UNIVERSIDAD NACIONAL AGRARIA DE LA SELVA

FACULTAD DE INGENIERIA EN INFORMATICA Y SISTEMAS

DEPARTAMENTO ACADÉMICO DE CIENCIAS EN INFORMÁTICA Y SISTEMAS

"MANTENIBILIDAD DE PRODUCTOS DE SOFTWARE SEGÚN EL MODELO SQUARE ISO/IEC 25000"

Tesis

Para optar el título de:

INGENIERO EN INFORMATICA Y SISTEMAS

SAMUEL RICARDO PARDO MESIAS

ASESORES:

ING. RONALD EDUARDO IBARRA ZAPATA

ING. BRIAN CESAR PANDO SOTO

TINGO MARÍA – PERÚ

2018

Universidad Nacional Agraria de la Selva Facultad de Ingeniería en Informática y Sistemas	**ACTA DE SUSTENTACIÓN DE TESIS N.º 02-2018**	**COMISION DE GRADOS Y TITULOS** **Fecha:** ...../...../2018

PARTE 1. FASE INICIAL

Siendo las 5:10 pm.... horas del día ...26..... de Setiembre de 2018; en la Sala de Grados de la UNAS, se instala el jurado calificador conformado por:

Jurado 1. ING. PEDRO CRISOLOGO TRUJILLO NATIVIDAD (Presidente)

Jurado 2. MSC. NOEL JUIPA CAMPO

Jurado 3. ING. WILMER BERMUDEZ PINO

Oficializado mediante **Resolución N.º 100-2018-D-FIIS-UNAS** del 22 de agosto de 2018, para el proceso de sustentación del informe final de Tesis del bachiller **Samuel Ricardo PARDO MESIAS,** titulado: **"MANTENIBILIDAD DE PRODUCTOS DE SOFTWARE SEGÚN EL MODELO SQUARE ISO/IEC 25000".** ASESOR**: Ing. Ronald Eduardo Ibarra Zapata,** CO-ASESOR: **Ing. Brian Cesar Pando Soto.**

Se manifiesta que el bachiller cumple con los requisitos exigidos de Ley y se le invita a disertar su Tesis por espacio de 30 minutos, asimismo se dispondrá de igual tiempo para la absolver preguntas y sugerencias.

PARTE 2. FASE DE PREGUNTAS Y RESULTADO

Culminada la exposición se inicia la fase de preguntas por parte del jurado calificador; también se invita a los asistentes a formular preguntas sobre el tema de Tesis.

Absueltas todas las peticiones, el jurado calificador procede a deliberar en privado la calificación y resultado.

Concluida la deliberación y en presencia del público asistente, el jurado calificador anuncia que el resultado de la Sustentación de Tesis es: Aprobado por unanimidad.

(NOTA: consignar una de la siguientes: DESAPROBADO, APROBADO POR MAYORIA o APROBADO POR UNANIMIDAD)

Con calificativo de: ...Bueno.....

(NOTA: consignar una de la siguientes: EXCELENTE, MUY BUENO, BUENO, DEFICIENTE, MUY DEFICIENTE)

Por lo que se comunicará a las instancias correspondientes para el trámite respectivo.

PARTE 3. CONFORMIDAD

De todo lo mencionado se firma al pie en señal de conformidad, siendo las 6:30 horas.

Firma:	Firma:	Firma:
Jurado 1: Pedro Crisologo Natividad	Jurado 2: Noel Juipa Campo	Jurado 3: Wilmer Bermúdez Pino
Firma:		
Sustentante: Samuel Ricardo Pardo Mesias		

DEDICATORIA

A DIOS

Por lo que soy y lo que tengo, por darme fuerzas, salud y voluntad, por siempre enseñarme el camino correcto a seguir y por ayudarme a mantener vivo el deseo de seguir adelante.

A MI PADRE: Ricardo Pardo Caja.

Por ser el mejor ejemplo de vida, por su amor, por su apoyo incondicional y por estar siempre a mi lado.

A MI ABUELITA: Dominga Caja Borquez.

Por confiar en mí, por su apoyo incondicional, por su amor, por ser mi motivo de seguir superándome día a día y por cuidarme siempre.

A MIS HERMANOS: Santiago, Liliana, Lanyer y Benny.

Por confiar en mí, porque estuvieron apoyándome moralmente y por ser mis motivos a seguir superándome día a día.

A MIS ABUELITOS y MI TIO: Pedro Caja Borquez, Santos Caja Borquez y Daniel Espinoza Caja.

Porque estuvieron apoyándome incondicionalmente, por enseñarme el camino correcto y porque siempre estuvieron a mi lado.

A MIA TIA: Marcela Narváez Espinoza.

Por ser un ejemplo de vida, por confiar en mi desde el inicio de mi carrera y por el apoyo incondicional que siempre me dio.

A MI FAMILIA HERMANOS CARLOS NARVAEZ: Alan, Jhosep, Junior, Omar, Steven, Román y Jhoana.

Porque confiaron en mí, porque estuvieron apoyándome moralmente, por ser ejemplo de superación y por su solidaridad que siempre tuvieron.

AGRADECIMIENTOS

A DIOS, por darme sabiduría y fe para seguir adelante con mis metas.

A mi padre (Ricardo), por brindarme el apoyo moral de seguir adelante estando en las buenas y en las malas conmigo. A mi madre (Elsa), por brindarme el apoyo en el inicio de mi carrera profesional y por cuidarme.

A los Profesores de la Facultad de Ingeniería en Informática y Sistemas, en especial a mis asesores el Ing. Ronald Eduardo Ibarra Zapata y el Ing. Brian Cesar Pando Soto por brindarme todo el conocimiento impartido, orientaciones y asesorías.

A mi tía Marcela Narvaez Espinoza, mis primos Junior Carlos Narvaez y Jhosep Carlos Narvaez por la confianza, por el apoyo incondicional desde el inicio de mi carrera y por la solidaridad que siempre tuvieron.

A mis compañeros de la Promoción FIIS 2007, por brindarme su ayuda en momentos difíciles y sobre todo por haber compartido una gran amistad con todos ellos.

A mis amigos Norris y Erick por compartir conocimientos mutuos, por motivarme y porque estuvieron apoyándome en cada momento de mis días en la Universidad. También personas que me apoyaron incondicionalmente a lo largo de mi carrera universitaria.

A toda la familia de la Empresa Lead Working Partner por darme la oportunidad de desarrollarme profesionalmente, por compartir grandes momentos y porque somos un gran equipo.

A las señoras Esther López Cabra y Julia Bailón Trujillo por su apoyo incondicional, por sus consejos, por su solidaridad que siempre tuvieron.

INDICE

INDICE DE CUADROS

INDICE DE FIGURAS

RESUMEN

Está investigación enfoca en el problema de la industria del software respecto al desconocimiento de la calidad de software que se está produciendo, omitiendo la calidad interna del producto, dado que no es un aspecto de mayor interés por muchos usuarios y clientes de software. El objetivo del estudio fue medir el grado de la mantenibilidad de los productos de software basado en el modelo de calidad de la ISO/IEC 25000. Se realizó un mapeo de modelos propuestos para la medición de la mantenibilidad y uno de ellos fue elegido, consiguiendo las propiedades de calidad y las herramientas para hacer el análisis de código fuente. El tipo de investigación es aplicada, nivel de investigación es descriptivo simple y con diseño descriptivo. La estrategia de medición de la mantenibilidad está basada en casos de estudio aplicados a productos de software producidos por egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva. Se obtuvieron resultados positivos indicando que los productos de software evaluados tienen una mantenibilidad alta, pero deben mejorar subcaracterísticas como capacidad de ser probado y modularidad.

ABSTRACT

This research focuses on the problem of the software industry regarding the lack of knowledge of software quality that is occurring, omitting the internal quality of the product, since it is not an aspect of greater interest for many users and software customers. The objective of the study was to measure the degree of maintainability of software products based on the quality model of the ISO / IEC 25000. A mapping of proposed models was carried out for the measurement of maintainability and one of them was chosen, getting the quality properties and the tools to do the source code analysis. . The type of research is applied, level of investigation is simple descriptive and with descriptive design. The measurement strategy of maintainability is based on case studies applied to software products produced by graduates of the Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva. Positive results were obtained indicating that the evaluated software products have high maintainability, but they must improve sub-characteristics such as ability to be tested and modularity.

INTRODUCCION

La evaluación de la calidad del software es un campo de gran actividad tanto investigadora como en el sector industrial, la mayor parte del esfuerzo realizado se ha centrado en la calidad de los procesos, habiéndose desarrollado gran cantidad de modelos y estándares de referencia, evaluación y mejora de procesos de software: ISO 90003, ISO 12207, ISO 15504, CMM, CMMI, IDEAL, SCAMPI.Sin embargo existe poca información de trabajos de investigación sobre la evaluación de la calidad de producto de software basada en la nueva familia de normas ISO/IEC 25000 (Rodríguez y Piattini, 2014).

Generalmente se presta mayor atención a la calidad externa de producto de software y no a la calidad interna, lo cual tiene como consecuencia mayores costos de mantenimiento de producto software. Así mismo al no contar con información sobre la mantenibilidad de los productos de software, por lo menos en el contexto nacional, es difícil poder realizar un proceso de mejora continua por un lado en la industria de software y por otro lado en la academia (universidades, institutos que forman profesionales en aspectos relacionados al desarrollo de software).

El contenido de la presente tesis consta de diez capítulos: En el capítulo I se presentan los aspectos generales de la tesis. En el capítulo II se describe el marco teórico en el cual se definen los conceptos básicos que fueron aplicados en el desarrollo de la tesis. En el capítulo III se describe los materiales y métodos, detallando el tipo de investigación. En el capítulo IV se describe los resultados de la tesis. En el capítulo V se discuten los resultados de la tesis. Finalmente se incluyen las conclusiones, recomendaciones, referencias bibliológicas y los anexos que se utilizaron en el desarrollo de la presente tesis.

I. ASPECTOS GENERALES

1.1. CONTEXTO DEL PROBLEMA

Desde hace varias décadas se han elaborado trabajos de investigación, normas y estándares, con el objetivo de crear modelos, procesos y herramientas de evaluación de la calidad del producto software. Entre los más representativos podemos mencionar a la familia de normas ISO/IEC 25000. Sin embargo, la certificación de la calidad del producto software sigue siendo relativamente joven de la Ingeniería del Software, en la que todavía no existe un consenso definitivo (Rodríguez, Fernández, y Pedreira, 2015).

Perú es uno de los países latinoamericanos que no tuvo mucho éxito en levantar una industria de software de calidad en comparación con los otros países de la región. Esta falta de éxito se debe a la poca importancia que se le viene dando a la calidad de software, principalmente por tema de costos y además por la no existencia de profesionales conocedores de herramientas o tecnologías que automaticen el proceso (Valdivia Huamán, 2017).

La mantenibilidad es un factor significativo en el éxito económico del producto de software. Además es un atributo importante de calidad, pero es difícil de estimar, ya que involucra predicciones acerca de cambios futuros. Por lo que si el modelo de predicciones de mantenibilidad es exacto, entonces el diseño de correcciones podría ser adoptado y ayudar a reducir esfuerzos futuros de mantenibilidad (Pérez-gonzález et al., 2015) .

Por otro lado la mantenibilidad es una de las características más demandadas hoy en día por los clientes de software, que piden que el producto software que se les desarrolle pueda ser después mantenido por ellos mismos o incluso por un tercero. El mantenimiento supone una de las fases del ciclo de vida de desarrollo de software más costosa llegando a alcanzar el 60% (Rodríguez y Piattini, 2014).

Generalmente se presta mayor atención a la calidad externa de producto de software y no a la calidad interna, lo cual tiene como consecuencia mayores costos de mantenimiento de producto software. Así mismo al no contar con información sobre la mantenibilidad de los productos de software, por lo menos en el contexto nacional, es difícil poder realizar un proceso de mejora continua por un lado en la industria de software y por otro lado en la academia (universidades, institutos que forman profesionales en aspectos relacionados al desarrollo de software).

1.2. FORMULACION DEL PROBLEMA

Se desconoce la calidad de producto de software que se viene produciendo en el contexto nacional, siendo la mantenibilidad una característica importarte a tomar en cuenta, por lo cual la pregunta principal que formula el presente trabajo es:

¿Cuál es el grado de mantenibilidad de productos de software desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva teniendo como referencia el modelo Square ISO/IEC 25000?

1.3. JUSTIFICACION

La presente investigación, por cuenta propia y además por posibles replicas en otras organizaciones, permitirá a los productores de software, tomar decisiones para mejorar sus procesos de desarrollo para asegurar la calidad interna del producto en cuanto a la mantenibilidad. Esto podría desencadenar un efecto positivo en cuanto a reducción de costos, satisfacción de cliente y con ello competir con grandes empresas de la industria de desarrollo de software.

Esta investigación permitirá proponer nuevas mejoras en la formación profesional en las universidades, institutos que forman profesionales en aspectos relacionados al desarrollo de software a partir de conocer que tanto mantenible es la producción de software de sus egresados.

