funcionalidades que integren estas APIs para evaluar y mejorar la mantenibilidad.

8. **¿Qué atributos considera necesarios para representar el grado de mantenibilidad?**

Para representar el grado de mantenibilidad, es esencial considerar tres atributos clave según la encuesta realizada: la visualización gráfica de métricas (74.2%), recomendaciones para mejorar la mantenibilidad basadas en métricas específicas (61.3%), y un informe detallado de la evaluación aplicada (67.7%). Estos resultados subrayan la necesidad de herramientas que permitan visualizar datos de manera efectiva, así como de orientación práctica para abordar áreas de mejora identificadas, junto con informes que documenten los resultados de las evaluaciones de mantenibilidad.

9. **¿Le gustaría recibir los informes de mantenibilidad en un formato PDF?**

Según la encuesta la encuesta muestra preferencia de recibir informes de mantenibilidad en formato PDF, el 100% indicó preferir esta opción, mientras que ningún encuestado optó por otras alternativas.

10. **¿Es importante que el informe detalle las métricas utilizadas para la evaluación?**

Se muestra que el 100% de los encuestados considera importante que el informe detalle las métricas utilizadas para la evaluación. Esto indica la necesidad de claridad y transparencia en la presentación de las métricas empleadas para evaluar los resultados.

11. **¿Le gustaría que el informe proporcione recomendaciones específicas para mejorar la mantenibilidad?**

De los encuestados, la totalidad (100%) manifestó que considera importante que el informe incluya recomendaciones específicas para mejorar la mantenibilidad.

12. **¿Es importante que el software incluya un historial de evaluaciones para seguimiento a lo largo del tiempo?**

De los encuestados, en un 96.8% considera importante esta funcionalidad, mientras que solo un 3.2% no lo considera necesario.

- **Validez de la entrevista**

Yo, **Gilson Orlando Quezada**, estudiante de la Universidad Nacional de Loja en calidad de tesista, declaro en forma libre y voluntaria que la presente entrevista ha sido realizada a los estudiantes del itinerario de ingeniería de software de la carrera de Computación de la Universidad Nacional de Loja.

En el siguiente enlace se puede tomar como evidencia y valides de la aplicación de la entrevista mediante un documento xls: https://docs.google.com/spreadsheets/d/1JdPKCS1JGobdV2O2t1Mez64u3ue8kfZPjtmEOq3ezpI/edit?usp=sharing, además del formulario digital de la aplicación de la encuesta: https://forms.gle/9UHMT2sFJtrDRhFp9.

Anexo 17. Historias de usuario

Ingeniería en Ciencias de la Computación

Especificación de historias de usuario

Proyecto: Evaluación de mantenibilidad para aplicaciones web en producción de los laboratorios de la carrera de computación de la UNL

Versión: 1.1

Elaborado por:

Gilson Orlando Quezada

Revisado y Aprobado por:

Ing. Francisco J Álvarez-Pineda Mg. Sc

Contenido

Historial de versiones

Versión	Fecha	Revisión	Historial de cambios	Autor	Verificación
1.0	12/07/2024	Presencial	Primera revisión	Gilson Orlando Quezada	Ing. Francisco J Álvarez Pineda Mg. Sc
1.1	12/08/2024	Presencial	Segunda revisión	Gilson Orlando Quezada	Ing. Francisco J Álvarez Pineda Mg. Sc

Legalización del documento

Por medio de la presente, se confirma que la documentación de historias de usuario para el Trabajo de Integración Curricular, titulado "*Evaluación de mantenibilidad para aplicaciones web en producción de los laboratorios de la carrera de computación de la UNL*", ha sido revisada y aprobada por el director del proyecto, el Ing. Francisco J Álvarez Pineda Mg. Sc, docente de la carrera de Computación de la Universidad Nacional de Loja.

Se adjunta las firmas de los participantes.

F: _____

Ing. Francisco J Álvarez Pineda Mg. Sc

Director del Proyecto

F: _____

Gilson Orlando Quezada

Autor

1. Introducción

El presente documento es una especificación de las historias de usuario para el desarrollo del del software para verificar la mantenibilidad de las aplicaciones web. Esta especificación se ha estructurado basándose en los esquemas propuestos para el desarrollo de historias de usuario. La presente sección muestra la descripción del propósito, el personal involucrado en la realización del presente documento y las definiciones y acrónimos.

1.1. Propósito

El presente documento muestra la especificación de las historias de usuario con el propósito determinar el flujo de las funcionalidades esenciales del prototipo para dar paso a la determinación de los requisitos del mismo. La documentación fue revisada por la parte interesada: Ing. Juan Carlos Solano Jiménez y el director del trabajo de integración curricular: Ing. José Oswaldo Guamán Quinche, Mg. para determinar si los requerimientos han sido recopilados de manera efectiva.

1.2. Personal involucrado

Nombre	Gilson Orlando Quezada
Rol	Analista y desarrollador.
Categoría profesional	Estudiante de la carrera de Computación.
Responsabilidades	Análisis, diseño y construcción de los requisitos del aplicativo.
Información de contacto	gilson.quezada@unl.edu.ec
Aprobación	Aprobado

Nombre	Francisco Javier Álvarez Pineda
Rol	- Director de Trabajo de Integración Curricular. - Analista de requerimientos.
Categoría profesional	- Docente de la carrera de Ingeniería en Computación.
Responsabilidades	Asesorar el desarrollo del Trabajo de Integración Curricular.
Información de contacto	fjalvarez@unl.edu.ec
Aprobación	Aprobado

1.3. Definiciones, acrónimos y abreviaturas

Nombre	Descripción
Usuario	Persona que hará uso del sistema.
HU	Historia de usuario
UNL	Universidad Nacional de Loja.

2. Descripción general

1.1. Perspectiva del producto

El software está enfocado en la web; por lo tanto, el diseño e implementación de las funcionalidades se basan en un enfoque netamente de navegación. Esto facilitará que sea utilizado de manera sencilla únicamente con el acceso a internet desde cualquier zona y hogar de la ciudad de Loja.

1.2. Actores identificados

Actor	Descripción
Usuario	Es aquel actor que hace uso del Sistema y necesita estar registrado.
Administrador	Es aquel usuario que necesita tener su cuenta y que va hacer la administración.

3. Especificación de las historias de usuario

3.1. Historias de usuario

La siguiente sección proporciona la descripción de las historias de usuario:

Tabla_1_Anexo_17. Modelo de historia de usuario - Inicio de sesión.

<table>
<tr><th colspan="5">Historia de usuario</th></tr>
<tr><td>ID: HU01</td><td colspan="4">Inicio de sesión</td></tr>
<tr><td>Usuario/Rol</td><td colspan="4">Administrador</td></tr>
<tr><td colspan="2">Prioridad en negocio</td><td>Alta</td><td>Riesgo en desarrollo</td><td>Alta</td></tr>
<tr><td>Descripción</td><td colspan="4">Como administrador, deseo iniciar sesión utilizando el servicio de la carrera de autenticación de Aerobase IAM, de la misma forma para el perfil usuario.</td></tr>
<tr><td>Nro.</td><td>Escenario</td><td colspan="3">Criterio de aceptación</td></tr>
<tr><td>1</td><td>Acceder al software con las credenciales de usuario</td><td colspan="3">● El servicio debe mostrar un formulario para ingresar los datos de inicio de sesión: correo y contraseña.
● El servicio permite dar paso al inicio de sesión en el software una vez se hayan ingresado las credenciales.
● Si las credenciales están vacías o no existen, se mostrará una alerta o mensaje de error.
● El servicio realizara la verificación de las credenciales como estudiantes de la carrera de computación para determinar el acceso al sistema.</td></tr>
<tr><td>2</td><td>Redirección según el rol del administrador y usuario</td><td colspan="3">● El sistema para el perfil administrador lo redirige a la página de administración de pautas, lista de verificación, métricas y subcaracterísticas.
● El sistema redirige al usuario a la página principal para verificar la mantenibilidad.</td></tr>
</table>

Tabla_2_Anexo_17. Modelo de historia de usuario – Administrar pauta.

Historia de usuario			
ID: HU02	Administrar pauta		
Usuario/Rol	Administrador		
Prioridad en negocio	Media	Riesgo en desarrollo	Media
Descripción	Como administrador, quiero agregar, modificar y eliminar pautas de mantenibilidad en el software, pertenecientes a las listas de verificación.		
Nro.	Escenario	Criterio de aceptación	
1	Registrar nueva pauta de mantenibilidad	● Como administrador, accedo a la página de administración de pautas de mantenibilidad. ● El sistema muestra la opción para agregar una nueva pauta de mantenibilidad. ● Ingreso el texto de la nueva pauta de mantenibilidad. ● Si existen parámetros por completar se debe mostrar la alerta respectiva. ● Guardo la nueva pauta de mantenibilidad. ● Se muestra un mensaje temporal indicando que la pauta de mantenibilidad se ha creado exitosamente. ● La nueva pauta de mantenibilidad se agrega a la lista de pauta de mantenibilidad disponibles.	
2	Modificar pauta de mantenibilidad existente	● Como administrador, accedo a la página de administración de pautas de mantenibilidad. ● Selecciono la pauta de mantenibilidad que quiero actualizar/modificar. ● Modifico el texto de la pauta de mantenibilidad o su opción de respuesta. ● Guardo los cambios. ● Se muestra un mensaje temporal indicando que la pauta de mantenibilidad se ha modificado exitosamente. ● La pauta de mantenibilidad modificada se actualiza en la lista de preguntas.	
3	Eliminar pauta de mantenibilidad existente	● Como administrador, accedo a la página de pautas de mantenibilidad. ● Selecciono la pauta de mantenibilidad que quiero eliminar. ● El sistema muestra el modal de confirmación de eliminación. ● Confirmo la eliminación. ● Se muestra un mensaje temporal indicando que la pauta de mantenibilidad se ha eliminado exitosamente. ● La pauta de mantenibilidad se elimina de la lista de pautas de mantenibilidad disponibles.	

Tabla_3_Anexo_17. Modelo de historia de usuario – Administrar lista de verificación.

Historia de usuario			
ID: HU03	Administrar lista de verificación		
Usuario/Rol	Administrador		
Prioridad en negocio	Media	Riesgo en desarrollo	Media
Descripción	Como administrador, quiero poder agregar, modificar y eliminar las listas de verificación en el sistema, pertenecientes a las metricas. También, cada lista de verificación tendrá preguntas únicas asociadas.		

Nro.	Escenario	Criterio de aceptación
1	Agregar nueva lista de verificación	● Como administrador, accedo a la página de administración de listas de verificación. ● El sistema muestra la opción para agregar una nueva lista de verificación. ● Ingreso el texto de la nueva lista de verificación y las pautas de mantenibilidad disponibles/no perteneciente a otra lista de verificación. ● Guardo la nueva lista de verificación. ● Se muestra un mensaje temporal indicando que la lista de verificación se ha creado exitosamente. ● La nueva lista de verificación se agrega a la lista de prácticas disponibles.
2	Modificar lista de verificación existente	● Como administrador, accedo a la página de administración de listas de verificación. ● Selecciono la lista de verificación que quiero actualizar/modificar. ● Modifico el texto de la lista de verificación o sus pautas de mantenibilidad/ pautas de mantenibilidad no seleccionadas. ● Guardo los cambios. ● Se muestra un mensaje temporal indicando que la lista de verificación se ha modificado exitosamente. ● La lista de verificación se ha modificada se actualiza del registro de lista de verificación.
3	Eliminar lista de verificación existente	● Como administrador, accedo a la página de administración de listas de verificación. ● Selecciono la lista de verificación que quiero eliminar. ● El sistema muestra el modal de confirmación de eliminación. ● Confirmo la eliminación. ● Se muestra un mensaje temporal indicando que la lista de verificación se ha eliminado exitosamente. ● La lista de verificación se elimina del registro de lista de verificación disponibles.

Tabla_4_Anexo_17. Modelo de historia de usuario – Administrar métrica.

Historia de usuario			
ID: HU04	Administrar métrica		
Usuario/Rol	Administrador		
Prioridad en negocio	Media	Riesgo en desarrollo	Media
Descripción	Como administrador, quiero poder agregar, modificar y eliminar las métricas en el sistema, pertenecientes a las subcaracterísticas. También, cada métrica tendrá una lista de verificación única asociada.		

Nro.	Escenario	Criterio de aceptación
1	Agregar nueva métrica	● Como administrador, accedo a la página de administración de métricas. ● El sistema muestra la opción para agregar una nueva métrica. ● Ingreso el texto de la nueva métrica y la lista de verificación disponible/no perteneciente a otra lista de verificación. ● Guardo la nueva métrica. ● Se muestra un mensaje temporal indicando que la métrica se ha creado exitosamente. ● La nueva métrica se agrega a la lista de métricas disponibles.
2	Modificar métrica existente	● Como administrador, accedo a la página de administración de métricas. ● Selecciono la métrica que quiero actualizar/modificar. ● Modifico el texto de la métrica o su lista de verificación/ lista de verificación no seleccionadas. ● Guardo los cambios. ● Se muestra un mensaje temporal indicando que la métrica se ha modificado exitosamente. ● La métrica modificada se actualiza en la lista de métricas.
3	Eliminar métrica existente	● Como administrador, accedo a la página de administración de métricas. ● Selecciono la métrica que quiero eliminar. ● El sistema muestra el modal de confirmación de eliminación. ● Confirmo la eliminación. ● Se muestra un mensaje temporal indicando que la métrica se ha eliminado exitosamente. ● La métrica se elimina de la lista de subcaracterística disponibles.

Tabla_5_Anexo_17. Modelo de historia de usuario – Administrar subcaracterística.

Historia de usuario			
ID: HU05	Administrar subcaracterística		
Usuario/Rol	Administrador		
Prioridad en negocio	Media	Riesgo en desarrollo	Media
Descripción	Como administrador, quiero poder agregar, modificar y eliminar las subcaracterística en el sistema, pertenecientes a una prueba de mantenibilidad. También, cada subcaracterística tendrá varias métricas asociadas.		

Nro.	Escenario	Criterio de aceptación
1	Agregar nueva subcaracterística	● Como administrador, accedo a la página de administración de subcaracterística. ● El sistema muestra la opción para agregar una nueva subcaracterística. ● Ingreso el texto de la nueva subcaracterística y las métricas disponibles. ● Guardo la nueva subcaracterística. ● Se muestra un mensaje temporal indicando que la subcaracterística se ha creado exitosamente. ● La nueva subcaracterística se agrega a la lista de subcaracterística disponibles.
2	Modificar subcaracterística existente	● Como administrador, accedo a la página de administración de subcaracterísticas. ● Selecciono la subcaracterística que quiero actualizar/modificar. ● Modifico el texto de la subcaracterística o sus métricas. ● Guardo los cambios. ● Se muestra un mensaje temporal indicando que la subcaracterística se ha modificado exitosamente. ● La subcaracterística modificada se actualiza en la lista de subcaracterística.
3	Eliminar subcaracterística existente	● Como administrador, accedo a la página de administración de subcaracterísticas. ● Selecciono la subcaracterística que quiero eliminar. ● El sistema muestra el modal de confirmación de eliminación. ● Confirmo la eliminación. ● Se muestra un mensaje temporal indicando que la subcaracterística se ha eliminado exitosamente. ● La subcaracterística se elimina de la lista de subcaracterística disponibles.

Tabla_6_Anexo_17. Modelo de historia de usuario – Administrar proyecto.

<table>
<tr><th colspan="5">Historia de usuario</th></tr>
<tr><td>ID: HU06</td><td colspan="4">Administrar proyecto</td></tr>
<tr><td>Usuario/Rol</td><td colspan="4">Administrador/usuario</td></tr>
<tr><td colspan="2">Prioridad en negocio</td><td>Media</td><td>Riesgo en desarrollo</td><td>Media</td></tr>
<tr><td>Descripción</td><td colspan="4">Como administrador o usuario quiero poder agregar, modificar y eliminar proyectos.</td></tr>
<tr><td>Nro.</td><td>Escenario</td><td colspan="3">Criterio de aceptación</td></tr>
<tr><td>1</td><td>Agregar nuevo proyecto</td><td colspan="3">● Como usuario/administrador, presiono en el botón de nuevo proyecto.
● Ingreso el nombre y descripción del proyecto.
● Y presiono el botón de guardar.
● Se muestra un mensaje temporal indicando que se creó el proyecto exitosamente.
● El sistema redirecciona a mis proyectos.
● El nuevo proyecto se agrega a la lista de mis proyectos.</td></tr>
<tr><td>2</td><td>Modificar proyecto existente</td><td colspan="3">● Como usuario/administrador, accedo a la página de mis proyectos.
● Selecciono el proyecto que quiero actualizar/modificar.
● Modifico los datos que deseo.
● Guardo los cambios.
● Se muestra un mensaje temporal indicando que el proyecto se ha modificado exitosamente.
● El proyecto se actualiza en la lista de mis proyectos.</td></tr>
<tr><td>3</td><td>Eliminar proyecto existente</td><td colspan="3">● Como usuario/administrador, accedo a la página de mis proyectos.
● Selecciono el proyecto que quiero eliminar.
● El sistema muestra el modal de confirmación de eliminación.
● Confirmo la eliminación.
● Se muestra un mensaje temporal indicando que el proyecto se ha eliminado exitosamente.
● El proyecto se elimina de mis proyectos.</td></tr>
<tr><td>4</td><td>Lista de proyectos creados</td><td colspan="3">● Como usuario/administrador, accedo a la página de mis proyectos.
● Visualizo la lista completa de mis proyectos creados en el sistema con las opciones de ver, editar y eliminar.</td></tr>
</table>

Tabla_7_Anexo_17. Modelo de historia de usuario – Administrar prueba de mantenibilidad.

<table>
<tr><th colspan="5">Historia de usuario</th></tr>
<tr><td>ID: HU07</td><td colspan="4">Administrar prueba de mantenibilidad</td></tr>
<tr><td>Usuario/Rol</td><td colspan="4">Administrador/Usuario</td></tr>
<tr><td colspan="2">Prioridad en negocio</td><td>Media</td><td>Riesgo en desarrollo</td><td>Media</td></tr>
<tr><td>Descripción</td><td colspan="4">Como administrador o usuario, quiero poder configurar el plan de prueba para verificar la mantenibilidad de aplicaciones web mediante el cumplimiento de pautas de mantenibilidad.</td></tr>
<tr><td>Nro.</td><td>Escenario</td><td colspan="3">Criterio de aceptación</td></tr>
<tr><td>1</td><td>Acceder a administración de pruebas de mantenibilidad</td><td colspan="3">● Como administrador/usuario, accedo a la lista de proyectos.
● El sistema muestra la opción para direccionar a prueba de mantenibilidad.
● Selecciono pruebas y el sistema direcciona la página de administración de pruebas de mantenibilidad.</td></tr>
<tr><td></td><td>Configurar estructura de la lista de verificación para implementar prueba de mantenibilidad</td><td colspan="3">● El software muestra en una pestaña configuración lista de verificación.
- Fase de planificación.
- Fase de ejecución
● En la fase de planificación de muestra cuatro opciones para estructurar la prueba de mantenibilidad.
- Subcaracterísticas.
- Subcaracterística especifica
- Métricas
- Métrica especifica.
● En la fase de ejecución muestra dos opciones para ejecutar la lista de verificación.
- Resolver prueba
- Ver pruebas.
● Selecciono la estructura de la prueba.</td></tr>
<tr><td>2</td><td>Asignar un parámetro de evaluación a una prueba de mantenibilidad existente</td><td colspan="3">● Como administrador, accedo a la página de administración de pruebas de mantenibilidad.
● El sistema muestra la opción para direccionar a ver respuestas de las pruebas de mantenibilidad.
● Selecciono ver respuestas y el sistema direcciona la página de administración de resultados o respuestas de las pruebas de mantenibilidad.
● Selecciono la prueba que se desea asignar un parámetro de evaluación.
● Se muestra un mensaje temporal indicando que si desea agregar otra métrica subcaracterística.
● Seleccionamos la métrica o subcaracterística y el sistema estructura la prueba.</td></tr>
</table>

Tabla_8_Anexo_17. Modelo de historia de usuario – Ejecutar prueba de mantenibilidad.

Historia de usuario			
ID: HU08	Ejecutar prueba de mantenibilidad		
Usuario/Rol	Administrador/usuario,		
Prioridad en negocio	Media	Riesgo en desarrollo	Media
Descripción	Como administrador o usuario, quiero ejecutar la prueba de mantenibilidad en cada uno de los proyectos.		
Nro.	Escenario	Criterio de aceptación	
1	Acceder a administración de pruebas de mantenibilidad	● Como usuario/administrador, accedo a la página de administración de pruebas de mantenibilidad. ● Selecciono el proyecto que para evaluar. ● El software permite administrar la lista de verificación (ID: HU07). ● Selecciono ejecutar prueba de mantenibilidad.	
2	Responder lista de verificación de mantenibilidad	● Agrego un título a prueba aplicada. ● Respondo las pautas de mantenibilidad según las opciones proporcionadas. ● Opcional: Registrar comentarios por cada pauta de mantenibilidad ● Operacional del software: promedio del cumplimiento de las pautas de mantenibilidad y almacena los resultados. ● Selecciono guarda resultados de evaluación. ● Se muestra un mensaje temporal indicando que las respuestas se han guardado exitosamente.	

Tabla_9_Anexo_17. Modelo de historia de usuario – Informe lista de verificación.

Historia de usuario			
ID: HU09	Informe lista de verificación		
Usuario/Rol	Administrador/usuario		
Prioridad en negocio	Media	Riesgo en desarrollo	Media
Descripción	Como Administrador o usuario, quiero ver el informe del cumplimiento de listas de verificación de cada proyecto.		
Nro.	Escenario	Criterio de aceptación	
1	Acceder a administración de proyectos	● Como usuario/administrador, accedo a la página administración de proyectos. ● Como usuario/administrador selecciono ver respuestas relacionada al proyecto seleccionado. ● Selecciono ver cálculos.	
2	Ver informe de cumplimiento de listas de verificación.	● El sistema redirecciona a la página de visualización del cumplimiento de lista de verificación. ● El sistema muestra anotaciones relacionada a cada pauta de mantenibilidad y el promedio de cumplimiento de pautas de mantenibilidad, además de un gráfico de barras.	

Tabla_10_Anexo_17. Modelo de historia de usuario – Generar PDF.

Historia de usuario			
ID: HU010	Generar PDF		
Usuario/Rol	Administrador/usuario		
Prioridad en negocio	Media	Riesgo en desarrollo	Media
Descripción	Como Administrador o usuario, quiero descargar el pdf generado por el software de la página de administración de pruebas de mantenibilidad		
Nro.	Escenario	Criterio de aceptación	
1	Acceder a administración de pruebas de mantenibilidad	● Como usuario/administrador, accedo a la página administración de pruebas de mantenibilidad. ● En la fase de ejecución selecciono ver respuestas relacionada a un proyecto. ● Selecciono ver cálculos.	
2	Seleccionar descargar PDF	● Como usuario/administrador, presiono la opción generar pdf para descargar el archivo. ● El sistema descarga el pdf correspondiente con el fin de que el usuario pueda realizar un análisis detallado del proyecto. ● Operacional del software**:** en el informe PDF se agregará cada uno de los comentarios relacionado a cada pauta de mantenibilidad.	

Tabla_11_Anexo_17. Modelo de historia de usuario – Calcular grado de mantenibilidad.

Historia de usuario			
ID: HU011	Calcular grado de mantenibilidad		
Usuario/Rol	Administrador/usuario		
Prioridad en negocio	Media	Riesgo en desarrollo	Media
Descripción	Como administrador o usuario deseo obtener las métricas de calidad de la herramienta SonarQube, mediante el análisis de código estático de mi página web para calcular el grado de mantenibilidad del código.		

Nro.	Escenario	Criterio de aceptación
1	Obtener propiedades de calidad de la herramienta de Sonarqube	● El software debe permitir obtener las propiedades de calidad mediante el servicio de APIs de la herramienta de Sonarqube. ● Se mostrará el botón “Guardar” para registrar la información de Sonarqube en la base de datos.
2	Calcular el valor de las métricas	● El software mediante fórmulas matemáticas realizará el cálculo del valor de las métricas utilizando propiedades como: Debt Ratio, duplicated lines, coverage lines, cyclomatic complexity, deuda técnica (Debt), code smells, cognitive complexity y comment lines. ● El sistema almacenara el valor de las métricas en la base de datos.
3	Calcular el valor de subcaracterísticas	● El sistema mediante fórmulas matemáticas realizará el cálculo del valor de las subcaracterísticas utilizando propiedades como: Mala práctica, redundante, complejidad cognitiva, obsoleto, confuso diseño, dificultad encontrada, densidad de comentarios, duplicaciones y cobertura. ● El sistema almacenara el valor de las subcaracterísticas en la base de datos.
4	Calcular el grado de mantenibilidad	● El sistema mediante fórmulas matemáticas realizará el cálculo el grado de mantenibilidad utilizando el valor de las subcaracterísticas como: Modularidad, reusabilidad, anulabilidad, capacidad de ser probado y capacidad de ser modificado. ● El sistema almacenara el valor del grado de mantenibilidad en la base de datos.
5	Mostrar grado de mantenibilidad	● El software mostrará mediante un análisis de métricas cuales son los defectos relacionados con la calidad interna del código y la norma ISO/IEC 25000

Anexo 18. Requisitos del software

Ingeniería en Ciencias de la Computación

Especificación de requisitos de software

Proyecto: Evaluación de mantenibilidad para aplicaciones web en producción de los laboratorios de la carrera de computación de la UNL

Versión: 1.1

Elaborado por:

Gilson Orlando Quezada

Revisado y Aprobado por:

Ing. Francisco J Álvarez-Pineda Mg. Sc

Contenido

Historial de versiones

Versión	Fecha	Revisión	Historial de cambios	Autor	Verificación
1.0	03/07/2024	17/07/2024	Primera revisión	Gilson Orlando Quezada Guartizaca	Ing. Francisco J Álvarez-Pineda Mg. Sc.
1.1	23/07/2024	17/09/2024	Segunda revisión	Gilson Orlando Quezada Guartizaca	Ing. Francisco J Álvarez-Pineda Mg. Sc.

Legalización del documento

Por medio de la presente, se confirma que la documentación de requisitos para el Trabajo de Integración Curricular, titulado "**Evaluación de mantenibilidad para aplicaciones web en producción de los laboratorios de la carrera de computación de la UNL**", ha sido revisada y aprobada por el director del proyecto, el Ing. Francisco J Álvarez-Pineda Mg.Sc.

Se adjunta las firmas de los participantes.

Firmado electrónicamente por:
FRANCISCO JAVIER ALVAREZ PINEDA

F: _____

Ing. Francisco J Álvarez-Pineda Mg. Sc

DIRECTO TRABAJO DE INTEGRACION CURRICULAR

Firmado electrónicamente por:
GILSON ORLANDO QUEZADA GUARTIZACA

F: _____

Gilson Orlando Quezada Guartizaca

AUTOR

1. Introducción

El presente documento es una especificación de requisitos de software (ERS) para el desarrollo de una aplicación web para verificar la mantenibilidad de aplicaciones web basado en la familia de la norma ISO/IEC 25000. Esta especificación se ha estructurado basándose en las directrices dadas por el estándar IEEE para las Especificaciones de Requisitos Software ANSI/IEEE 830, 1998 [1]. La presente sección muestra la descripción del propósito, alcance, personal involucrado, las definiciones y acrónimos, las referencias, así mismo un resumen general del aplicativo.

1.1. Propósito

El presente documento muestra la especificación de los requerimientos funcionales y no funcionales del sistema, el cual tiene como objetivo verificar la mantenibilidad de aplicaciones web basado en la familia de la norma ISO/IEC 25000. El presente documento tiene como propósito determinar las necesidades de las partes interesadas y generar la especificación de los requerimientos esenciales para el desarrollo de las actividades.

La documentación fue revisada por el director del trabajo de integración curricular: Ing. Francisco J Álvarez Pineda Mg.Sc. para determinar si los requerimientos han sido recopilados de manera efectiva.

1.2. Alcance

El desarrollo de una aplicación web para verificar la mantenibilidad de aplicaciones web en los laboratorios de la carrera de computación basado en la norma ISO/IEC 25000 tiene como objetivo implementar listas de verificación para evaluar el cumplimiento de las pautas de mantenibilidad del software y uso de herramientas de análisis de código estático, como APIs, para calcular el grado de mantenibilidad del código. Para este propósito se busca realizar un análisis y operaciones en base a métricas numéricas necesarias para calcular nivel de calidad.

1.3. Personal involucrado

Nombre	Gilson Orlando Quezada Guartizaca
Rol	Analista y desarrollador.
Categoría profesional	Estudiante de la carrera de Computación.
Responsabilidades	Análisis, diseño y construcción de los requisitos del aplicativo.
Información de contacto	gilson.quezada@unl.edu.ec
Aprobación	Aprobado

Nombre	Francisco J Álvarez Pineda
Rol	- Director de Trabajo de Integración Curricular. - Analista de requerimientos.
Categoría profesional	- Master. - Docente de la carrera de Ingeniería en Computación.
Responsabilidades	Asesorar el desarrollo del Trabajo de Integración Curricular.
Información de contacto	fjalvarez@unl.edu.ec
Aprobación	Aprobado

1.4. Definiciones, acrónimos y abreviaturas

Nombre	Descripción
Usuario	Persona que hará uso del sistema.
ERS	Especificación de requisitos software.
RF	Requisito funcional.
RNF	Requisito no funcional.
UNL	Universidad Nacional de Loja.

1.5. Referencias

Referencia	Título
Standard IEEE 830 - 1998 [1]	IEEE

2. Descripción general

2.1. Perspectiva del producto

La aplicación está enfocada en la web; por lo tanto, el diseño e implementación de las funcionalidades se basan en un enfoque netamente de navegación. Esto facilitará que sea utilizado de manera sencilla únicamente con el acceso a internet desde cualquier zona y hogar de la ciudad de Loja.

2.2. Actores identificados

Actor	Descripción
Usuario	Es aquel actor que hace uso del sistema y necesita tener una cuenta registrada.
Administrador	Es aquel usuario que necesita tener su cuenta y que va hacer la administración.

2.3. Restricciones

La aplicación consta de las siguientes restricciones:

- Para el desarrollo del software se utilizará base de datos y lenguajes de programación de código abierto entre MongoDB, Vite JS, Express JS y Node JS.
- Será desarrollado y podrá utilizar en navegador web, especialmente para ordenadores de mesa o portátiles.
- La aplicación tendrá una interfaz sencilla y fácil de usar para la comodidad del usuario general.
- Para acceder al aplicativo web los usuarios deberán contar con una contraseña y un rol para acceder al mismo

3. Requisitos específicos

3.1. Requisitos funcionales

La siguiente sección proporciona el resumen de los requisitos funcionales:

Tabla_1_Anexo_IV. Especificación de requisitos funcionales.

Requisitos funcionales	
Nombre de requisito	**Número de requisito**
Inicio de sesión	RF-01
Administrar pauta de mantenibilidad	RF-02
Administrar lista de verificación	RF-03
Administrar métrica	RF-04
Administrar subcaracterística	RF-05
Administrar proyecto	RF-06
Administrar prueba de mantenibilidad	RF-07
Ejecutar prueba de mantenibilidad	RF-18
Informe lista de verificación	RF-09
Generar PDF	RF-10
Calcular grado de mantenibilidad	RF-11

La siguiente sección proporciona la descripción de los requisitos funcionales.