Adicionalmente la presente investigación generara conocimiento empírico de modelos propuestos para la medición de la mantenibilidad, herramientas que permiten la automatización del proceso de análisis de código fuente de los productos de software y así como también las métricas y/o propiedades de calidad para cada subcaracterística de la mantenibilidad.

1.4. PROBLEMA GENERAL Y ESPECIFICOS

1.4.1. PROBLEMA GENERAL

¿Cuál es el grado de mantenibilidad de los productos de software desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva teniendo como referencia el modelo Square ISO/IEC 25000?

1.4.2. PROBLEMAS ESPECIFICOS

¿Cuál es el grado de modularidad de los productos de software desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva teniendo como referencia el modelo Square ISO/IEC 25000?

¿Cuál es el grado de reusabilidad de los productos de software desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva teniendo como referencia el modelo Square ISO/IEC 25000?

¿Cuál es el grado de analizabilidad de los productos de software desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva teniendo como referencia el modelo Square ISO/IEC 25000?

¿Cuál es el grado de capacidad de ser modificado de los productos de software desarrollados por los egresados de la Facultad de Ingeniería en

Informática y Sistemas de la Universidad Nacional Agraria de la Selva teniendo como referencia el modelo Square ISO/IEC 25000?

¿Cuál es el grado de capacidad de ser probado de los productos de software desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva teniendo como referencia el modelo Square ISO/IEC 25000?

1.5. OBJETIVOS

1.5.1. OBJETIVOS GENERAL

Determinar el grado de mantenibilidad de los productos de software desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva teniendo como referencia modelo Square ISO/IEC 25000.

1.5.2. OBJETIVOS ESPECIFICOS

Determinar el grado de “modularidad” de los productos de software desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva teniendo como referencia modelo Square ISO/IEC 25000.

Determinar el grado de “reusabilidad” de los productos de software desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva teniendo como referencia modelo Square ISO/IEC 25000.

Determinar el grado de “analizabilidad” de los productos de software desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva teniendo como referencia modelo Square ISO/IEC 25000.

Determinar el grado de “capacidad de ser modificado” de los productos de software desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva teniendo como referencia modelo Square ISO/IEC 25000.

Determinar el grado de “capacidad de ser probado” de los productos de software desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva teniendo como referencia modelo Square ISO/IEC 25000.

1.6. HIPOTESIS Y VARIABLES

1.6.1. HIPOTESIS GENERAL

El grado de mantenibilidad de los productos de software desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva de software teniendo como referencia modelo Square ISO/IEC 25000 es baja.

1.6.2. OPERACIONALIZACION DE VARIABLES E INDICADORES

Tabla 1. Operacionalización de variables

Variables	Dimensiones	Indicadores
X: Productos de software	Tecnología	Lenguaje de programación, motor de base datos, frameworks.
	Dominio de aplicación	Categoría de dominio
	Modularidad	Diseño, duplicaciones, cobertura
	Reusabilidad	Mala práctica, redundante, diseño, duplicaciones
Y: Mantenibilidad	Analizabilidad	Mala práctica, redundante, complejidad cognitiva, confuso, diseño, dificultad encontrada, falto de ingenio

	Capacidad de ser modificado	Mala práctica, obsoleto, diseño, duplicaciones
	Capacidad de ser probado	Redundante, diseño, dificultad encontrada, cobertura

Fuente: Elaboración propia.

Tabla 2. Explicación de variables

Variables	**Dimensiones**	**Definición**
X: Productos de software	Tecnología	Hace referencia a la forma de cómo se presenta el software en cuanto a frameworks, motor de base de datos, lenguaje programación utilizados para su desarrollo.
	Dominio de aplicación	Hace referencia al rubro o sector que pertenece el software como bancario, comercial, educativo, gestión administrativa, académico.
Y: Mantenibilidad	Modularidad	Se define como el grado, en el que un sistema se encuentra dividido en módulos de forma que el impacto que causa una modificación en un módulo sea mínimo para el resto. (Rodríguez y Piattini, 2014)
	Reusabilidad	Se define como el grado en que un activo (módulo, paquete, clase, etc.) puede ser usado en más de un sistema o en la construcción de otros activos. (Rodríguez y Piattini, 2014)
	Analizabilidad	Se define como la facilidad para identificar las partes de un sistema que se deben modificar debido a deficiencias o fallos, o la capacidad de evaluar el

	impacto que puede provocar un cambio en el sistema. (Rodríguez y Piattini, 2014)
Capacidad de ser modificado	Se define como el grado en el que se pueden realizar cambios en un producto software de forma efectiva y eficiente, sin introducir defectos ni degradar su rendimiento. (Rodríguez y Piattini, 2014)
Capacidad de ser probado	Se define como la facilidad para establecer criterios de prueba para un sistema y realizar las pruebas que permitan comprobar que se cumplen esos criterios. (Rodríguez y Piattini, 2014)

Fuente: Elaboración propia.

II. REVISION DE LITERATURA

2.1. ANTECEDENTES

(Galán Pausic, 2017)**. ESTUDIO DE LA PLATAFORMA SONARQUBE PARA LA IMPLEMENTACION DE ISO/IEC 25000.** En esta investigación se explora en profundidad el modelo y las métricas propuestas por la familia de la norma ISO/IEC 25000 y paralelamente una herramienta de software libre para medir las propiedades de calidad en forma estática a partir del código fuente, llamada Sonarqube. También se analizó el grado de cobertura que Sonarqube da a las características planteadas en el modelo de la ISO/IEC 25000. A partir de los puntos de coincidencia se elaboró una propuesta de fórmulas que permiten calcular la Mantenibilidad tomadas del modelo de la ISO/IEC 25010 utilizando Sonarqube. La propuesta se aplicó en un producto de software Wild Pizzería teniendo como resultado una mantenibilidad de 87.15% y una falta de mantenibilidad de 12.85%, donde los problemas que más afectan a este caso de estudio están relacionados en primer orden con la falta de cobertura de test unitarios, luego duplicaciones de código y problemas de diseño.

(Balseca Chisaguano, 2009)**. EVALUACIÓN DE CALIDAD DE PRODUCTOS SOFTWARE EN EMPRESAS DE DESARROLLO DE SOFTWARE APLICANDO LA NORMA ISO/IEC 25000.** En esta investigación aplicó el modelo de evaluación de la calidad ISO/IEC 25040 para evaluar la calidad del producto software LogiNotificador de la empresa Logiciel Cia. Para facilitar el proceso de evaluación se utilizó una matriz de calidad, la cual permitió al evaluador realizar la evaluación de la calidad del producto de software de una manera completa y concisa. La matriz consta de cuatro secciones calidad interna, calidad externa, calidad de uso y resultado de final del análisis de la calidad de producto de software. El resultado final del análisis de la calidad del producto software LogiNotificador, fue un valor total de 8,36 sobre 10 lo que representa que el sistema tuvo un nivel de puntuación aceptable. De acuerdo a los valores obtenidos de las características de calidad, la calidad externa obtuvo un valor de 8,63 con una puntuación de aceptable, la calidad de uso obtuvo un

valor de 9,07 con una puntación de cumple con los requisitos y la calidad interna obtuvo un valor de 7,36 con una puntación de aceptable, el resultado de la mantenibilidad obtuvo un valor de 1,73 en un rango de 0 a 2.5 lo cual hace un 69.20% el cual resulta exitoso. Se recomendó utilizar el modelo de calidad ISO/IEC 25000 para evaluar productos software, ya que el modelo presenta una mayor información sobre las características de calidad de un producto software y por ser un modelo integrado con el proceso de evaluación.

(Irrazábal, 2012)**. CONSTRUCCIÓN DE UN ENTORNO PARA LA MEDICIÓN AUTOMATIZADA DE LA CALIDAD DE LOS PRODUCTOS SOFTWARE.** En esta investigación se propone un entorno metodológico Kybele Environment Measurement Information System (KEMIS) basado en software libre para implementar un sistema de medición de la mantenibilidad software a nivel operativo, táctico y estratégico. Así como también un soporte metodológico para la evaluación de la mantenibilidad del producto software capaz de calcular el valor y el retorno de inversión de las refactorizaciones necesarias para mejorar dicha mantenibilidad. El modelo de medición está basado en la norma ISO/IEC 9126 e ISO/IEC 25010 para obtener valores indicadores de la calidad de un producto software. Por otra parte, el entorno KEMIS permitirá configurar los elementos del modelo de medición (medidas de calidad, funciones de medición, umbrales, etc.), adaptándolo a las características de cada organización. La propuesta se implanto en una empresa española que brinda servicios de tecnología de la información y telecomunicaciones teniendo como resultado obtención de métricas, a partir de la generación de manera automatizada y periódica de informes, que permiten fácilmente el cálculo de los indicadores de calidad como Densidad de Código Repetido, Densidad de Complejidad Ciclomatica, Ratio de Cobertura de Pruebas Unitarias.

(Chawla, 2015)**. SQMMA: SOFTWARE QUALITY MODEL FOR MAINTAINABILITY ANALYSIS.** En esta investigación propone un modelo de calidad SQMMA para la evaluación de la mantenibilidad que ofrece fórmulas matemáticas para cuantificar los atributos de calidad capacidad de ser

analizabilidad, modificabilidad, estabilidad y capacidad de ser probado propuestas por el modelo de medición de la norma ISO/IEC 9126. También se propuso un modelo de evaluación de la mantenibilidad basado en el modelo de calidad de la ISO/IEC 25010. La propuesta se aplicó en cuatro versiones de un producto de software Apache Software Foundation (ASF) donde los resultados de la mantenibilidad se muestran de la siguiente manera ASF el valor es 1 donde las siguientes versiones se fueron obteniendo teniendo como referencia la línea base, ASF el valor es 1.02, ASF el valor es 1.05 y ASF el valor es 1.10 usando el modelo de calidad de la ISO/IEC 25010.

(Rodríguez y Piattini, 2014)**. ENTORNO PARA LA EVALUACION Y CERTIFICACION DE LA CALIDAD DE PRODUCTO SOFTWARE.** En esta investigación propone un entorno integrado que permite llevar a cabo la evaluación de la calidad de producto de software basándose en la familia de norma ISO/IEC 25000. Donde dicho entorno se ha validado en un laboratorio de evaluación acreditado y se han realizado evaluaciones de varios proyectos piloto con una entidad de certificación, cuyos resultados han sido la obtención de los primeros productos de software certificados en mantenibilidad a nivel internacional.

2.2. MARCO TEORICO

2.2.1. INGENIERIA DE SOFTWARE

Según (Bayona Zubieta, Pineda Samaca, y Pardo Mahecha, 2016), la ingeniería de software es una nueva área de la ingeniería y es considerada como una disciplina que se encarga de crear y mantener las aplicaciones de software haciendo uso de tecnologías, prácticas, métodos y técnicas para el desarrollo de programas informáticos con calidad, apoyándose en las herramientas y los procedimientos que provee la informática.

Según (INTECO, 2009), la ingeniería del software como aplicación práctica del conocimiento científico al diseño y construcción de programas de computadora y a la documentación asociada requerida para desarrollar, operar

y mantenerlos. Se conoce también como desarrollo de software o producción de software.

Según (Pressman, 2010), la ingeniería de software como aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software. La ingeniería de software consta de varias capas como se puede apreciar en la Figura 1, cualquier enfoque de ingeniería debe basarse en un compromiso organizacional con la calidad. El fundamento en el que se apoya la ingeniería de software es el compromiso con la calidad.

Figura 1. Capas de la Ingeniería de Software
Fuente: (Pressman, 2010).

El fundamento para la ingeniería de software es la capa proceso. El proceso de ingeniería de software es el aglutinante que une las capas de la tecnología y permite el desarrollo racional y oportuno del software de cómputo. El proceso define una estructura que debe establecerse para la obtención eficaz de tecnología de ingeniería de software. El proceso de software forma la base para el control de la administración de proyectos de software, y establece el contexto en el que se aplican métodos técnicos, se generan productos del trabajo (modelos, documentos, datos, reportes, formatos), se establecen puntos de referencia, se asegura la calidad y se administra el cambio de manera apropiada.

Los métodos de la ingeniería de software proporcionan la experiencia técnica para elaborar software e incluyen un conjunto amplio de tareas, como comunicación, análisis de los requerimientos, modelación del diseño, construcción del programa, pruebas y apoyo. Los métodos de la

ingeniería de software se basan en un conjunto de principios fundamentales que gobiernan cada área de la tecnología e incluyen actividades de modelación y otras técnicas descriptivas. Las herramientas de la ingeniería de software proporcionan un apoyo automatizado o semiautomatizado para el proceso y los métodos. Cuando se integran las herramientas de modo que la información creada por una pueda ser utilizada por otra, queda establecido un sistema llamado ingeniería de software asistido por computadora que apoya el desarrollo de software.

2.2.2. CALIDAD DE SOFTWARE

Según (Pressman, 2010) es el proceso eficaz de software que se aplica de manera que se crea un producto útil que proporciona valor medible a quienes lo producen y a quienes lo utilizan.