Número de requisito	RF-01
Nombre de requisito	Inicio de sesión
Tipo	Requisito
Característica	Los usuarios deben autentificarse para acceder a cualquier proceso del sistema.
Descripción	**Inicio de sesión:** El sistema permitirá el ingreso de los usuarios, mediante los servicios de autenticación de Aerobase IAM de la carrera, utilizando correo electrónico y la contraseña, que son recolectados en el servicio.
Prioridad del requisito	Alta

Número de requisito	RF-02
Nombre de requisito	Administrar pauta de mantenibilidad
Tipo	Requisito
Característica	Los usuarios deben estar logueados en el sistema.
Descripción	**Registrar, editar y eliminar pauta de mantenibilidad:** El sistema permitirá registrar y editar información, además de eliminar las pautas de mantenibilidad.
Prioridad del requisito	Media

Número de requisito	RF-03
Nombre de requisito	Administrar lista de verificación
Tipo	Requisito
Característica	Los usuarios deben estar logueados en el sistema.
Descripción	**Registrar, editar y eliminar de lista de verificación:** El sistema permitirá registrar y modificar la información asociada, además de eliminar la lista de verificación, lo cual es necesario para aplicar pruebas de mantenibilidad.
Prioridad del requisito	Media

Número de requisito	RF-04
Nombre de requisito	Administrar métrica
Tipo	Requisito
Característica	Los usuarios deben estar logueados en el sistema.
Descripción	**Registrar, editar y eliminar métrica:** El sistema permitirá registrar y editar información relacionada a la misma, además de eliminar la métrica, lo cual es importante para estructurar la lista de verificación.
Prioridad del requisito	Media

Número de requisito	RF-05
Nombre de requisito	Administrar subcaracterística
Tipo	Requisito
Característica	Los usuarios deben estar logueados en el sistema.
Descripción	**Registrar, editar y eliminar las subcaracterística:** El sistema permitirá registrar y modificar información relacionada a la misma, además de eliminar la métrica, lo cual es importante para estructurar la lista de verificación.
Prioridad del requisito	Media

Número de requisito	RF-06
Nombre de requisito	Administrar proyecto
Tipo	Requisito
Característica	Los usuarios deben estar logueados en el sistema.
Descripción	**Registrar, editar y eliminar proyecto:** El sistema permitirá registrar y editar relacionada a la mismas, además de eliminar el proyecto relacionado a cada usuario.
Prioridad del requisito	Media

Número de requisito	RF-07
Nombre de requisito	Administrar prueba de mantenibilidad
Tipo	Requisito
Característica	Los usuarios deben estar logueados en el sistema.
Descripción	**Registrar y editar prueba de mantenibilidad:** El sistema permitirá registrar y editar las pruebas de mantenibilidad de cada una de los proyectos registrados. El registro y editar consiste en establecer la estructura de la prueba.
Prioridad del requisito	Media

Número de requisito	RF-08
Nombre de requisito	Ejecutar prueba de mantenibilidad
Tipo	Requisito
Característica	Los usuarios deben estar logueados en el sistema.
Descripción	**Ver lista de proyectos y ejecutar prueba de mantenibilidad:** El sistema permitirá visualizar la lista de proyectos, con el propósito de seleccionar un proyecto y responder las listas de verificación.
Prioridad del requisito	Media

Número de requisito	RF-09
Nombre de requisito	Informe lista de verificación
Tipo	Requisito
Característica	Los usuarios deben estar logueados en el sistema.
Descripción	**Ver lista de proyectos y ver informe de mantenibilidad:** El sistema permitirá visualizar la lista de proyectos, con el propósito de seleccionar un proyecto y visualizar los resultados de la aplicación de la lista de verificación.
Prioridad del requisito	Media

Número de requisito	RF-10
Nombre de requisito	Generar PDF
Tipo	Requisito
Característica	Los usuarios deben estar logueados en el sistema.
Descripción	**Generar y descargar PDF:** El sistema permitirá generar y descargar el informe del resultado de verificación de mantenibilidad de los proyectos registrados. Generado el PDF.
Prioridad del requisito	Media

Número de requisito	RF-11
Nombre de requisito	Calcular el grado de mantenibilidad
Tipo	Requisito
Característica	Los usuarios deben estar logueados en el sistema.
Descripción	**Calcular grado de mantenibilidad:** El sistema permitirá realizar el cálculo del grado de mantenibilidad utilizando las siguientes actividades. - Obtener propiedades de calidad de la herramienta de Sonarqube. - Calcular el valor de las métricas. - Calcular el valor de subcaracterísticas. - Calcular el grado de mantenibilidad. - Mostrar grado de mantenibilidad
Prioridad del requisito	Media

3.2. Requisitos no funcionales

3.2.1. Seguridad

- La aplicación web garantiza la confiabilidad, la seguridad y el desempeño a los usuarios mediante el servició de Aerobase IAM.
- El prototipo garantiza la seguridad en las operaciones mediante la implementación de controles y mensajes de confirmación para evitar el uso malintencionado de las operaciones.

3.2.2. Usabilidad

- El prototipo ofrece una interfaz gráfica sencilla y fácil de usar para los usuarios generales, por lo tanto, se desarrollará de acuerdo a las especificaciones proporcionadas por los beneficiarios del sistema.

3.2.3. Requisitos no funcionales específicos

Número de requisito	RNF-1
Nombre de requisito	Interfaz Gráfica
Tipo	Requisitos
Descripción	La interfaz gráfica es intuitiva y fácil de usar, siguiendo las convenciones de diseño estándar y proporcionando retroalimentación visual clara para todas las interacciones del usuario.
Prioridad del requisito	Alta

Número de requisito	RNF-2
Nombre de requisito	Accesibilidad
Tipo	Requisitos
Descripción	La aplicación permite a los usuarios acceder a las funcionalidades de manera efectiva.
Prioridad del requisito	Alta

Número de requisito	RNF-3
Nombre de requisito	Operacional
Tipo	Requisitos
Descripción	El prototipo cumple en su totalidad con las funciones planificadas.
Prioridad del requisito	Alta

Número de requisito	RNF-4
Nombre de requisito	Confidencialidad
Tipo	Requisitos
Descripción	El sistema almacenará la información de manera íntegra y segura, donde únicamente los administradores y usuarios con cuentas registradas podrán acceder a los datos almacenados en la base de datos.
Prioridad del requisito	Alta

4. Bibliografía

[1] E. Stephen y E. Mit, “Evaluation of software requirement specification based on IEEE 830 quality properties”, Int. J. Adv. Sci. Eng. Inf. Technol., vol. 10, núm. 4, p. 1396, 2020.

Anexo 19. Solución de defectos – Caso de estudio Gestion VR

Ingeniería en Ciencias de la Computación

Solución de defectos - Caso de estudio “Gestion VR”, relacionado a la característica de la mantenibilidad de la norma ISO/ICE 25010.

TIC: Evaluación de mantenibilidad para aplicaciones web en producción de los laboratorios de la carrera de computación de la UNL.

Versión: 1.1

Elaborado por:

Gilson Orlando Quezada

Revisado y Aprobado por:

Ing. Francisco J Álvarez-Pineda Mg. Sc

Julio 2024

Contenido

Historial de versiones

Versión	Fecha	Revisión	Historial de cambios	Autor	Verificación
1.0	10/09/2024	30/09/2024	Primera revisión	Gilson Orlando Quezada	Ing. Francisco J Álvarez Pineda Mg. Sc
1.1	16/10/2024	16/11/2024	Segunda revisión	Gilson Orlando Quezada	Ing. Francisco J Álvarez Pineda Mg. Sc

Legalización del documento

Por medio de la presente, se confirma que la documentación relacionado con la solución de defectos del caso de estudio “Gestion VR”, relacionado a la característica de la mantenibilidad de la norma ISO/ICE 25010 para el Trabajo de Integración Curricular, titulado "**Evaluación de mantenibilidad para aplicaciones web en producción de los laboratorios de la carrera de computación de la UNL**", ha sido revisada y aprobada por el director del proyecto, el Ing. Francisco J Álvarez Pineda Mg. Sc, docente de la carrera de Computación de la Universidad Nacional de Loja.

Se adjunta las firmas de los participantes.

F: _____

Ing. Francisco J Álvarez Pineda Mg. Sc

Director del Proyecto

F: _____

Gilson Orlando Quezada

Tesista

F: _____

Ing. Jorge Pucha Banegas

Colaborador

1. Introducción

El presente informe documenta el plan de estrategias para la solución de los defectos, resultado de la evaluación de mantenibilidad del caso de estudio **Gestion VR** implementando el modelo de medición de mantenibilidad adaptado. Para abordar estos problemas, se implementó una aplicación web para verificar la mantenibilidad mediante listas de verificación basadas en pautas de mantenibilidad y el cálculo del grado de mantenibilidad mediante el análisis de código estático utilizando la API de SonarQube. La lista de verificación se ha estructurado siguiendo los lineamientos de la norma ISO/IEC 25000 y ISTQB resultado de la investigación aplicada para el presente TIC. En esta sección, se describen el propósito de la investigación, el equipo involucrado en la realización del documento.

1.1. Propósito

El propósito del documento es aplicar la solución de los defectos de mantenibilidad identificados en el caso de estudio "Gestión VR", para lo cual se utilizará las pautas de mantenibilidad recopiladas en la revisión bibliográfica, el enfoque está dirigido específicamente al backend, como la área encargada de la lógica de la aplicación web. Para cumplir lo siguiente, se hace uso de SonarQube, GitHub, Excel y la aplicación web desarrollada (**QualityCCode**).

1.2. Personal involucrado

Nombre	Gilson Orlando Quezada
Rol	Analista, desarrollador y tester.
Categoría profesional	Estudiante de la carrera de Computación.
Responsabilidades	Análisis, diseño y construcción de los requisitos del aplicativo.
Información de contacto	gilson.quezada@unl.edu.ec
Aprobación	Aprobado

Nombre	Francisco Javier Álvarez Pineda
Rol	- Director de Trabajo de Integración Curricular. - Analista de requerimientos.
Categoría profesional	- Docente de la carrera de Ingeniería en Computación.
Responsabilidades	Asesorar el desarrollo del Trabajo de Integración Curricular.
Información de contacto	fjalvarez@unl.edu.ec
Aprobación	Aprobado

Nombre	Jorge Pucha Banegas
Rol	Tester.
Categoría profesional	Ex estudiante de la carrera de Computación.
Responsabilidades	Pruebas de calidad
Información de contacto	Jorge.pucha@unl.edu.ec
Aprobación	Aprobado

2. Descripción metodológica

2.2. Descripción del software para verificación de la mantenibilidad

El software **QualityCCode** para verificar la mantenibilidad del software se estructura en basa a las subcaracterística de la mantenibilidad de la norma ISO/IEC 25010, utilizando métricas como: Mala práctica, redundante, complejidad cognitiva, obsoleto, confuso, diseño, dificultad encontrada, densidad de comentarios, duplicaciones y cobertura, es importante mencionar que cada una de las métricas está relacionada a una o varias subcaracterísticas.

Cada métrica está asociada a una lista de verificación que contiene pautas y opciones que permite evaluar el grado de cumplimiento, el usuario tendrá la facultad de poder configurar las pruebas del software según sus necesidades, eligiendo evaluar por métricas o por sub características, lo que permitirá una personalización detallada del análisis de mantenibilidad.

Para implementar la o las listas de verificación para la recopilar de pautas de desarrollo, se solicitó la colaboración del Ing. Jorge Pucha Banegas como desarrollador del software, su conocimiento sobre el desarrollo del código fue fundamental para responder adecuadamente a las listas de verificación.

En la **Figura 1-A19** se muestra la participación del Ing. Jorge Pucha.

Figura 1-A19. Colaboración del Ing. Jorge Pucha.

2.3. Método de prueba

El plan de estrategias para la solución de los defectos**,** incluyó la recopilación de datos sobre los defectos identificados, instalación de herramientas (**SonarQube**, **Gestion VR** y **QualityCCode**) y la creación de casos de prueba adaptados para cada defecto.

Las especificaciones para el análisis del código se lo hacen directamente repositorio GitHub en donde se aloja el proyecto, creando archivos de configuración YML y properties, además de los secretos y variables de SonarQube que permita establecer comunicación y analizar las propiedades de calidad del código.

La gestión de cambios para la solución de los defectos, se utilizó la implementación de commits, los cuales se pueden evidenciar en el repositorio git del caso de estudio. A continuación, se evidencia el enlace del repositorio, https://github.com/gilsonOrlando/vr-backend/commits/main/.

Para analizar el cumplimiento de las pautas de mantenibilidad en la corrección de los defectos, se establecieron criterios de valoración de la lista de verificación. Esto permitirá evaluar de manera objetiva si la aplicación de las pautas ha dado lugar a resultados favorables, en la mejora de la calidad del código y la mantenibilidad del software.

Valor (%)	**Representación**
1 – 25	Deficiente
26 – 50	Insatisfactorio
51 – 75	Aceptable
76 – 100	Excelente

3. Especificación de la solución de defectos

3.1. Preparación

En la siguiente sección, se muestra los pasos ejecutados para la preparación de los casos de prueba.

- **Recopilación de datos relacionado con los defectos**

Los defectos identificados se derivan del análisis de las propiedades de calidad mediante el análisis de código estático de la herramienta de SonarQube. Estos defectos son clasificados como tales, debido a que no cumplen con los porcentajes aceptables del modelo de medición, es decir, cada una de las propiedades de calidad desencadena un defecto, sino cumple el porcentaje de aceptación.

Los cuales se menciona a continuación:

- Defecto 1 - Código duplicado.
- Defecto 2 - Falta de pruebas unitarias.
- Defecto 3 - Malas prácticas de codificación.
- Defecto 4 - Comentarios desactualizados o innecesarios.

En la **Figura 2-A19** se muestra la matriz de defectos encontrados acorde a la norma ISO/IEC 25000 respecto a la mantenibilidad.

	Subcaracterística				
Defecto	**Modularidad**	**Reusabilidad**	**Analizabilidad**	**CM**	**CP**
Código duplicado	X	x		x	
Falta de pruebas unitarias					x
Malas prácticas de codificación			x		
Comentarios desactualizados o innecesarios			x		

Capacidad de ser modificado (**CM**); Capacidad de ser probado (**CP**).

Figura 2-A19. Matriz de defectos acorde a la norma ISO/IEC 25000.

Para seleccionar una lista de verificación que se desea aplicar a un caso de estudio, se plantea la siguiente recomendación; según la columna de los defectos se toma las métricas o subcaracterísticas la cual está relacionada a una lista de verificación, como se visualiza en **Figura 3-A19**.

Propiedades de calidad SonarQube	**Defectos**	**Métricas**	**Subcaracterísticas**
Líneas duplicadas (Duplicated Lines)	Código duplicado	Duplicaciones	Modularidad / Reusabilidad / Capacidad ser modificado
Líneas de cobertura (Coverage lines)	Código no probado	Cobertura	Modularidad / Capacidad ser probado
Complejidad ciclomática (Cyclomatic complexity)	Complejidad del código	Mala practica	Reusabilidad / Analizabilidad / Capacidad ser modificado
Olores de código (Code smells)	Malas prácticas de codificación	Confuso, Dificultad encontrada	Analizabilidad / Capacidad ser modificado / Capacidad ser probado
Complejidad cognitiva (Cognitive Complexity)	Cualitativamente complejidad del código	Complejidad cognitiva	Analizabilidad
Líneas de comentadas (Comment lines)	Errores de documentación	Densidad de comentarios	Analizabilidad

Figura 3-A19. Seleccionar lista de verificación.

- **Ejecución de herramientas:**

En los siguientes puntos se especifica la ejecución de SonarQube, caso de estudio Gestión VR y el software de verificación de mantenibilidad implementado QualityCCode.

- **SonarQube**

Para la ejecución y despliegue de Sonarqube de ha utilizado la plataforma Docker.

En la **Figura 4-A19** se muestra el código para crear los contenedores de SonarQube en Docker.

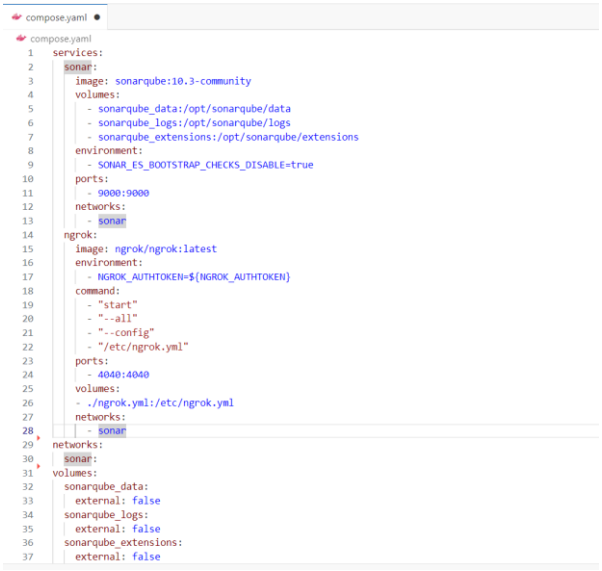


Figura 4-A19. Codificación de contenedores.

En la **Figura 5-A19** se muestra el despliegue de SonarQube a Internet.

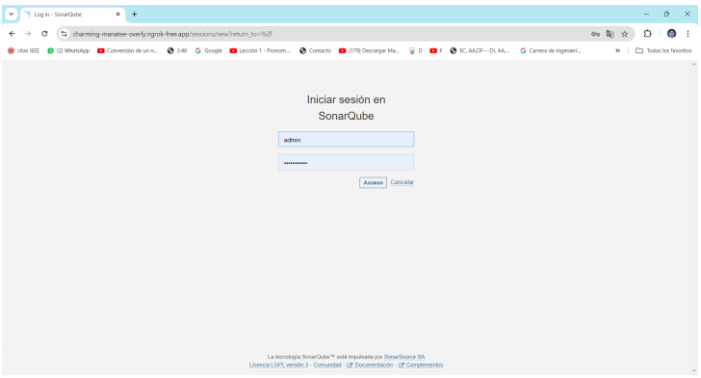


Figura 5-A19. Despliegue de SonarQube.

- **Caso de estudio Gestion VR**

Para entorno para el caso de estudio, se establece considerando los criterios de evaluación del objetivo 1, es decir basado en el análisis de código estático y calidad interna. Por tal razón, el software no exactamente debe estar en producción, pero en este caso el proyecto se lo ejecutara en un servidor local y almacenado en el repositorio GitHub para la gestión de la solución de defectos del mismo.

En la **Figura 6-A19** se muestra la ubicación local del caso de estudio para la ejecución en el servidor local, en este caso el enfoque es el backend.

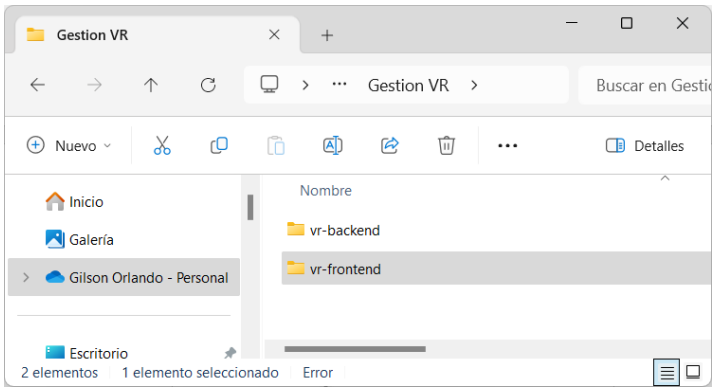


Figura 6-A19. Ubicación local del caso de estudio, computador personal.

En la **Figura 7-A19** se muestra la creación del repositorio en GitHub, para subir el caso de estudio y su respectiva gestión en la solución de defectos mediante commits.

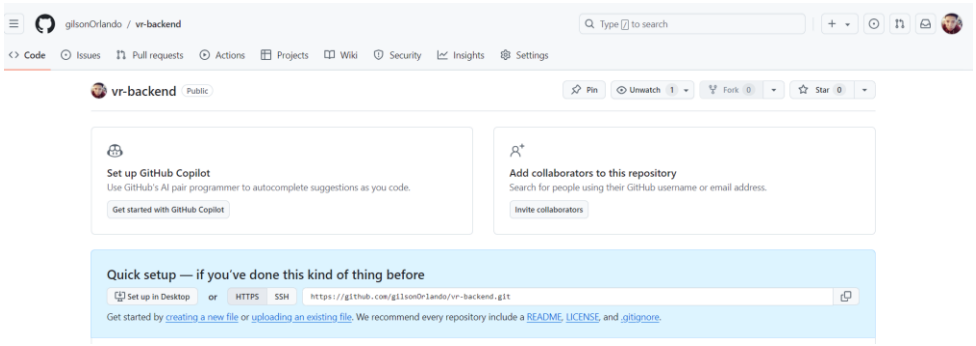


Figura 7-A19. Creación del repositorio en GitHub.

En la **Figura 8-A19** y **Figura 9-A19** se muestra la aplicación de comandos para subir los archivos del caso de estudio al repositorio GitHub.

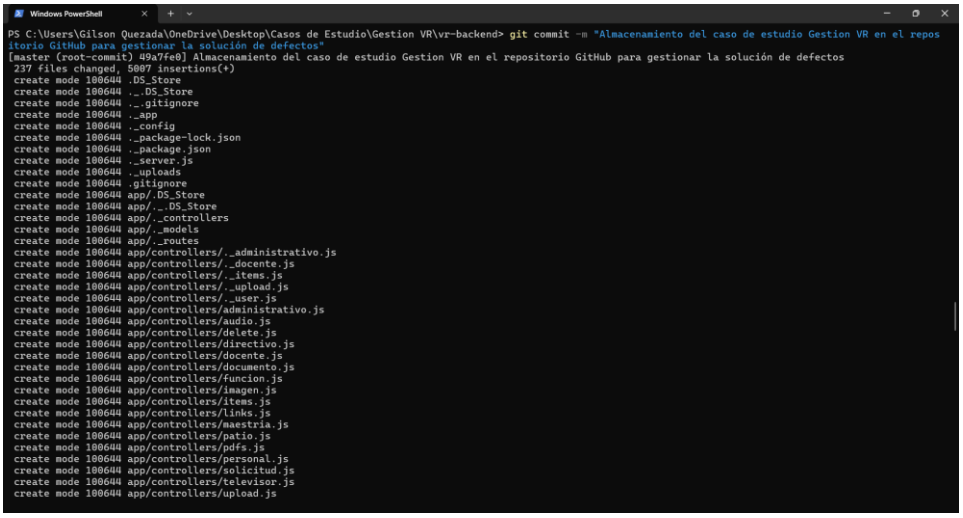


Figura 8-A19. Primer commit en el repositorio GitHub.

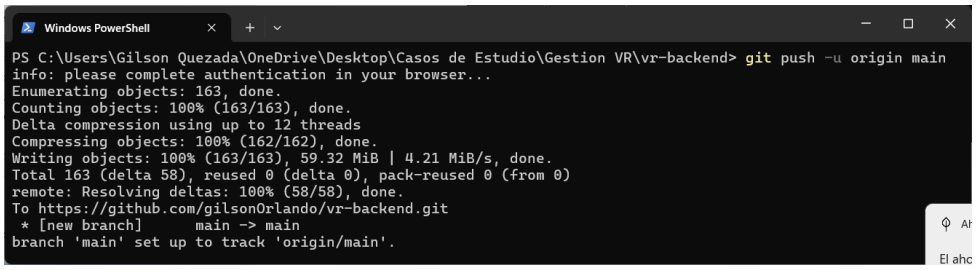


Figura 9-A19. Almacenamiento del primer commits.

En la **Figura 10-A19** se muestra la ejecución del caso del caso de estudio en el servidor local, sin aplicar la solución de defectos.

```
PS C:\Users\Gilson Quezada\OneDrive\Desktop\Casos de Estudio\Gestion VR\vr-backend> npm start

> video-node@1.0.0 start
> nodemon server.js

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
La aplicacion esta en linea!
MongoDB connected to my_app_node
```

Figura 10-A19. Ejecución del caso de estudio en el servidor local.

En la **Figura 11-A19** se muestra el despliegue del caso del caso de estudio ya en el navegador.

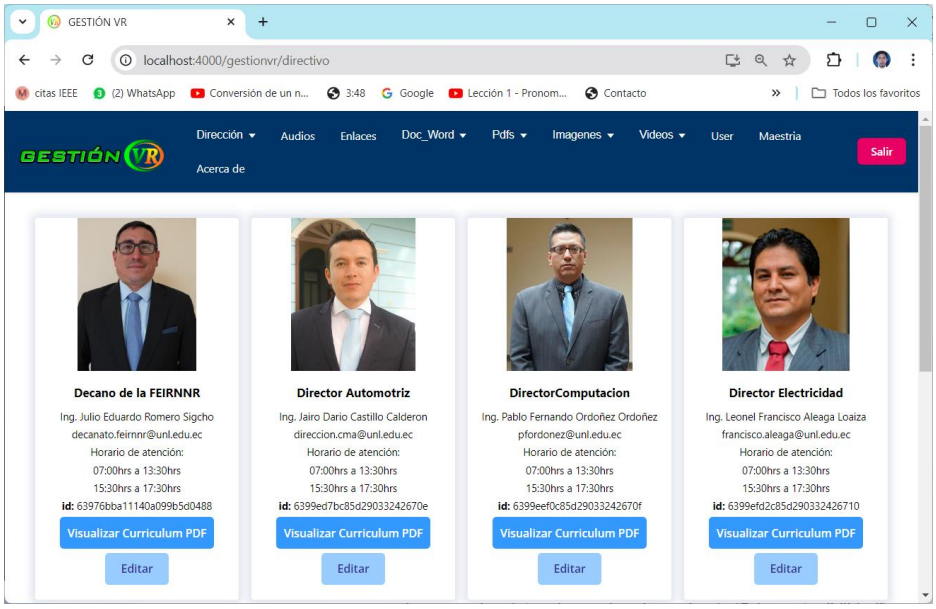


Figura 11-A19. Despliegue del caso de estudio en el navegador.

- **Software para verificar la mantenibilidad**

Para el software desarrollado denominado “QualityCCode”, se estableció los siguientes pasos.

En **Figura 12-A19** se muestra la ejecución de la aplicación QualityCCode, en el servidor local, puerto “5000”.

```
PS C:\Users\Gilson Quezada\OneDrive\Desktop\QualityCC\backend-qa> npm run dev

> projectqa@1.0.0 dev
> nodemon ./src/app.js

[nodemon] 3.1.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node ./src/app.js`
Server running on port 5000
MongoDB connected
```

Figura 12-A19. Ejecución del caso de estudio en el servidor local.

En **Figura 13-A19** se muestra el despliegue de la aplicación web en el navegador, servidor local.

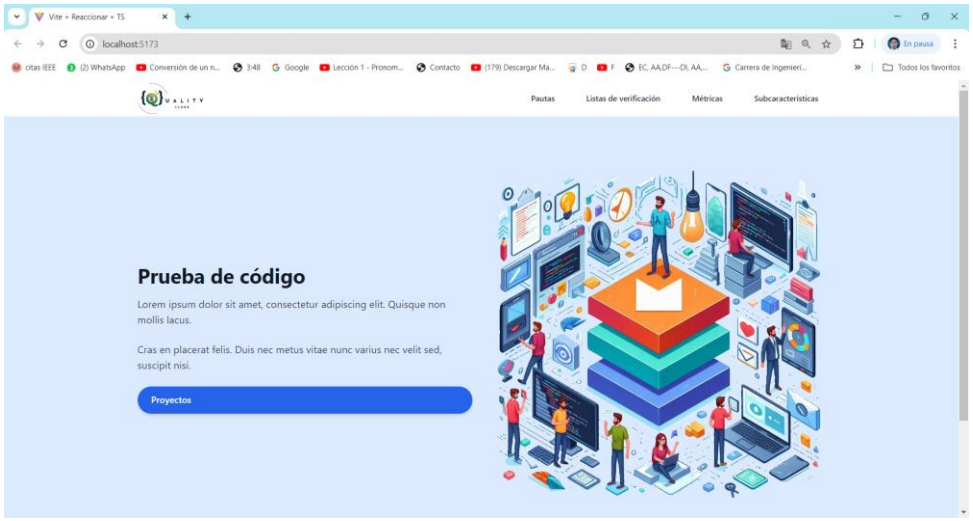


Figura 13-A19. Despliegue de QualityCCode en el navegador.

3.2. Creación de casos de prueba

En esta sección se lleva a cabo la planificación de los casos prueba para analizar y solucionar los defectos.

Para el registro y control del grado de mantenibilidad del backend del caso de estudio, se ha implementado un documento Excel, el cual va permitir registrar y actualizar según la aplicación de las pautas para la solución de los defectos.

En el siguiente enlace, se presente los registros de los valores obtenidos de la aplicación del modelo de medición de mantenibilidad de cada una de las métricas para calcular el grado de mantenibilidad del caso de estudio, es decir, estado inicial sin aplicar estrategias para la solución de defectos, https://docs.google.com/spreadsheets/d/1HoK3rSUGG2tKUElxcsqXBIwP8XBCLY71/edit?usp=sharing&ouid=103537596169053493808&rtpof=true&sd=true.

1. **Caso de prueba 1**

- **Defecto identificado**
  Código duplicado.
- **Subcaracterística de mantenibilidad afectada**
  Modularidad, reusabilidad y capacidad de ser modificado**.**
- **Criterio de Verificación:**
  El código debe estar dividido en módulos y métodos específicos sin duplicación.
- **Procedimiento de la prueba**
  - Implementación de la lista de verificación al caso de estudio, tomando en consideración a las subcaracterística o las métricas afectadas por el defecto.
  - Analizar y recopilar métodos o técnicas, resultado de aplicar lista de verificación.
  - Identificar áreas afectadas utilizando la herramienta de Sonarqube.
  - Aplicar métodos o técnicas en áreas identificadas para la solución de los defectos.
  - Segunda implementación de la lista de verificación

2. **Caso de prueba 2**

- **Defecto identificado**
  Malas prácticas de codificación.
- **Subcaracterística de mantenibilidad afectada**
  Analizabilidad, capacidad de ser modificado y capacidad de ser probado**.**
- **Criterio de Verificación:**
  Código limpio.
- **Procedimiento de la prueba**
  - Implementación de la lista de verificación al caso de estudio, tomando en consideración a las subcaracterística o las métricas afectadas por el defecto.
  - Analizar y recopilar métodos o técnicas, resultado de aplicar lista de verificación.
  - Identificar áreas afectadas utilizando la herramienta de Sonarqube.
  - Aplicar métodos o técnicas en áreas identificadas para la solución de los defectos.

- Segunda implementación de la lista de verificación

3.3. Ejecución de Pruebas

En esta sección se lleva a cabo la ejecución de los casos prueba para analizar y solucionar los defectos.

Para iniciar con la ejecución de los casos de prueba, se registró el proyecto en la herramienta de QualityCCode el cual se lo ha denominado como “VR-Backend”. En la **Figura 14-A19** se muestra el proyecto registrado el cual hace referencia al caso de estudio.

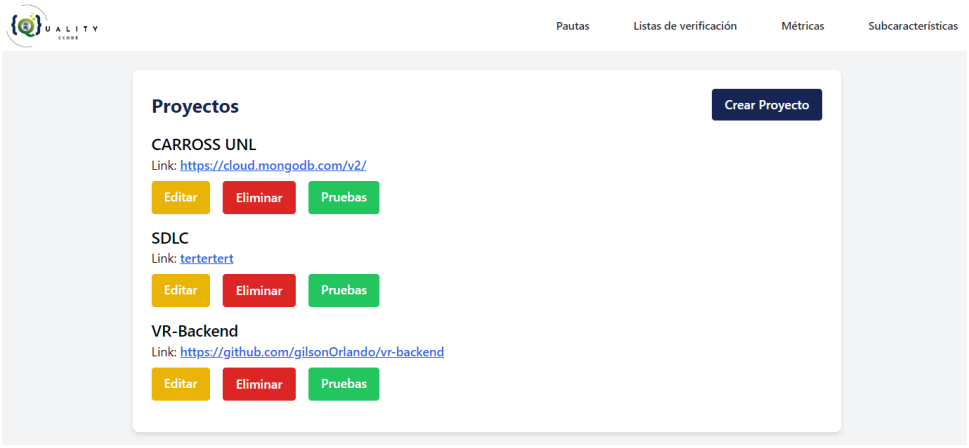


Figura 14-A19. Proyectos registrados en QualityCCode.

a. Defecto 1 – código duplicado

- **Implementación de la lista de verificación**

En esta sección se muestra la aplicación de las listas de verificación, relacionada a cada una de las subcaracterísticas o métricas afectadas respecto al defecto.

Para seleccionar la lista de verificación es pertinente tomar en cuenta la **Figura 2-A19**. En la **Figura 15-A19** se muestra la selección de la métrica de duplicaciones la cual está relacionada al defecto código duplicado.

Propiedades de calidad SonarQube	Defectos	Métricas	Subcaracterísticas
Líneas duplicadas (Duplicated Lines)	Código duplicado	Duplicaciones	Modularidad / Reusabilidad / Capacidad ser modificado
Líneas de cobertura (Coverage lines)	Código no probado	Cobertura	Modularidad / Capacidad ser probado
Complejidad ciclomática (Cyclomatic complexity)	Complejidad del código	Mala practica	Reusabilidad / Analizabilidad / Capacidad ser modificado
Olores de código (Code smells)	Malas prácticas de desarrollo	Redundante, Confuso, Dificultad encontrada	Analizabilidad / Capacidad ser modificado / Capacidad ser probado
Complejidad cognitiva (Cognitive Complexity)	Cualitativamente complejidad del código	Complejidad cognitiva	Analizabilidad
Líneas de comentadas (Comment lines)	Errores de documentación	Densidad de comentarios	Analizabilidad

Figura 15-A19. Método para seleccionar las subcaracterísticas o métricas.

Seleccionada la métrica de “duplicaciones” relacionada al defecto código duplicado, se procede a seleccionar lista de verificación en la herramienta QualityCCode.

En la **Figura 16-A19** se muestra las alternativas que presenta la herramienta para seleccionar una lista de verificación.

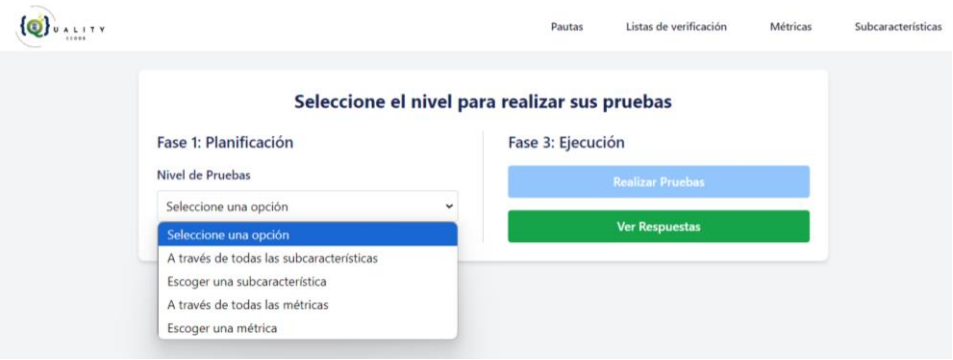


Figura 16-A19. Selección de subcaracterísticas o métricas.

Para este caso se selecciona la métrica de duplicaciones, conforme la **Figura 17-A19**, la cual está relacionada con las pautas de mantenibilidad pertenecientes al código duplicado**.** El enfoque permitirá evaluar el cumplimiento de las mejores prácticas para identificar y eliminar la duplicación de código en el proyecto.

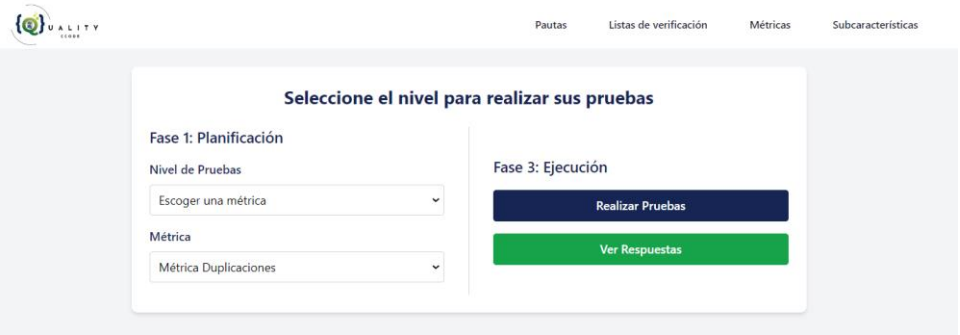


Figura 17-A19. Selección de la métrica duplicaciones.

En la **Figura 18-A19** se ilustra la ejecución de la lista de verificación, la cual se la titula como Prueba-CodigoDuplicado V1.

Figura 18-A19. Resolución de la lista de verificación.

- **Analizar y recopilar métodos, principios o técnicas, resultado de aplicar lista de verificación**

A partir de los resultados obtenidos de la lista de verificación de QualityCCode (Ver PDF https://drive.google.com/file/d/18N6sS0pFXdXzbtdy4RpGAKJyRgAiM9od/view?usp=sharing) relacionada con el defecto “código duplicado”, muestra un porcentaje del 38.89% de cumplimiento de pautas de mantenibilidad (Considerado insuficiente), la cual es la inferencia del usuario en base al análisis de la herramienta de SonarQube.

A continuación, se detalla cada uno de los métodos, principios o técnicas para la solución de los defectos, para conocer y aplicar cada uno de los mismo es necesario revisar los documentos resultado de la revisión bibliográfica (Ver documentos https://drive.google.com/drive/folders/1TySGU0tRZyklbN-1u2a5xbla5-n_oJGo?usp=drive_link) de los cuales se obtuvo las de pautas de mantenibilidad.

- **DRY**: Evita la duplicación de código creando funciones auxiliares (Ver **Documento 22**).
- **Single Responsibility Principle (SRP):** Cada función se encarga de una sola tarea específica **(**Ver **Documento 22**).
- **Code Readability and Clarity**: Código más limpio y fácil de entender **(**Ver **Documento 5**).
- **Encapsulation**: Encapsulación de la lógica común en funciones separadas **(**Ver **Documento 11**).
- **Modularization**: Dividir el código en módulos manejables **(**Ver **Documento 11**).

- **Error Handling**: Manejo de errores consistente en todo el código. Identificar **(**Ver **Documento 29**).

- **Identificar áreas afectadas utilizando la herramienta de Sonarqube**

Para identificar las áreas en donde existe código duplicado se hace uso de sonarqube, mediante el análisis de código estático. Esta técnica es fundamental para abordar y solucionar los defectos detectados.

En la **Figura 19-A19** se detalla la cantidad y el porcentaje de líneas duplicadas, específicamente en el área del backend, con una cantidad de 1,228 líneas, lo que representa un 56,86% del total de 2,161 líneas de código.

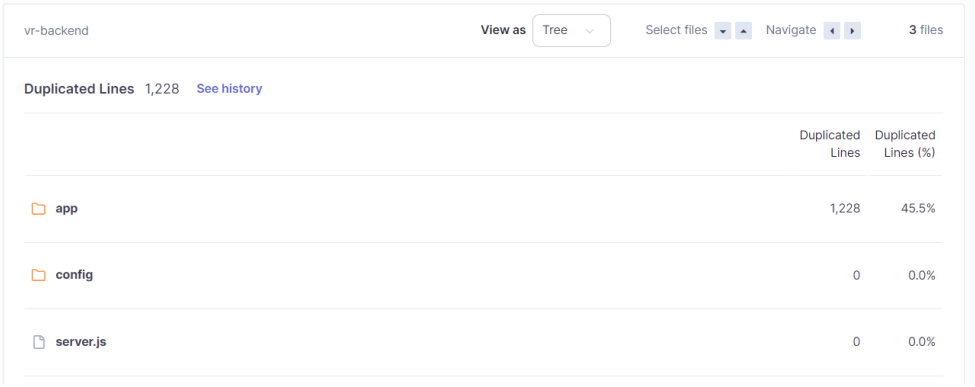


Figura 19-A19 Líneas duplicadas utilizando SonarQube.

En la **Figura 20-A19** se ilustran las áreas del backend donde se presenta el defecto, detalladamente se identifica que la sección de controladores es la que muestra una mayor incidencia del mismo.

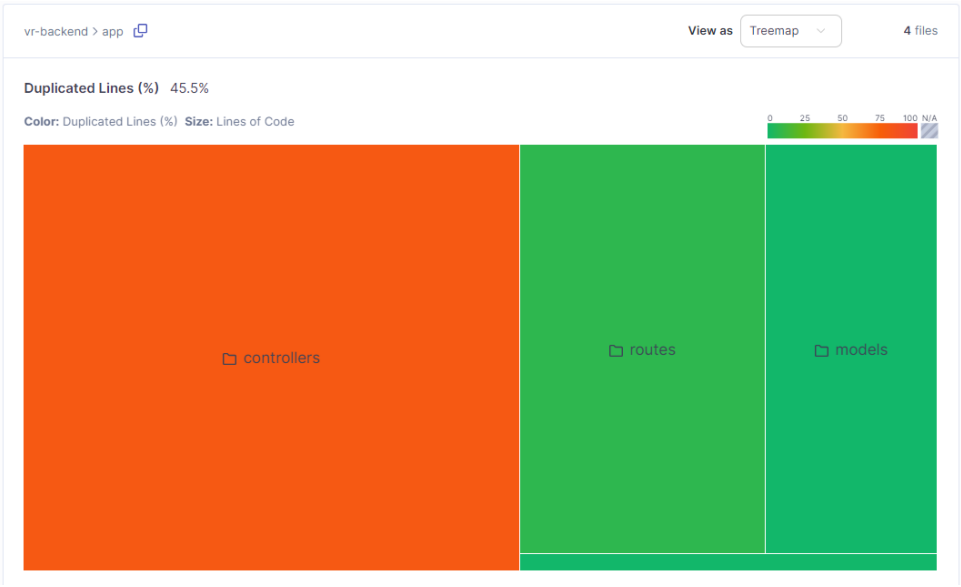


Figura 20-A19 Líneas duplicadas en el área del backend utilizando SonarQube.

En la **Figura 21-A19** se ilustran las clases del módulo controladores donde se presenta el defecto, la figura resalta áreas específicas del código en las que se manifiesta el defecto, ofreciendo una visualización clara que facilita su identificación y análisis.

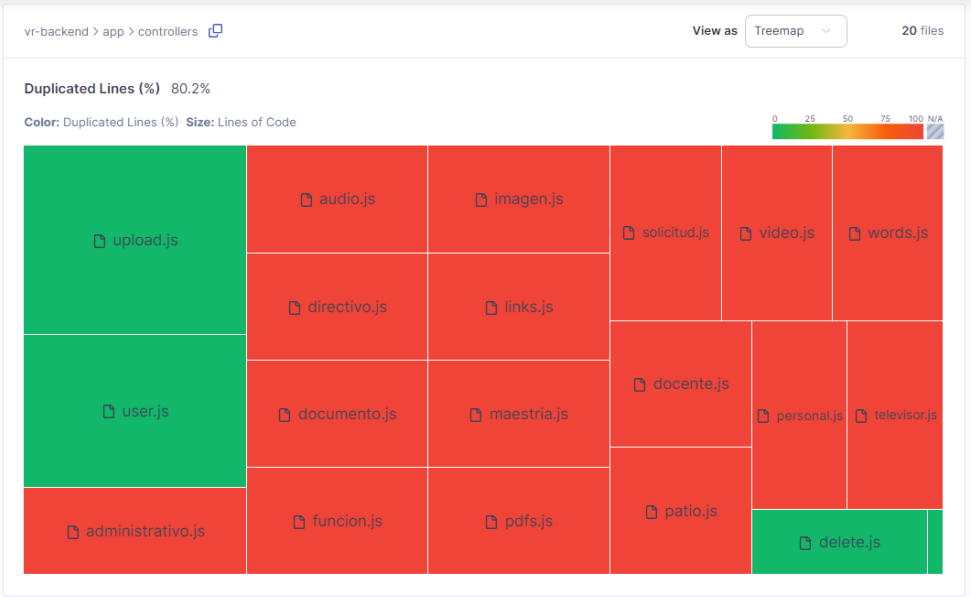


Figura 21-A19. Líneas duplicadas modulo controladores.

- **Aplicar métodos o técnicas en áreas identificadas para la solución de defectos.**

Para la solución de defectos, se consideran las recomendaciones establecidas en las pautas de mantenibilidad incluidas en la lista de verificación, cada pauta está relacionada un documento recuperado de la revisión bibliográfica que proporciona directrices claras y prácticas que facilitan corrección de defectos en el código.

En la **Figura 21-A19** se destacan áreas específicas del módulo controladores donde se manifiestan los defectos. Para abordar la solución de los defectos en el caso de estudio, se tomará como punto de partida la clase "**administrativo.js**". Esta clase o todas son fundamentales en la estructura del proyecto, ya que contiene funcionalidades clave acerca de la lógica de aplicación.

Para aplicar las recomendaciones planteadas es necesario considerar los métodos y técnicas recopilados de la lista de verificación aplicada (Ver **estrategias**).

En la **Figura 22-A19** se muestra las secciones específicas de la clase administrativo donde existe código duplicado.

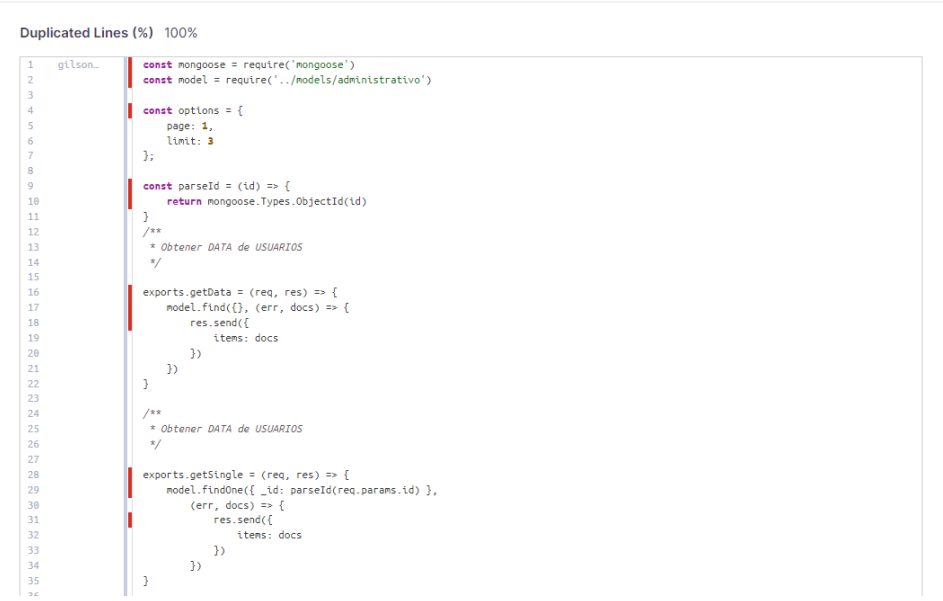


Figura 22-A19. Líneas duplicadas clase administrativo.

En la **Figura 23-A19** se ilustra el método implementado para manejar las respuestas del cliente.

```
 * Función para manejar la respuesta al cliente
 * @param {Object} res - Objeto de respuesta Express
 * @param {Object} data - Datos a enviar en la respuesta
 * @param {number} [statusCode=200] - Código de estado HTTP (predeterminado: 200 - OK)
 */
const handleResponse = (res, data, statusCode = 200) => {
  res.status(statusCode).send(data);
};
```

Figura 23-A19. Método para el manejo de respuesta al cliente.

En la **Figura 24-A19** se ilustra el método implementado para manejar errores y respuestas de error.

```
/**
 * Función para manejar errores y enviar respuestas de error
 * @param {Object} res - Objeto de respuesta Express
 * @param {Error} err - Error que ocurrió
 */
const handleError = (res, err) => {
  console.error(err); // Log del error en la consola para depuración
  handleResponse(res, { error: 'Ocurrió un error al procesar la solicitud.' }, 500);
};
```

Figura 24-A19. Método para el manejo de errores y respuestas de error.

Aplicado los cambios a la clase administrativo se procede a actualizar el repositorio git mediante el uso de commits.

En la **Figura 25-A19** se muestra la actualización del repositorio git mediante la creación de un commit con los cambios aplicados en la clase administrativo.

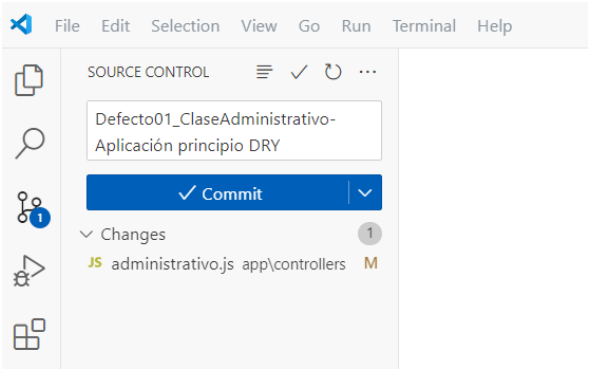


Figura 25-A19. Creación del primer commit relacionado al defecto código duplicado.

El commit se puede evidenciar ingresando al repositorio git del caso de estudio (https://github.com/gilsonOrlando/vr-backend).

En la **Figura 26-A19** se muestra el commit en la plataforma git.

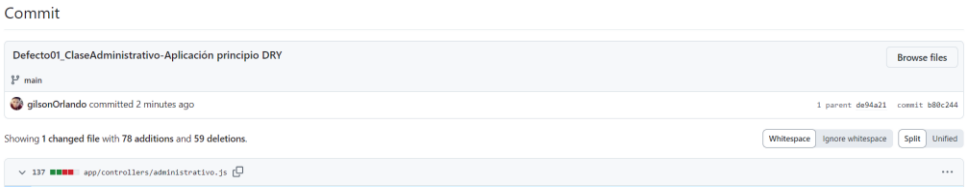


Figura 26-A19. Commit 13 repositorio git.

En el siguiente enlace **https://youtu.be/lokKGQ8V-PE** se puede evidenciar el funcionamiento del caso de estudio luego de aplicar los métodos o técnicas para solucionar el defecto denominado como código duplicado.

A sí mismo para evidenciar si el defecto ha sido tratado correctamente, se aplica nuevamente el análisis de código estático con SonarQube, en donde se muestra resultados positivos.

En la **Figura 27-A19** se muestra el treemap de SonarQube, en donde la clase de administrativo ha cambiado de color, rojo a verde, el cual representa que ya no existe código duplicado.

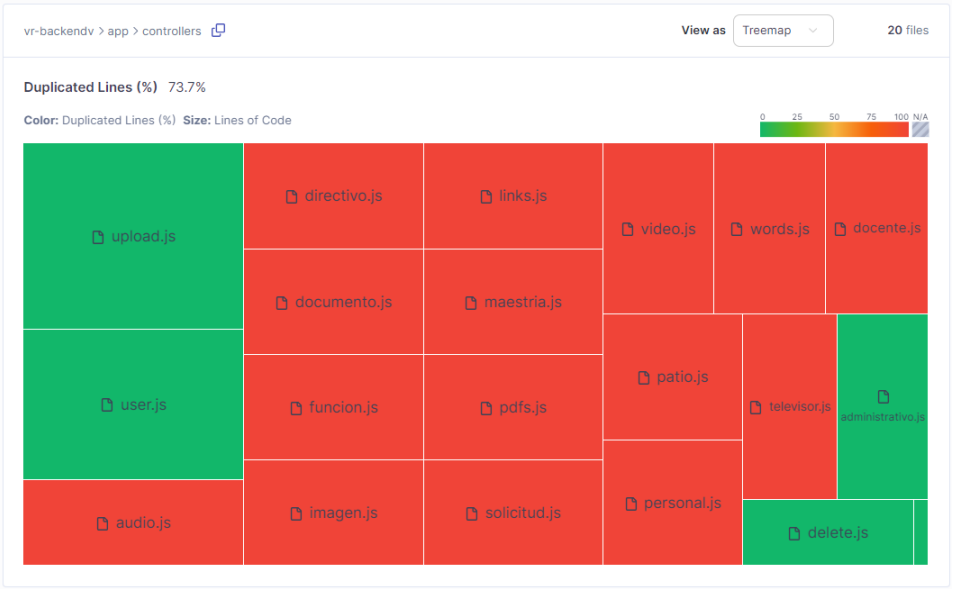


Figura 27-A19. Resultado en la solución del defecto código duplicado.

En la **Figura 28-A19** se evidencia, una vez aplicado las soluciones respectivas la cantidad de líneas duplicas disminuye significativamente de 1,228 a 1,148, así como las líneas de código del proyecto, en comparación con el estado inicial del caso de estudio Ver **Figura 19-A19**.

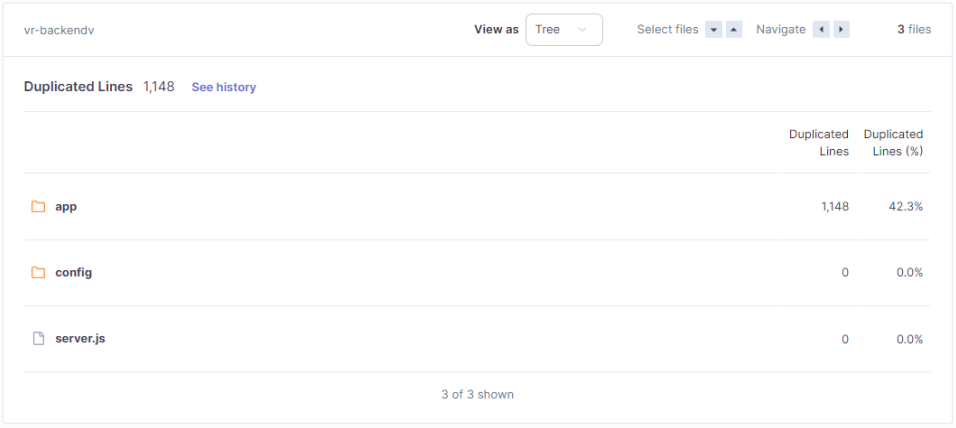


Figura 28-A19. Disminución de líneas duplicadas.

De la misma forma, se aplican los mismos métodos y técnicas para solucionar los defectos de las siguientes clases del módulo de controladores. Para tener constancia de los cambios, se creará un commit por cada clase del módulo que se aplique cambios.

En la **Tabla 1**, se presenta un resumen de commits aplicados en la solución de defectos, de las clases del módulo "controllers". Este módulo contiene clases fundamentales del caso de estudio que han sido afectadas y seleccionadas para este proceso.

Tabla 1. Lista de commits aplicados

Solución de defectos	
Clase	**Nombre del commit**
Administrativo	Defecto01_ClaseAdministrativo-Aplicación principio DRY
Documento	Defecto01_ClaseDocumento-Solucióndefecto
Audio	Defecto01_ClaseAudio- Solucióndefecto
Directivo	Defecto01_ClaseDirectivo- Solucióndefecto
Función	Defecto01_ClaseFuncion- Solucióndefecto
Imagen	Defecto01_ClaseImagen- Solucióndefecto
Links	Defecto01_ClaseLinks- Solucióndefecto
Maestría	Defecto01_ClaseMaestria- Solucióndefecto
Pdfs	Defecto01_ClasePdfs- Solucióndefecto
Personal	Defecto01_ClasePersonal- Solucióndefecto
Video	Defecto01_ClaseVideo- Solucióndefecto
Words	Defecto01_ClaseWords- Solucióndefecto
Docente	Defecto01_ClaseDocente- Solucióndefecto
Patio	Defecto01_ClasePatio- Solucióndefecto

En la **Figura 29-A19** se muestra gráficamente los resultados de aplicar las respectivas soluciones en comparación con la **Figura 231-A19** en donde no se había aplicado los método o técnicas para solucionar los mismo.

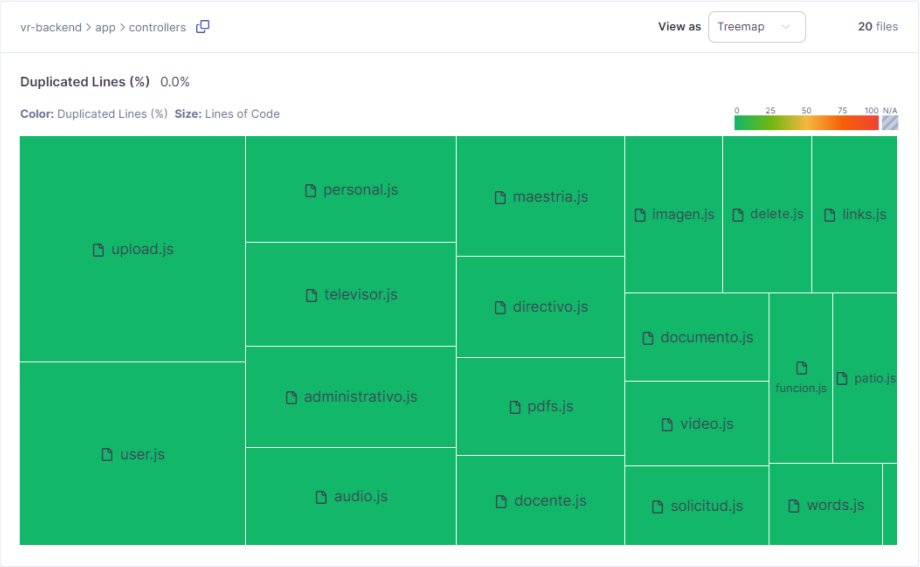


Figura 29-A19. Resultado de la aplicación de pautas de mantenibilidad para eliminar líneas duplicadas

A continuación, se documenta el nuevo valor de la propiedad de calidad relacionada con la cantidad de líneas de código duplicadas. Para ello, se ha creado un archivo de **Excel** que permite registrar y actualizar el grado de mantenibilidad del backend tras la implementación de las pautas de mantenibilidad.

Líneas duplicadas **62**, de 1870 líneas de código, equivalente al 3.31%.

Enlace,
https://docs.google.com/spreadsheets/d/1MpQICqJyBhKjncv_MjdPYDzYmtBwIh_d/edit?usp=sharing&ouid=103537596169053493808&rtpof=true&sd=true.

Es importante tomar en cuenta, al aplicar solución a un defecto este contribuye a la mejora de otras propiedades de calidad, que en efecto mejora la calidad del producto final.

- **Segunda implementación de la lista de verificación**

La segunda implementación de la lista de verificación en la herramienta QualityCCode, correspondiente al defecto código duplicado, tiene como objetivo evaluar el impacto que ha tenido la aplicación de pautas de mantenibilidad, validando a través del análisis de código estático realizado con la herramienta de Sonarqube.

En la **Figura 30-A19** se muestra la implementación de la lista de verificación, la cual es denominada como Prueba-CodigoDuplicado V2.

Figura 30-A19. Resultado de la aplicación de pautas de mantenibilidad defecto “código duplicado”.

Los resultados obtenidos de la segunda aplicación de la lista de verificación de QualityCCode (Ver PDF https://drive.google.com/file/d/1SizWjK8_fV8HMqbJqPGlKSpINasRbG85/view?usp=sharing) muestra una mejora significativa en la eliminación de líneas duplicadas, el porcentaje alcanzado es del 83.33%, lo que se considerado como excelente como criterios establecidos, en comparación, de la primera aplicación de la lista de verificación arrojó un porcentaje del 38.89%, evidenciando un incremento sustancial en la efectividad de las soluciones implementadas para abordar este defecto.

b. Defecto 3 – malas prácticas de codificación

- **Aplicar lista de verificación**

En la **Figura 31-A19** se ilustra la selección de la lista de verificación acorde al defecto.

Propiedades de calidad SonarQube	Defectos	Métricas	Subcaracterísticas
Líneas duplicadas (Duplicated Lines)	Código duplicado	Duplicaciones	Modularidad / Reusabilidad / Capacidad ser modificado
Líneas de cobertura (Coverage lines)	Código no probado	Cobertura	Modularidad / Capacidad ser probado
Complejidad ciclomática (Cyclomatic complexity)	Complejidad del código	Mala practica	Reusabilidad / Analizabilidad / Capacidad ser modificado
Olores de código (Code smells)	Malas prácticas de desarrollo	Redundante, Confuso, Dificultad encontrada	Analizabilidad / Capacidad ser modificado / Capacidad ser probado
Complejidad cognitiva (Cognitive Complexity)	Cualitativamente complejidad del código	Complejidad cognitiva	Analizabilidad
Líneas de comentadas (Comment lines)	Errores de documentación	Densidad de comentarios	Analizabilidad

Figura 31-A19. Selección de la lista de verificación, defecto malas prácticas de codificación.

En este caso se ha seleccionado las métricas de “confuso y redundante” relacionada al defecto de malas prácticas de codificación, a continuación, se procede a seleccionar lista de verificación en la herramienta QualityCCode.

En la **Figura 32-A19** se muestra las alternativas que presenta la herramienta para seleccionar una lista de verificación.

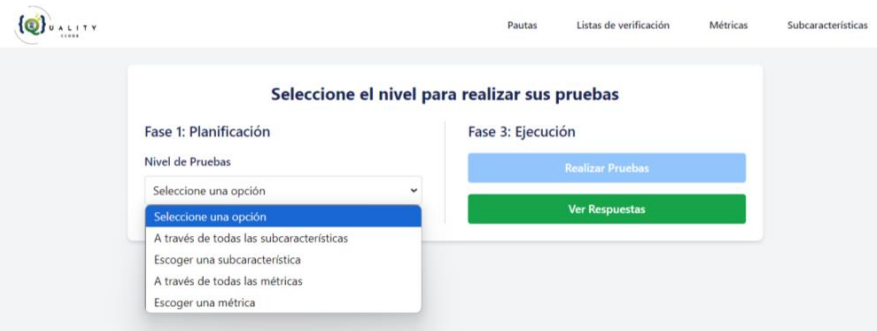


Figura 32-A19. Selección de subcaracterísticas o métricas.

Se selecciona la métrica código confuso, como se muestra en la **Figura 33-A19**. Al iniciar la prueba de la lista de verificación, esta incluirá pautas de mantenibilidad que se alinean con las buenas prácticas de codificación.

Figura 33-A19. Selección de la métrica confuso y redundante.

En la **Figura 34-A19** se ilustra la ejecución de la lista de verificación, la cual la es denominada como Prueba-MalasPracticasCodificacion V1.

Figura 34-A19. Resolución de la lista de verificación.

- **Analizar y recopilar métodos, principios o técnicas, resultado de aplicar lista de verificación**

A partir de los resultados obtenidos de la lista de verificación de QualityCCode (Ver PDF https://drive.google.com/file/d/1733hUFAY1s5UEGbOminDM_6TCXmbpX1q/view?usp=sharing) relacionada con el defecto “malas prácticas de codificación”, muestra un porcentaje del 51.14% de cumplimiento de pautas de mantenibilidad (Considerado como aceptable), la misma que es una inferencia del usuario en base al análisis de la herramienta de SonarQube.

A continuación, se describe cada uno de los métodos, principios o técnicas para la solución de los defectos, para lo cual es necesario conocer y aplicar cada uno de los mismo, esto impulsa a revisar los documentos resultado de la revisión bibliográfica (Ver documentos https://drive.google.com/drive/folders/1TySGU0tRZyklbN-1u2a5xbla5-n_oJGo?usp=drive_link) para la obtención de pautas de mantenibilidad.

- **Manejo de errores (err.message, log de error):** Se refiere a las técnicas y estrategias utilizadas para manejar situaciones inesperadas o excepcionales que pueden ocurrir durante la ejecución de un programa **(**Ver **Documento 11**).
- **Construcción de rutas de archivos:** path.join Es una técnica utilizada en programación, especialmente en Node.js, para crear rutas de archivos de manera segura y eficiente **(**Ver **Documento 11**).
- **Utilización de excepciones:** Es una técnica de programación que permite a los desarrolladores controlar y gestionar los errores que pueden ocurrir durante la ejecución de un programa **(**Ver **Documento 5**).
- **Manejo de variables y control de alcance:** Es forma de declarar, utilizar y gestionan las variables en un programa **(**Ver **Documento 5**).
- **DRY**: Evita la duplicación de código creando funciones auxiliares (Ver **Documento 22**).
- **Single Responsibility Principle (SRP):** Cada función se encarga de una sola tarea específica **(**Ver **Documento 22**).
- **Encapsulation**: Encapsulación de la lógica común en funciones separadas **(**Ver **Documento 11**).
- **Error Handling**: Manejo de errores consistente en todo el código. Identificar **(**Ver **Documento 29**).

- **Identificar áreas afectadas utilizando la herramienta de Sonarqube**

Para identificar las áreas en donde existe malas prácticas de codificación se hace uso de SonarQube, aplicando el análisis de código estático. Esta técnica es fundamental para abordar y solucionar los defectos detectados.

En la **Figura 35-A19** se presentan los defectos relacionados con las “malas prácticas de codificación”, se ha detectado una cantidad de 0 instancias de malas prácticas, lo que representa un 0% del total de 1,870 líneas de código.

Figura 35-A19 Líneas duplicadas utilizando SonarQube.

En la **Figura 36-A19** se ilustran las 3 primeras malas prácticas de codificación del backend.

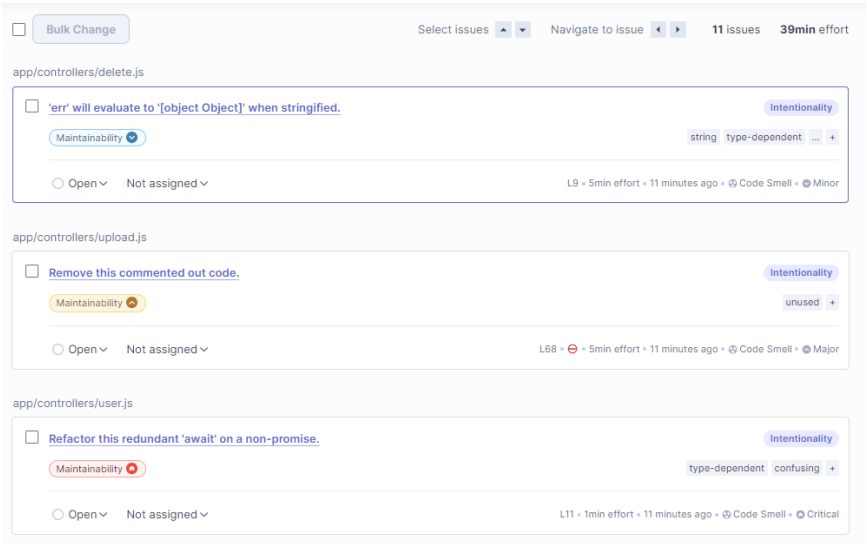


Figura 36-A19. Malas prácticas de codificación identificadas en SonarQube.

- **Aplicar métodos o técnicas en áreas identificadas para la solución de defectos.**

Para la solución de defectos, se consideran las recomendaciones establecidas en las pautas de mantenibilidad incluidas en la lista de verificación, cada pauta está relacionada un documento recuperado de la revisión bibliográfica que proporciona directrices claras y prácticas que facilitan corrección de defectos en el código.

En la **Figura 36-A19** se destacan áreas específicas en donde existe malas prácticas de codificación, para abordar la solución de los defectos en el caso de estudio, se tomará como punto de partida el defecto titulado " 'err' will evaluate to '[object Object]' when stringified". Cada sección identificada contiene funcionalidades clave que requiere atención.

Para aplicar las recomendaciones planteadas es necesario considerar los métodos y técnicas recopilados de la lista de verificación aplicada (Ver **estrategias**).

En la **Figura 37-A19** se muestra el área específica del primer caso de mala práctica de codificación, clase delete.

Figura 37-A19. Mala práctica de codificación clase delete.

En la **Figura 38-A19** se ilustra el método modificado para eliminar archivos.

```
// Función para eliminar archivos por su nombre (asíncrona)
const remove = (req, res) => {
  const fileName = req.params.name;
  const directoryPath = path.join(__basedir, "uploads", fileName); // Usar path para mayor seguridad y consistencia

  fs.unlink(directoryPath, (err) => {
    if (err) {
      return res.status(500).send({
        message: "Could not delete the file. Error: " + err.message // Usar err.message para un error más claro
      });
    }

    res.status(200).send({
      message: "File is deleted."
    });
  });
};
```

Figura 38-A19. Mejora del para eliminar archivos por su nombre.

La modificación consiste en lugar de concatenar el objeto err directamente en los mensajes de error se utiliza err.message, esto evita que el objeto err se convierta en una cadena como [object Object], lo que SonarQube detecta como un problema (code smell).