Según (Roa, Morales, y Gutierrez, 2015) es la concordancia con los requerimientos funcionales cumpliendo con las políticas y el rendimiento establecidos por el negocio. Desarrollar un software con calidad implica la utilización de estándares, metodologías y procesos para análisis, diseño, programación y pruebas, con el fin de lograr confiabilidad, efectividad y productividad para el control de la calidad del software.

2.2.2.1. CALIDAD DE PROCESO

Según (Galán Pausic, 2017) las compañías a menudo consiguen las mejoras en la calidad definiendo y desarrollando un conjunto de capacidades estratégicas. Los problemas aparecen cuando la compañía trata de transferir las lecciones de calidad aprendidas en un proceso de proceso de desarrollo de software. En el caso de la manufactura se desarrollan modelos de calidad recolectando gran cantidad de datos de los puestos de trabajo donde se realiza la misma función una y otra vez.Los modelos de desarrollo de software no pueden ser construidos de la misma forma que los modelos de manufactura, con su dependencia en las lecciones aprendidas de las repeticiones masivas del

mismo proceso. Los modelos de software proveen la habilidad de aprender de otros proyectos de desarrollo de software. Una empresa puede manejar la calidad de un sistema de software de dos formas. Primero puede mejorar la efectividad del proceso de desarrollo reduciendo la cantidad de reelaboración y reusando componentes a través de los segmentos de un proyecto o de diferentes proyectos. Segundo, puede desarrollar e implementar planes para mejoras continuas basadas en hechos y datos.

2.2.2.2. CALIDAD DE PRODUCTO

Según (Galán Pausic, 2017) hay poca evidencia de que cumplir un modelo de procesos asegure la calidad del producto software y aunque la estandarización de los procesos garantiza la uniformidad en la salida de los mismos, podría llegar a darse el caso de que institucionalizara la creación de malos productos. La idea de que las evaluaciones del software deberían basarse en evidencias directas del propio producto y no sólo en evidencias de los procesos que se utilizan para su construcción se está extendiendo en el sector. Por ello, es cada día mayor el número de organizaciones que se interesan no sólo por la calidad de los procesos que se siguen en el desarrollo de software, sino también por la calidad de los productos que desarrollan y adquieren. Y es que, una vez que el producto ha sido implantado en sus instalaciones se pueden encontrar con graves problemas de calidad y dificultades para corregirlo, adaptarlo o evolucionarlo.

2.2.3. NORMAS DE CALIDAD

(ISO 25000, 2017) Son documentos que proporcionan requisitos, especificaciones, directrices o características que se pueden utilizar de manera consistente para asegurar que los materiales, productos, procesos y servicios son adecuados para su propósito.

2.2.3.1. NORMA ISO/IEC 25000

(Roa et al., 2015) Las normas ISO/IEC 25000 también llamadas SQuare (System and Software Quality Requirements and Evaluation) están

conformadas por las normas ISO/IEC 9126 e ISO/IEC 14598, surgen para crear modelos, métricas, procesos y herramientas de evaluación de calidad del software como producto, por medio de la especificación de los requisitos, podemos ver la división de la ISO/IEC 25000 en la Tabla 3 .

Es una familia de normas que tiene por objetivo la creación de un marco de trabajo común para evaluar la calidad del producto software.

Tabla 3. Divisiones de la ISO/IEC 25000

ISO/IEC 25000 (SQuaRE)		
Modelos N	**Modelos**	**Nombres**
Administración de calidad	ISO 25000	Guía de SQUARE
ISO 2500n	ISO 25001	Planeación y gestión
Modelo de calidad 2501n	ISO 25010	Calidad del modelo
	ISO 25020	Calidad de las métricas
Medidas de Calidad 2502n	ISO 25021	Modelo de referencia de las métricas primitivas de medición
	ISO 25022	Medidas de calidad interna
	ISO 25023	Medidas de calidad externa
Requerimientos de calidad 2503n	ISO 25024	Medidas de calidad de uso
	ISO 25030	Requerimientos de calidad
	ISO 25040	Visión General de valuación de calidad
Evaluación de calidad 2504n	ISO 25041	Módulo de evaluación
	ISO 25042	Proceso para desarrolladores
	ISO 25043	Proceso para adquirientes
	ISO 25044	Proceso de evaluadores

Fuente: (Acosta, Espinel, y Garcia, 2017)

a. Estructura de la Familia ISO/IEC 25000

La ISO/IEC 25000 se encuentra compuesta por varias divisiones, entre las que se destacan las siguientes:

Figura 2. División de la Norma ISO/IEC 25000

Fuente: (ISO 25000, 2017)

b. Modelo de calidad

Según (Roa et al., 2015) a la hora de establecer la calidad de un producto de software es importante definir un modelo que permita realizar una evaluación detallada con una secuencia específica, que además permita estructurar los puntos a evaluar. La norma ISO 2501n presenta un modelo de calidad detallado donde incluye las características de calidad interna, externa y para la calidad en uso.

La ISO/IEC 25010 - Modelos del sistema y calidad del software: Modelo de calidad representa la piedra angular en torno a la cual se establece el sistema para la evaluación de la calidad del producto. En este modelo se determinan las características de calidad que se van a tener en cuenta a la hora de evaluar las propiedades de un producto software determinado.

La calidad del producto software se puede interpretar como el grado en que dicho producto satisface los requisitos de sus usuarios aportando de esta

manera un valor. Son precisamente estos requisitos (funcionalidad, rendimiento, seguridad, mantenibilidad, etc.) los que se encuentran representados en el modelo de calidad, el cual categoriza la calidad del producto en características y subcaracterísticas. El modelo de calidad del producto definido por la ISO/IEC 25010 se encuentra compuesto por las ocho características de calidad que se muestran en la siguiente Figura 3.

Figura 3. Estructura de la ISO/IEC 25010

Fuente: (ISO 25000, 2017)

Adecuación funcional: Permite medir la capacidad que tiene un producto de software para proveer las funciones que satisfacen requerimientos explícitos e implícitos cuando el software se usa en determinadas condiciones.

Eficiencia de desempeño: Es el comportamiento del sistema funcionalidad, capacidad, utilización de recursos y respuesta temporal. Dentro de sus características se encuentra que el sistema requiere la utilización de un mínimo de recursos (por ejemplo tiempo de CPU) para ejecutar una tarea determinada.

Compatibilidad: Es el proceso en el cual dos o más sistemas intercambian información y llevan a cabo funciones requeridas en cuanto a su entorno hardware o software compartido.

Usabilidad: Algunas de las características que la conforman son: comprensibilidad, aprendibilidad, operabilidad, atractivo, cumplimiento de usabilidad, capacidad para reconocer su adecuación, capacidad de aprendizaje, capacidad de ser usado, protección contra errores de usuario, estética de la interfaz de usuario, accesibilidad. También fácil de usar, fácil de aprender, atractivo para el usuario, conforme a normas, uso intuitivo.

Fiabilidad: Madurez, tolerancia a defectos, recuperabilidad, cumplimiento de fiabilidad. En determinadas condiciones, el software sistema mantendrá su capacidad-funcionalidad a lo largo de un periodo de tiempo.

Seguridad: Capacidad de proteger la información y los datos de manera que no puedan ser leídos o modificados por personas o sistemas no autorizados. La autenticidad, la confidencialidad y la responsabilidad son sus principales características.

Mantenibilidad: Es la medida del esfuerzo requerido para realizar cambios en los componentes de un sistema de manera efectiva y eficiente. Algunas de sus características son modularidad, analizabilidad, reusabilidad, capacidad de ser modificado, capacidad de ser probado.

Portabilidad: Es la capacidad del software de ser transferido a un nuevo entorno (software, hardware, organización). Es fácil de instalar y desinstalar, además permite ser adaptado de forma efectiva a diferentes entornos de hardware o software.

c. Medición de calidad

(Roa et al., 2015) Presenta un modelo de referencia para medir la calidad de un producto de software, por medio de definiciones matemáticas y métricas.

ISO/IEC 25020-Modelo de referencia para la medida con guía: Presenta un modelo de referencia para las medidas de calidad interna y externa.

ISO/IEC 25021-Elementos de medida de calidad: Define y especifica un conjunto de métricas para ser usadas durante el ciclo de vida del producto.

ISO/IEC 25022-Medición de la calidad en uso: Especifica las métricas para realizar la medición de la calidad del uso de un producto.

ISO/IEC 25023-Medición de sistemas y software de calidad del producto: Define específicamente las métricas para realizar la medición de la calidad de sistemas de software y productos.

ISO/IEC 25024-Medición de la calidad de los datos: Especifica las métricas para realizar la medición de la calidad de datos.

d. Requisitos de calidad

Según (Roa et al., 2015) la norma ISO/IEC 25030–Requisitos de calidad, ayuda a especificar más claramente los requisitos de calidad del producto software o como entrada del proceso de evaluación.

e. Evaluación de calidad

Según (Roa et al., 2015) define como el proceso para llevar a cabo la evaluación del producto software. Dentro de los modelos referenciales que ayudan a llevar a cabo un proceso de evaluación de calidad del producto software encontramos las siguientes:

ISO/IEC 25040–Modelo de referencia, evaluación y guía: Es un modelo de referencia para la evaluación, el cual considera las entradas, las restricciones y los recursos necesarios para obtener las salidas.

ISO/IEC 25041-Guía de evaluación para los desarrolladores: Compradores y evaluadores independientes: Describe las recomendaciones desde el punto de vista del desarrollador, los compradores y los evaluadores, para la puesta en práctica de la evaluación del producto software.

ISO/IEC 25042–Módulos de evaluación: En este módulo se tiene en cuenta la evaluación y la documentación, la estructura y el contenido que se deben utilizar para definir los módulos completos.

ISO/IEC 25045–Módulo de evaluación de recuperabilidad: Es un módulo para la evaluación de la recuperabilidad de todo tipo de información.

2.2.3.2. NORMA ISO/IEC 9126

Según (Acosta et al., 2017) la norma ISO/IEC 9126, tiene como fin cuantificar los productos de software, esta norma nos indica las características de la calidad del software y fue elaborado para cubrir las necesidades de error que genera. La norma fue diseñada en los siguientes factores: calidad de

proceso, calidad del producto, calidad del software y calidad de uso. Por otro lado, presenta dos modelos de calidad, el primero referido a la calidad interna y externa y el segundo modelo referido a la calidad en uso; a continuación se describe cada uno de ellos.

ISO/IEC 9126-1: Hace referencia al modelo de calidad de un producto de software, en primera instancia especifica las características del modelo que están divididas en seis tanto en calidad interna como externa, además dentro de estas se dividen en otras categorías. La calidad externa es aquella que es medible con el uso del producto, y la calidad interna realiza una revisión relacionada con los atributos del software como se puede ver en la Figura 4. Permite especificar y calificar la calidad de un software desde diferentes puntos de vista, los cuales se relacionan con los requerimientos, uso, adquisición, desarrollo, auditoria de software y mantenimiento.

Figura 4. Características comunes a las vistas internas y externas

Fuente: (Acosta et al., 2017)

ISO/IEC TR 9126-2:2003 (métricas externas): Suministra unos parámetros que hacen posible la medición a través de los seis atributos que tiene la norma; los parámetros externos hacen posible cuantificar el comportamiento de un sistema basada en un PC a través de software. Estos parámetros proporcionan a los verificadores, usuarios, desarrolladores y evaluadores la capacidad para evaluar la calidad del software durante su funcionamiento. Los parámetros los pueden definir los evaluadores o el área usuaria, con el fin de

obtener unos parámetros básicos y así aplicarlos a los productos, basados en el modelo estándar de calidad.

ISO/IEC TR 9126-3:2003 (métricas internas): El fundamento de esta métrica es lograr que la calidad externa y la calidad de uso pretendida, esta métrica hace posible evaluar la calidad del software relacionado con los inconvenientes presentados antes de su puesta en producción. Este tipo de métrica utiliza número de elementos de construcciones de software, donde se visualizan las sentencias de código fuente, gráficos, flujo de datos, estado de procesos, entre otros.

ISO/IEC TR 9126-4:2004 (calidad en uso): Las métricas de calidad en uso calculan los efectos de uso del software en un determinado campo de uso; estas métricas evalúan si el producto cumple con las necesidades del usuario. Las caracteristicas de la calidad en uso ser puede ver en la Figura 5.

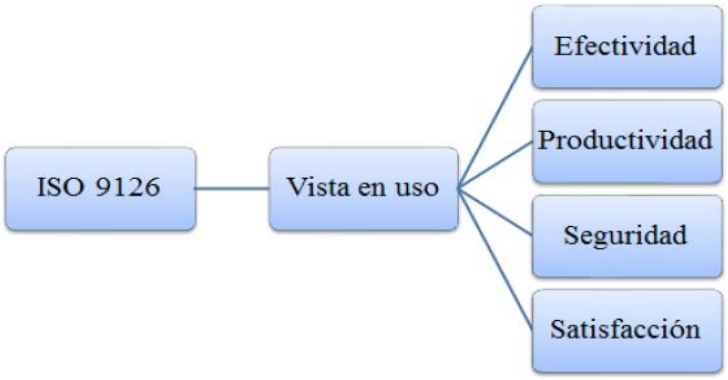


Figura 5. Características de la vista en uso
Fuente: (Valenciano López, 2015)

2.2.3.3. NORMA ISO/IEC 14598

(Acosta et al., 2017) La ISO/IEC 14598 es utilizada actualmente como una metodología para evaluar el productor de software. La norma establece una serie de etapas e involucra el marco de trabajo donde se manipula el software y se evalúa la calidad del producto, definiendo dentro de esto las siguientes características primordiales en los procesos de evaluación.