En la **Figura 39-A19** se ilustra la mejora para construir rutas de archivos con path.join().

```
6   const fileName = req.params.name;
7   const directoryPath = path.join(__basedir, "uploads", fileName); // Usar path para mayor seguridad y consistencia
8
```

Figura 39-A19. Rutas de archivos con path.join().

La modificación consiste en concatenar cadenas para formar la ruta del archivo, para lo cual se utilizó la función path.join() de Node.js, esto asegura que las rutas de archivos se manejen de forma adecuada en diferentes sistemas operativos (evitando posibles errores en Windows o Linux debido a las diferencias en las rutas de archivos).

En la **Figura 40-A19** se ilustra la modificación para el retorno de respuestas.

```
10  if (err) {
11    return res.status(500).send({
12      message: "Could not delete the file. Error: " + err.message // Usar err.message para un error más claro
13    });
14  }
```

Figura 40-A19. Modificación para el retorno de respuestas.

La solución consiste en agregar al método remove un return después de enviar la respuesta cuando se produce un error, esto detiene la ejecución de más código y evita problemas lógicos.

Aplicado los cambios a la clase delete se procede a actualizar el repositorio git mediante el uso de commits.

En la **Figura 41-A19** se muestra la actualización del repositorio git, con los cambios aplicados en la clase delete, mismo que permitirá evidenciar la solución del defecto relacionado con malas prácticas de codificación.

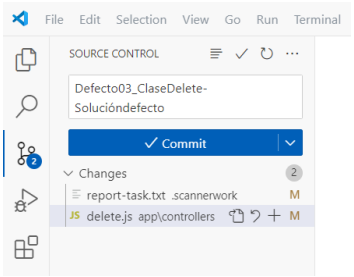


Figura 41-A19. Creación del primer commit relacionado a las malas prácticas de codificación.

El commit se puede evidenciar ingresando al repositorio git del caso de estudio (https://github.com/gilsonOrlando/vr-backend).

En la **Figura 42-A19** se muestra el commit en la plataforma git.

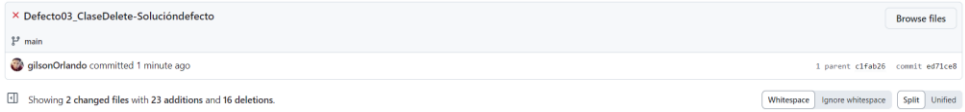


Figura 42-A19. Commit repositorio git.

A sí mismo, para evidenciar si el defecto ha sido tratado correctamente, se aplica nuevamente el análisis de código estático con SonarQube, en donde se muestra resultados positivos.

En la **Figura 43-A19** se ilustra después de haber aplicado las respectivas soluciones al defecto 1 (código duplicado), las incidencias de malas prácticas de codificación (Code Smell) disminuyen, de 11 casos a 8 casos.

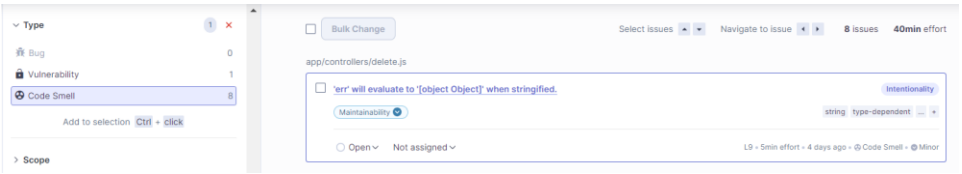


Figura 43-A19. Reducción de malas prácticas de codificación.

En la **Figura 44-A19** muestra los resultados obtenidos después de aplicar las soluciones correspondientes al primer caso de mala práctica de codificación. Se puede observar una respuesta positiva en la solución del mismo, en comparación con el estado inicial del caso de estudio Ver **Figura 37-A19**.

Figura 44-A19. Resultado de la solución del primer caso de mala práctica de codificación.

En la **Figura 45-A19** de la misma forma se muestra la reducción de las malas prácticas de codificación de 8 a 7.

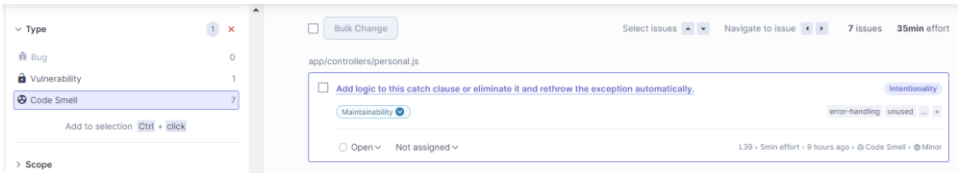


Figura 45-A19. Reducción de malas prácticas de codificación.

De la misma forma, se aplican los mismos métodos y técnicas para solucionar las siguientes malas prácticas de codificación. Para tener constancia de los cambios, para lo cual se creará un commit por cada mala práctica.

En la **Tabla 2** presenta un resumen de commits creados, como evidencia de la solución de los defectos de malas prácticas de codificación mencionadas anteriormente. Las malas prácticas están en clases fundamentales del caso de estudio, lo que es importante para garantizar un código mantenible.

Tabla 2. Lista de commits aplicados.

Solución de defectos	
Clase	**Nombre del commit**
Delete	Defecto02_ClaseDelete-Solucióndefecto
Personal	Defecto02_ClasePersonal-Solucióndefecto
Upload	Defecto02_ClaseUpload-Solucióndefecto
Authjwt	Defecto02_ClaseAuthjwt -Solucióndefecto
Passport	Defecto02_Clase Passport -Solucióndefecto

En la **Figura 46-A19** se muestra gráficamente los resultados luego de aplicar las respectivas soluciones de malas prácticas de codificación en comparación con la **Figura 43-A19** en donde no se había aplicado los método o técnicas para solucionar los mismo.

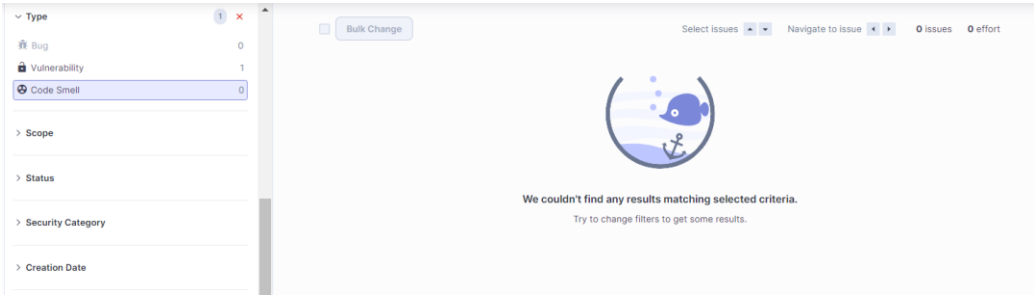


Figura 46-A19. Resultado de la aplicación de pautas de mantenibilidad defecto “malas prácticas de codificación”.

A continuación, se documenta el nuevo valor de la propiedad de calidad relacionada con las malas prácticas de codificación. Para ello, se hace uso el archivo de Excel para registrar y actualizar el grado de mantenibilidad del backend tras la implementación de las pautas de mantenibilidad.

Malas prácticas de codificación 0, equivalente al 0%.

Enlace, **https://docs.google.com/spreadsheets/d/1MpQlCqJyBhKjncv_MjdPYDzYmtBwlh_d/edit?usp=sharing&ouid=103537596169053493808&rtpof=true&sd=true**.

- **Segunda implementación de la lista de verificación**

La segunda implementación de la lista de verificación en la herramienta QualityCCode, **correspondiente** al defecto malas prácticas de codificación, tiene como objetivo evaluar el impacto que ha tenido la aplicación de pautas de mantenibilidad en la solucionar de el defecto, validando a través del análisis de código estático realizado con la herramienta de Sonarqube.

En la **Figura 47-A19** se **muestra** la implementación de la lista de verificación, la cual es denominada como Prueba-MalasPrácticasCodificación V2.

Figura 47-A19. Resultado de la aplicación de pautas de mantenibilidad defecto “malas prácticas de codificación”.

Los resultados **obtenidos** de la segunda aplicación de la lista de verificación de QualityCCode (Ver PDF **https://drive.google.com/file/d/1K-Aw9rp95GJsiMLcQvafgk0dwoNxOiUJ/view?usp=sharing)** muestra una mejora significativa en la eliminación de las malas prácticas de codificación. El porcentaje alcanzado es del 60%, lo que se considerado como aceptable como criterios establecidos, en comparación, de la primera aplicación de la lista de verificación arrojó un porcentaje del 51.14%, evidenciando un incremento sustancial en la efectividad de las soluciones implementadas para abordar este defecto.

3.4. Documentación de resultados

En esta sección se lleva a cabo la documentación de resultados de los casos prueba.

Para calcular el grado de **mantenibilidad** del caso de estudio después de la aplicación del plan de estrategias, se ha implementado un documento Excel (**https://docs.google.com/spreadsheets/d/1MpQICqJyBhKjncv_MjdPYDzYmtBwIh_d/edit?usp=sharing&ouid=103537596169053493808&rtpof=true&sd=true**) que permite realizar un seguimiento del progreso a medida que se aplican las modificaciones para la solución de los defectos. Este documento se basa en cálculos matemáticos utilizando fórmulas para evaluar las métricas y subcaracterísticas de mantenibilidad, recopiladas durante la investigación previa del presente TIC.

El documento excel permitió cuantificar el grado de mantenibilidad del software de manera objetiva y sistemática, utilizando una escala de valoración del 0% al 100%. Según los resultados después de aplicar el plan de estrategias para la solución de los defectos de mayor relevancia del

backend se ha logrado alcanzar un grado de mantenibilidad del caso de estudio “Gestion VR” del 82,34% al 89,62%, el cual se ubica con una escala de valoración de altamente mantenible, este rango representa un nivel de calidad en el que el software es fácil de entender, modificar y probar.

Así mismo, para calcular el grado de mantenibilidad del backend durante la implementación del plan de estrategias para solución de defectos, se elaboró un documento de Excel. En el siguiente enlace se puede visualizar el documento sin aplicar solución de defectos **https://docs.google.com/spreadsheets/d/1V81NxLrq5v3ywRO9fmxbup-vKphecqSM/edit?usp=sharing&ouid=103537596169053493808&rtpof=true&sd=true**. De igual forma en el siguiente enlace se puede visualizar el documento después de aplicar solución de defectos **https://docs.google.com/spreadsheets/d/1DulHSAjkIUAC7fZfjhaN7FiY1kNpRlXT/edit?usp=sharing&ouid=103537596169053493808&rtpof=true&sd=true**.

- **Defecto 1: Código duplicado**

Las modificaciones realizadas, ha permitido evidenciar mejoras significativas en la calidad del código, especialmente en lo que respecta al "código duplicado" en las clases del módulo "controllers", uno de los más críticos del proyecto. La implementación de las sugerencias basadas en pautas de mantenibilidad ha sido fundamental para lograr estos avances, esto se evidencia en la reducción del porcentaje de código duplicado del 56,77% al 3,1%, lo que incrementa el grado de mantenibilidad del backend del 73,07% al 87,56%.

Las sugerencias incluyen principios como DRY (Don't Repeat Yourself), SRP (Single Responsibility Principle), encapsulación y modularización, ha permitido crear un código más limpio, legible y fácil de mantener. La aplicación de estas pautas ha tenido un impacto positivo en la calidad del software, facilitando la identificación y corrección de errores, y sentando las bases para una mayor escalabilidad y flexibilidad en el futuro. Estos resultados demuestran la efectividad de las pautas de mantenibilidad y la importancia de adoptar buenas prácticas de programación.

- **Defecto 3: Malas prácticas de codificación**

La aplicación de buenas prácticas de programación ha permitido evidenciar mejoras significativas en la calidad del código, especialmente en lo que respecta al defecto de "malas prácticas de codificación", en las clases del módulo "controllers", "middlewares" y "config", unos de los más críticos del proyecto. La implementación de las sugerencias basadas en pautas de mantenibilidad ha sido fundamental para lograr estos avances, esto se evidencia en la eliminación por completo de estas apariciones, reduciendo la cantidad de las malas prácticas, lo que incrementa el grado de mantenibilidad del backend del 87,56% al 87,63%.

La implementación de técnicas como el manejo de errores, a través de err.message y registros de error ha permitido gestionar de manera efectiva las situaciones excepcionales que pueden surgir durante la ejecución del programa. Además, la construcción de rutas de archivos mediante path.join garantiza que las rutas se generen de forma segura y eficiente, evitando problemas relacionados con la portabilidad entre diferentes sistemas operativos. La utilización de excepciones proporciona un marco para controlar y gestionar errores, lo que mejora la robustez del software, asimismo, el manejo de variables y control de alcance asegura que las variables se declaren y utilicen de manera adecuada, minimizando conflictos y errores.

Así mismo, como en la solución del defecto 1, se aplicó el principio DRY (Don't Repeat Yourself), el cual contribuyo a la reducción de código duplicado mediante la creación de funciones auxiliares, mientras que el Single Responsibility Principle (SRP) aseguro que cada función se encargue de una única tarea específica. En conjunto, estas prácticas no solo mejoran la calidad del software, sino que también facilitan la identificación y corrección de errores, estableciendo una base sólida para la escalabilidad y flexibilidad futura del sistema.

3.5. Impacto de las soluciones en la mantenibilidad

Los resultados de la aplicación de pautas de mantenibilidad para abordar los defectos de código duplicado y malas prácticas de codificación son altamente positivos. Esto refleja en una significativa disminución de los defectos en el código: el porcentaje de líneas duplicadas ha caído del 56,77% al 3,1%, mientras que las malas prácticas de codificación se han reducido de 6 a 0. Estos avances demuestran la efectividad de las estrategias aplicadas y su impacto en la calidad general del código. Un ejemplo notable es el aumento en la calidad de la documentación del código, que pasó del 20% al 21.60%, como resultado de la efectividad de la corrección de los dos defectos, este incremento resalta la importancia de seguir prácticas de mantenibilidad para mejorar no solo el código en sí, sino también la documentación que lo acompaña.

La corrección de los defectos como "código duplicado" y "malas prácticas de codificación" ha tenido un impacto significativo en la mejora del grado de mantenibilidad general del código. Antes de la aplicación de las pautas de mantenibilidad, el grado de mantenibilidad se situaba en el 73.07%, sin embargo, después de implementar las soluciones y corregir estos defectos, el índice de mantenibilidad aumentó notablemente hasta alcanzar el 87,63%.

Este incremento del 14,56% en el grado de calidad del backend afecta significativamente en grado de mantenibilidad del producto del 82,34% al 89,62%. Los resultados reflejan de manera contundente la efectividad de las pautas de mantenibilidad implementadas, que incluyen principios como

la modularización, la encapsulación y la eliminación de redundancias, demostrando su capacidad para optimizar la estructura y legibilidad del código.

Anexo 20. Prueba de aceptación director del proyecto.

EVALUACION DE ACEPTACION DE LA APLICACIÓN QUALITYCCODE

INTRODUCCION

El presente cuestionario tiene como objetivo la evaluación de aceptación de la aplicación web QualityCCode, la cual se enfoca en verificar la mantenibilidad de aplicaciones web en la carrera de computación de la Universidad Nacional de Loja.

DESCRIPCION

La evaluación se realiza tomando los criterios de aceptación de las historias de usuario para determinar el cumplimiento de las necesidades para el usuario final por parte del director, Ing. Francisco Alvarez Pineda.

Si los criterios de aceptación de las historias de usuario se cumplen en su totalidad se debe marcar con un visto la historia como completada, caso contrario con una X.

EVALUACION

N.	Historia de usuario	¿Completada?	Observación
1	Inicio de sesión	✓	
2	Administrar pauta	✓	
3	Administrar lista de verificación	✓	Ninguna
4	Administrar métrica	✓	
5	Administrar subcaracterística	✓	
6	Administrar proyecto	✓	
7	Administrar prueba de mantenibilidad	✓	Ninguna
8	Ejecutar prueba de mantenibilidad	✓	
9	Informe lista de verificación	✓	
10	Generar PDF	✓	

11	Calcular grado de mantenibilidad	✓	Ninguna

Se adjunta la firma de los participantes:

Firmado electrónicamente por:
FRANCISCO JAVIER ALVAREZ PINEDA

F: _____

Ing. Francisco J Álvarez Pineda Mg. Sc

Director del Proyecto

Firmado electrónicamente por:
GILSON ORLANDO QUEZADA GUARTIZACA

F: _____

Gilson Orlando Quezada

Autor

Anexo 21. Cuestionario para medir la aceptación de la aplicación web

El propósito de la encuesta es evaluar la aceptación de la aplicación web diseñada para verificar la mantenibilidad del software, conforme a la norma ISO/IEC 25000. La herramienta es parte del itinerario de Ingeniería de Software de la Carrera de Computación de la Universidad Nacional de Loja.

En el siguiente enlace se muestra el formulario del cuestionario: https://forms.gle/qnDbJsgWKHjn5mUT6.

1. ¿Considera que el software es sencillo y fácil de usar?
2. ¿Encuentra intuitiva la interfaz de pruebas para verificar la mantenibilidad de aplicaciones web?
3. ¿Cree que el software proporciona instrucciones claras para realizar el análisis de código estático?
4. ¿Encuentra intuitiva la interfaz que resuelve el cálculo del grado de mantenibilidad de aplicaciones web?
5. ¿La navegación entre cada sección del software es sencilla?
6. ¿El software proporciona datos relevantes sobre los resultados de las pruebas destinadas para verificar la mantenibilidad de aplicaciones web?
7. ¿El PDF generado proporciona información relevante sobre los resultados de las pruebas destinadas para verificar la mantenibilidad de aplicaciones web?
8. ¿Considera que el software permite conocer pautas de mantenibilidad y propiedades de calidad para la solución de defectos de aplicaciones web?
9. ¿Considera el software una herramienta útil para verificar la mantenibilidad de las aplicaciones web?
10. ¿El software posee tiempos de respuesta buenos?

Por medio de la presente, se confirma que las preguntas para la implementación de pruebas de aceptación para el Trabajo de Integración Curricular, titulado "Evaluación de mantenibilidad para aplicaciones web en producción de los laboratorios de la carrera de computación de la UNL", ha sido revisada y aprobada por el director del proyecto, el Ing. Francisco J Álvarez-Pineda Mg.Sc.

Se adjunta las firmas de los participantes.

F: _____

Ing. Francisco J Álvarez-Pineda Mg. Sc

DIRECTO TRABAJO DE

INTEGRACION CURRICULAR

F: _____

Gilson Orlando Quezada Guartizaca

AUTOR

Anexo 22. Resumen de la encuesta aplicada

En la **Figura 1-A22** se muestra el resumen de la encuesta aplicada para obtener el nivel de aceptación de la aplicación web QualityCCode, , mismos que pueden ser accedidos en el siguiente enlace: https://docs.google.com/spreadsheets/d/18OOUYjb0C796KWSAFO1aczcPsp0VboLj4XGpYvfPzpo/edit?usp=sharing.

Marca temporal	Dirección de correo electrónico	1. ¿Considera que el software es sencillo y fácil de usar?	2. ¿Encuentra intuitiva la interfaz de pruebas para verificar la mantenibilidad	3. ¿Cree que el software proporciona instrucciones claras para realizar el análisis de	4. ¿Encuentra intuitiva la interfaz que resuelve el cálculo del grado de mantenibilidad	5. ¿La navegación entre cada sección del software es sencilla?	6. ¿El software proporciona datos relevantes sobre los resultados de las pruebas destinadas para verificar la mantenibilidad de	7. ¿El PDF generado proporciona información relevante sobre los resultados de las pruebas destinadas	8. ¿Considera que el software permite conocer pautas de mantenibilidad y propiedades de calidad para la solución de	9. ¿Considera el software una herramienta útil para verificar la mantenibilidad	10. ¿El software posee tiempos de respuesta buenos?
9/23/2024 9:51:39	wagner.castillo@unl.edu.ec	En parte	En parte	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
9/23/2024 10:04:20	edinson.quizhpe@unl.edu.ec	Sí	Sí	Sí	En parte	Sí	En parte	En parte	Sí	Sí	En parte
9/23/2024 10:06:14	jack.rojas@unl.edu.ec	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
9/23/2024 10:13:41	victor.a.jimenez@unl.edu.ec	Sí	En parte	Sí	Sí	En parte	Sí	Sí	Sí	Sí	Sí
9/23/2024 10:20:57	vladimir.celi@unl.edu.ec	Sí	Sí	Sí	En parte	Sí	Sí	Sí	Sí	Sí	Sí
9/23/2024 11:00:44	jorge.poma@unl.edu.ec	Sí	Sí	En parte	Sí	En parte	Sí	Sí	Sí	En parte	Sí
9/23/2024 11:10:29	lenin.yanangomez@unl.edu.ec	Sí	Sí	Sí	En parte	Sí	Sí	Sí	Sí	Sí	En parte
9/23/2024 11:18:02	jhandry.santorum@unl.edu.ec	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
9/23/2024 11:35:39	fsriosg@unl.edu.ec	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
9/23/2024 11:37:03	danny.a.martinez@unl.edu.ec	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	En parte
9/23/2024 13:41:16	jean.x.agreda@unl.edu.ec	Sí	En parte	En parte	Sí	Sí	En parte	Sí	En parte	Sí	En parte
9/23/2024 13:43:06	yeurenah@unl.edu.ec	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
9/23/2024 14:48:07	rubier.padilla@unl.edu.ec	En parte	Sí	Sí	En parte	Sí	Sí	En parte	En parte	En parte	En parte
9/23/2024 17:13:30	byron.herrera@unl.edu.ec	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
9/23/2024 17:16:25	cristian.e.armijos@unl.edu.ec	Sí	Sí	En parte	En parte	Sí	Sí	Sí	En parte	Sí	Sí
9/23/2024 18:23:57	kevin.sarango@unl.edu.ec	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
9/23/2024 20:03:54	kevin.j.jaramillo@unl.edu.ec	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
9/23/2024 21:12:34	jandry.hernandez@unl.edu.ec	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
9/23/2024 21:23:30	jorge.l.cevallos@unl.edu.ec	Sí	Sí	Sí	Sí	Sí	Sí	Sí	En parte	Sí	Sí
9/23/2024 21:29:28	israel.calva@unl.edu.ec	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
9/23/2024 22:42:21	lourdes.vinamagua@unl.edu.ec	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
9/24/2024 0:12:29	roy.leon@unl.edu.ec	En parte	En parte	Sí	En parte	En parte	Sí	Sí	En parte	Sí	Sí
9/24/2024 0:31:56	luis.villalta@unl.edu.ec	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
9/24/2024 2:11:03	jose.villavicencio@unl.edu.ec	Sí	Sí	En parte	En parte	Sí	Sí	Sí	Sí	Sí	Sí
9/24/2024 6:17:45	alexander.canar@unl.edu.ec	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí

Figura 1-A22 Encuesta aplicada para obtener el nivel de aceptación de QualityCCode.

Anexo 23. Implementación de tareas de la fase de codificación del sistema

A continuación, se muestra la implementación de tareas para el proceso de codificación de la aplicación web, cada iteración se organiza de manera eficiente y asegura que cada aspecto del desarrollo sea abordado adecuadamente.

- **Tarea 1: Inicio de sesión**

  - **Diseño**

En la **Figura 1-A23** se muestra el diseño para el inicio de sesión, que brinda el autenticador centralizado de la Carrera de Computación.

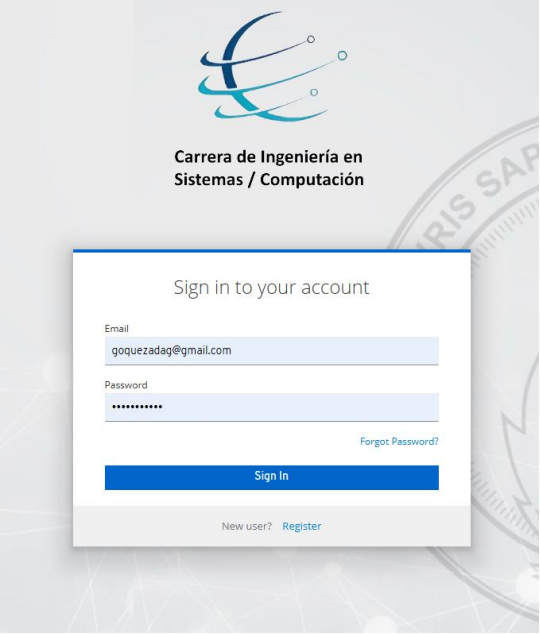


Figura 1-A23. Diseño de inicio de sesión.

  - **Codificación**

En la **Figura 2-A23** se muestra la iniciación de las funciones para administrar el inicio de sesión utilizando keycloak, la codificación del inicio de sesión no se muestra por cuestiones de privacidad. En este caso de muestra configuración de variables del cliente y el estado de autenticación del usuario mediante el hook llamado useAuth y también la función de logout para cerra la sesión y restablecer el estado.

useAuth.tsx

src > hooks > useAuth.tsx > ...

```tsx
import { useState, useEffect, useRef, useCallback } from "react";
import KeycloakConfig from "keycloak-js";

// Configuración del cliente de Keycloak
const client = new KeycloakConfig({
    url: import.meta.env.VITE_KEYCLOAK_URL,
    realm: import.meta.env.VITE_KEYCLOAK_REALM,
    clientId: import.meta.env.VITE_KEYCLOAK_CLIENT,
});

// Tipos para el estado de autenticación
interface AuthData {
    isLogin: boolean;
    userId: string | null;
    roles: string[];
}

const useAuth = () => {
    const isGo = useRef(false);
    const [authData, setAuthData] = useState<AuthData>({
        isLogin: false,
        userId: null,
        roles: []
    });
```

Figura 2-A23. Iniciación de las funciones para administrar el inicio de sesión utilizando keycloak.

- **Pruebas**

Para el inicio de sesión se registró un caso de prueba unitaria, como se detalla en la **Tabla 1-A23**, la misma que tiene su respectiva la ejecución.

Tabla 1-A23. Prueba unitaria inicio de sesión PU-01.

Prueba Unitaria – PU-01			
Número: PU-01		**Versión**: 1.0	
Componente evaluado: Caso de uso: Inicio de sesión			
Identificador: Inicio de sesión		**Historia de usuario**: HU01	
Caso de prueba: Verificación del flujo de inicio de sesión mediante Keycloak			
Datos de entrada	**Descripción**	**Salida esperada**	**Resultado**
Body(x-www-form-urlencoded) grant_type: password client_id: myappclient username: {} password: {]	Verificar si el usuario puede iniciar sesión correctamente.	El servidor debe devolver un código de estado 200 con un Access_token en la respuesta.	Aceptado

Referencia de historia de usuario (RHU).

En la **Figura 3-A23** se muestra el objeto json de prueba y la API para verificar el inicio de sesión.

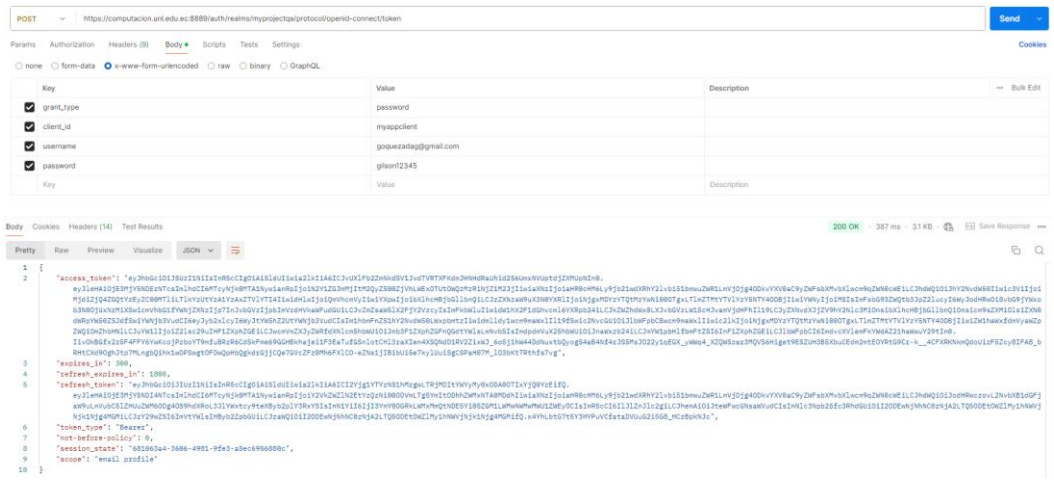


Figura 3-A23. Objeto json de prueba y la API para verificar el inicio de sesión.

- **Tarea 2: Administrar pauta**

  - **Diseño**

En la **Figura 4-A23** se muestra el diseño para registrar una nueva pauta de mantenibilidad.

Figura 4-A23. Diseño, crear pauta de mantenibilidad.

En la **Figura 5-A23** se muestra el diseño para actualizar y eliminar una nueva pauta de mantenibilidad.

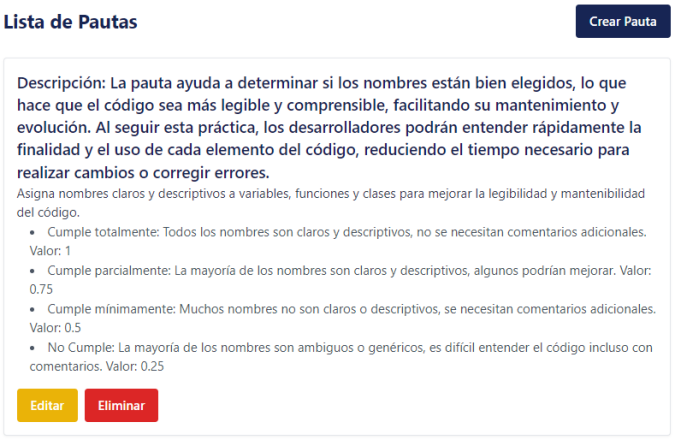


Figura 5-A23. Diseño, actualizar y eliminar una pauta de mantenibilidad.

Los casos de pruebas están relacionados a cada uno de los diseños **Figura 4-A23** y **5-A23**, cuyo objetivo es verificar el modelo para crear y llevar a cabo modificación, eliminación y obtener datos.

- **Codificación**

En la **Figura 6-A23** se destaca las funciones para registrar una nueva pauta, obtener los registros de todas las pautas y en la **Figura 7-A23** las funciones para permitir la actualización de los datos existentes de una pauta, así como su eliminación.

PautaController.js

src > controllers > PautaController.js > ...

```
1   const express = require('express');
2   const router = express.Router();
3   const Pauta = require('../models/Pauta');
4
5   // Create a new Pauta
6   router.post('/', async (req, res) => {
7       try {
8           const newPauta = new Pauta(req.body);
9           const pauta = await newPauta.save();
10          res.json(pauta);
11      } catch (err) {
12          res.status(500).send(err.message);
13      }
14  });
15
16  // Get all Pautas
17  router.get('/', async (req, res) => {
18      try {
19          const pautas = await Pauta.find();
20          res.json(pautas);
21      } catch (err) {
22          res.status(500).send(err.message);
23      }
24  });
```

Figura 6-A23. Código para crear y obtener las pautas de mantenibilidad.

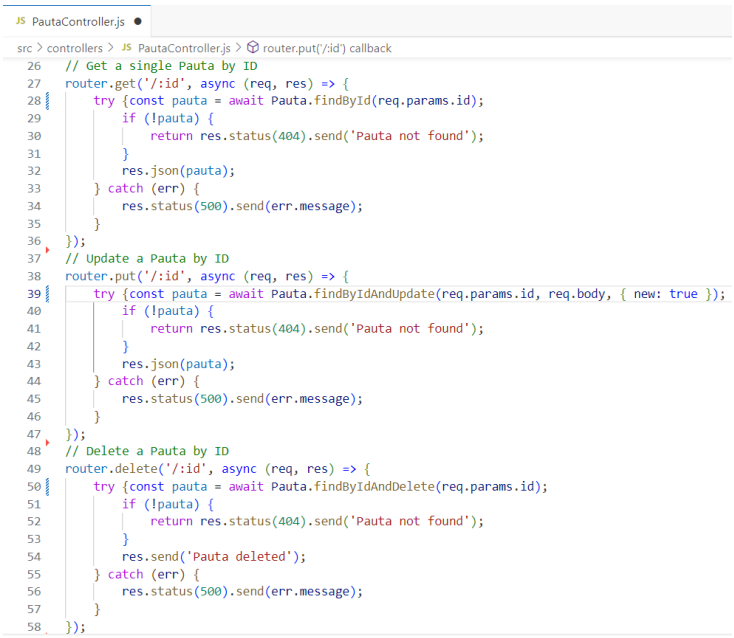


Figura 7-A23. Código para actualizar y elimina una pauta de mantenibilidad.

- **Pruebas**

Para las pruebas unitarias se registró dos casos de prueba, presentando la ejecución de manera más detallada.

La **Tabla 2-A23** se detalla el caso de prueba para verificar el modelo pauta, mediante el escenario crear pauta.

Tabla 2-23. Prueba unitaria administrar pauta de mantenibilidad- PU-02.

Prueba Unitaria – PU-02			
Número: PU-02		**Versión**: 1.0	
Componente evaluado: Caso de uso: Administrar pauta			
Identificador: Administrar pauta		**Historia de usuario**: HU02	
Caso de prueba: Verificación del modelo Pauta			
Datos de entrada	**Descripción**	**Salida esperada**	**Resultado**
Json pauta	Verificar si el modelo Pauta se crea correctamente	Las instancias del modelo Pauta deben crearse correctamente y verificar el estado del registro.	Aceptado

Referencia de historia de usuario (RHU).