La norma ISO/IEC 14598 es un estándar que proporciona un marco de trabajo para evaluar la calidad de todo tipo de producto software e indica los

requisitos para los métodos de medición y el proceso de evaluación, la estructura de la ISO/IECE 14598 se puede ver en la Figura 6.

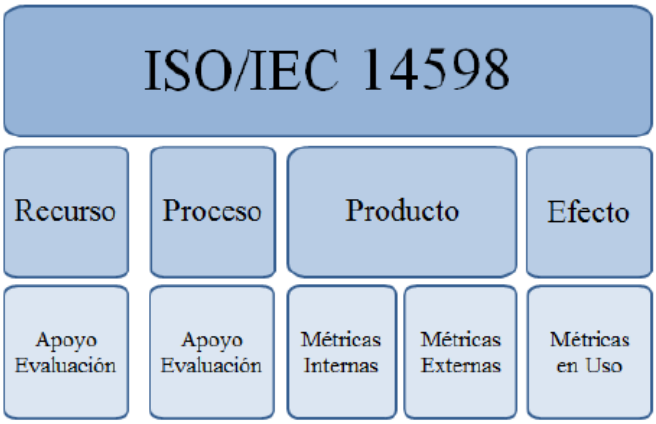


Figura 6. Estructura de la ISO/IEC 14598

Fuente: (Valenciano López, 2015)

La Norma ISO/IEC 14598 define el proceso para evaluar un producto de software, el mismo consta de seis partes:

ISO/IEC 14598-1 (visión general): Suministra una visión general de las otras cinco partes a continuación mencionadas y explica la relación entre la evaluación del producto software y el modelo de calidad definido en la norma ISO/IEC 9126.

ISO/IEC 14598-2 (planeamiento y gestión): Está compuesto por guías y requisitos para determinar las funciones de soporte como lo es la planificación y gestión de la evaluación del producto.

ISO/IEC 14598-3 (proceso para desarrolladores): Determina los requisitos y guías para la evaluación del producto software cuando esta es llevada a cabo en paralelo con el desarrollo por parte del desarrollador.

ISO/IEC 14598-4 (proceso para adquirientes): Suministra las guías y requisitos para llevar a cabo la evaluación del producto de software mostrándole a los compradores que desean adquirir o utilizar el producto existente.

ISO/IEC 14598-5 (proceso para evaluadores): Determina las guías y requisitos para la evaluación del producto software cuando la evaluación es realizada por evaluadores independientes.

ISO/IEC 14598-6 (documentación de módulos): Suministra las guías para la documentación del módulo de evaluación. La evaluación de los productos de software va relacionada con las necesidades que tengan los usuarios finales o proveedor. Tiene como actividades introducción, alcance, entradas y resultados.

2.2.4. MANTENIBILIDAD

Según (Erazo, Florez, y Pino, 2016) la mantenibilidad es uno de los atributos de calidad esenciales, ya que las tareas de mantenimiento consumen una gran proporción del esfuerzo total gastado en el ciclo de vida del software. El costo de esta etapa consume entre el 50% y el 80% de los recursos del proyecto y el 66% de los costos del ciclo de vida del software son invertidos en el mantenimiento del producto. Además, el 61% del tiempo que dedican los programadores al desarrollo es invertido en la etapa de mantenimiento, y sólo el 39% es empleado en nuevos desarrollos. Lo anterior refleja que la etapa de mantenimiento: requiere el mayor porcentaje de los costos del ciclo de vida del software, incrementa el esfuerzo realizado, impide que una gran parte del tiempo sea utilizado para nuevos desarrollos.

Dado que la etapa de mantenimiento genera altos costos durante el ciclo de vida del desarrollo de software, para facilitar su ejecución es conveniente tener en cuenta la característica de mantenibilidad de software. ISO/IEC 25010 define la mantenibilidad como el grado de efectividad o eficiencia con la que un producto o sistema puede ser modificado. La mantenibilidad de software se descompone en una serie de atributos que pueden ser considerados durante diferentes etapas del proceso de desarrollo. ISO/IEC 25010 define atributo como propiedad inherente o característica de una entidad que puede ser distinguida cualitativa o cuantitativamente por medios humanos o automatizados (Erazo et al., 2016).

La mantenibilidad de un producto software es afectada por principios de diseño y arquitectura, describe varios factores propuestos por diferentes investigadores en modelos de mantenibilidad de software, los cuales son de gran importancia en las evaluaciones de mantenibilidad. Entre estos factores se destacan: la estabilidad, facilidad de cambio, facilidad de análisis y facilidad de prueba, establecidos por la norma ISO 9126.

(Valenciano López, 2015) la mantenibilidad es uno de los principales atributos de calidad de un sistema software, se conoce como una sub-dimensión de calidad y se evalúa durante la revisión del producto. Se define como la facilidad con que un sistema software o componente puede ser modificado para corregir faltas, mejorar rendimiento u otros atributos, o adaptar a un entorno cambiante, pero que se puede resumir como fácil de entender y cambiar. Una aplicación puede ser usable, rápida, eficiente y estética todas ellas características relativas al estado o comportamiento actuales. Pero el software tiene una fuerte componente de evolución en el tiempo y ahí es donde entra la mantenibilidad.

Esta característica, que afecta enormemente al futuro de una aplicación, es obviada a menudo, a pesar de ser una de las que más diferencia el producto creado por un desarrollador aficionado del creado por uno profesional. Si queremos adaptar la aplicación a una evolución del sistema operativo, a una nueva resolución de pantalla, añadir, cambiar o corregir una funcionalidad estamos poniendo a prueba la mantenibilidad del software. Durante el desarrollo de una aplicación software se recomienda hacer las cosas sencillas, claras, que se puedan entender fácilmente.

2.2.5. SUBCARACTERISICAS DE LA MANTENIBILIDAD

2.2.5.1. MODULARIDAD

Según (Rodríguez y Piattini, 2014) define como el grado, en el que un sistema se encuentra dividido en módulos de forma que el impacto que causa una modificación en un módulo sea mínimo para el resto.

2.2.5.2. REUSABILIDAD

Según (Rodríguez y Piattini, 2014) define como el grado en que un activo (módulo, paquete, clase, etc.) puede ser usado en más de un sistema o en la construcción de otros activos.

2.2.5.3. ANALIZAILIDAD

Según (Rodríguez y Piattini, 2014) define como la facilidad para identificar las partes de un sistema que se deben modificar debido a deficiencias o fallos, o la capacidad de evaluar el impacto que puede provocar un cambio en el sistema.

2.2.5.4. CAPACIDAD DE SER MODIFICADO

Según (Rodríguez y Piattini, 2014) define como el grado en el que se pueden realizar cambios en un producto software de forma efectiva y eficiente, sin introducir defectos ni degradar su rendimiento.

2.2.5.5. CAPACIDAD DE SER PROBADO

Según (Rodríguez y Piattini, 2014) define como la facilidad para establecer criterios de prueba para un sistema y realizar las pruebas que permitan comprobar que se cumplen esos criterios.

2.2.6. EVALUACION DE LA CALIDAD DE PRODUCTO DE SOFTWARE

2.2.6.1. MODELO DE CALIDAD

Según (Rodríguez et al., 2015) el objetivo del modelo es definir las características del producto software que se pueden evaluar y que por tanto influyen en la calidad del mismo, es decir, el modelo deja claro los puntos de vista desde los que se puede considerar la calidad de un producto software. Es importante remarcar que el modelo define el qué evaluar (características de calidad), pero no el cómo evaluarlas. Aspectos como la funcionalidad que tiene un producto, la usabilidad, la seguridad o la mantenibilidad del producto software podrían ser algunos de estos tipos de características.

Es importante que el modelo no solo defina las características de calidad, sino que también desglose éstas en subcaracterísticas de más bajo nivel y a su vez estas en indicadores y métricas que se puedan medir a partir de los elementos del producto software. El modelo también debe identificar, si es que existen, las relaciones entre las características de calidad del producto. De manera que podamos saber si mejorar una característica supone mejorar a su vez otra, o si por el contrario una mejora en una característica (Rodríguez et al., 2015).

2.2.6.2. PROCESO DE EVALUACION

Se dice (Rodríguez et al., 2015) el objetivo del proceso de evaluación es definir las actividades que se deben realizar para poder llevar a cabo la evaluación de la calidad del producto software. Es importante remarcar que el proceso de evaluación define el cómo evaluar, pero no el qué evaluar, que como veíamos antes estaba definido por el modelo.

El proceso de evaluación suele iniciarse por la selección de las partes del producto software que se van a evaluar: el código fuente, los requisitos, los modelos, los casos de pruebas, el producto en ejecución, etc. podrían ser algunos de los ejemplos de partes del producto software que se pueden evaluar. Además, durante el proceso se debe también identificar qué

características se van a evaluar. Características de calidad que se han definido previamente en un modelo de calidad como hemos visto en el apartado anterior, de ahí la relación entre el proceso de evaluación y el modelo de calidad. El proceso de evaluación debe definir el conjunto de pasos que el evaluador debe seguir, identificar las herramientas que se utilizarán, así como identificar qué personas participarán en la evaluación y cuáles serán las actividades en las que participarán (Rodríguez et al., 2015).

2.2.6.3. ENTORNO TECNOLOGICO

Se dice (Rodríguez et al., 2015) el objetivo del entorno tecnológico es precisamente el de asistir al proceso de evaluación y a la recolección de las métricas y umbrales definidos en el modelo de calidad. Las herramientas que forman este tipo de entornos sirven para facilitar la obtención de los datos, ya sea parseando automáticamente el producto software o permitiendo al evaluador introducir los datos de una manera amigable. Estas herramientas deben poder también automatizar los algoritmos de medición, de manera que a partir de las métricas base que se hayan tomado, se puedan ir escalando dichos valores para obtener los indicadores de calidad. Además, estas herramientas también deben permitir presentar los resultados de una manera entendible dependiendo el público objetivo de los mismos.

Dentro de este tipo de herramientas hay que diferenciar aquellas que realizan medición de la calidad del producto software, obteniendo métricas de bajo nivel e indicadores. De aquellas herramientas que evalúan la calidad del producto software, escalando las métricas anteriores para dar una valoración de las subcaracterísticas y características de calidad del modelo. De aquí la relación también entre el modelo y el entorno tecnológico (Rodríguez et al., 2015).

2.3. DEFICIONES OPERACIONALES

2.3.1. PROCESO DE SOFTWARE

Es un conjunto de actividades relacionadas que conduce a la creación de un producto de software. Estas actividades pueden incluir el desarrollo de software utilizando un lenguaje de programación.

2.3.2. PROYECTO DE SOFTWARE

Es el procedimiento del desarrollo de software, desde la recogida de requisitos, pasando por las pruebas y el mantenimiento, y llevado a cabo en acorde a las metodologías de ejecución en un tiempo estimado para lograr el producto software deseado.

2.3.3. PRODUCTO DE SOFTWARE

Es un conjunto de programas de cómputo y documentación asociada que forman parte de un sistema de cómputo. Los productos de software se pueden desarrollar para un cliente en particular o un mercado general.

2.3.4. HERRAMIENTA DE ANALSIS ESTATICO

Son herramientas de software licenciado o software libre que se utilizaran para evaluar el código fuente de los productos de software.

2.3.5. METRICA

Es una característica de un producto de software, documentación de sistema o proceso de desarrollo que puede medirse de manera objetiva. También las métricas son usadas para medir los atributos internos de un producto de software.

2.3.6. PROPIEDADES DE CALIDAD

Las propiedades medibles relacionadas con la calidad se llaman propiedades de la calidad, las cuales tienen asociadas mediciones. Para llegar a medir una característica de calidad o una subcaracterística será necesario identificar una colección de propiedades que juntas cubran las características o

subcaracterísticas, obtener mediciones de calidad para cada una y combinarlas computacionalmente (Galán Pausic, 2017).

2.3.7. CALIDAD

Es un conjunto de características inherentes de un producto para cumplir los requisitos de los clientes y de otras partes interesadas.

2.3.8. CARACTERISTICAS DE CALIDAD

Son un conjunto de cualidades que se tienen que tomar en cuenta al momento de evaluar la calidad de un producto de software.

2.3.9. SUBCARATERICTICAS DE CALIDAD

Son un conjunto de cualidades que pertenecen a una caracteristica en su forma invidivial y que estas permiten evaluar al producto de software.

2.3.10. ENTORNO TECNOLOGICO

Es el conjunto de herramientas que automatizan el proceso de analisis de código fuente de producto de software para obtener los resultados de las propiedades de calidad.

2.3.11. MODELO DE CALIDAD

Es el conjunto de características del producto de software que pueden evaluar, para la investigación se tomó la característica de mantenibilidad basado en la ISO/IEC 25010.

2.3.12. MATENIBILIDAD

Es facilidad o dificultad para modificar producto de software, ya sea a causa de un arreglo que se debe realizar o de una mejora que se desea implementar.

III. MATERIALES Y METODOS

3.1. TIPO Y DISEÑO

3.1.1. TIPO DE INVESTIGACION

Según (Espinoza Montes, 2014) Investigación Aplicada, porque se utilizó una propuesta del modelo de medición de la mantenibilidad, donde a través de este se evaluaron los productos de software que fueron desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva.

3.1.2. NIVEL DE INVESTIGACION

Según (Espinoza Montes, 2014) Descriptivo, porque se describió la mantenibilidad de los productos de software, siendo estas evaluados sin manipular las variables de investigación.

3.1.3. DISEÑO DE INVESTIGACION

Según (Espinoza Montes, 2014) Descriptivo Simple, porque de acuerdo a los resultados obtenidos de la evaluación de los productos de software se determinó la Mantenibilidad.

3.2. POBLACION Y MUESTRA

3.2.1. POBLACION

Todos los productos de software que se encuentren en producción en diversas organizaciones y que hayan sido desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva conforman la población, sin embargo no es posible cuantificar esta población dado que sería técnicamente necesario esfuerzos no disponibles.

3.2.2. UNIDAD DE ANALISIS

Los productos de software han sido desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva y fueron seleccionados para la investigación.

En la Tabla 4. Resumen de descripción de activos de software se describen los productos de software según los criterios que se tomaron en cuenta en la ficha de descripción de activos de software cual se detalla en el Anexo 1.

Tabla 4. Resumen de descripción de activos de software

Producto	**Tecnología**	**Categoría de dominio**
SIMBA	Lenguaje programación: Java. Manejador de base de datos: PostgreSQL. Frameworks: PrimeFaces y JavaServer Faces	Dominio Comercial
SYS-CEIUNAS	Lenguaje programación: Java. Manejador de base de datos: PostgreSQL. Frameworks: PrimeFaces y JavaServer Faces	Dominio Gestión Administrativa.
SPPP	Lenguaje programación: Java. Manejador de base de datos: PostgreSQL. Frameworks: PrimeFaces y JavaServer Faces	Dominio Gestión Administrativa.
SYS-EPGV2.0	Lenguaje programación: Java. Manejador de base de datos: PostgreSQL. Frameworks: Spring	Dominio Gestión Académica.
SYS-CIUNAS	Lenguaje programación: Java.	Dominio Gestión Administrativa.

	Manejador de base de datos: PostgreSQL.

Fuente: Elaboración propia.

3.2.3. TAMAÑO DE MUESTRA

Debido a que la población es cuantitativamente indeterminada, se opto por un muestreo no probabilístico, por conveniencia (Marshall, 1996) y (Hernandez Sampieri, Fernandez Collado, y Baptista Lucio, 2010).

Para determinar el número de muestra se tomó en cuenta por conveniencia aquellos productos de software pertenecientes a organizaciones conocidas por el autor, dispuestas a ceder el código fuente y tomando en consideración las características de la Tabla 5 para la selección de productos. Además se tuvo en cuenta el esfuerzo disponible para la medición de la mantenibilidad en el marco de la investigación. El número de muestra para la investigación es de 5 productos de software.

Tabla 5. Características de selección por conveniencia de los productos de software.

Características	**Descripción**
Código Fuente	Para realizar la evaluación es importante que los autores de producto de software puedan ceder el código fuente.
Productos en Producción	Para la realzar la evaluación se tomó solo los productos de software que se encuentren en producción.
Lenguaje de Programación	Para la realizar la evaluación solo se tomaron los productos de software desarrollados en el lenguaje de programación Java.

Fuente: Elaboración propia.

3.3. TECNICAS E INSTRUMENTOS DE INVESTIGACIÓN

- Observación: Se utilizó esta técnica la cual permitió percibir los resultados de cada una de las propiedades de calidad al realizar la evaluación de los productos de software.

- **Fichas descriptiva convenio:** Se recolecto información de los productos de software mediante las fichas de descripción del activo de software (Ver Anexo I).

- **Ficha de evaluación:** Se recolecto la información de los resultados obtenidos después de la evaluación de los productos de software (Ver Anexo III).

3.4. PROCEDIMIENTO DE LA INVESTIGACION

La investigación se ha desarrollado teniendo en cuenta con 5 fases, las cuales se detallan a continuación.

3.4.1. SELECCIÓN DE PRODUCTOS DE SOFTWARE

Se realizó una búsqueda de información sobre los productos de software desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas con la finalidad de entrevistar al propietario del producto de software e informarle sobre la investigación. Para asegurar la no divulgación de código fuente, se realizó un convenio de uso de activos de software con fines de investigación, ver Anexo I y se obtuvieron los siguientes productos de software que se describen en la Tabla 6.

Tabla 6. Productos de Software

Producto	Descripción
SIMBA	Es un producto de software orientado a las ventas, cotizaciones, pedidos, guía de remisión, apertura caja, cierre caja, almacén y reportes.

SYS-CEIUNAS	Es un producto de software orientado al control de pagos de estudiantes, gestión de cursos, docentes y aulas de un centro de idiomas.
SPPP	Es un producto de software orientado a registros de prácticas, gestiona los procesos de aprobación y seguimiento hasta la sustentación del alumno practicante de la Facultad de Ingeniería en Informática y Sistemas.
SYS-EPG V2.0	Es un producto de software orientado a registro de datos docentes, académicos, reportes y movimientos de pagos.
SYS-CIUNAS	Es un producto de software orientado a realizar seguimiento y control de la ejecución de los proyectos de investigación de docentes y estudiantes de la Universidad Nacional Agraria de la Selva.

Fuente: Elaboración propia.

3.4.2. SELECCIÓN DE MODELO DE MEDICIÓN DE LA MANTENIBILIDAD

Se realizó un análisis de cinco modelos de medición de la Mantenibilidad basados en el modelo de la ISO/IEC 25010, se aplicó cuatro criterios de selección para determinar el modelo que se utilizó en la investigación las cuales:

- **Cobertura de modelo de la ISO/IEC 25010:** Los modelos propuestos para la medición de la mantenibilidad deben considerar las subcaracterísticas de la mantenibilidad para realizar la evaluación de los productos de software. (Rodríguez et al., 2015) para realizar la evaluación de productos de software es importante definir el modelo de calidad a usarse y definir las subcaracterísticas.

- **Cobertura de Propiedades o Métricas de calidad:** Para realizar la evaluación de la mantenibilidad se tienen que definir métricas o propiedades de calidad que permitan determinar el valor de las subcaracterísticas de la

mantenibilidad. Para (Rodríguez et al., 2015), no solo es importante definir las subcaracterísticas sino que el desglose de estas al más bajo nivel donde se define indicadores y métricas que se pueden medir a través de los productos de software.

- **Entorno tecnológico:** La evaluación de productos de software es tedioso ya que se tienen una gran de cantidad de líneas de código fuente, entonces para automatizar la evaluación es importante utilizar herramientas que faciliten este trabajo. Para (Rodríguez y Piattini, 2014) y (Rodríguez et al., 2015) es un elemento que permite asistir al proceso de evaluación de forma automatizada permitiendo así la recolección de métricas y visualización de resultados.

- **Función de la mantenibilidad:** Para determinar el valor de la mantenibilidad de los productos de software es importante definir la función que permita realizar este cálculo a través de los valores obtenidos de las subcaracterísticas de la mantenibilidad. (Irrazábal, 2012) define la función de la mantenibilidad como la combinación de valores obtenidos de las subcaracterísticas que pertenecen a la mantenibilidad.

A continuación se detallan los cinco modelos propuestos para la medición de la mantenibilidad.

- **Modelo 1:** Este modelo fue propuesto por (Galán Pausic, 2017) . Muestra una cobertura en su totalidad de todas las subcaracterísticas de la mantenibilidad basados en el modelo de calidad de la ISO/IEC 25010, también propone las propiedades de calidad que cobertura todas las subcaracterísticas de la mantenibilidad, para realizar el análisis de código fuente propone un entorno tecnológico donde utilizo la herramienta de análisis estático Sonarqube y propone una función de la mantenibilidad, se puede observar el modelo 1 en la Tabla 7.

Tabla 7. Propuesta del modelo 1

Propiedades	Analizabilidad	Modularidad	Capacidad de ser modificado	Reusabilidad	Capacidad de ser probado
Mala practica	x		x	x	
Redundante	x		x		x
Complejidad cognitiva	x				
Obsoleto				x	
Confuso	x				
Diseño	x	x	x	x	x
Dificultad encontrada	x				x
Falto ingenio	x				
Duplicaciones		x	x	x	
Cobertura		x			x

Fuente: (Galán Pausic, 2017).

- **Modelo 2:** Este modelo fue propuesto por (Irrazábal, 2012) . Este modelo no muestra una cobertura de subcaracterísticas de la mantenibilidad basados en el modelo de calidad de la ISO/IEC 25010, como se puede apreciar en el cuadro solo propone un modelo con 3 subcaracterísticas, también propone las métricas de calidad que cobertura 3 subcaracterísticas de la mantenibilidad propuestas por su modelo de medición, para realizar el análisis de código fuente propone un entorno tecnológico donde utilizo las herramientas como JavaNCSS, PWD, Junit, Jdepend, y propone una función de la mantenibilidad, se puede observar el modelo 2 en la Tabla 8.

Tabla 8. Propuesta del modelo 2

Métricas	Analizabilidad	Capacidad de ser modificado	Capacidad de ser probado
Densidad de complejidad ciclomatica	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	

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	

Fuente: (Irrazábal, 2012).

- **Modelo 3:** Este modelo fue propuesto por (Chawla, 2015) . Este modelo muestra una cobertura en su totalidad de todas las subcaracterísticas de la mantenibilidad basados en el modelo de calidad de la ISO/IEC 25010, también propone las propiedades de calidad que cobertura todas las subcaracterísticas de la mantenibilidad, para realizar el análisis de código fuente propone un entorno tecnológico donde utilizo las herramientas como Metrics y Ckjm, donde ser muestra dificultad en la configuración, soporte y disponibilidad de las herramientas Metrics y Ckjm y propone una función de la mantenibilidad, se puede observar el modelo 3 en la Tabla 9.

Tabla 9. Propuesta del modelo 3

Propiedades	**Analizabilidad**	**Modularidad**	**Capacidad de ser modificado**	**Reusabilidad**	**Capacidad de ser probado**
Número ciclomático	x				x
Número de declaraciones	x			x	
Frecuencia Comente	x				
Métodos ponderados por clase	x				
Herencia del método	x				
Tamaño del promedio de declaraciones			x		

Numero de anidados			x		x
Acoplamiento entre objetos				x	
Falta de cohesión			x	x	
Profundidad de herencia			x		
Factor de polimorfismo			x		
Número de hijos			x		x
Cantidad de entrada / salida			x		
Componentes directamente llamados			x		
Respuesta de una clase					x
Acoplamiento hacia afuera					x
Falta de cohesión / número de módulos	x				

Fuente: (Chawla, 2015).

- **Modelo 4:** Este modelo fue propuesto por (Rodríguez y Piattini, 2014) . Este modelo muestra una cobertura en su totalidad de todas las subcaracterísticas de la mantenibilidad basados en el modelo de calidad de la ISO/IEC 25010, también propone las propiedades de calidad que cobertura todas las subcaracterísticas de la mantenibilidad, para realizar el análisis de código fuente propone un entorno tecnológico pero no muestra un disponibilidad de este y no propone una función de la mantenibilidad, se puede observar el modelo 4 en la Tabla 10.

Tabla 10. Propuesta del modelo 4

Propiedades	Analizabilidad	Modularidad	Capacidad de ser modificado	Reusabilidad	Capacidad de ser probado
Incumplimiento de Reglas	x	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

Fuente: (Rodríguez y Piattini, 2014).

- **Modelo 5:** Este modelo fue propuesto por (Balseca Chisaguano, 2009). Este modelo muestra una cobertura en su totalidad de todas las subcaracterísticas de la mantenibilidad basados en el modelo de calidad de la ISO/IEC 25010, también propone las propiedades de calidad que cobertura todas las subcaracterísticas de la mantenibilidad, para realizar el análisis de código fuente no propone un entorno tecnológico y no propone una función de la mantenibilidad, se puede observar el modelo 5 en la Tabla 11.

Tabla 11. Propuesta del modelo 5

Propiedades	Analizabilidad	Modularidad	Capacidad de ser modificado	Reusabilidad	Capacidad de ser probado
Capacidad de condensación		x			
Acoplamiento de clases		x			

Ejecución de reusabilidad			x		
Capacidad de pistas de auditoria	x				
Diagnóstico de funciones suficientes	x				
Complejidad ciclomatica		x			
Profundidad de herencia		x			
Grado de localización de corrección de impacto		x			
Complejidad de modificación		x			
Índice de éxito de modificación		x		x	
Completitud funcional de funciones de prueba				x	
Capacidad de prueba autónoma				x	
Capacidad de reinicio de pruebas				x	

Fuente: (Balseca Chisaguano, 2009).