En la **Figura 8-A23** se muestra la API y el objeto json de prueba para crear el modelo pauta.

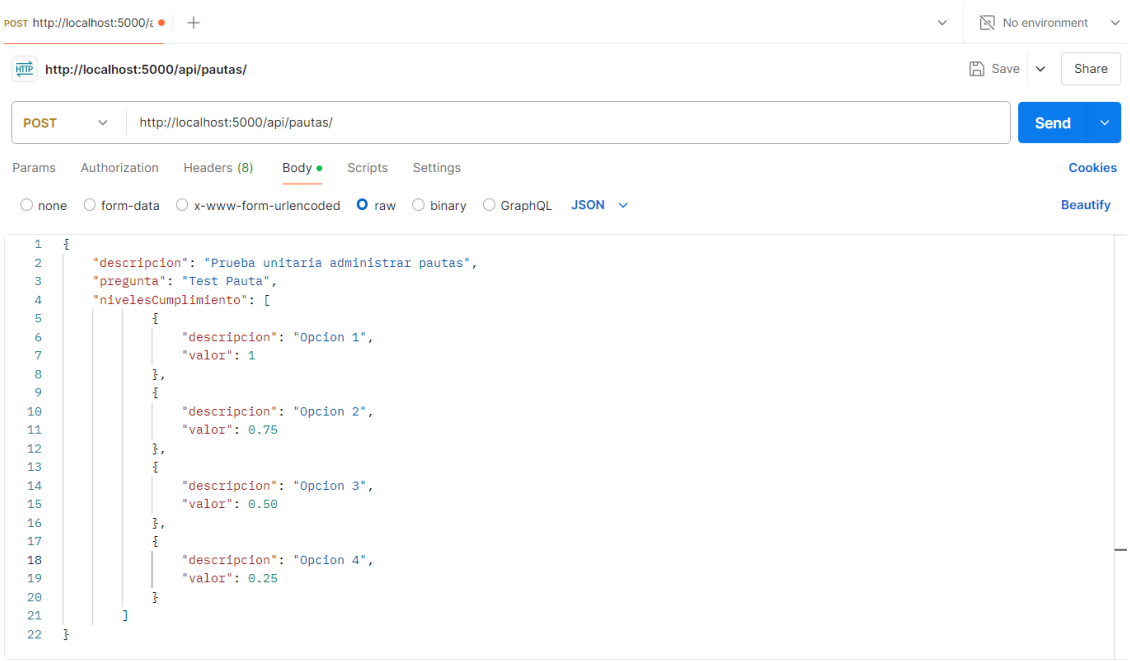


Figura 8-A23. Api y objeto json Pauta para crear un modelo pauta - PU-02.

En la **Figura 9-A23** se muestra la verificación del modelo pauta, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

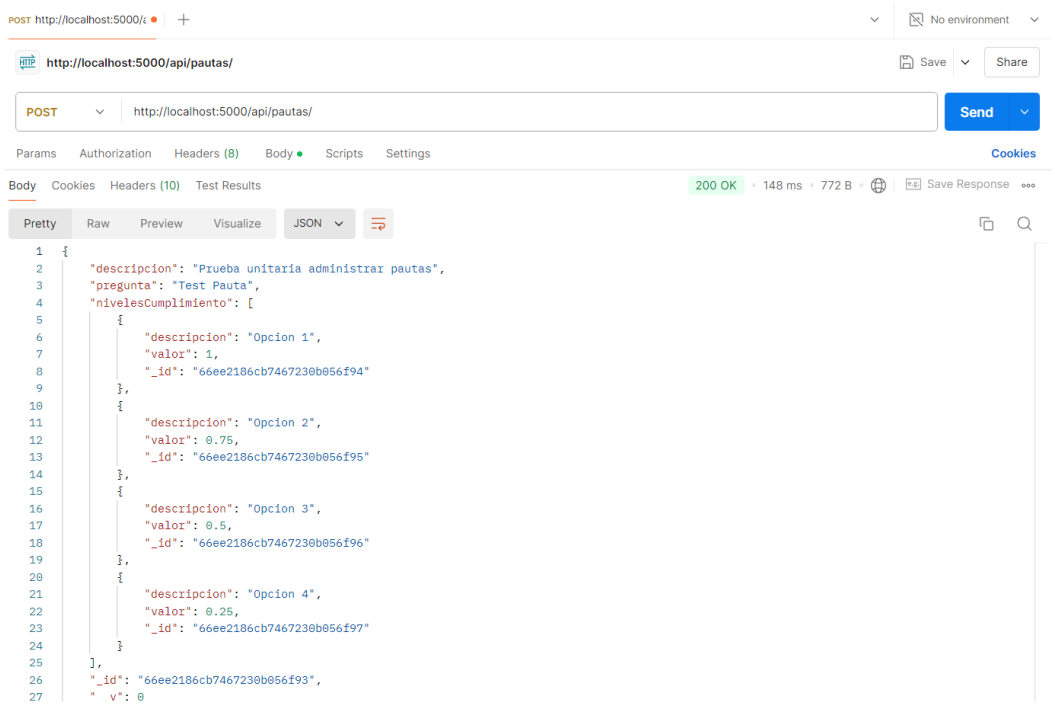


Figura 9-A23. Verificación del modelo pauta - PU-02.

En la **Figura 10-A23** se muestra la pauta de mantenibilidad registrada, desde la ejecución del diseño del frontend.

Figura 10-A23. Verificación del registro pauta - PU-02.

La **Tabla 3-A23** se detalla el caso de prueba del modelo para llevar a cabo modificaciones, eliminaciones y obtener datos.

Tabla 3-A23. Prueba unitaria administrar pauta de mantenibilidad- PU-03.

Prueba Unitaria – PU-03			
Número: PU-03		**Versión**: 1.0	
Componente evaluado: Caso de uso: Administrar pauta			
Identificador: Administrar pauta		**Historia de usuario**: HU02	
Caso de prueba: Modificación, eliminación y obtención de datos			
Datos de entrada	**Descripción**	**Salida esperada**	**Resultado**
Modelo pauta	Verificar si se efectúan los métodos de modificación, eliminación y listar los datos de los registros del modelo de pauta	Obtener los datos de cada uno de los registros para tener la opción de eliminarlos o modificarlos	Aceptado

Referencia de historia de usuario (RHU).

En la **Figura 11-A23** se muestra la API para listar los datos de los registros del modelo pauta.

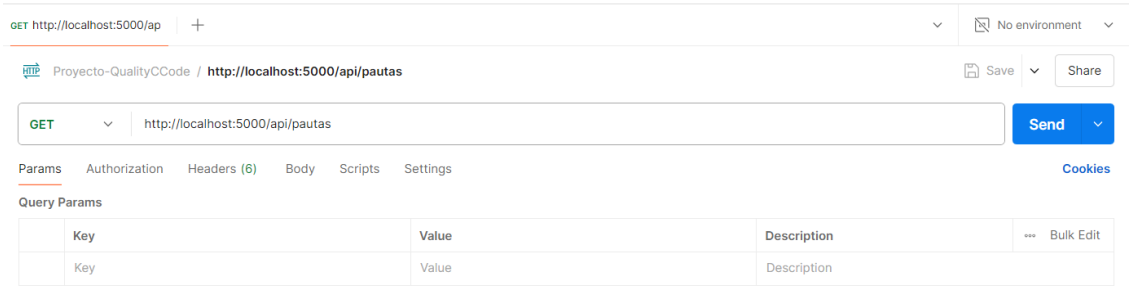


Figura 11-A23. Api para listar los datos de los registros del modelo pauta - PU-03.

En la **Figura 12-A23** se muestra la verificación de listar los registros de pautas, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

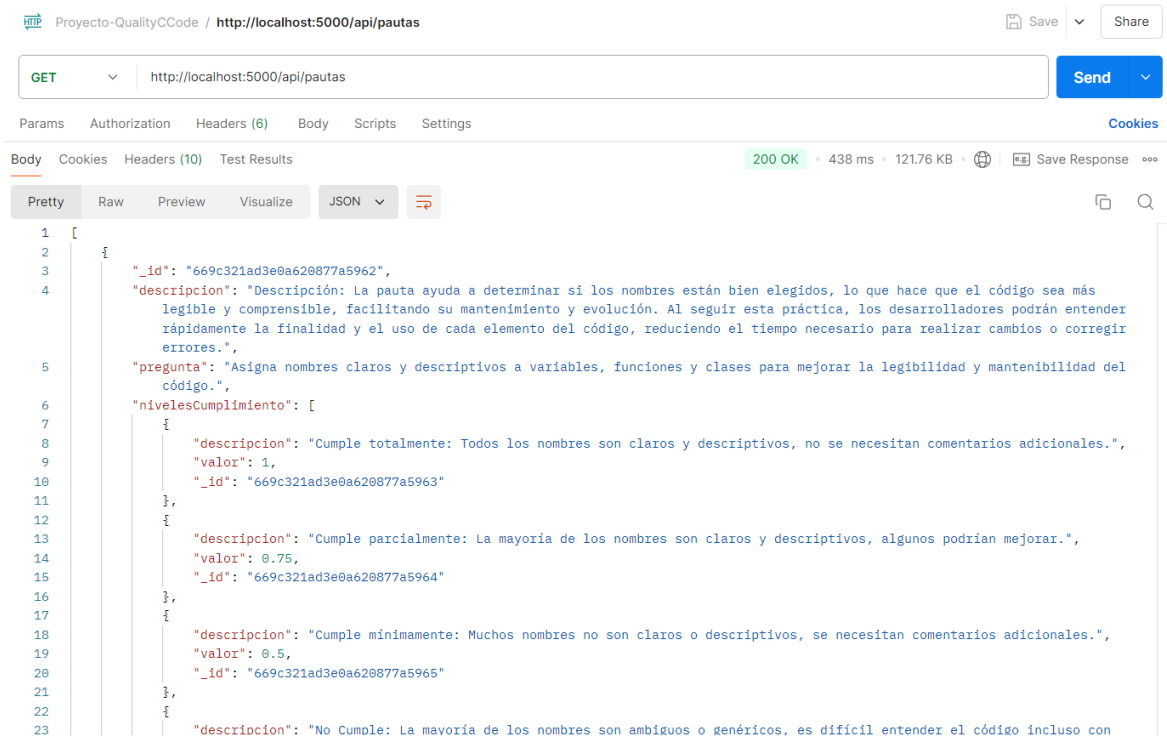


Figura 12-A23. Verificación de listar los registros de pautas - PU-03.

En la **Figura 13-A23** se muestra la API en la cual establece la variable Id para modificar los datos y eliminar un registro del modelo pauta.

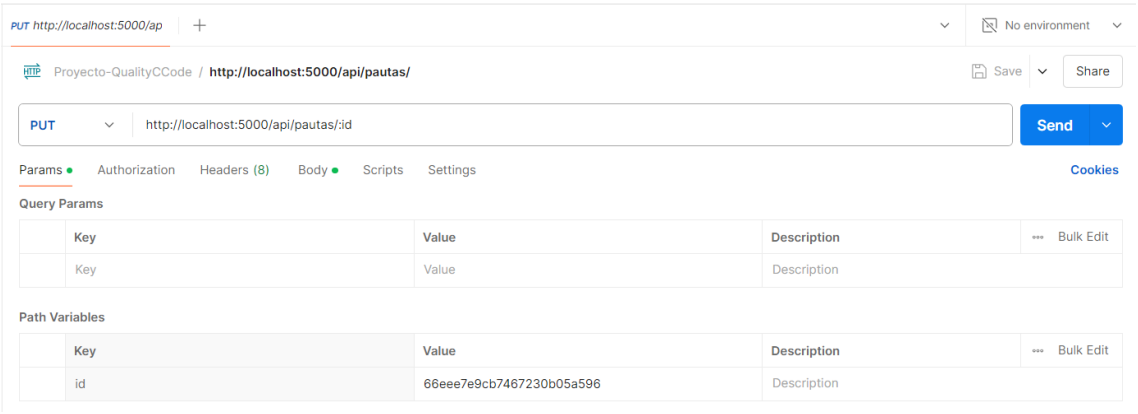


Figura 13-A23. Variable Id para modificar y eliminar un registro del modelo pauta - PU-03.

En la **Figura 14-A23** se muestra la API y el objeto Json para modificar los datos de un registro modelo pauta.

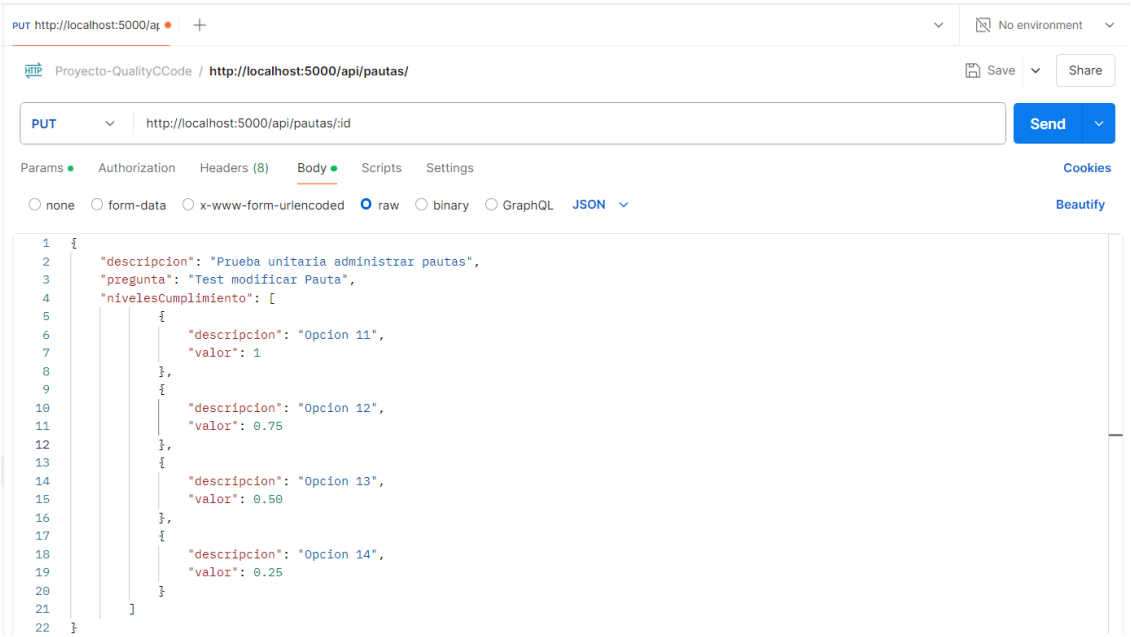


Figura 14-A23. API y el objeto Json para modificar los datos de un registro del modelo pauta - PU-03.

En la **Figura 15-A23** se muestra la verificación de modificar un registro pauta, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

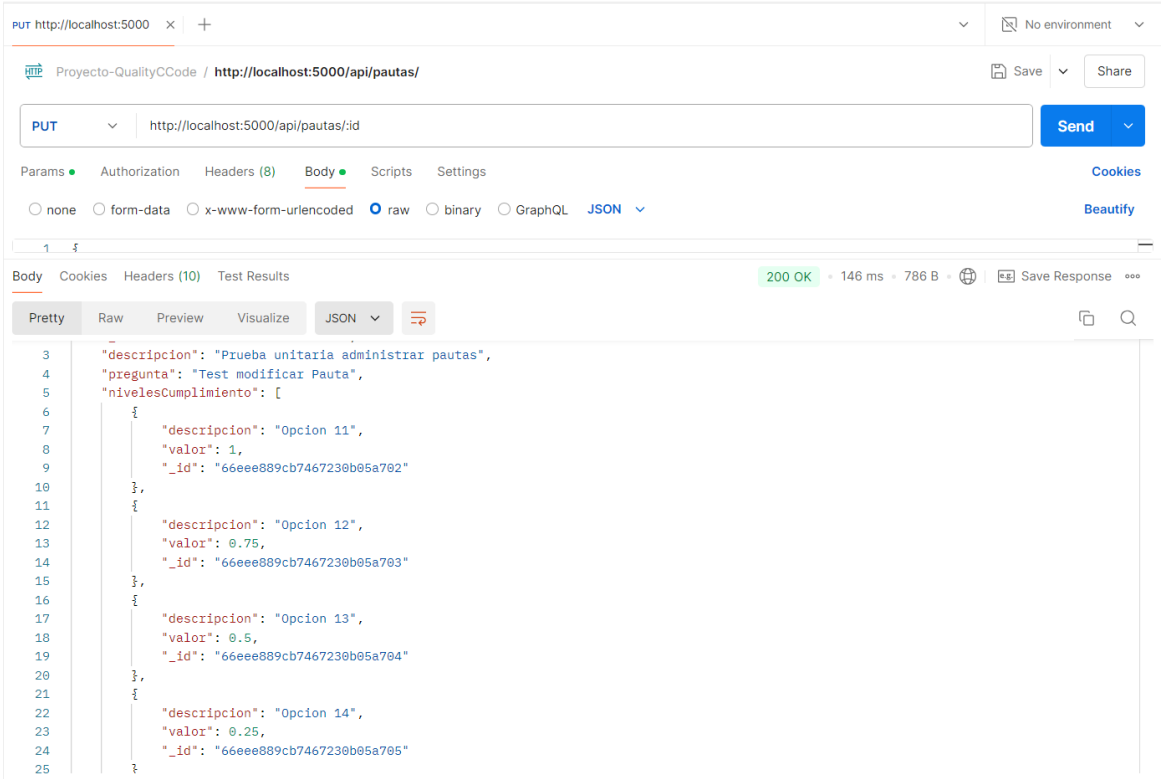


Figura 15-A23. Verificación de modificar un registro del modelo pauta - PU-03.

En la **Figura 16-A23** se muestra la pauta de mantenibilidad registrada en la PU-02, en ejecución del diseño del frontend se ha actualizado correctamente.

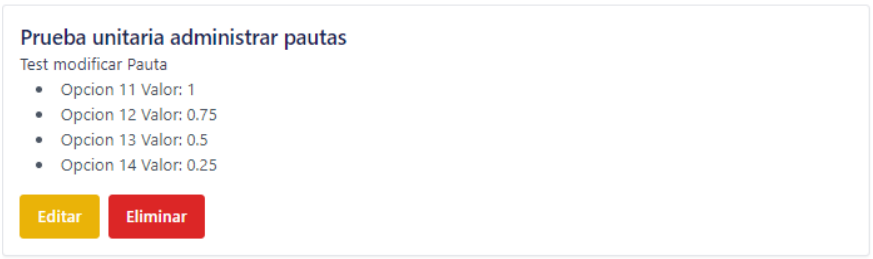


Figura 16-A23. Verificación de la modificación del registro pauta - PU-03.

En la **Figura 17-A23** se muestra la API para eliminar un registro modelo pauta.

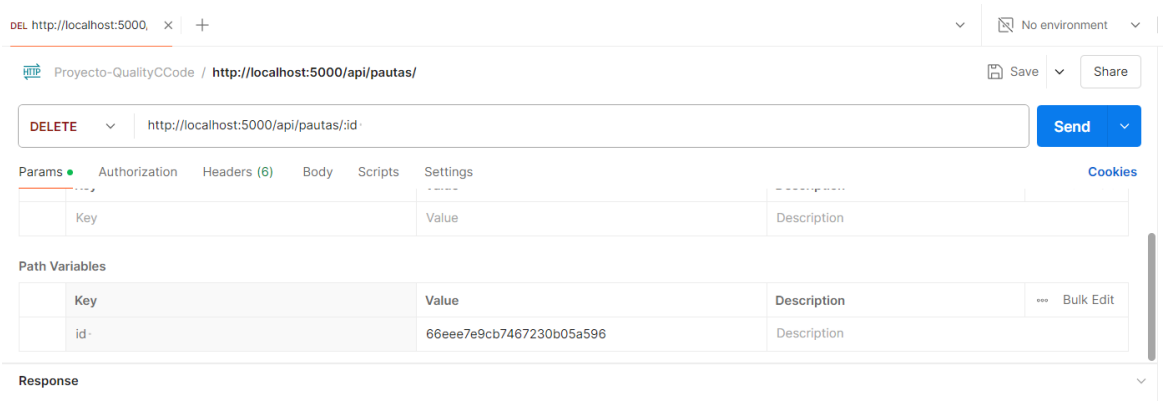


Figura 17-A23. API para eliminar un registro del modelo pauta - PU-03.

En la **Figura 18-A23** se muestra la verificación de eliminar un registro pauta, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

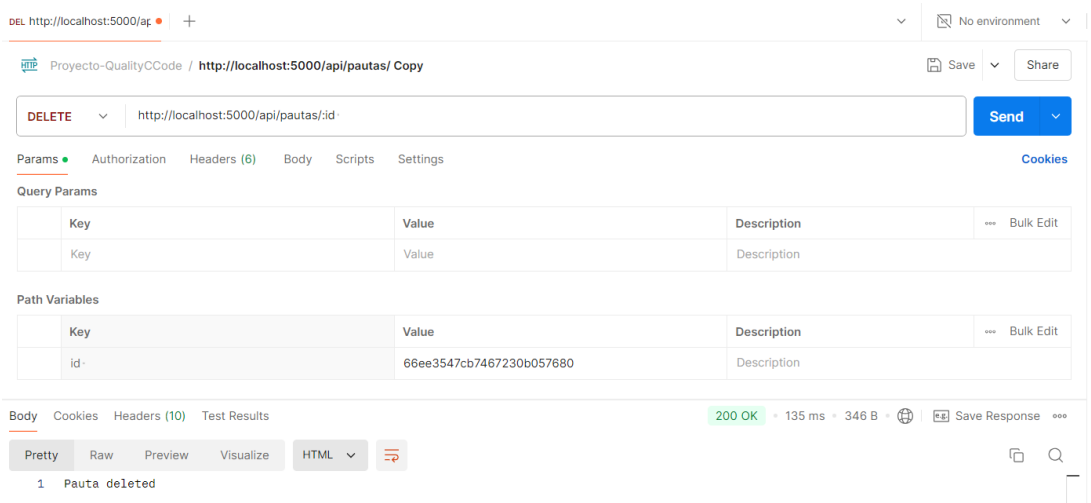


Figura 18-A23. Verificación de eliminar un registro del modelo pauta - PU-03.

- **Tarea 3: Administrar lista de verificación**

  - **Diseño**

En la **Figura 19-A23** se muestra el diseño para registrar una nueva lista de verificación**.**

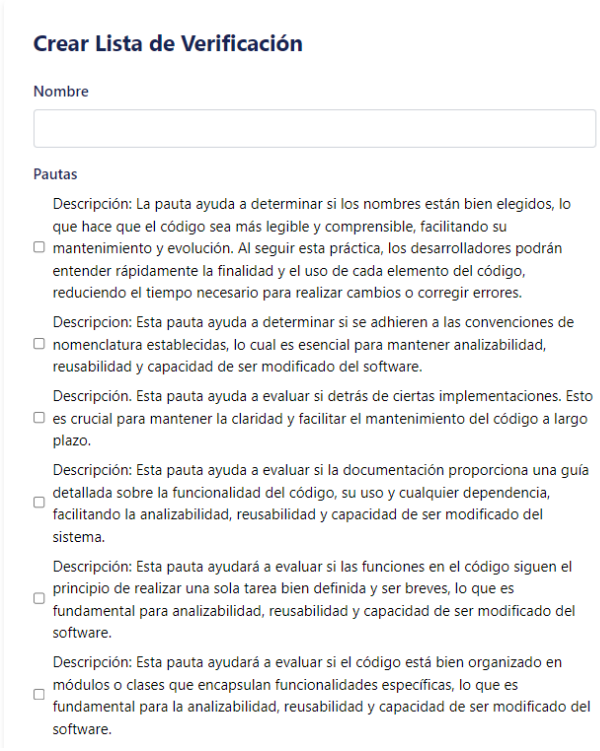


Figura 19-A23. Diseño, crear lista de verificación.

En la **Figura 20-A23** se muestra el diseño para actualizar y eliminar una nueva lista de verificación.

Figura 20-A23. Diseño, actualizar y eliminar una lista de verificación.

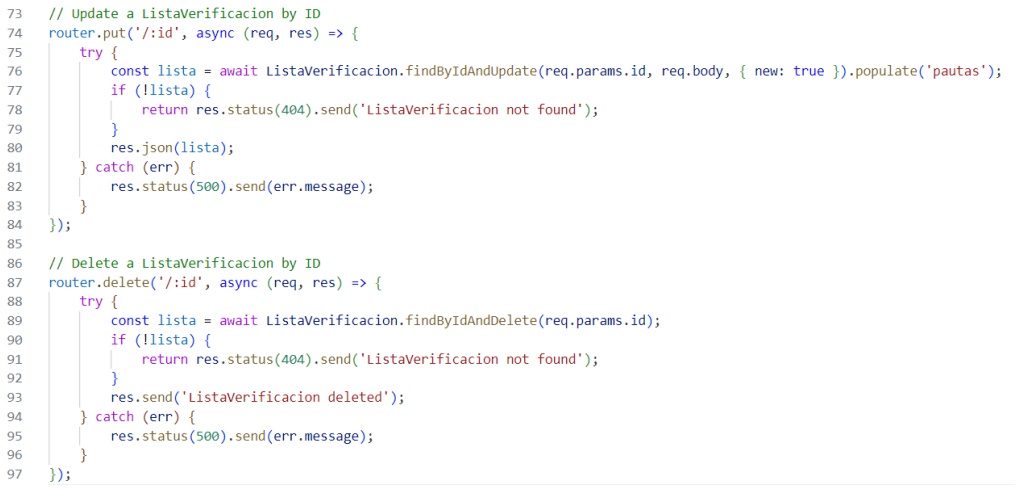
Los casos de prueba están relacionados a cada uno de los diseños **Figura 19-A23 y 20-A23**, cuyo objetivo es verificar el modelo para crear y llevar a cabo modificación, eliminación y obtener datos.

- **Codificación**

En la **Figura 21-A23** se destaca las funciones para registrar una nueva lista de verificación, obtener los registros de todas las listas de verificación y en la **Figura 22-A23** las funciones para permitir la actualización de los datos existentes de una lista, así como su eliminación.

```
JS ListaVerificacionController.js ×
src > controllers > JS ListaVerificacionController.js > router.post('/') callback
1   const express = require('express');
2   const router = express.Router();
3   const ListaVerificacion = require('../models/ListaDeVerificacion');
4   const mongoose = require('mongoose');
5
6   // Create a new ListaVerificacion
7   router.post('/', async (req, res) => {
8       try {
9           const newLista = new ListaVerificacion(req.body);
10          const lista = await newLista.save();
11          res.json(lista);
12      } catch (err) {
13          res.status(500).send(err.message);
14      }
15  });
16
17  // Get all ListasVerificacion
18  router.get('/', async (req, res) => {
19      try {
20          const listas = await ListaVerificacion.find().populate('pautas');
21          res.json(listas);
22      } catch (err) {
23          res.status(500).send(err.message);
24      }
25  });
```

Figura 21-A23. Código para crear y obtener las listas de verificación.



```
73  // Update a ListaVerificacion by ID
74  router.put('/:id', async (req, res) => {
75      try {
76          const lista = await ListaVerificacion.findByIdAndUpdate(req.params.id, req.body, { new: true }).populate('pautas');
77          if (!lista) {
78              return res.status(404).send('ListaVerificacion not found');
79          }
80          res.json(lista);
81      } catch (err) {
82          res.status(500).send(err.message);
83      }
84  });
85
86  // Delete a ListaVerificacion by ID
87  router.delete('/:id', async (req, res) => {
88      try {
89          const lista = await ListaVerificacion.findByIdAndDelete(req.params.id);
90          if (!lista) {
91              return res.status(404).send('ListaVerificacion not found');
92          }
93          res.send('ListaVerificacion deleted');
94      } catch (err) {
95          res.status(500).send(err.message);
96      }
97  });
```

Figura 22-A23. Código para actualizar y elimina una lista de verificación.

- **Pruebas**

Para las pruebas unitarias se registró dos casos de prueba con la respectiva ejecución de manera detallada.

La **Tabla 4-A23** se detalla el caso de prueba para verificar el modelo lista de verificación, mediante el escenario crear lista de verificación.

Tabla 4-A23. Prueba unitaria administrar lista de verificación- PU-04.

Prueba Unitaria – PU-04			
Número: PU-04		**Versión**: 1.0	
Componente evaluado: Caso de uso: Administrar lista de verificación			
Identificador: Administrar lista de verificación		**Historia de usuario**: HU03	
Caso de prueba: Verificación del modelo Lista de verificación			
Datos de entrada	**Descripción**	**Salida esperada**	**Resultado**
Json lista de verificación	Verificar si el modelo Lista de verificación se crea correctamente	Las instancias del modelo lista de verificación deben crearse correctamente y verificar el estado del registro.	Aceptado

Referencia de historia de usuario (RHU).

En la **Figura 23-A23** se muestra la API y el objeto json de prueba para crear el modelo lista de verificación, al registrar una lista es necesario agregar una pauta de mantenibilidad para ello se hace uso el modelo pauta implementado en la PU-02.

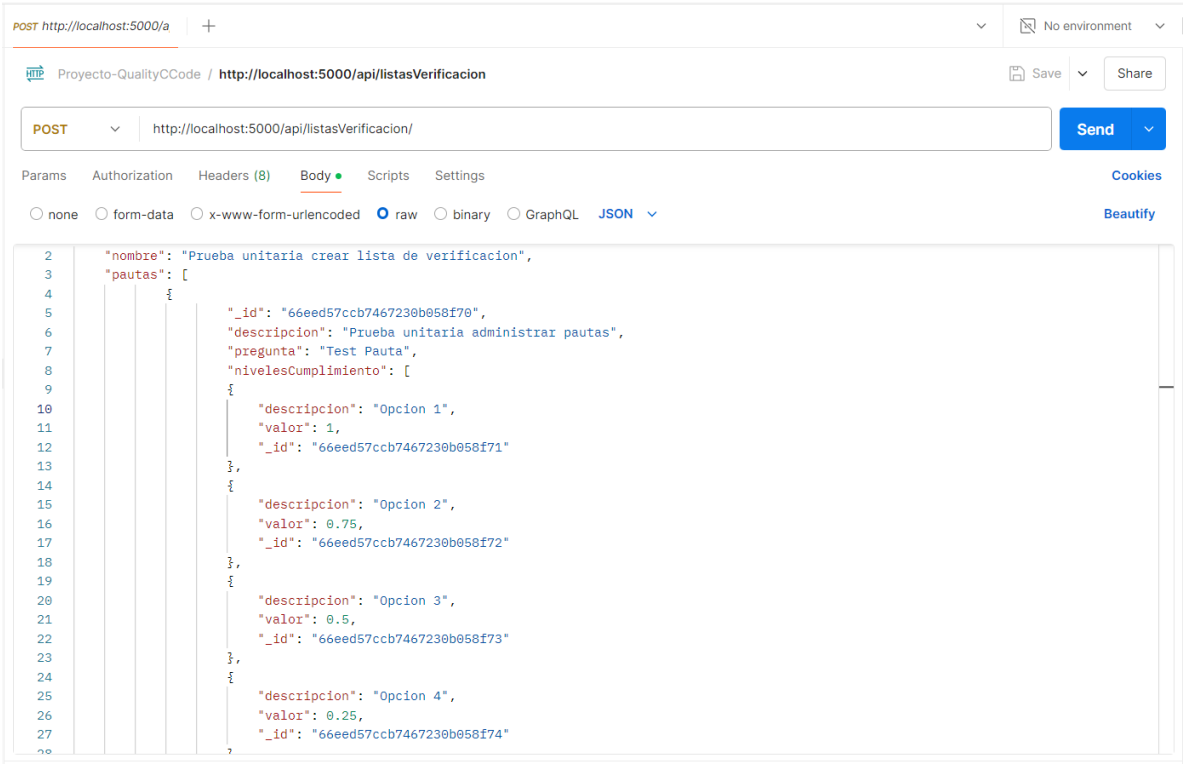


Figura 23-A23. Api y objeto json Pauta para crear un modelo lista de verificación - PU-04.

En la **Figura 24-A23** se muestra la verificación del modelo lista de verificación, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

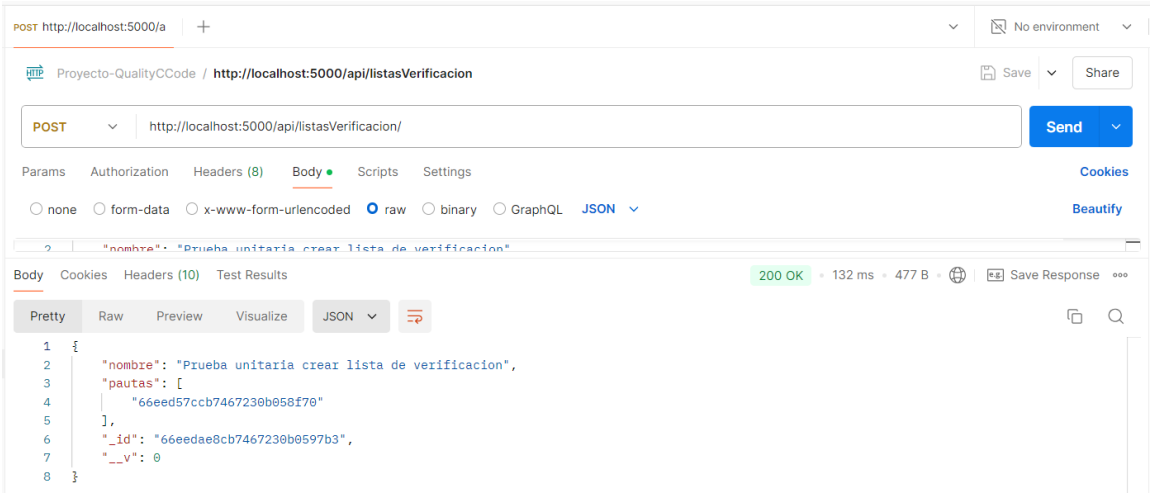


Figura 24-A23. Verificación del modelo lista de verificación - PU-04.