Para determinar el modelo de medición de la mantenibilidad que se utilizó en la evaluación de la mantenibilidad de los productos de software, se realizó una comparación de modelos basándonos en los cuatro criterios mencionados anteriormente.

Tabla 12. Comparativo de Modelos según criterios de selección

Modelo	Cobertura del modelo calidad ISO/IEC 25010	Cobertura de Propiedades o Métricas de calidad	Entorno tecnológico	Función de Mantenibilidad
Modelo 1	x	x	x	x
Modelo 2	-	-	x	x
Modelo 3	x	x	-	x
Modelo 4	x	x	-	-
Modelo 5	x	x	-	x

Fuente: Elaboración propia.

Entonces podemos concluir de la Tabla 12, que el modelo 1 propuesto por (Galán Pausic, 2017) cumple con todos los criterios de selección, siendo así el modelo de medición de la mantenibilidad que se utilizó para la evaluación de la mantenibilidad de los productos de software basados en el modelo de calidad de la ISO/IEC 25000.

Para determinar el grado de la mantenibilidad de producto de software se utilizó la valoración de la mantenibilidad la Tabla 13 cual se adaptó de (Rodríguez y Piattini, 2014) llegando a obtener lo siguiente en la Tabla 13 .

Tabla 13. Escala de valoración de la Mantenibilidad

Grado	Valor de mantenibilidad (%)	Descripción
A	0-25	Deficiente
B	26-50	Bajo
C	51-75	Bueno
D	76-95	Alto
E	96-100	Excelente

Fuente: Elaboración propia.

El modelo 1 propuesto por (Galán Pausic, 2017) define las propiedades de calidad de la siguiente manera y para obtener más detalles sobre las propiedades, ver Anexo II donde la documentación fue obtenida de (SonarSource, 2018).

- **Mala práctica:** Es la codificación que complica la capacidad de lectura del código fuente. También es un código que probablemente funciona como fue diseñado, pero de la forma que fue diseñado es ampliamente reconocido como una mala idea.
- **Redundante:** Código redundante en un punto del programa si se ha calculado previamente y se garantiza su resultado estar disponible en ese punto. Si tales cálculos pueden ser identificados, pueden obviamente ser eliminado sin afectar el comportamiento del programa.
- **Complejidad cognitiva:** Esta propiedad está asociada a la complejidad cognitiva, dificultad de mantenibilidad ya que es difícil entender el flujo de control. También se refiere a la complejidad ciclomatica.
- **Obsoleto:** Corresponde a clases e interfaces del lenguaje que han caído en desuso, estas deben ser evitadas ya que serán substituidas y eventualmente removidas. El periodo de obsolescencia permite hacer una transición suave.
- **Confuso:** Es la codificación confusa como por ejemplo definir variables con el mismo nombre en clases relacionadas. Los mantenedores tardarán más en comprender de lo que realmente está justificado por lo que el código realmente hace.
- **Diseño:** Es cuestionable sobre el diseño del código fuente que hacen que sea difícil de comprender. Los problemas de diseño repercutirán en una mayor dificultad de refactoreo.
- **Dificultad encontrada:** Es el código tramposo. Si bien es técnicamente correcto reduce la capacidad de ser leído y es confuso entender lo que hacer.
- **Falto de ingenio:** Esta propiedad está asociada a código escrito con cierta torpeza o falto de ingenio. Se deben usar pasos adicionales para lograr algo que podría hacerse de forma más clara y concisa. Por ejemplo clausulas “if” que pueden ser unidas en una sola expresión lógica y sin embargo se encuentran separadas en varias cláusulas que producen confusión en la lectura de código fuente.

- **Duplicaciones:** Son bloques duplicados a lo largo de código fuente. También ocasiona grandes riesgos de errores al hacer mantenimiento. Indica el porcentaje de código duplicado que hay en software analizado.
- **Cobertura:** Es la proporción de código fuente cubierto con test unitarios. Indica el grado de cobertura que tiene el código fuente con pruebas unitarias.

3.4.3. INSTRUMENTO DE MEDICIÓN

De acuerdo al modelo de medición de la mantenibilidad seleccionado, siento este el Modelo 1, donde para automatizar el proceso de análisis de código fuente utilizó la herramienta de análisis estático Sonarqube. Para esta investigación se utilizó la herramienta propuesta por el modelo para obtener los datos de las propiedades de calidad los productos de software, ver Anexo VII.

Se utilizó Sonarqube en su versión 7.2 donde para realizar el análisis de código fuente, donde se requieren de lo siguiente: Inputs: La herramienta Sonarqube automatiza el proceso de análisis de código fuente para lo cual como entrada se necesita un producto de software a evaluar sin la necesidad de realizar un despliegue. Outputs: Luego de realizar el análisis de código fuente de los productos de software la herramienta Sonarqube muestra los resultados finales de las propiedades de calidad en una interfaz de usuario amigable.

- **Validación de modelo de medición de la mantenibilidad:** Se desarrolló un software piloto considerando las reglas de codificación (cobertura de código, evitar duplicidad de código, etc.), donde posteriormente se realizó la evaluación de la mantenibilidad considerando el modelo de medición de la mantenibilidad seleccionada, los resultados fueron favorables teniendo una mantenibilidad de 97.63% y se pueden observar en el Anexo VI.

- **Creación utilitario User Friendly:** Para tener una mejor visualización de los resultados de evaluación de los productos de software, se

desarrolló una aplicación web User Friendly donde se muestra el valor de la mantenibilidad y el valor de todas las subcaracterísticas según el modelo de calidad de la ISO/IEC 25000 y se pueden observar en el Anexo V.

3.4.4. PROCEDIMIENTO Y ANALISIS DE DATOS

Luego de obtener los resultados de las propiedades de calidad de cada uno de los productos de software, se procedió a determinar el valor de la Mantenibilidad y de cada una de las subcaracterísticas, se utilizó el aplicativo Microsoft Excel en la que se elaboró una ficha de evaluación de productos de software (Anexo III), y cuya función de medición de cada propiedad de calidad podemos observar en el (Anexo II), para obtener el resultado de la Mantenibilidad y las subcaracterísticas se utilizó la función propuesta por el modelo seleccionado (Anexo IV).

Se analizaron los resultados obtenidos de la evaluación de los 5 productos de software, para determinar el grado de mantenibilidad y de cada subcaracterística se utilizó la escala de valoración de la Mantenibilidad (Tabla 13).

3.4.5. ELABORACIÓN DE INFORME FINAL

Con los resultados se prosiguió a la interpretación y discusión. Luego se procedió con la redacción del informe final donde se muestran los resultados de forma adecuada y de fácil interpretación del público lector.

IV. RESULTADOS

4.1. MANTENIBILIDAD DE PRODUCTOS DE SOFTWARE

Los productos de software fueron evaluados y se encontró la mantenibilidad de cada producto de software donde los resultados obtenidos se pueden apreciar en la Tabla 14.

Tabla 14. Mantenibilidad de productos de software evaluados

Productos de Software	Mantenibilidad	Valor Media
SIMBA	85.50%	83. 26%
SYS-CEIUNAS	74.84%	
SPPP	86.26%	
SYS-EPG	84.47%	
SYS-CIUNAS	85.21%	

Fuente: Elaboración propia.

En la Tabla 14 se observa los resultados finales de la evaluación de la mantenibilidad de cada producto de software y la media de la mantenibilidad con un valor de 83.26%.

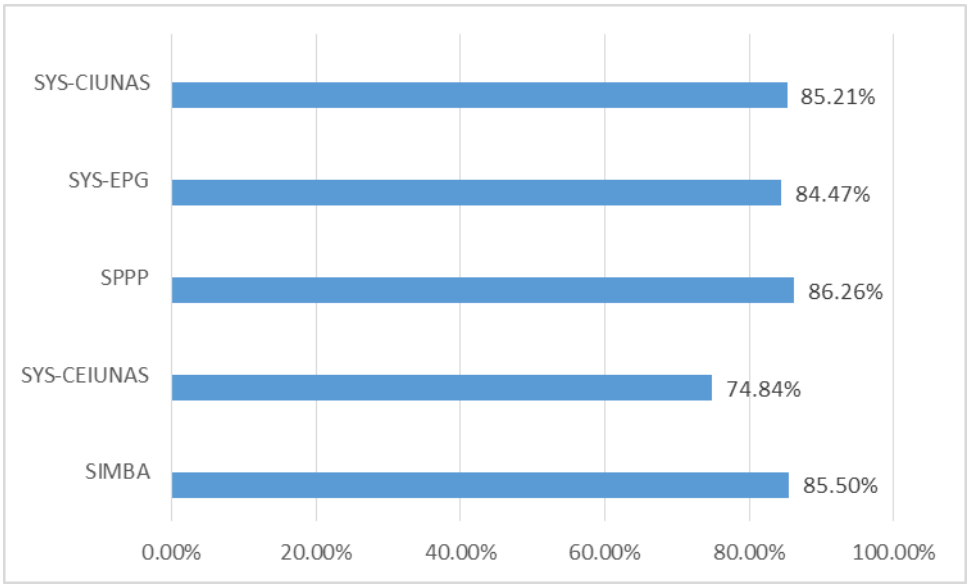


Figura 7. Mantenibilidad de productos de software evaluados

El la Figura 7 se observa los resultados finales de la evaluación de la Mantenibilidad de cada producto de software donde se presentan de la siguiente manera.

- El producto de software “SYS-CEIUNAS” tiene una mantenibilidad de 74.84% que es la más baja con respectos a los demás productos de software.
- Los productos de software “SYS-EPG”, “SYS-CIUNAS”, “SPPP”, “SIMBA” tienen una Mantenibilidad que está por encima del 80%.

4.2. MANTENIBILIDAD DE PRODUCTOS DE SOFTWARE SEGÚN SUBCARACTETISTICAS DE LA MANTENTENIBILIDAD

Tabla 15. Subcaracterísticas de la Mantenibilidad de los productos de software

	PRODUCTOS DE SOFTWARE					
Subcaracteristicas	**SIMBA**	**SYS-CEIUNAS**	**SPPP**	**SYS-EPG**	**SYS-CIUNAS**	**MEDIA**
Analizabilidad	99.56%	99.87%	99.88%	99.49%	99.36%	99.63%
Modularidad	61.61%	39.82%	62.63%	59.50%	61.09%	56.93%
Reusabilidad	95.74%	79.77%	96.93%	94.37%	95.72%	92.50%
Capacidad de ser Probado	74.81%	74.95%	74.89%	74.68%	74.17%	74.70%
Capacidad de ser Modificado	95.80%	79.77%	96.94%	94.33%	95.72%	92.51%

Fuente: Elaboración propia.

En la Tabla 15 se observa los resultados finales de la evaluación de productos de software según las subcaracterísticas de la mantenibilidad y los valores medios por cada subcaracterística.

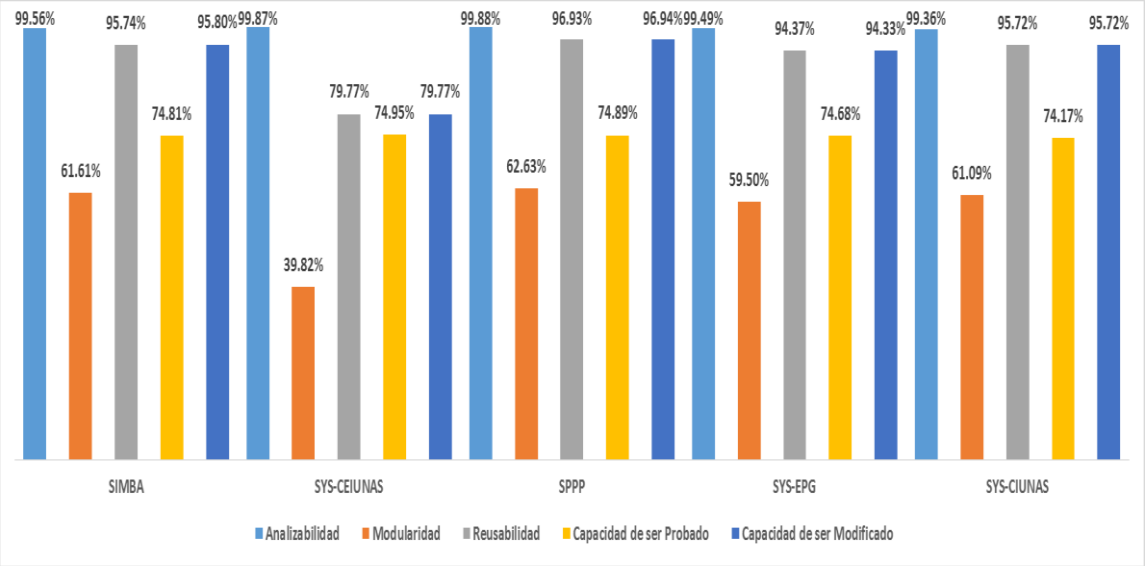


Figura 8. Subcaracterísticas de la Mantenibilidad de los productos de software

En la Tabla 15 y Figura 8 se observa que la Modularidad y Capacidad de ser Probado son las subcaracterísticas más bajas. En tanto la Analizabilidad, Reusabilidad y Capacidad de ser Modificado son las subcaracterísticas con resultados más altos.