En la **Figura 25-A23** se muestra la lista de verificación, mediante la ejecución del diseño del frontend.

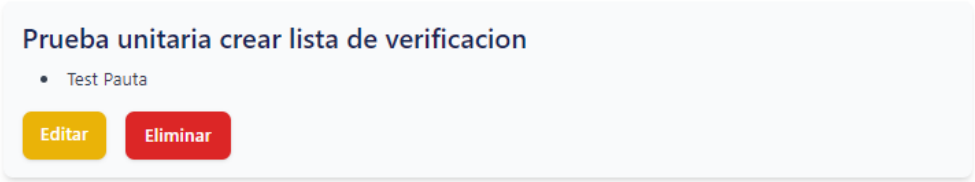


Figura 25-A23. Verificación del registro lista de verificación - PU-04.

La **Tabla 5-A23** se detalla el caso de prueba del modelo para llevar a cabo modificaciones, eliminaciones y obtener datos.

Tabla 5-A23. Prueba unitaria administración lista de verificación- PU-05.

Prueba Unitaria – PU-05			
Número: PU-05		**Versión**: 1.0	
Componente evaluado: Caso de uso: Administrar lista de verificación			
Identificador: Administrar lista de verificación		**Historia de usuario**: HU03	
Caso de prueba: Modificación, eliminación y obtención de datos			
Datos de entrada	**Descripción**	**Salida esperada**	**Resultado**
Modelo pauta. Modelo lista de verificación.	Verificar si se efectúan los métodos de modificación, eliminación y listar los datos de los registros del modelo de pauta	Obtener los datos de cada uno de los registros para tener la opción de eliminarlos o modificarlos	Aceptado

Referencia de historia de usuario (RHU).

En la **Figura 26-A23** se muestra la API para listar los datos de los registros del modelo lista de verificación.

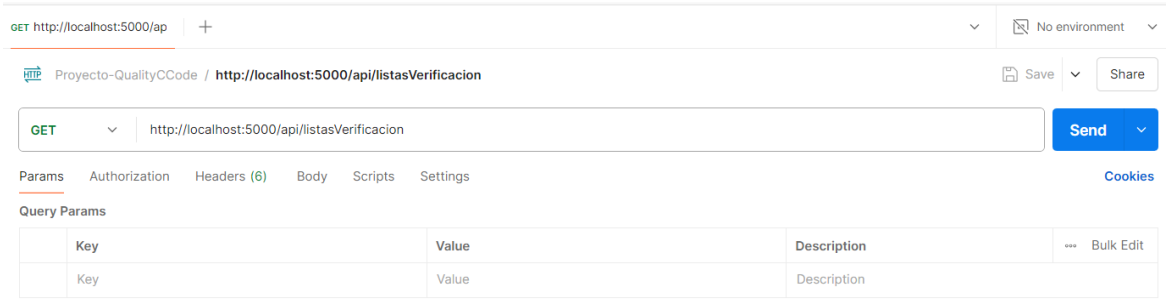


Figura 26-A23. Api para listar los datos de los registros del modelo pauta - PU-05.

En la **Figura 27-A23** se muestra la verificación de listar los registros de las listas de verificación, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

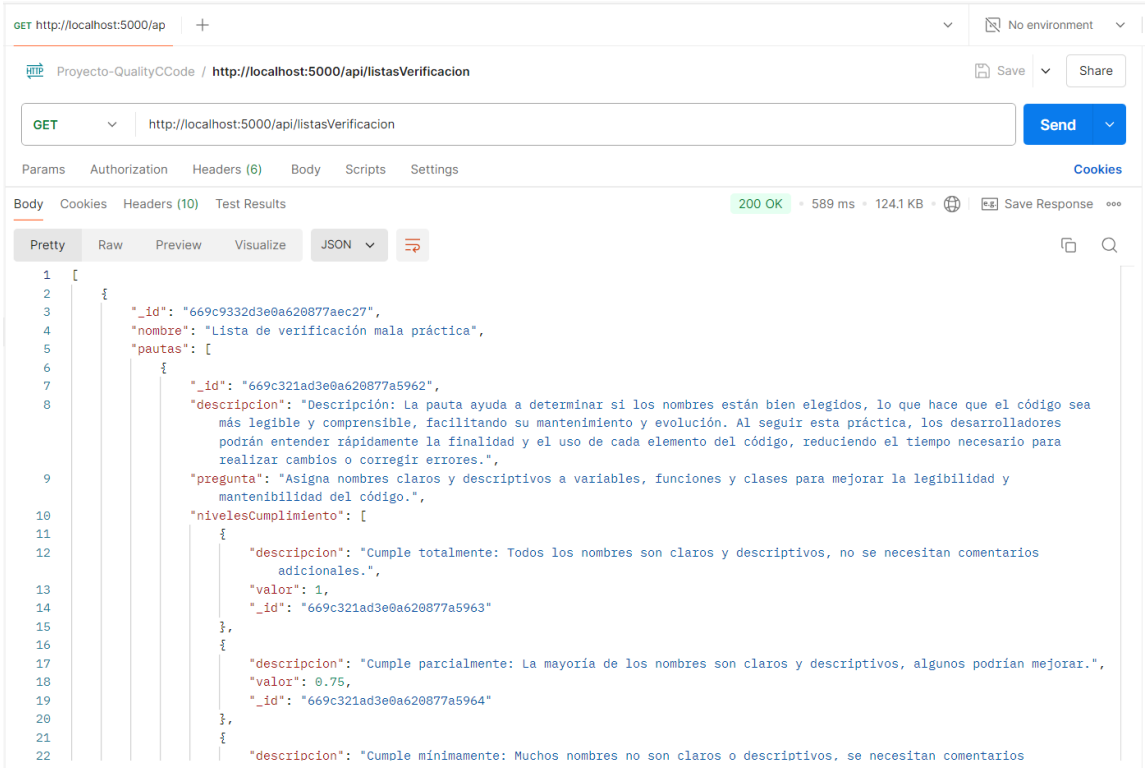


Figura 27-A23. Verificación de listar los registros de pautas - PU-05.

En la **Figura 28-A23** se muestra la API en la cual establece la variable Id para modificar los datos y eliminar un registro del modelo lista de verificación.

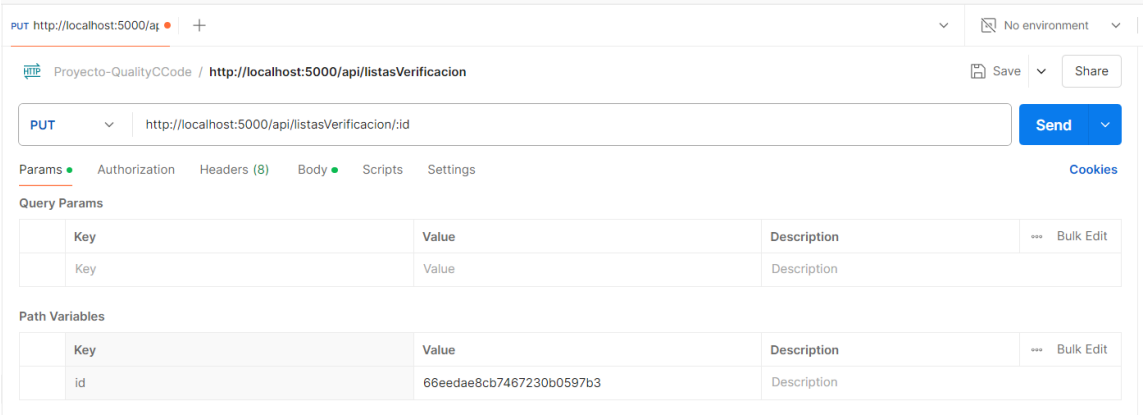


Figura 28-A23. Variable Id para modificar y eliminar un registro del modelo lista de verificación - PU-05.

En la **Figura 29-A23** se muestra la API y el objeto Json para modificar los datos de un registro modelo lista de verificación.

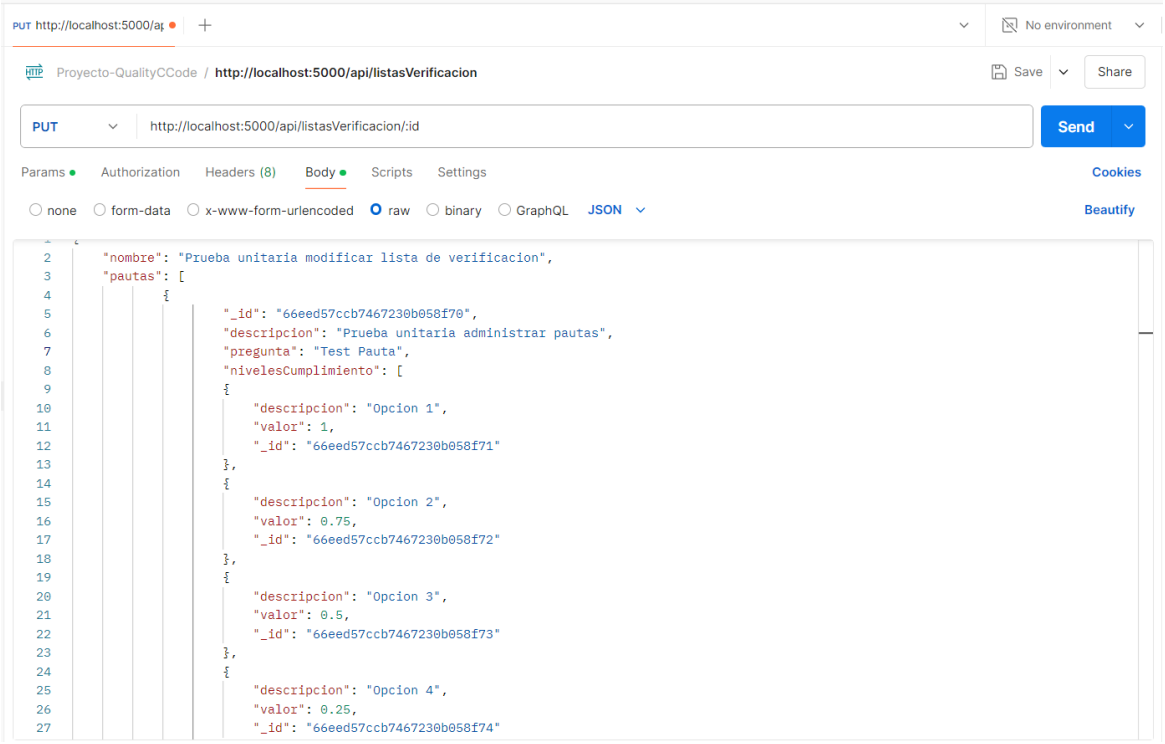


Figura 29-A23. API y el objeto Json para modificar la modelo lista de verificación - PU-05.

En la **Figura 30-A23** se muestra la verificación de modificar un registro lista de verificación, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

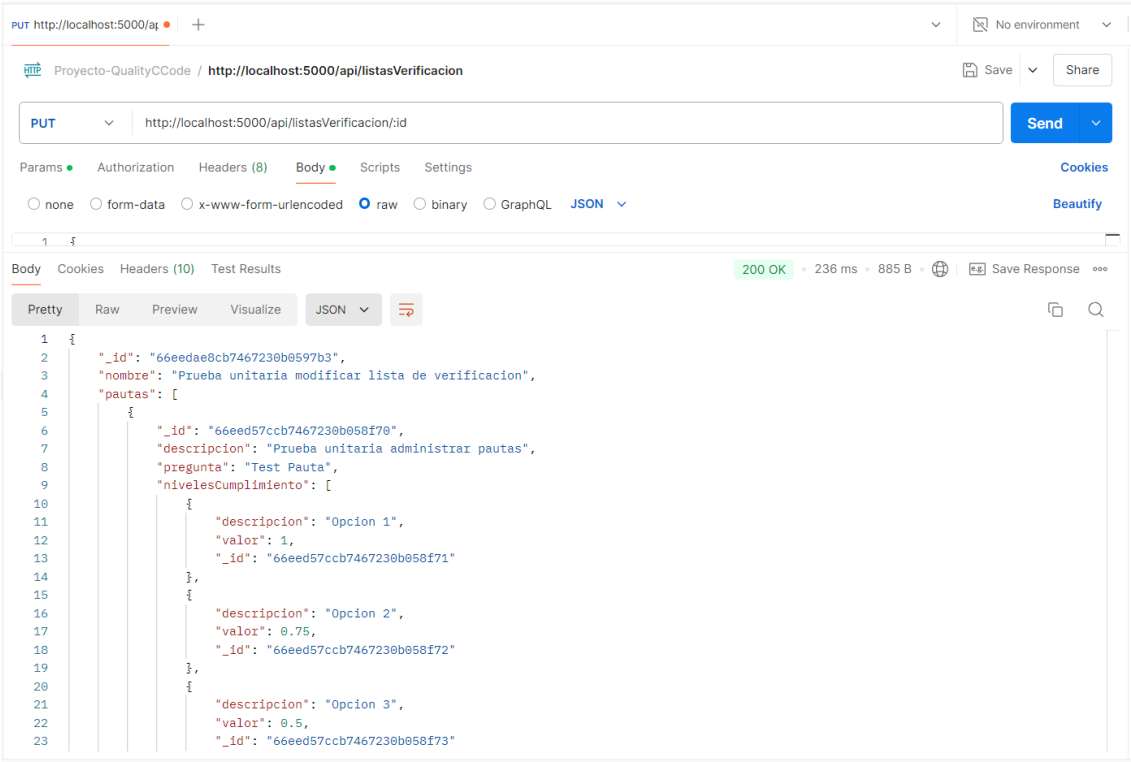


Figura 30-A23. Verificación de modificar un registro del modelo lista de verificación - PU-05.

En la **Figura 31-A23** se muestra la lista de verificación registrada en la PU-04, mediante la ejecución del diseño del frontend.

Figura 31-A23. Verificación de la modificación del registro lista de verificación - PU-05.

En la **Figura 32-A23** se muestra la API para eliminar un registro modelo lista de verificación.

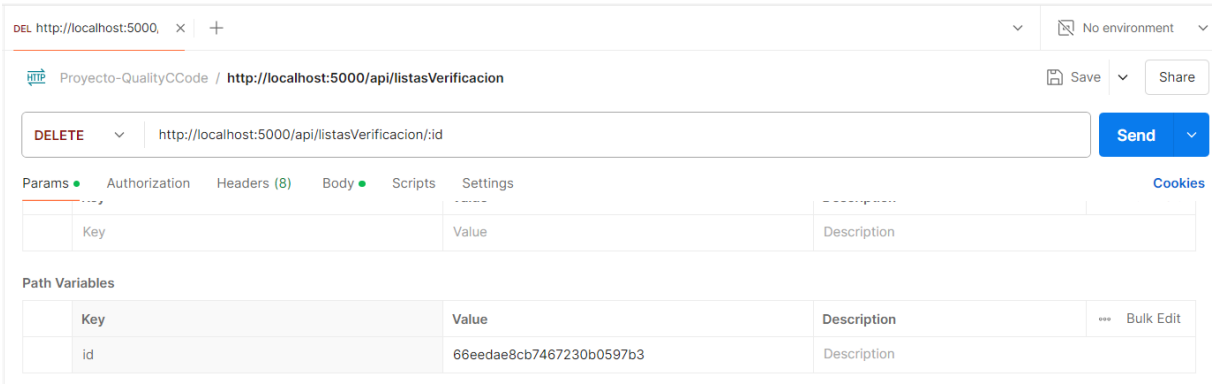


Figura 32-A23. API para eliminar un registro del modelo lista de verificación - PU-05.

En la **Figura 33-A23** se muestra la verificación de eliminar un registro lista de verificación, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

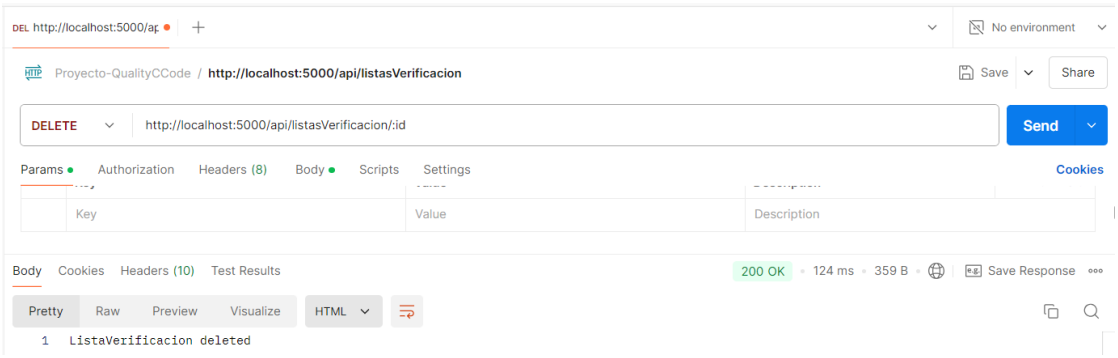


Figura 33-A23. Verificación de eliminar un registro del modelo lista de verificación - PU-05.

- **Tarea 4: Administrar métrica**

  - **Diseño**

En la **Figura 34-A23** se muestra el diseño para registrar una nueva métrica.

Figura 34-A23. Diseño, crear lista de verificación.

En la **Figura 35-A23** se muestra el diseño para actualizar y eliminar una nueva métrica.

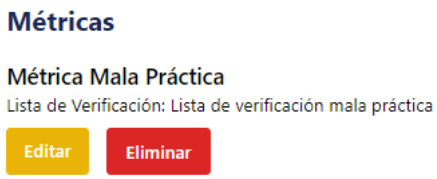


Figura 35-A23. Diseño, actualizar y eliminar una métrica.

Los casos de pruebas están relacionados a cada uno de los diseños de la **Figura 34-A23** y 65, cuyo objetivo es verificar el modelo para crear y llevar a cabo modificación, eliminación y obtener datos.

- **Codificación**

En la **Figura 36-A23** se destaca las funciones para registrar una nueva métrica, obtener los registros de todas las listas de las métricas y en la **Figura 37-A23** las funciones para permitir la actualización de los datos existentes de una métrica, así como su eliminación.

Figura 36-A23. Código para crear y obtener métricas.

```
JS MetricaController.js ×
src > controllers > JS MetricaController.js > ...
153  // Update a Metrica by ID
154  router.put('/:id', async (req, res) => {
155      try {
156          const metrica = await Metrica.findByIdAndUpdate(req.params.id, req.body, { new: true }).populate('listaVerificacion');
157          if (!metrica) {
158              return res.status(404).send('Metrica not found');
159          }
160          res.json(metrica);
161      } catch (err) {
162          res.status(500).send(err.message);
163      }
164  });
165
166  // Delete a Metrica by ID
167  router.delete('/:id', async (req, res) => {
168      try {
169          const metrica = await Metrica.findByIdAndDelete(req.params.id);
170          if (!metrica) {
171              return res.status(404).send('Metrica not found');
172          }
173          res.send('Metrica deleted');
174      } catch (err) {
175          res.status(500).send(err.message);
176      }
177  });
```

Figura 37-A23. Código para actualizar y eliminar una métrica.

- **Pruebas**

Para las pruebas unitarias se registró dos casos de prueba, presentando la ejecución de manera detallada.

La **Tabla 6-A23** se detalla el caso de prueba para verificar el modelo métrica, mediante el escenario crear métrica.

Tabla 6-A23. Prueba unitaria administrar métrica- PU-06.

Prueba Unitaria – PU-06			
Número: PU-06		**Versión**: 1.0	
Componente evaluado: Caso de uso: Administrar métrica			
Identificador: Administrar métrica		**Historia de usuario**: HU04	
Caso de prueba: Verificación del modelo métrica			
Datos de entrada	**Descripción**	**Salida esperada**	**Resultado**
Json lista de verificación	Verificar si el modelo métrica se crea correctamente	Las instancias del modelo métrica deben crearse correctamente y verificar el estado del registro.	Aceptado

Referencia de historia de usuario (RHU).

En la **Figura 38-A23** se muestra la API y el objeto json de prueba para crear el modelo métrica, al registrar una métrica es necesario agregar una lista de verificación para ello se hace uso el modelo lista de verificación implementado en la PU-04.

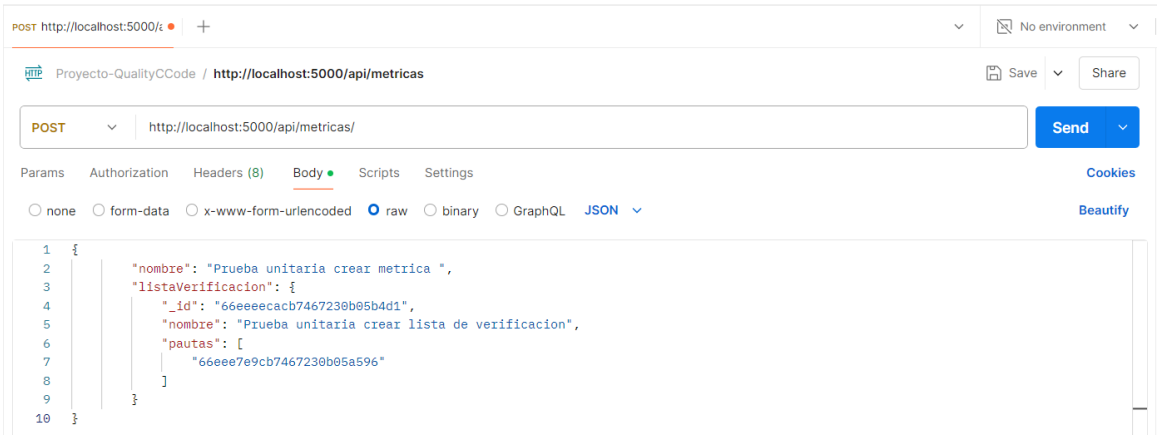


Figura 38-A23. Api y objeto json métrica para crear un modelo métrica - PU-06.

En la **Figura 39-A23** se muestra la verificación del modelo métrica, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

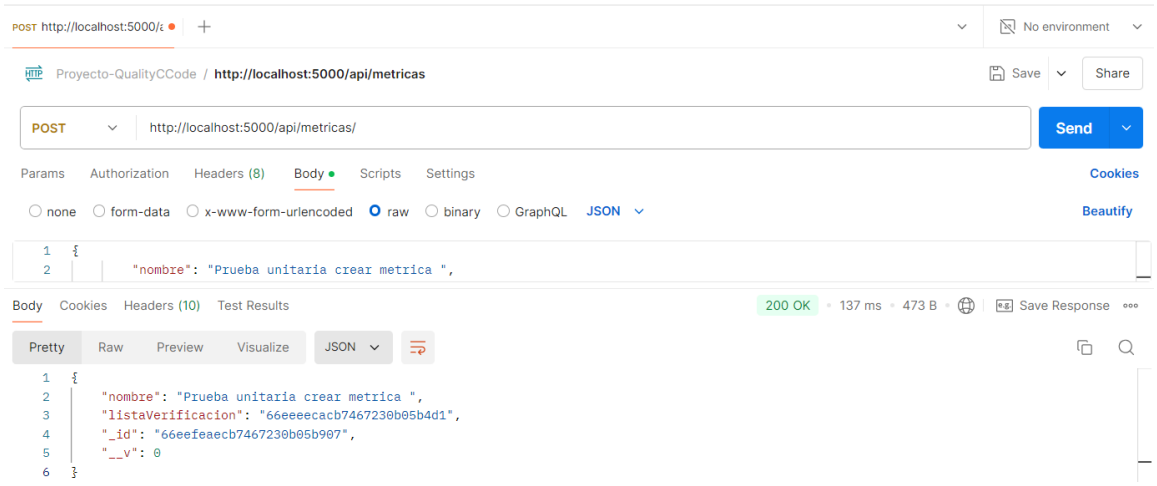


Figura 39-A23. Verificación del modelo métrica - PU-06.

En la **Figura 40-A23** se muestra la métrica registrada, mediante la ejecución del diseño del frontend.

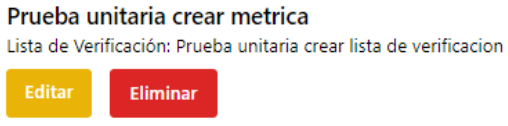


Figura 40-A23. Verificación del registro lista de verificación - PU-06.

La **Tabla 7-A23** se detalla el caso de prueba del modelo para llevar a cabo modificaciones, eliminaciones y obtener datos.

Tabla 7-A23. Prueba unitaria administrar de métrica- PU-07.

Prueba Unitaria – PU-07			
Número: PU-07		**Versión**: 1.0	
Componente evaluado: Caso de uso: Administrar métrica			
Identificador: Administrar métrica		**Historia de usuario**: HU04	
Caso de prueba: Modificación, eliminación y obtención de datos			
Datos de entrada	**Descripción**	**Salida esperada**	**Resultado**
Modelo pauta. Modelo lista de verificación. Modelo métrica.	Verificar si se efectúan los métodos de modificación, eliminación y listar los datos de los registros del modelo métrica	Obtener los datos de cada uno de los registros para tener la opción de eliminarlos o modificarlos	Aceptado

Referencia de historia de usuario (RHU).

En la **Figura 41-A23** se muestra la API para listar los datos de los registros del modelo métrica.

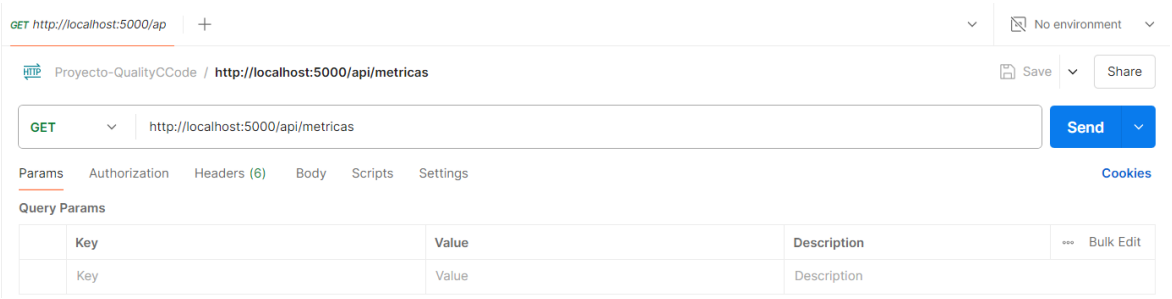


Figura 41-A23. Api para listar los datos de los registros del modelo métrica - PU-07.

En la **Figura 42-A23** se muestra la verificación de listar los registros de las métricas, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

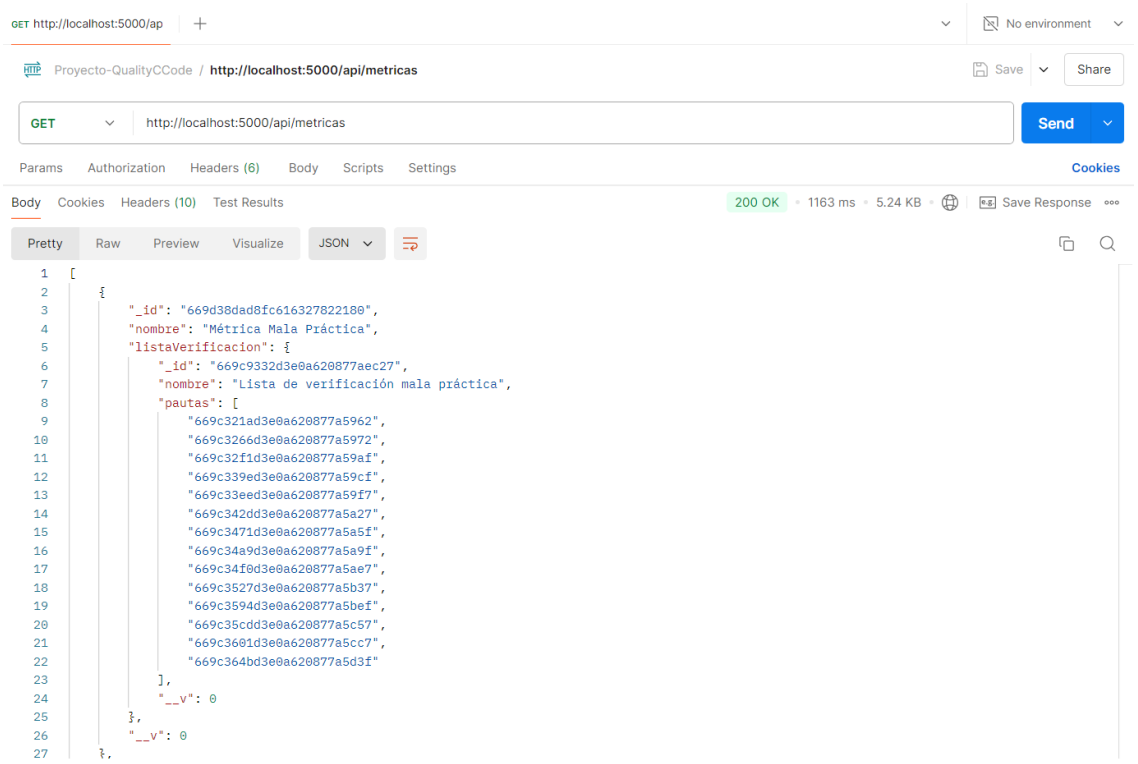


Figura 42-A23. Verificación de listar los registros de las métricas - PU-07.

En la **Figura 43-A23** se muestra la API en la cual establece la variable Id para modificar los datos y eliminar un registro del modelo métrica.

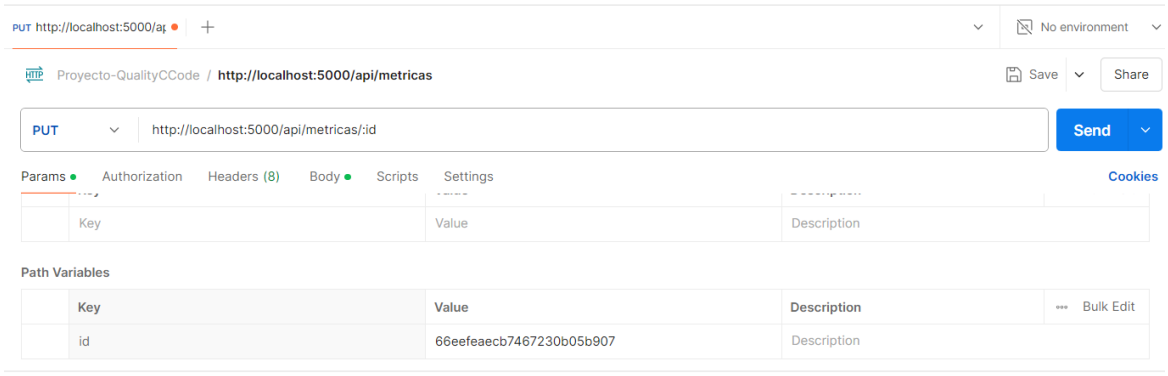


Figura 43-A23. Variable Id para modificar y eliminar un registro del modelo métrica - PU-07.

En la **Figura 44-A23** se muestra la API y el objeto Json para modificar los datos de un registro del modelo métrica.

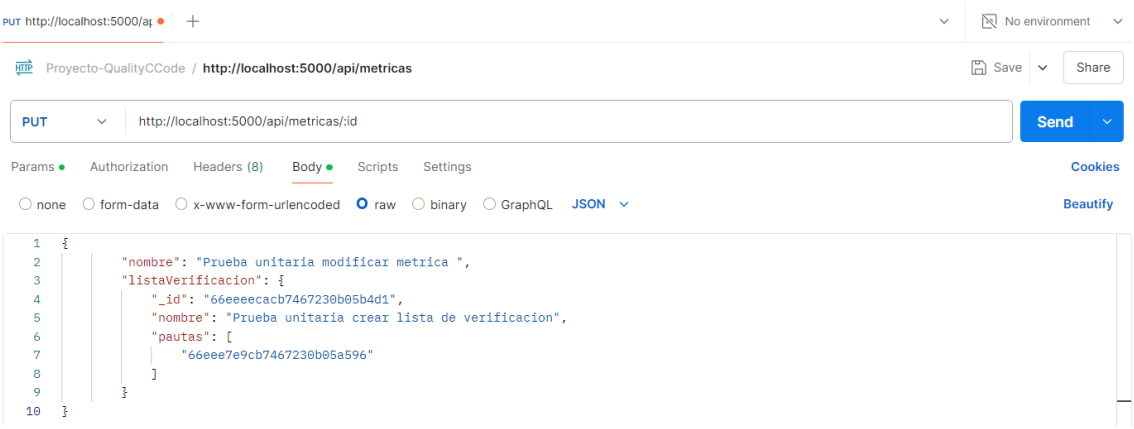


Figura 44-A23. API y el objeto Json para modificar los datos de un registro del modelo métrica - PU-07.