El producto de software “SYS-CEIUNAS”, tiene la Modularidad más baja con 39.82%. En tanto la Modularidad y Capacidad de ser Probado con valor medio de 56,93% y 74.70% respectivamente.

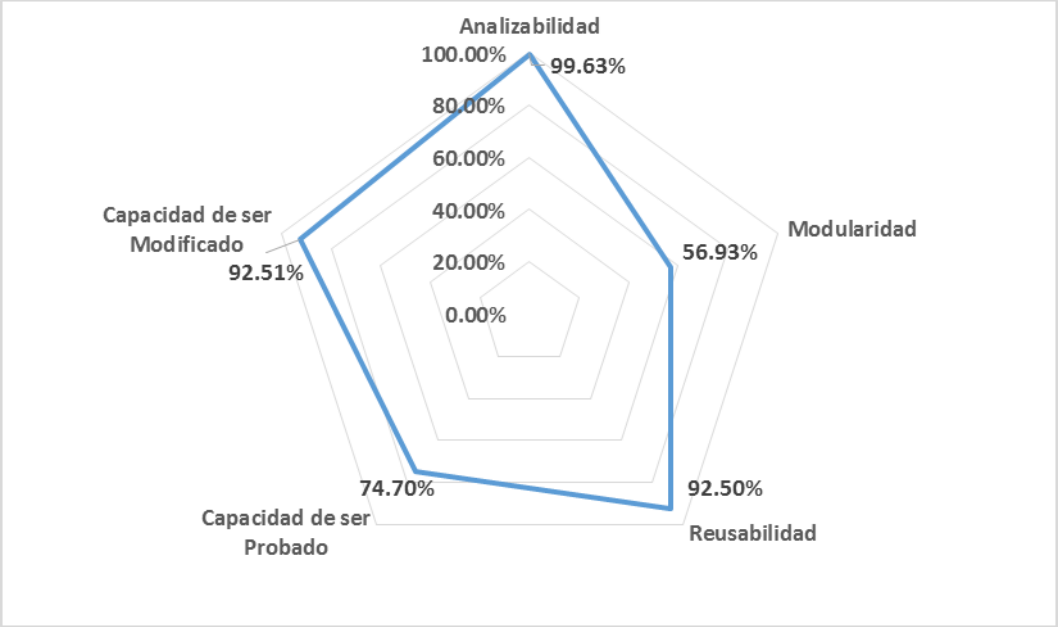


Figura 9. Identificación de subcaracterísticas fuertes y débiles

En la Figura 9 se observa que la subcaracterística que influye negativamente en el resultado de la Mantenibilidad es la Modularidad con un 56.93%, seguido por la Capacidad de ser Probado con un 74.70%, luego tenemos a la Analizabilidad, Reusabilidad, Capacidad de ser Modificado como subcaracterísticas fuertes que benefician el resultado de la Mantenibilidad.

La Analizabilidad tiene un valor más alto con 99.63% esta subcaracterística es la que aporta más a la Mantenibilidad.

4.3. PROPIEDADES DE CALIDAD SEGÚN SUBCARACTERISTICAS DE MANTENIBILIDAD

Se analizaran las subcaracterísticas de la Mantenibilidad a nivel de las propiedades de calidad de los productos de software evaluados.

4.3.1. ANALIZABILIDAD

Tabla 16. Propiedades de calidad de la Analizabilidad

Propiedades	Productos de Software SIMBA	SYS-CEIUNAS	SYS-SPPP	SYS-EPG	SYS-CIUNAS	MEDIA
Mala practica	1.57%	0.36%	0.07%	1.01%	0.42%	0.69%
Redundante	0.04%	0.02%	0.07%	0.17%	0.00%	0.06%
Complejidad cognitiva	0.12%	0.03%	0.02%	0.12%	0.09%	0.08%
Confuso	0.07%	0.17%	0.14%	0.58%	0.28%	0.25%
Diseño	0.48%	0.05%	0.20%	0.70%	1.22%	0.53%
Dificultad encontrada	0.22%	0.13%	0.19%	0.70%	2.09%	0.67%
Falto ingenio	0.58%	0.14%	0.13%	0.53%	0.40%	0.35%

Fuente: Elaboración propia.

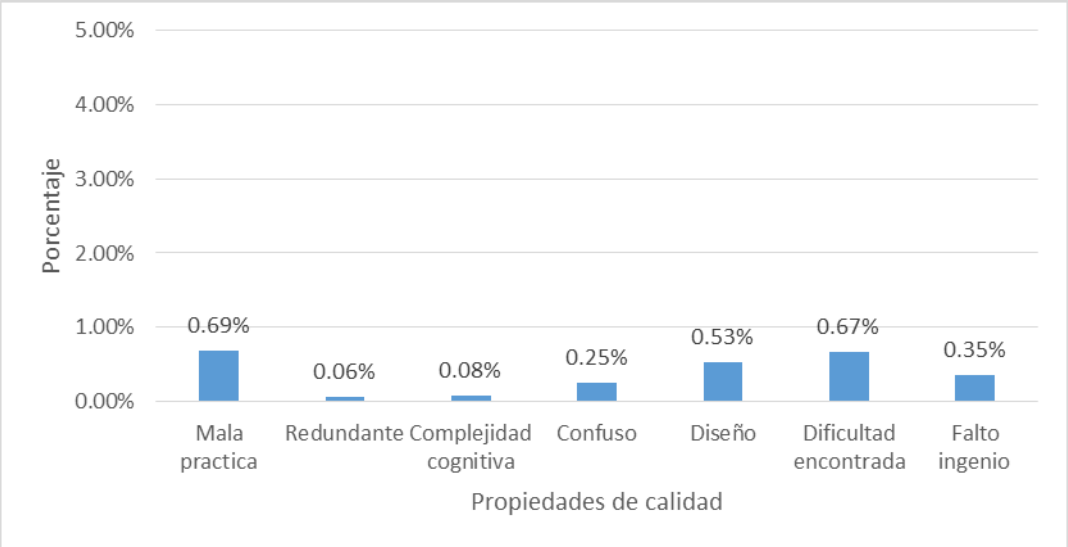


Figura 10. Propiedades de calidad de la Analizabilidad

En la Tabla 16 y Figura 10 se observa que los resultados de las propiedades de calidad están por debajo del 1% esto refleja que la Analizabilidad aporta positivamente a la mantenibilidad.

4.3.2. MODULARIDAD

Tabla 17. Propiedades de calidad de la Modularidad

Propiedades	Productos de Software SIMBA	SYS-CEIUNAS	SYS-SPPP	SYS-EPG	SYS-CIUNAS	MEDIA
Diseño	0.48%	0.05%	0.20%	0.70%	1.22%	0.53%
Duplicaciones	14.70%	80.50%	11.90%	20.80%	15.50%	28.68%
Cobertura	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%

Fuente: Elaboración propia.

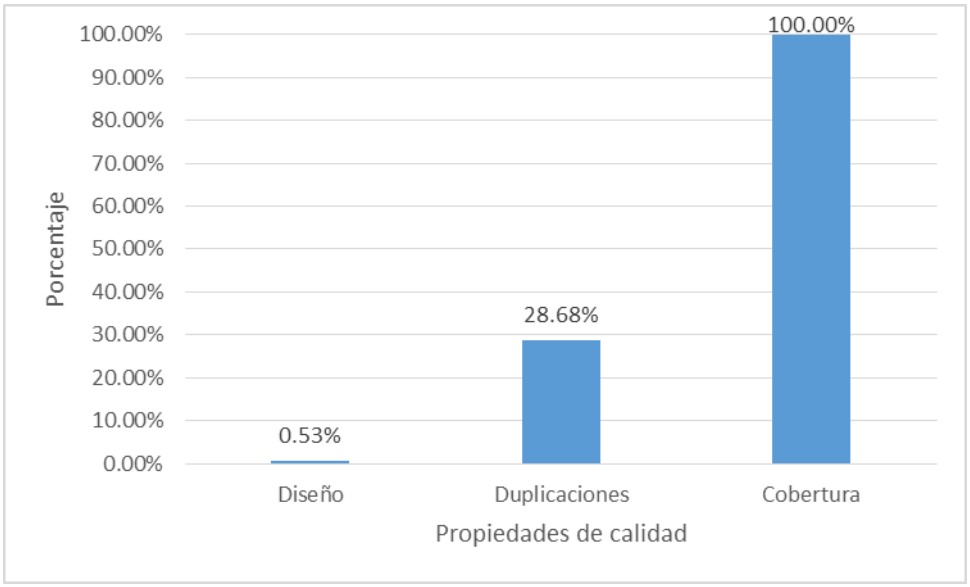


Figura 11. Propiedades de calidad Modularidad

En la Tabla 17 y Figura 11 se observa que la propiedad de calidad: Cobertura, presenta un valor de 100% lo cual indica que ninguno de los proyectos cuentan con pruebas unitarias a nivel de código y Duplicaciones con 28.68 % lo cual indica que en los productos de software se introdujeron código duplicado, estas propiedades afectan negativamente a la Modularidad. En tanto la propiedad de calidad Diseño con 0.53% lo cual indica que se tiene código de fácil comprensión, esta propiedad aporta positivamente a la Modularidad.

4.3.3. REUSABILIDAD

Tabla 18. Propiedades de calidad de la Reusabilidad

Propiedades	Productos de Software SIMBA	SYS-CEIUNAS	SYS-SPPP	SYS-EPG	SYS-CIUNAS	MEDIA
Mala practica	1.57%	0.36%	0.07%	1.01%	0.42%	0.69%
Obsoleto	0.30%	0.03%	0.09%	0.00%	0.00%	0.09%
Diseño	0.48%	0.05%	0.20%	0.70%	1.22%	0.53%
Duplicaciones	14.70%	80.50%	11.90%	20.80%	15.50%	28.68%

Fuente: Elaboración propia.

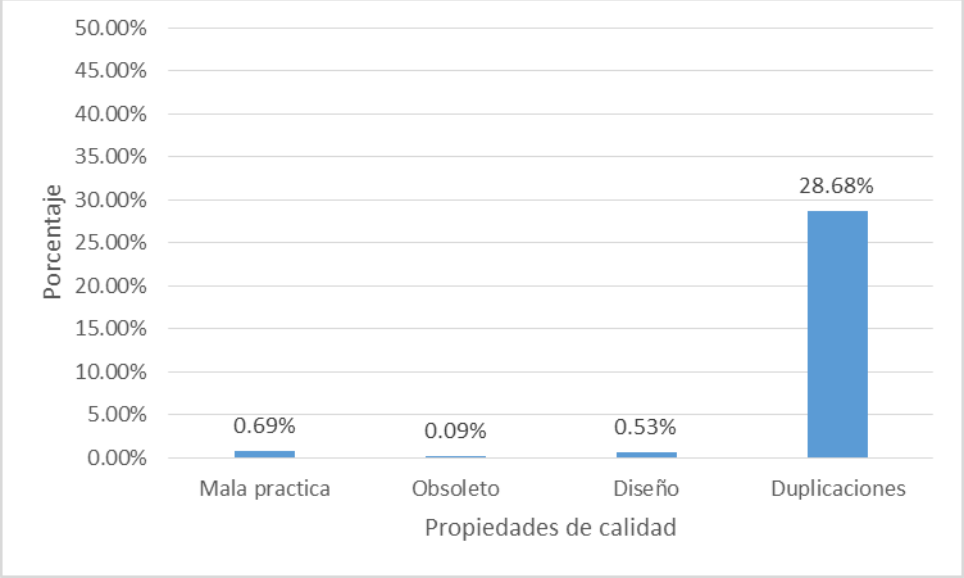


Figura 12. Propiedades de calidad de la Reusabilidad

En la Tabla 18 y Figura 12 se observa que la propiedad de calidad: Duplicaciones, presenta un valor de 28.68% lo cual indica que en los productos de software se introdujeron código duplicado, esta propiedad afecta negativamente a la Reusabilidad. En tanto las propiedades de calidad Mala práctica, Obsoleto y Diseño tienen por debajo del 1% lo cual aportan positivamente a la Reusabilidad.

4.3.4. CAPACIDAD DE SER MODIFICADO

Tabla 19. Propiedades de calidad de la Capacidad de ser Modificado

Propiedades	Productos de Software SIMBA	SYS-CEIUNAS	SYS-SPPP	SYS-EPG	SYS-CIUNAS	MEDIA
Mala practica	1.57%	0.36%	0.07%	1.01%	0.42%	0.69%
Redundante	0.04%	0.02%	0.07%	0.17%	0.00%	0.06%
Diseño	0.48%	0.05%	0.20%	0.70%	1.22%	0.53%
Duplicaciones	14.70%	80.50%	11.90%	20.80%	15.50%	28.68%

Fuente: Elaboración propia.