En la **Figura 45-A23** se muestra la verificación de modificar un registro métrica, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

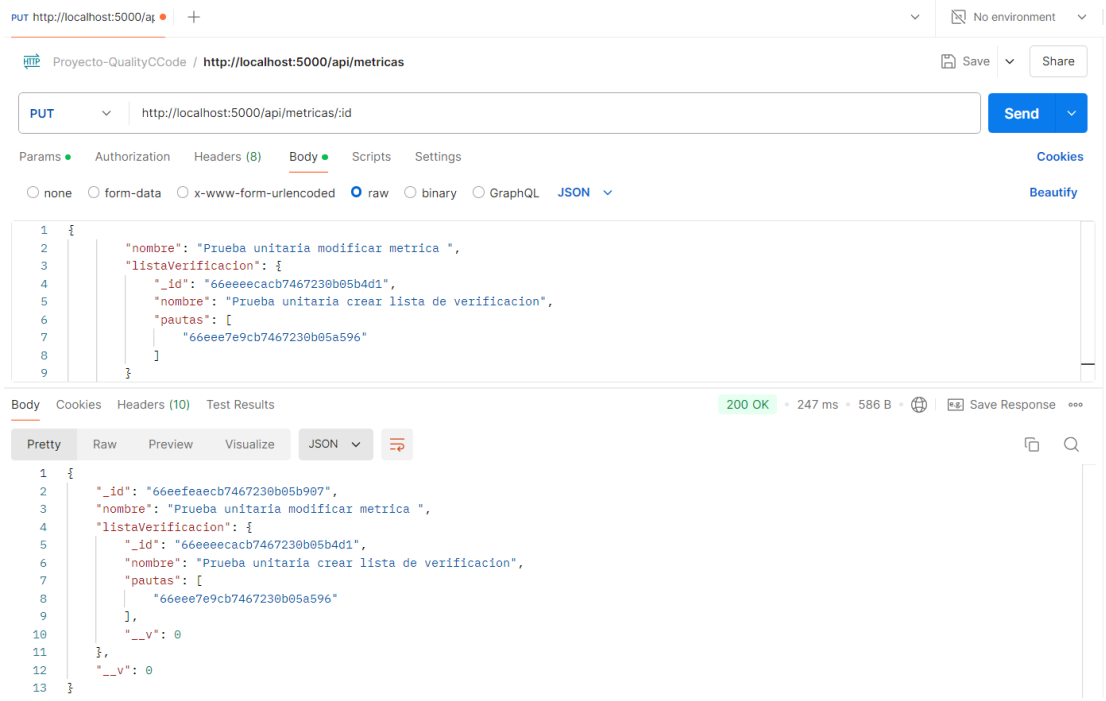


Figura 45-A23. Verificación de modificar un registro métrica - PU-07.

En la **Figura 46-A23** se muestra la métrica registrada en la PU-06, en ejecución del diseño del frontend se ha actualizado correctamente.

Figura 46-A23. Verificación de la modificación del registro métrica - PU-07.

En la **Figura 47-A23** se muestra la API para eliminar un registro modelo métrica.

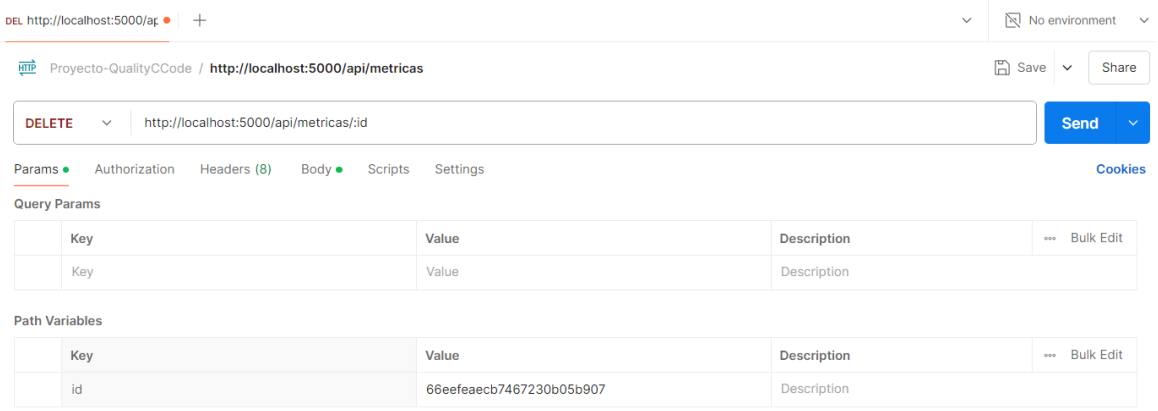


Figura 47-A23. API para eliminar un registro del modelo métrica - PU-07.

En la **Figura 48-A23** se muestra la verificación de eliminar un registro métrica, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

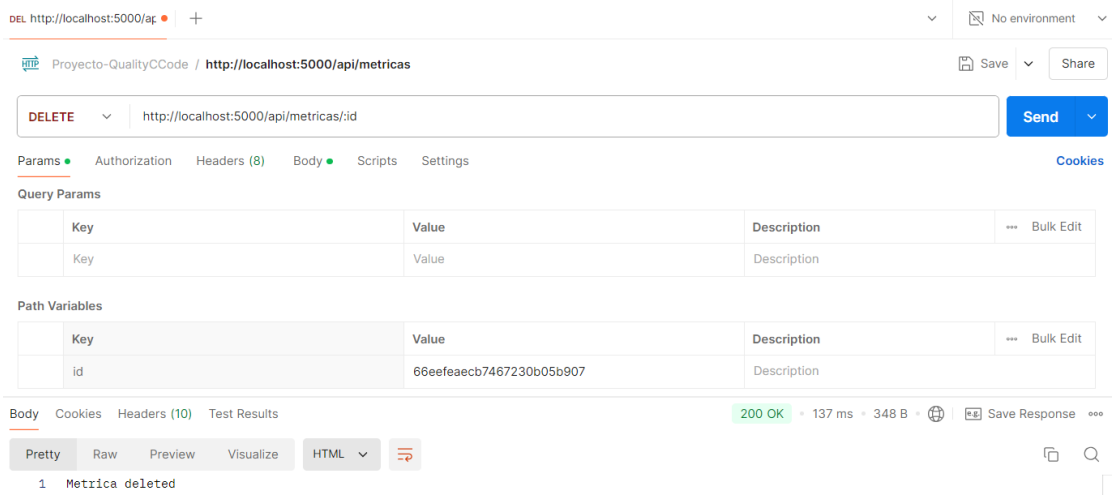


Figura 48-A23. Verificación de eliminar un registro métrica - PU-07.

- **Tarea 5: Administrar subcaracterística**

  - **Diseño**

En la **Figura 49-A23** se muestra el diseño para registrar una nueva subcaracterística.

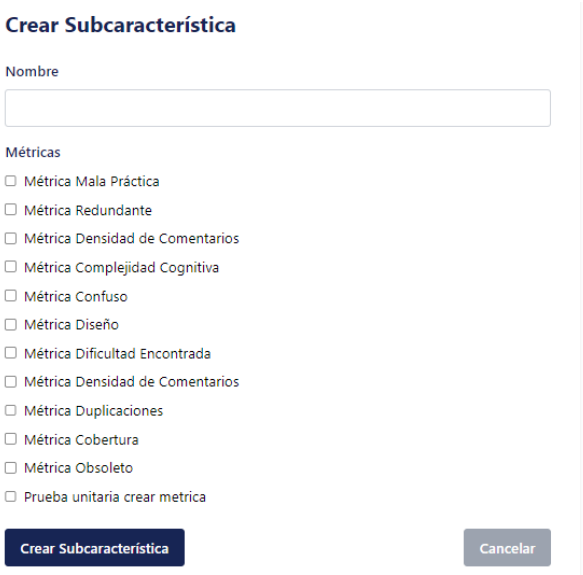


Figura 49-A23. Diseño, crear subcaracterística.

En la **Figura 50-A23** se muestra el diseño para actualizar y eliminar una métrica subcaracterística.

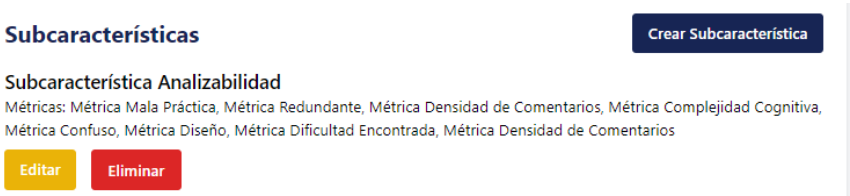


Figura 50-A23. Diseño, actualizar y eliminar una subcaracterística.

Los casos de pruebas están relacionados a cada uno de los diseños de la figura 79 y 80, cuyo objetivo es verificar el modelo para crear y llevar a cabo modificación, eliminación y obtener datos.

  - **Codificación**

En la **Figura 51-A23** se destaca las funciones para registrar una nueva subcaracterística, obtener los registros de todas las listas de las subcaracterísticas y en la **Figura 52-A23** las

funciones para permitir la actualización de los datos existentes de una subcaracterística, así como su eliminación.

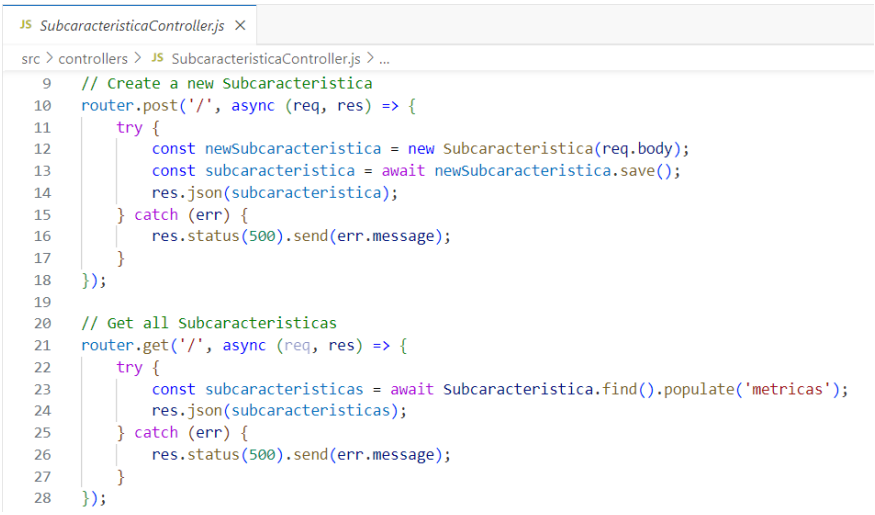


Figura 51-A23. Código para crear y obtener subcaracterísticas.

Figura 52-A23. Código para actualizar y eliminar una subcaracterística.

- **Pruebas**

Para las pruebas unitarias se registró dos casos de prueba, presentando la ejecución de manera detallada.

En la **Tabla 8-A23** se detalla el caso de prueba para verificar el modelo subcaracterística, mediante el método subcaracterística.

Tabla 8-A23. Prueba unitaria administrar subcaracterística - PU-08.

Prueba Unitaria – PU-08			
Número: PU-08		**Versión**: 1.0	
Componente evaluado: Caso de uso: Administrar subcaracterística			
Identificador: Administrar subcaracterística		**Historia de usuario**: HU05	
Caso de prueba: Verificación del modelo subcaracterística			
Datos de entrada	**Descripción**	**Salida esperada**	**Resultado**
Json subcaracterística	Verificar si el modelo subcaracterística se crea correctamente	Las instancias del modelo subcaracterística deben crearse correctamente y verificar el estado del registro.	Aceptado

Referencia de historia de usuario (RHU).

En la **Figura 53-A23** se muestra la API y el objeto json de prueba para crear el modelo subcaracterística, al registrar una subcaracterística es necesario agregar una métrica para ello se hace uso del modelo métrica implementado en la PU-06.

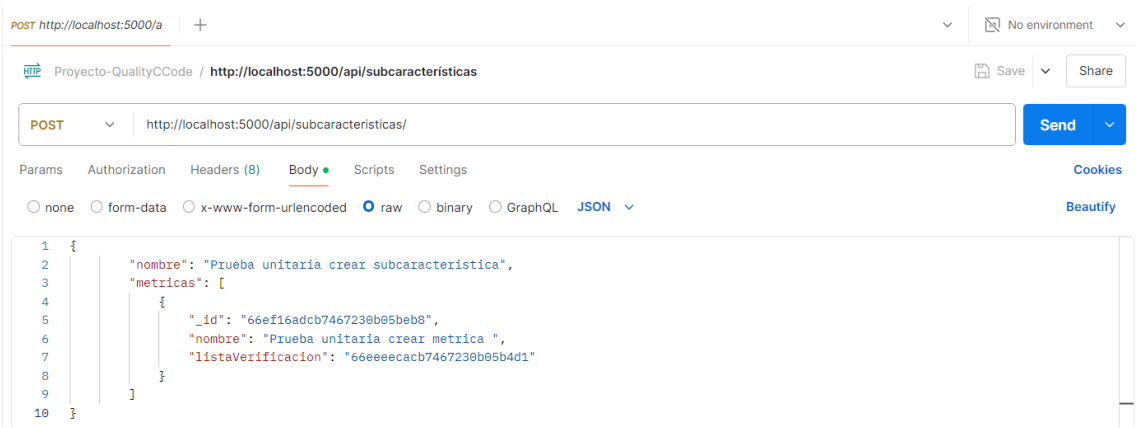


Figura 53-A23. Api y objeto json métrica para crear un modelo subcaracterística - PU-08.

En la **Figura 54-A23** se muestra la verificación del modelo subcaracterística, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

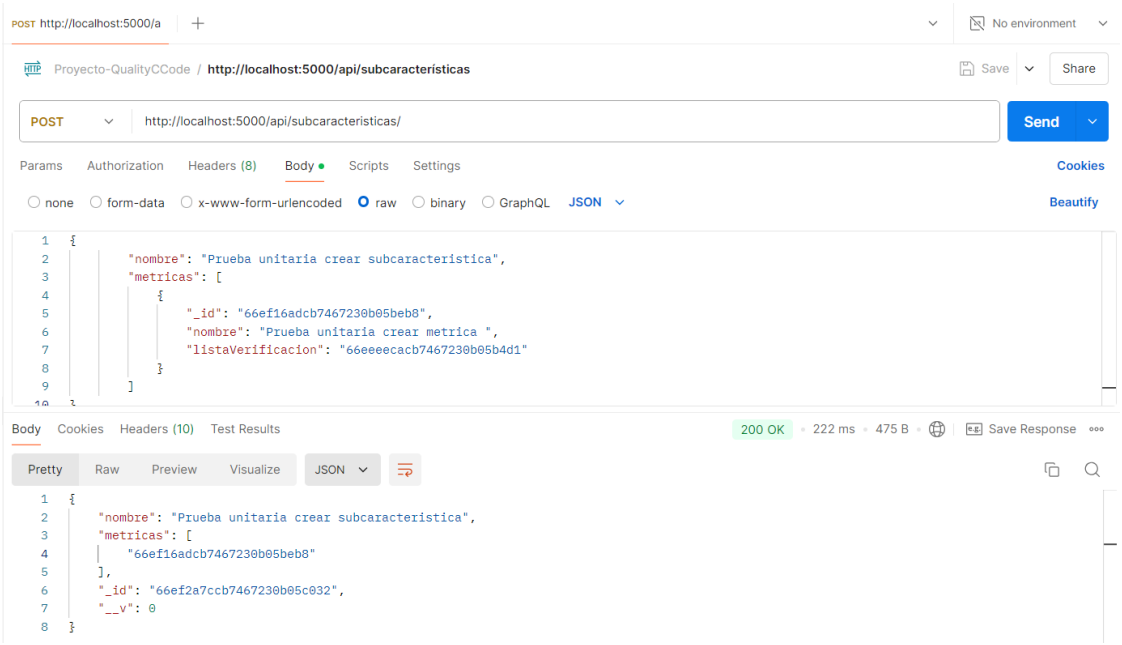


Figura 54-A23. Verificación del modelo subcaracterística - PU-08.

En la **Figura 55-A23** se muestra la subcaracterística registrada, mediante la ejecución del diseño del frontend.

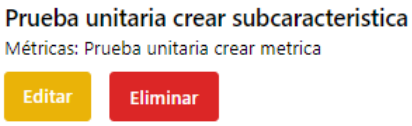


Figura 55-A23. Verificación del registro subcaracterística - PU-08.

La **Tabla 9-A23** se detalla el caso de prueba del modelo para llevar a cabo modificaciones, eliminaciones y obtener datos.

Tabla 9-A23. Prueba unitaria administrar de subcaracterística - PU-09.

Prueba Unitaria – PU-09			
Número: PU-09		**Versión**: 1.0	
Componente evaluado: Caso de uso: Administrar subcaracterística			
Identificador: Administrar subcaracterística		**Historia de usuario**: HU05	
Caso de prueba: Modificación, eliminación y obtención de datos			
Datos de entrada	**Descripción**	**Salida esperada**	**Resultado**
Modelo pauta. Modelo lista de verificación. Modelo métrica. Modelo subcaracterística	Verificar si se efectúan los métodos de modificación, eliminación y listar los datos de los registros del modelo subcaracterística	Obtener los datos de cada uno de los registros para tener la opción de eliminarlos o modificarlos	Aceptado

Referencia de historia de usuario (RHU).

En la **Figura 56-A23** se muestra la API para listar los datos de los registros del modelo subcaracterística.

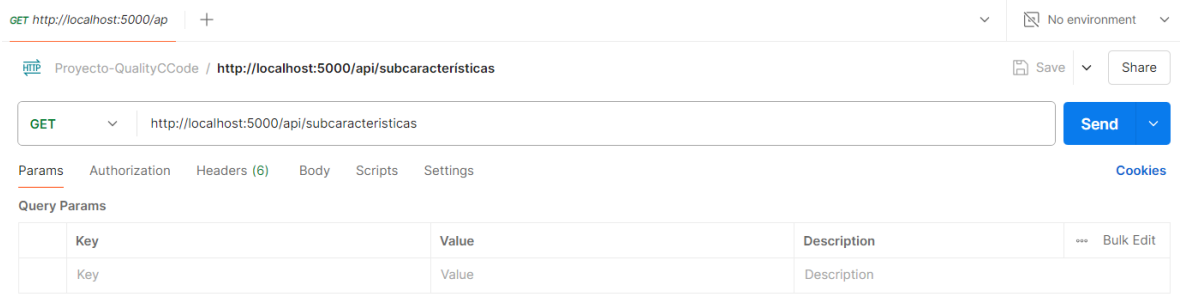


Figura 56-A23. Api para listar los datos de los registros del modelo subcaracterística - PU-09.

En la **Figura 57-A23** se muestra la verificación de listar los registros de las subcaracterísticas, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

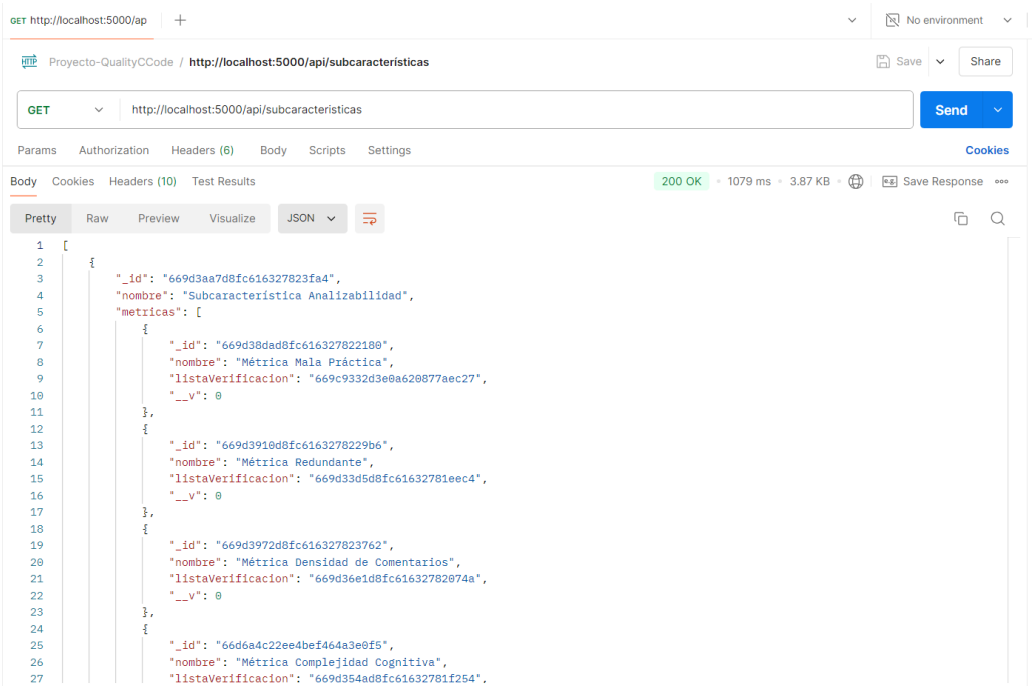


Figura 57-A23. Verificación de listar los registros de las subcaracterísticas - PU-09.

En la **Figura 58-A23** se muestra la API en la cual establece la variable Id para modificar los datos y eliminar un registro del modelo subcaracterística.

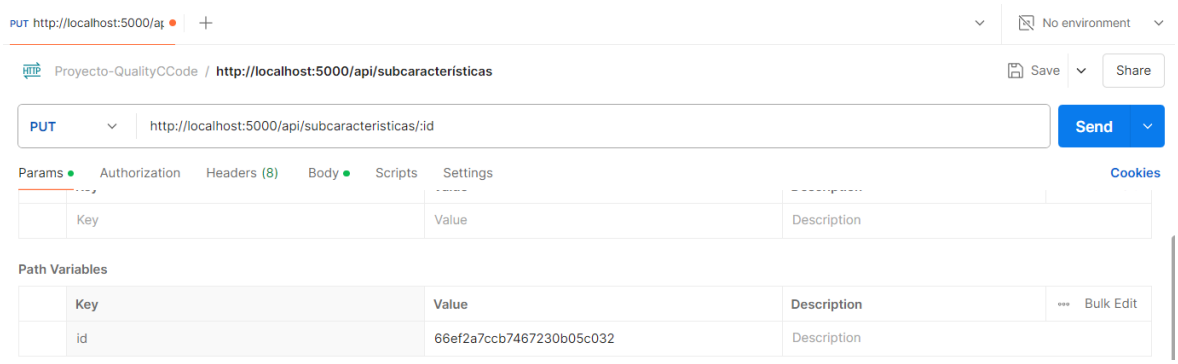


Figura 58-A23. Variable Id para modificar y eliminar un registro del modelo subcaracterísticas - PU-09.

En la **Figura 59-A23** se muestra la API y el objeto Json para modificar los datos de un registro del modelo subcaracterística.

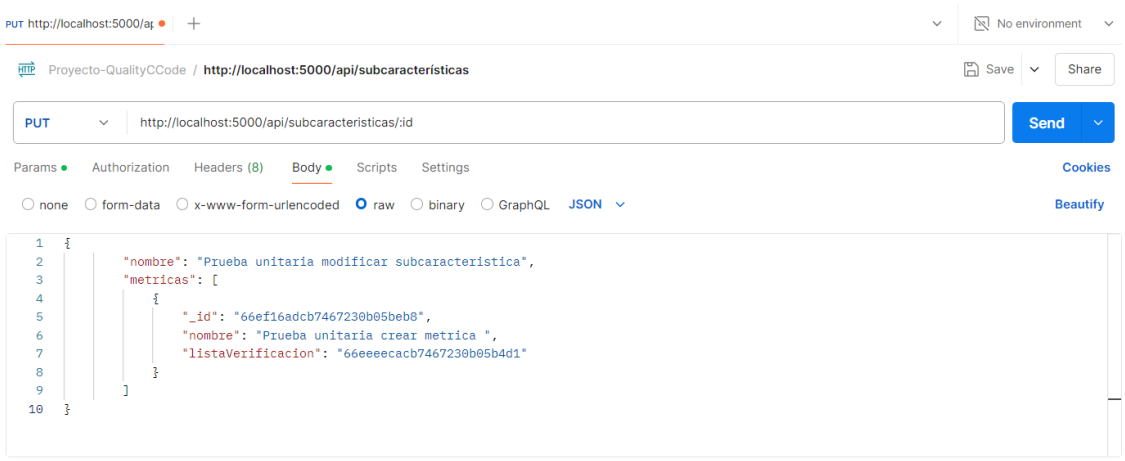


Figura 59-A23. API y el objeto Json para modificar modelo subcaracterística - PU-09.

En la **Figura 60-A23** se muestra la verificación de modificar un registro subcaracterística, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

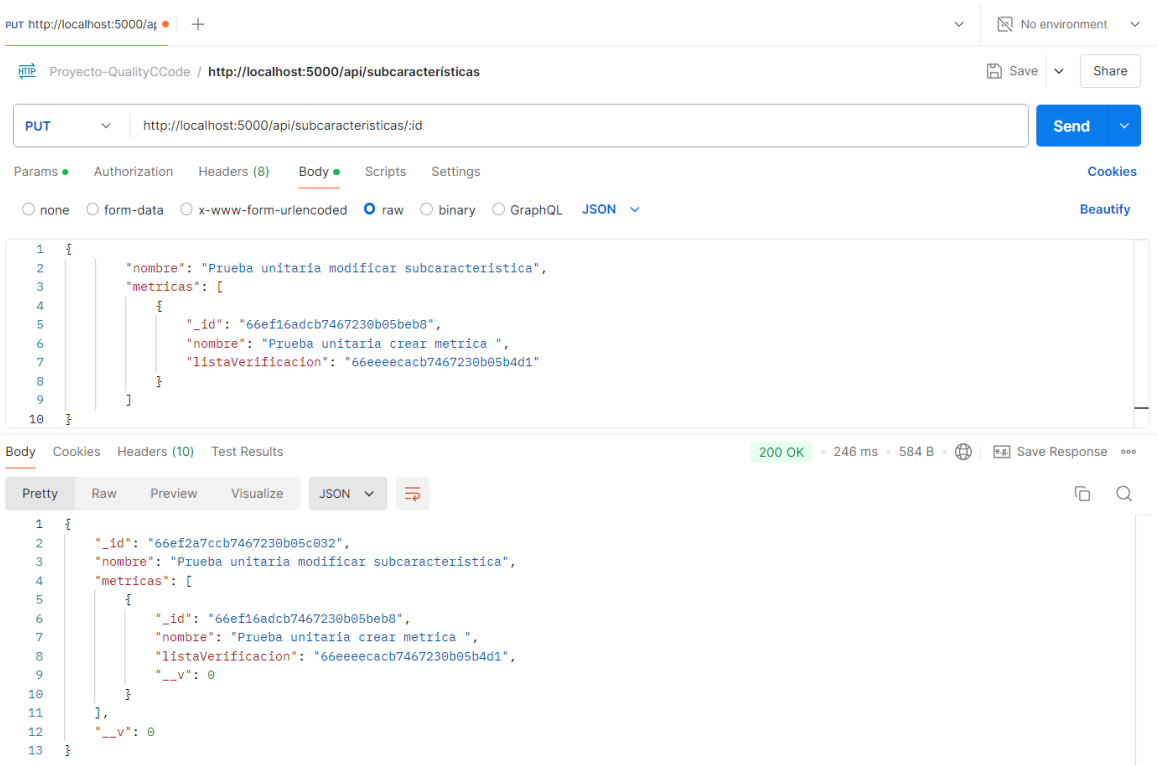


Figura 60-A23. Verificación de modificar un registro subcaracterística - PU-09.

En la **Figura 61-A23** se muestra la subcaracterística registrada en la PU-08, en ejecución del diseño del frontend se ha actualizado correctamente.

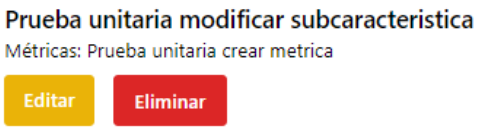


Figura 61-A23. Verificación de la modificación del registro subcaracterística - PU-09.

En la **Figura 62-A23** se muestra la API para eliminar un registro modelo subcaracterística.

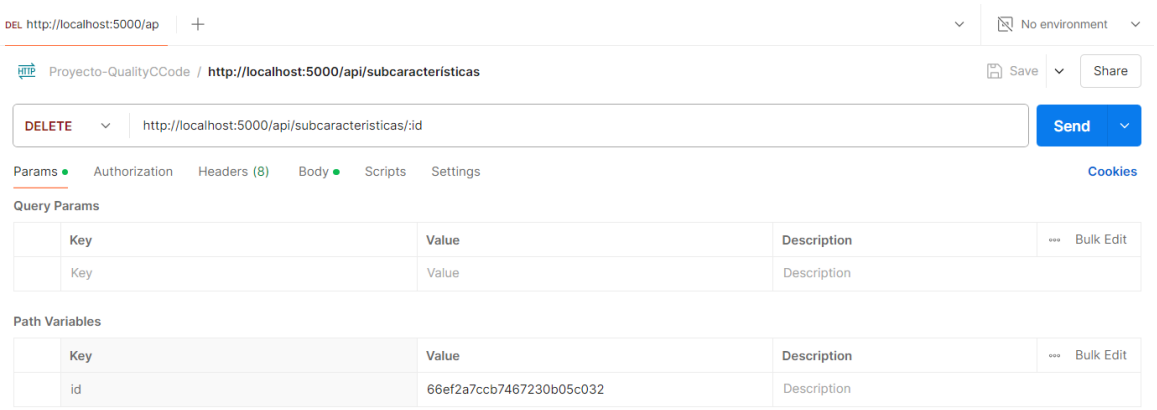


Figura 62-A23. API para eliminar un registro del modelo subcaracterística - PU-09.

En la **Figura 63-A23** se muestra la verificación de eliminar un registro subcaracterística, en el cual se evidencia que solicitud HTTP ha sido procesada con éxito por el servidor.

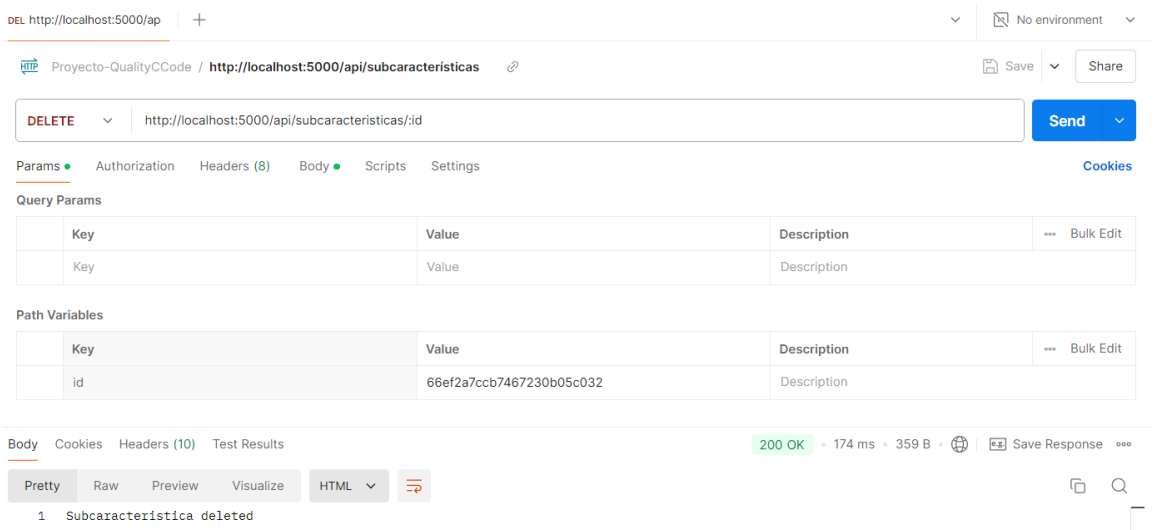


Figura 63-A23. Verificación de eliminar un registro subcaracterística - PU-09.

- **Tarea 6, 7 y 8: Administrar proyecto y prueba de mantenibilidad**

  - **Diseño**

En la **Figura 64-A23** se muestra el diseño para crear un proyecto, el formulario permite ingresar los detalles del proyecto, como nombre, enlace del repositorio de GitHub, la rama, y token de autenticación, lo que facilita la gestión de proyectos para el usuario.

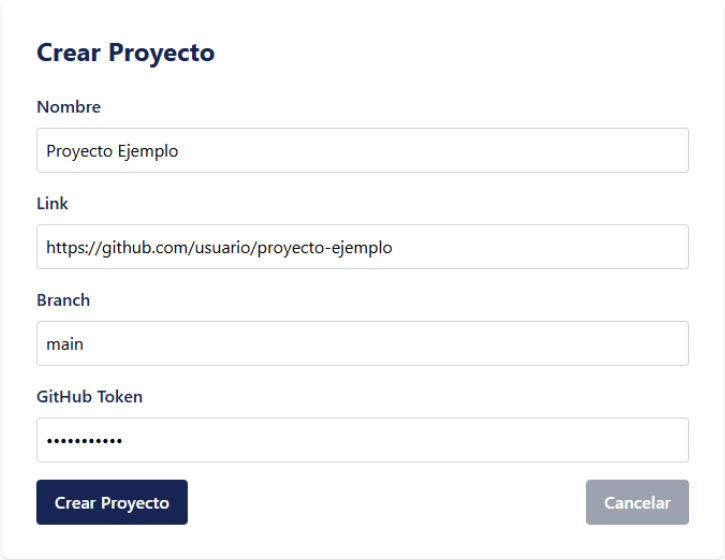
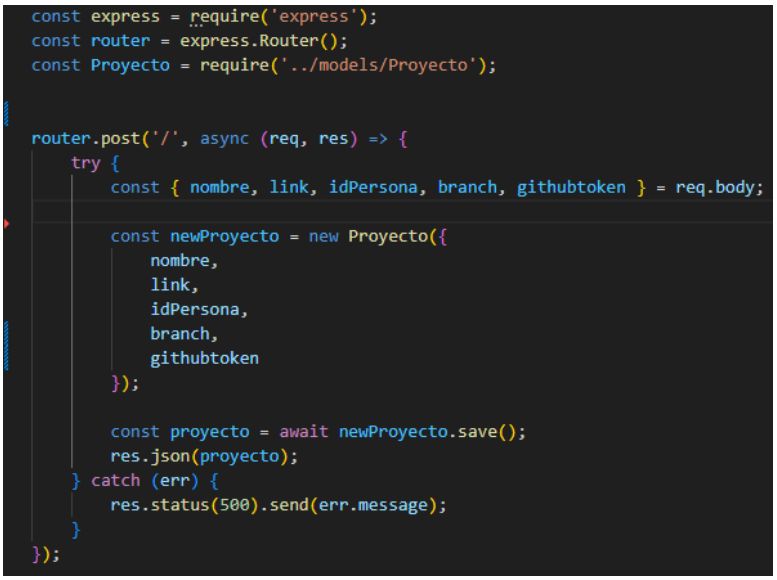


Figura 64-A23. Diseño de crear un proyecto.

  - **Codificación**

En la **Figura 65-A23** se destaca el código para registrar un nuevo proyecto, utilizando una ruta POST en el backend. Se procesan los datos enviados por el usuario a través del formulario, y se crea una nueva instancia del proyecto correspondientes.



```
const express = require('express');
const router = express.Router();
const Proyecto = require('../models/Proyecto');


router.post('/', async (req, res) => {
    try {
        const { nombre, link, idPersona, branch, githubtoken } = req.body;

        const newProyecto = new Proyecto({
            nombre,
            link,
            idPersona,
            branch,
            githubtoken
        });

        const proyecto = await newProyecto.save();
        res.json(proyecto);
    } catch (err) {
        res.status(500).send(err.message);
    }
});
```

Figura 65-A23. Código para registrar un nuevo proyecto.

- **Pruebas**

Para la prueba unitaria se registró un caso de prueba, como se detalla en la **Tabla 10-A23**, la misma tiene su respectiva figura presentando la ejecución de manera más detallada.

Tabla 10-A23. Prueba unitaria crear proyecto- PU-10.

Prueba Unitaria – PU-10			
Número: PU-10		**Versión**: 1.0	
Componente evaluado: Caso de uso: Crear proyecto			
Identificador: Administrar proyecto		**Historia de usuario**: HU06	
Caso de prueba: Verificación del modelo proyecto			
Datos de entrada	**Descripción**	**Salida esperada**	**Resultado**
Modelo Persona Modelo Proyecto	Verificar si el modelo Proyecto se crea correctamente.	El servidor debe devolver un código de estado 200 con los datos del proyecto	Aceptado

Referencia de historia de usuario (RHU).

En la **Figura 66-A23** se muestra el objeto json de prueba para verificar la creación del proyecto.

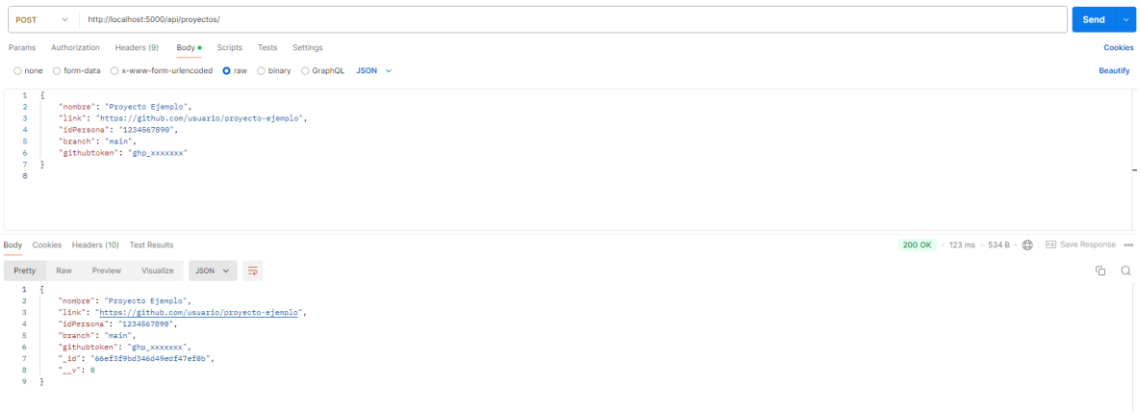


Figura 66-A23. Api y objeto json métrica para crear un modelo proyecto.

En la **Figura 67-A23** se muestra el diseño de la pantalla donde el usuario puede seleccionar y presionar el botón para realizar pruebas relaciones con subcaracterísticas, métricas, listas de verificación y pautas. En la **Figura 68-A23** se muestra una prueba con preguntas y respuestas, permitiendo al usuario registrar y registro de las respuestas.

Figura 67-A23. Diseño de seleccionar y realizar pruebas.

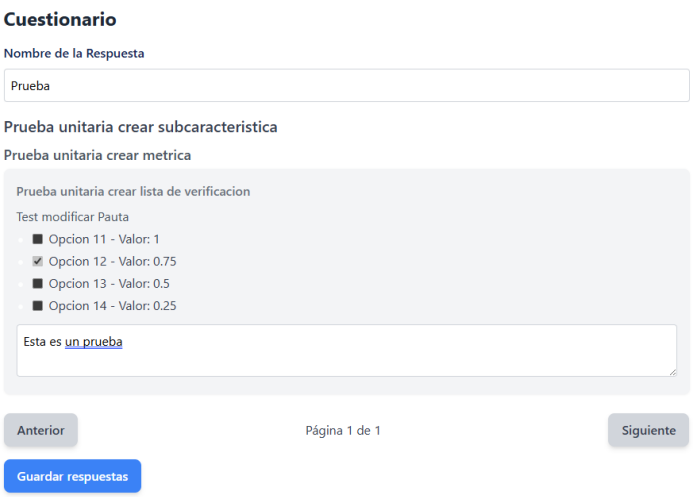


Figura 68-A23. Diseño de seleccionar y realizar pruebas.

En la **Figura 69-A23**, se destaca el código que permite obtener subcaracterísticas específicas por su ID. El endpoint /one/:id utiliza la función GET y verifica si los IDS proporcionados son válidos. Luego, busca las subcaracterísticas y las devuelve junto con las métricas y pautas asociadas, utilizando el método populate para obtener la información completa.

```
router.get('/one/:id', async (req, res) => {
    const ids = req.params.id.split(',');

    // Verificar si los IDs son ObjectId válidos
    if (!ids.every(id => mongoose.Types.ObjectId.isValid(id))) {
        return res.status(400).json({ message: 'Uno o más IDs son inválidos' });
    }

    try {
        // Buscar los documentos por los IDs
        const subcaracteristicas = await Subcaracteristica.find({ _id: { $in: ids } }).populate({
            path: 'metricas',
            populate: {
                path: 'listaVerificacion',
                populate: {
                    path: 'pautas'
                }
            }
        }).lean();

        // Comprobar si se encontraron los documentos
        if (!subcaracteristicas) {
            return res.status(404).json({ message: 'Subcaracterística(s) no encontrada(s)' });
        }

        // Enviar los documentos como respuesta
        res.status(200).json(subcaracteristicas);
    } catch (error) {
        console.error('Error al obtener subcaracterística:', error);
        res.status(500).json({ message: 'Error al obtener subcaracterística', error });
    }
});
```

Figura 69-A23. Código que permite obtener subcaracterísticas específicas.

En la **Figura 70-A23** se presenta la implementación de un endpoint POST en la ruta /guardar-respuestas. Esta función guarda las respuestas del usuario relacionadas con subcaracterísticas específicas, verificando el proyecto, subcaracterística, métrica, y pautas. El registro incluye el nombre de la respuesta y los intentos realizados.

```
const express = require('express');
const router = express.Router();
const Respuesta = require('../models/Respuesta');

router.post('/guardar-respuestas', async (req, res) => {
    try {
        const { proyectoId, tipo, subcaracteristicaId, metricaId, respuestas, nombre, intentos } = req.body;

        const nuevaRespuesta = new Respuesta({
            proyectoId,
            tipo,
            subcaracteristicaId,
            metricaId,
            respuestas,
            nombre,
            intentos
        });

        await nuevaRespuesta.save();
        res.status(201).json({ message: 'Respuestas guardadas exitosamente' });
    } catch (error) {
        res.status(500).json({ error: 'Error al guardar las respuestas', details: error.message });
    }
});
```

Figura 70-A23. Codificación para guardar respuestas.

Para las pruebas unitarias se registró un caso de prueba, como se detalla en la **Tabla 11-A23**, además de la ejecución de manera detallada.

Tabla 11-A23. Prueba unitaria administrar prueba de mantenibilidad- PU-011.

Prueba Unitaria – PU-011			
Número: PU-11		**Versión**: 1.0	
Componente evaluado: Caso de uso: Administrar prueba de mantenibilidad			
Identificador: Administrar prueba de mantenibilidad		**Historia de usuario**: HU02, HU03, HU04, HU05, HU06, HU07 y HU08.	
Caso de prueba: Verificación del flujo de obtención y registro de subcaracterísticas, métricas y respuestas.			
Datos de entrada	**Descripción**	**Salida esperada**	**Resultado**
Modelo Subcaracterísticas Modelo Listas de verificación Modelo Métricas Modelo Pautas	Verificar si las subcaracterísticas se obtienen correctamente.	El servidor debe devolver un código de estado 200 y los datos de la subcaracterística	Aceptado
Modelo prueba	Verificar si las respuestas se guardan correctamente.	El servidor debe devolver un código de estado 200 y también un mensaje de confirmación “Respuestas guardas exitosamente”	Aceptado

Referencia de historia de usuario (RHU).

En la **Figura 71-A23**, se muestra el objeto JSON utilizado para obtener la subcaracterística mediante su ID, y se presenta la respuesta del servidor mostrando información relacionada a la misma.

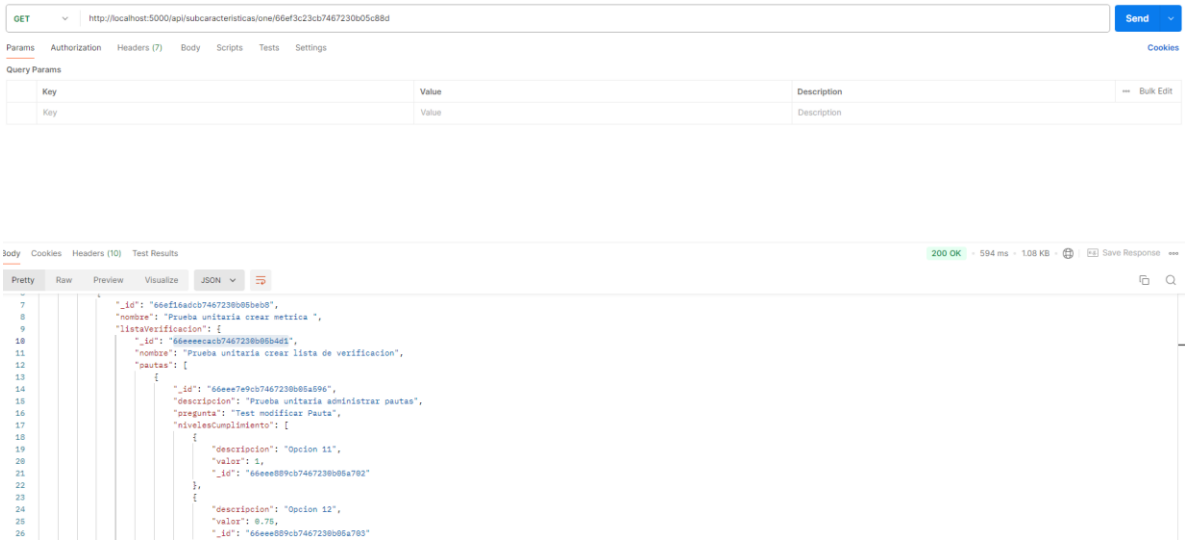


Figura 71-A23. Objeto json de prueba para obtener la subcaracterística.

En la **Figura 72-A23** se presenta el objeto JSON de prueba para guardar las respuestas, además se muestra la respuesta del servidor.

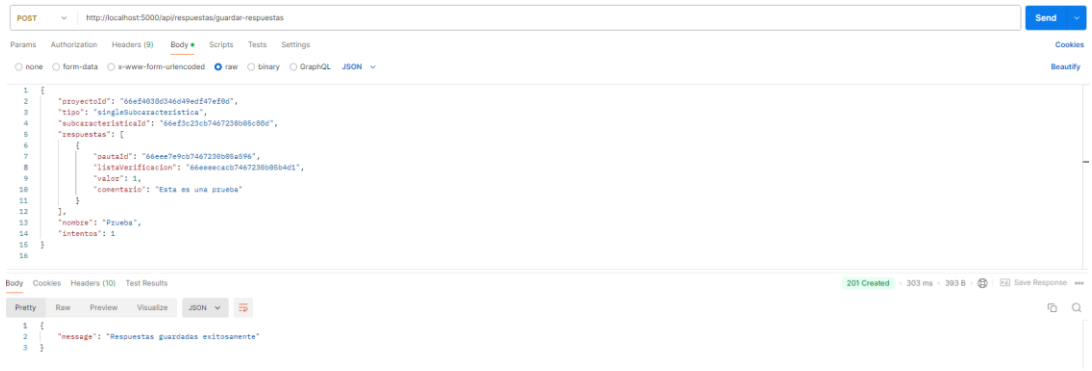


Figura 72-A23. Objeto json de prueba para guardar la respuesta.

- **Tarea 9: Informe lista de verificación**

  - **Diseño**

En la **Figura 73-A23** se muestra el diseño para el informe de cumplimiento de listas de verificación.

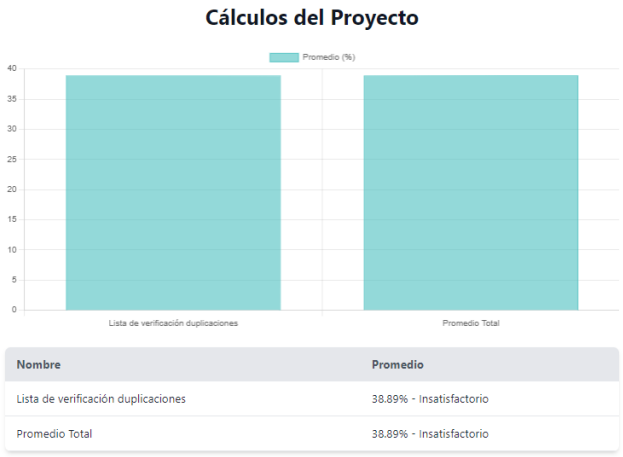
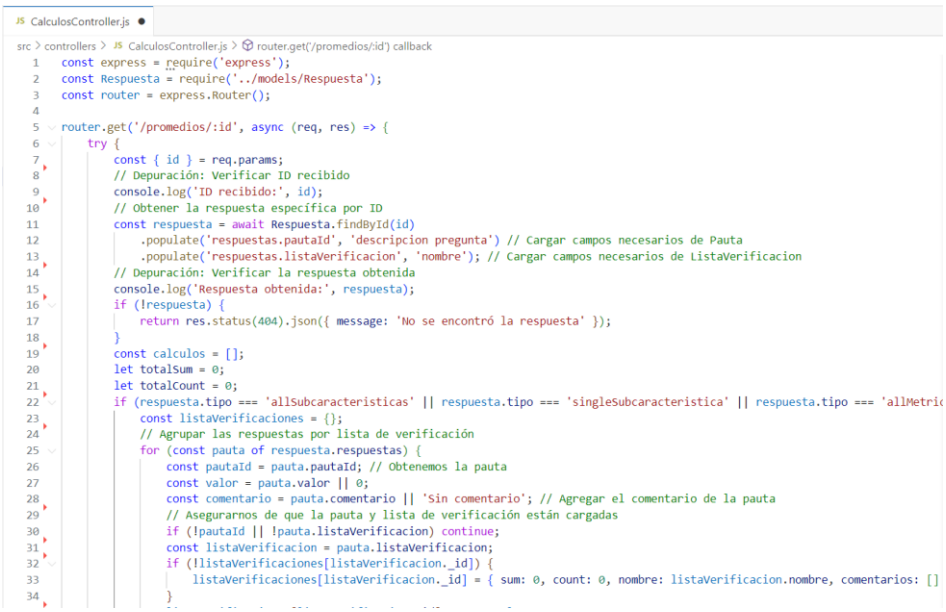


Figura 73-A23. Diseño del informe de cumplimiento de lista de verificación.

  - **Codificación**

En la **Figura 74-A23** se muestra el método principal para imprimir el informe de cumplimiento de listas de verificación.



```
CalculosController.js
src > controllers > CalculosController.js > router.get('/promedios/:id') callback
const express = require('express');
const Respuesta = require('../models/Respuesta');
const router = express.Router();

router.get('/promedios/:id', async (req, res) => {
    try {
        const { id } = req.params;
        // Depuración: Verificar ID recibido
        console.log('ID recibido:', id);
        // Obtener la respuesta específica por ID
        const respuesta = await Respuesta.findById(id)
            .populate('respuestas.pautaId', 'descripcion pregunta') // Cargar campos necesarios de Pauta
            .populate('respuestas.listaVerificacion', 'nombre'); // Cargar campos necesarios de ListaVerificacion
        // Depuración: Verificar la respuesta obtenida
        console.log('Respuesta obtenida:', respuesta);
        if (!respuesta) {
            return res.status(404).json({ message: 'No se encontró la respuesta' });
        }
        const calculos = [];
        let totalSum = 0;
        let totalCount = 0;
        if (respuesta.tipo === 'allSubcaracteristicas' || respuesta.tipo === 'singleSubcaracteristica' || respuesta.tipo === 'allMetric
            const listaVerificaciones = {};
            // Agrupar las respuestas por lista de verificación
            for (const pauta of respuesta.respuestas) {
                const pautaId = pauta.pautaId; // Obtenemos la pauta
                const valor = pauta.valor || 0;
                const comentario = pauta.comentario || 'Sin comentario'; // Agregar el comentario de la pauta
                // Asegurarnos de que la pauta y lista de verificación están cargadas
                if (!pautaId || !pauta.listaVerificacion) continue;
                const listaVerificacion = pauta.listaVerificacion;
                if (!listaVerificaciones[listaVerificacion._id]) {
                    listaVerificaciones[listaVerificacion._id] = { sum: 0, count: 0, nombre: listaVerificacion.nombre, comentarios: []
                }
```

Figura 74-A23. Método principal para imprimir el informe de cumplimiento de listas de verificación.

- **Tarea 10: Generar PDF**

  - **Diseño**

En la **Figura 75-A23** se muestra el PDF generado acerca del informe de cumplimiento de listas de verificación.

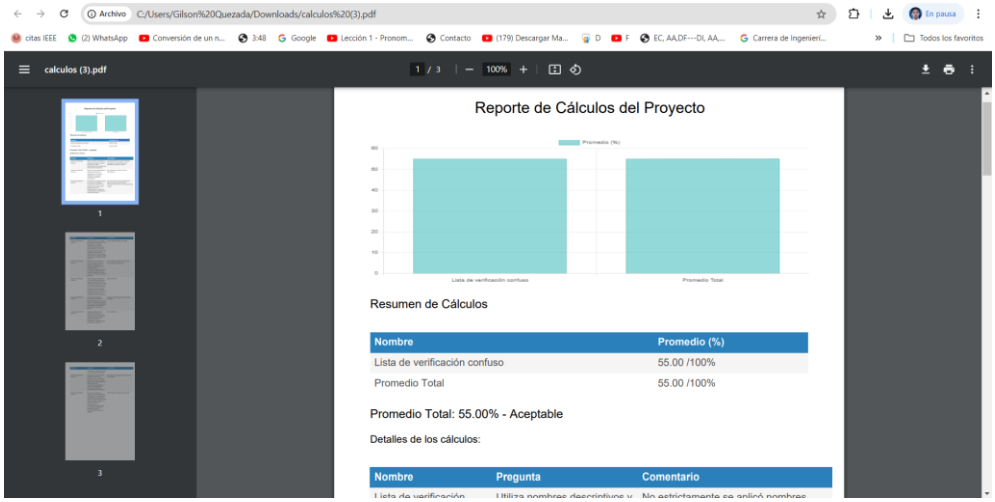


Figura 75-A23. PDF acerca del informe de cumplimiento de listas de verificación.

- **Codificación**

En la **Figura 76-A23** se muestra el método principal para generar el PDF acerca del cumplimiento de listas de verificación.

```
Calculos.tsx 1
src > components > Calculos.tsx > Calculos > generarPDF > useCallback() callback
23  const Calculos: React.FC = () => {
55      const generarPDF = useCallback(() => {
56          const doc = new jsPDF();
57          const pageWidth = doc.internal.pageSize.getWidth();
58          const pageHeight = doc.internal.pageSize.getHeight();    'pageHeight' is declared but its value is never read.
59          const marginLeft = 15;
60          const marginRight = 15;
61          let finalY = 20; // Variable para controlar la posición vertical
62
63          doc.setFontSize(18);
64          doc.text('Reporte de Cálculos del Proyecto', pageWidth / 2, finalY, { align: 'center' });
65          finalY += 10;
66
67          // Agregar gráfico si existe
68          if (chartRef.current) {
69              const chartCanvas = chartRef.current.canvas as HTMLCanvasElement;
70              const chartImage = chartCanvas.toDataURL('image/png');
71              doc.addImage(chartImage, 'PNG', marginLeft, finalY, pageWidth - marginLeft - marginRight, 60);
72              finalY += 70; // Espacio después del gráfico
73          }
74
75          doc.setFontSize(14);
76          doc.text('Resumen de Cálculos', marginLeft, finalY);
77          finalY += 10;
78
79          // Crear la tabla de resumen de cálculos
80          const tableColumn = ['Nombre', 'Promedio (%)'];
81          const tableRows: (string | number)[][] = [];
82
83          calculos.forEach(calculo => {
84              const calculoData = [calculo.nombre, `${(calculo.promedio * 100).toFixed(2)} /100% `];
85              tableRows.push(calculoData);
86          });
```

Figura 76-A23. Método principal para generar el PDF.

- **Tarea 11: Calcular grado de mantenibilidad**

  - **Diseño**

En la **Figura 77-A23** se muestra el diseño para calcular el grado de mantenibilidad de un proyecto alojado en el repositorio de GitHub, utilizando la API de sonarqube mediante el análisis de código estático.

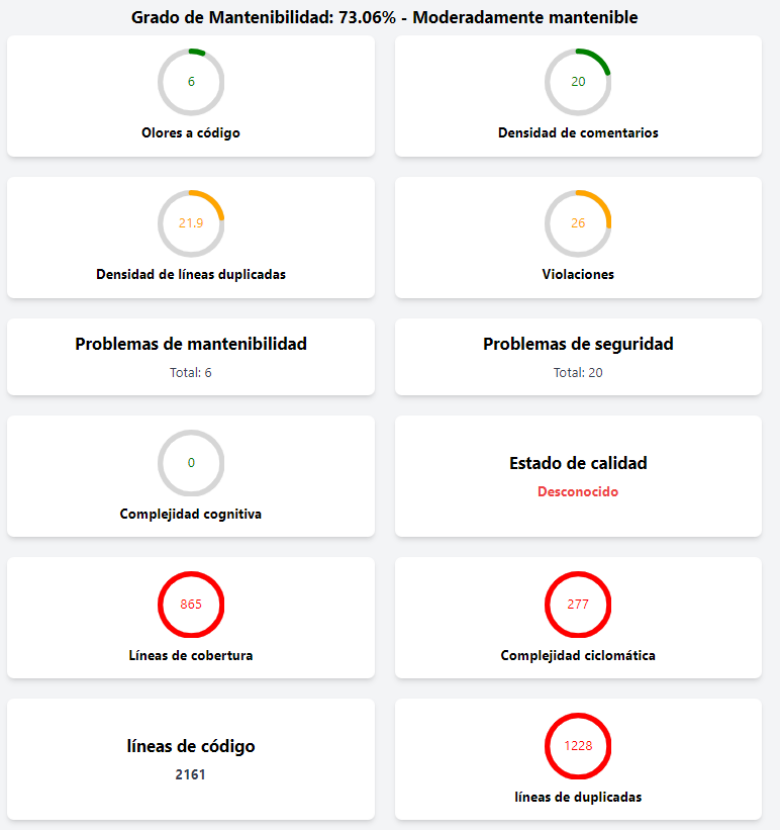


Figura 77-A23. Diseño para calcular el grado de mantenibilidad utilizando API SonarQube.

- **Codificación**

En la **Figura 78-A23** se muestra el método principal para calcular el grado de mantenibilidad.

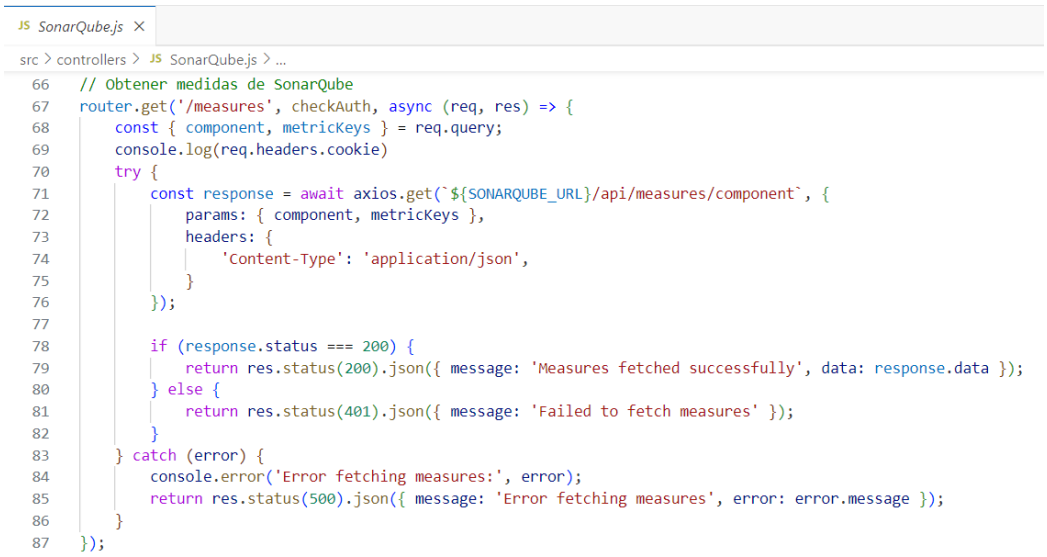


Figura 78-A23. Método principal para calcular el grado de mantenibilidad.

Anexo 24. Matriz para calcular el grado de mantenibilidad del caso de estudio

En la **Figura 1-A24**, se ilustra la matriz para calcular el grado de mantenibilidad del caso de estudio, en la aplicación del plan de estrategias para la solución de los defectos.

Propiedades de calidad	
Las propiedades son el resultados del analisis de codigo estatico utilizando la herramienta de SonarQube	
Lines de codigo banckend	1870
Lines de codigo frontend	26000

Propiedad de calidad	**Backend**	**Frontend**
Duplicated Lines	62	2823
Coverage lines	851	1534
Cyclomatic complexity	293	1021
Code smells	0	846
Cognitive Complexity	0	30
Comment lines	21,60%	0,40%

Obtencion de valores de las métricas

Metricas	**Backend**	**Frontend**
Diseño	15,67%	3,93%
Duplicaciones	3,32%	10,86%
Cobertura	45,51%	5,90%
Mala práctica	0,00%	3,25%
Obsoleto	0,00%	3,25%
Redundante	3,32%	10,86%
Cognitive Complexity	0,00%	0,12%
Confuso	0,00%	3,25%
Dificultad encontrada	0%	0,12%
Densidad de comentarios	78,40%	99,60%

	Subcaracterísticas				
	Modularidad	**Reusabilidad**	**Analizabilidad**	**Capacidad ser modificado**	**Capacidad ser probado**
	9,80%	1,63%	1,63%	1,63%	7,09%
	7,09%	1,63%	7,09%	7,09%	9,80%
	25,70%	9,80%	0,06%	9,80%	0,06%
		7,09%	1,63%	7,09%	25,70%
			9,80%		
			0,06%		
			89,00%		
Promedio	14,196%	5,035%	15,61%	6,40%	10,66%
Grado de calidad	85,80%	94,97%	84,39%	93,60%	89,34%

Grado de mantenibilidad	**89,62%**
Escala de valoración	*Altamente mantenible*

Figura 1-A24. Matriz para calcular el grado de mantenibilidad**.**

Anexo 25. Matriz para el cálculo del grado de mantenibilidad sin aplicar el plan de estrategias para la solución de defectos

En la **Figura 1-A25**, se ilustra la matriz para calcular el grado de mantenibilidad del caso de estudio, sin aplicar el plan de estrategias para la solución de defectos.

Propiedades de calidad	
Las propiedades son el resultados del analisis de codigo estatico utilizando la herramienta de SonarQube	
Lines de codigo del proyecto	2161

Propiedad de calidad	**Backend**
Duplicated Lines	1228
Coverage lines	865
Cyclomatic complexity	277
Code smells	6
Cognitive Complexity	0
Comment lines	20%

Obtencion de valores de las métricas	**Backend**
Diseño	12,82%
Duplicaciones	56,83%
Cobertura	40,03%
Mala práctica	0,28%
Obsoleto	0,28%
Redundante	56,83%
Cognitive Complexity	0,00%
Confuso	0,28%
Dificultad encontrada	0%
Densidad de comentarios	80,00%

	Subcaracterísticas				
	Modularidad	**Reusabilidad**	**Analizabilidad**	**Capacidad ser modificado**	**Capacidad ser probado**
	12,818%	0,278%	0,278%	0,278%	56,826%
	56,826%	0,278%	56,826%	56,826%	12,818%
	40,028%	12,818%	0,000%	12,818%	0,000%
		56,826%	0,28%	56,826%	40,028%
			12,818%		
			0,000%		
			80,00%		
Promedio	36,557%	17,550%	21,457%	31,687%	27,418%
Grado de calidad	63,44%	82,45%	78,543%	68,313%	72,582%
Grado de mantenibilidad	73,07%				

Figura 1-A25. Matriz para calcular el grado de mantenibilidad sin aplicar el plan de estrategias para la solución de defectos**.**

Anexo 26. Matriz para el cálculo del grado de mantenibilidad con la aplicación del plan de estrategias para la solución de defectos

En la **Figura 1-A26**, se ilustra la matriz para calcular el grado de mantenibilidad del caso de estudio, con la aplicación del plan de estrategias para la solución de defectos.

Propiedades de calidad	
Las propiedades son el resultados del analisis de codigo estatico utilizando la herramienta de SonarQube	
Lines de codigo del proyecto	1870

Propiedad de calidad	**Backend**
Duplicated Lines	62
Coverage lines	851
Cyclomatic complexity	293
Code smells	0
Cognitive Complexity	0
Comment lines	21,60%

Obtencion de valores de las métricas	**Backend**
Diseño	15,67%
Duplicaciones	3,32%
Cobertura	45,51%
Mala práctica	0,00%
Obsoleto	0,00%
Redundante	3,32%
Cognitive Complexity	0,00%
Confuso	0,00%
Dificultad encontrada	0%
Densidad de comentarios	78,40%

		Subcaracterísticas				
		Modularidad	**Reusabilidad**	**Analizabilidad**	**Capacidad ser modificado**	**Capacidad ser probado**
		15,668%	0,000%	0,000%	0,000%	3,316%
		3,316%	0,000%	3,316%	3,316%	15,668%
		45,508%	15,668%	0,000%	15,668%	0,000%
			3,316%	0,00%	3,316%	45,508%
				15,668%		
				0,000%		
				78,40%		
	Promedio	21,497%	4,746%	13,912%	5,575%	16,123%
Grado de calidad		78,50%	95,25%	86,088%	94,425%	83,877%
Grado de mantenibilidad		87,63%				

Figura 1-A26. Matriz para calcular el grado de mantenibilidad con la aplicación del plan de estrategias para la solución de defectos**.**

Anexo 27. Certificado de traducción del resumen

CERTIFICADO DE TRADUCCION

Lcdo. Luis Hernán Sánchez Villa
Licenciado en Ciencias de la Educación, especialización Idioma Inglés

CERTIFICO:

Que he realizado la traducción del idioma español al idioma inglés, del resumen derivado de la tesis denominada " EVALUACIÓN DE MANTENIBILIDAD PARA APLICACIONES WEB EN PRODUCCIÓN DE LOS LABORATORIOS DE LA CARRERA DE COMPUTACIÓN DE LA UNIVERSIDAD NACIONAL DE LOJA" de autoría del señor Gilson Orlando Quezada Guartizaca, con cedula de identidad número 1150006979, estudiante de la Carrera de Ingeniería en Computación de la facultad de Energía, las Industrias y los Recursos Naturales no Renovables de la Universidad Nacional de Loja.

Es todo cuanto puedo certificar en honor a la verdad, facultando al interesado, hacer uso del presente, en lo que considere pertinente.

Lcdo. Luis Hernán Sánchez Villa
CI: 1102404314
Senescyt: 1008-02-154120