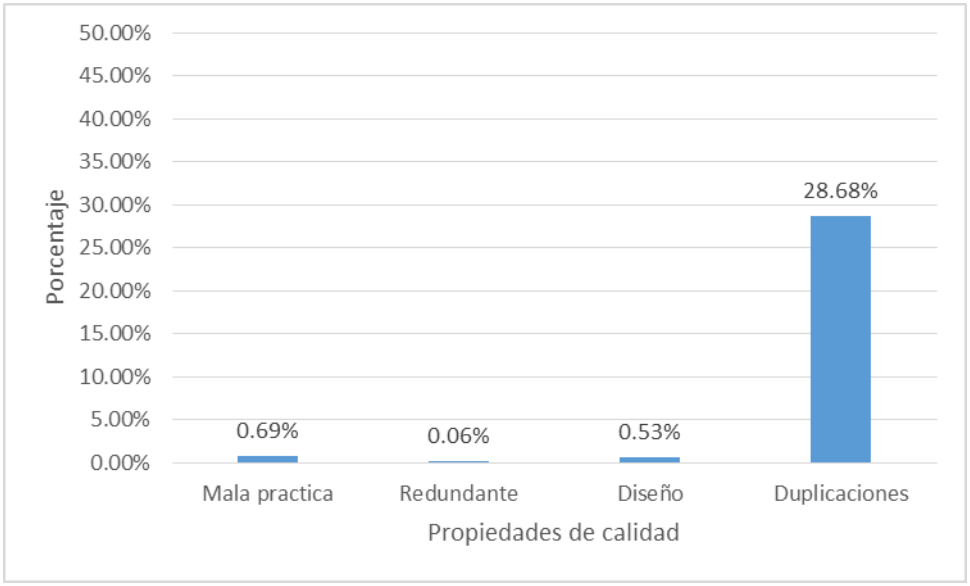


Figura 13. Propiedades de calidad de la Capacidad de ser Modificado

En la Tabla 19 y Figura 13 se observa que la propiedad de calidad Duplicaciones, presenta un valor de 28.68% lo cual indica que en los productos de software se introdujeron código duplicado, esta propiedad afecta negativamente a la Capacidad de ser Modificado. En tanto las propiedades de calidad Mala práctica, Redundante y Diseño tienen por debajo del 1% lo cual aportan positivamente a la Capacidad de ser Modificado.

4.3.5. CAPACIDAD DE SER PROBADO

Tabla 20. Propiedades de calidad de la Capacidad de ser Probado

Propiedades	Productos de Software SIMBA	SYS-CEIUNAS	SYS-SPPP	SYS-EPG	SYS-CIUNAS	MEDIA
Redundante	0.04%	0.02%	0.07%	0.17%	0.00%	0.06%
Diseño	0.48%	0.05%	0.20%	0.70%	1.22%	0.53%
Dificultad encontrada	0.22%	0.13%	0.19%	0.42%	2.09%	0.61%
Cobertura	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%

Fuente: Elaboración propia.

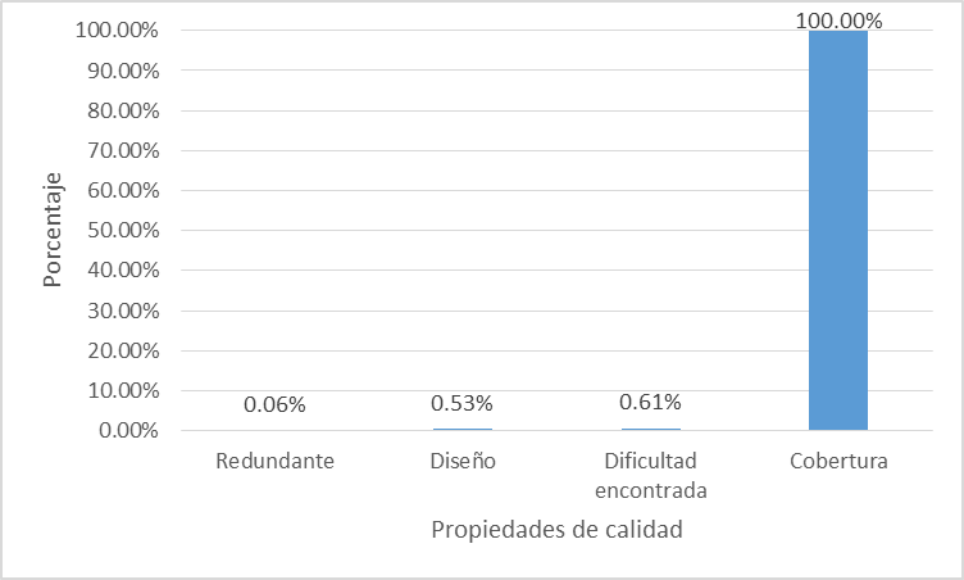


Figura 14. Propiedades de la Capacidad de ser Probado

En la Tabla 20 y Figura 14 se observa que la propiedad de calidad: Cobertura, presenta un valor de 100% lo cual indica que ninguno de los proyectos cuentan con pruebas unitarias a nivel de código, esta propiedad aporta negativamente a la Capacidad de ser Probado. En tanto las propiedades de calidad Redundante, Diseño y Dificultad encontrada tienen por debajo del 1% lo cual aportan positivamente a la Capacidad de ser Probado.

V. DISCUSIÓN

Según la evaluación que realizó (Galán Pausic, 2017), donde se utilizó el mismo modelo de medición de la mantenibilidad seleccionado para esta investigación, encuentra una mantenibilidad de 87.15% y evalúa solo un producto de software que pertenece al dominio comercial. Esta investigación evalúa 5 productos de software que pertenecen al dominio ventas, educativo y documentario resultando una mantenibilidad media de 83.26%. La evaluación que realizó (Balseca Chisaguano, 2009) propone un modelo de medición de mantenibilidad diferente, donde tiene como resultado una mantenibilidad de 1,73 en un rango de 0 a 2.5 lo cual hace un 69.20% el cual resulta exitoso y evalúa solo un producto de software que pertenece al dominio de gestión administrativa. En (Galán Pausic, 2017) y en esta investigación se utilizó la herramienta de análisis estático Sonarqube, la cual proporciona los valores de propiedades de calidad, mientras que (Balseca Chisaguano, 2009) realizo la evaluación con un matriz automatizada.

Los resultados de mantenibilidad de (Galán Pausic, 2017) y esta investigación utilizando el mismo modelo de medición de mantenibilidad, resulta superior a los resultados encontrados por (Balseca Chisaguano, 2009), de lo cual se puede plantear dos hipótesis: la primera “el producto de software de (Balseca Chisaguano, 2009) tiene menor mantenibilidad que los de (Galán Pausic, 2017) y esta investigación; y la segunda: “el modelo de medición de mantenibilidad propuesto por (Balseca Chisaguano, 2009) es menos optimista que el propuesto por (Galán Pausic, 2017) .

En (Galán Pausic, 2017) las propiedades de calidad más bajas fueron cobertura, duplicaciones y diseño, mientras que en la presente investigación, las propiedades de calidad más bajas fueron cobertura y duplicaciones en la cual se afirma que coincidimos en estos resultados. Por otra parte en la propiedad de calidad diseño se tiene una contradicción con los resultados.

Para la presente investigación los resultados de las propiedades de calidad que aportan positivamente a la mantenibilidad fueron mala práctica, redundante, complejidad cognitiva, confuso, dificultad encontrada, falto de ingenio donde estas coinciden con los resultados obtenidos por (Galán Pausic, 2017).

Según (Chawla, 2015) propone un modelo de medición de mantenibilidad diferente a la presente investigación, donde se tuvo como resultado una Mantenibilidad de 1, 1.02, 1.05, 1.10 respectivamente, la diferencia con la presente investigación es que se realizó un comparativo de versiones del mismo producto tomando como referencia la línea base de valor 1.

5.1.1. PRUEBA DE HIPOTESIS

Los resultados de la Figura 15 indican que los productos de software están en una valoración de alto según la escala de valores de la Tabla 13, con una media de 0.83, permitiendo esto rechazar la hipótesis nula y aceptar la alterna, es decir que la mantenibilidad de los productos de software evaluados que fueron desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas es alta.

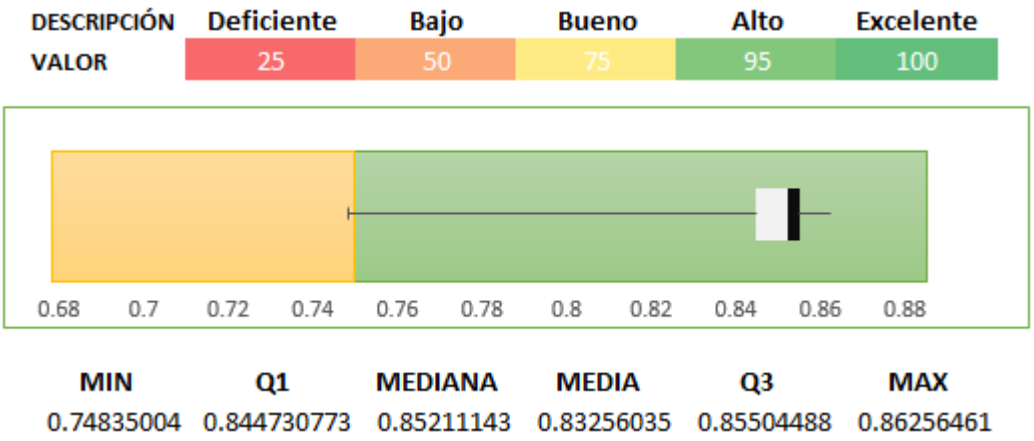


Figura 15. Estadística descriptiva de la mantenibilidad de los productos de software evaluados

En la Figura 16 se observa que 4 de los 5 productos de software superan la mantenibilidad media, solo 1 está por debajo, esto debido a las propiedades de duplicaciones y cobertura de código siendo estas un criterio a tener en cuenta en la enseñanza académica.

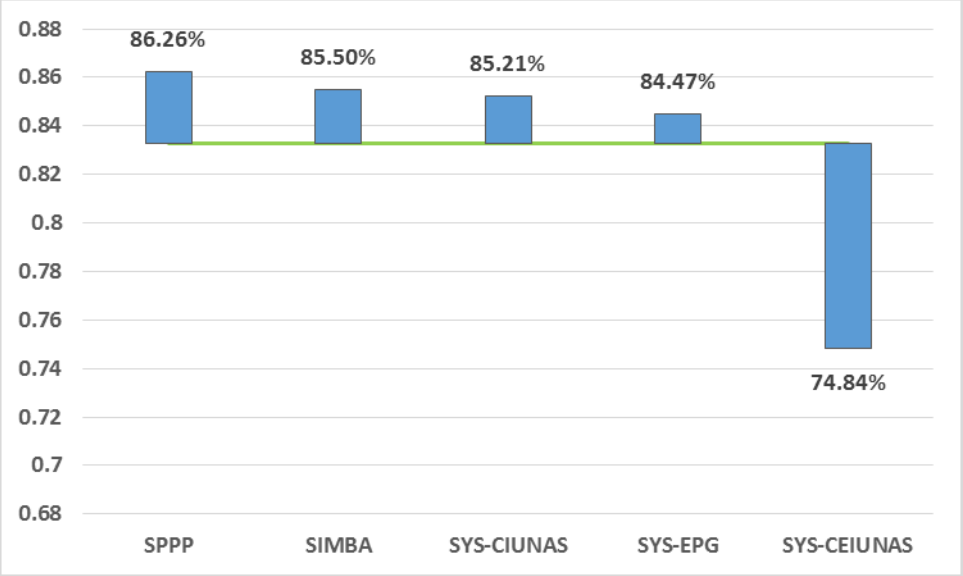


Figura 16. Comparación de la mantenibilidad de los productos de software con el valor de la media

VI. CONCLUSIONES

Se determinó el grado de Mantenibilidad de los 5 productos de software desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva teniendo como referencia el modelo Square ISO/IEC 25000, usando la herramienta de análisis estático Sonarqube cuyo valor medio fue de 83.26% y un falta de Mantenibilidad de 16.74%, donde la falta de mantenibilidad se da en las características de modularidad y capacidad de ser probado.

Se logró determinar el grado de Modularidad de los productos de software desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva teniendo como referencia el modelo Square ISO/IEC 25000 cuyo valor medio fue de 56.93% y un falta de Modularidad de 43.07%, donde las propiedades de calidad de menor desempeño son la cobertura y duplicaciones.

Se logró determinar el grado de Reusabilidad de los productos de software desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva teniendo como referencia el modelo Square ISO/IEC 25000 cuyo valor medio fue de 92.50% y un falta de Reusabilidad de 7.50%, donde la propiedad de calidad duplicaciones tiene que ser tomado más en cuenta.

Se logró determinar el grado de Analizabilidad de los productos de software desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva teniendo como referencia el modelo Square ISO/IEC 25000 cuyo valor medio fue de 99.63% y un falta de Mantenibilidad de 0.37%, esta característica aporta positivamente a la mantenibilidad.

Se logró determinar el grado de “Capacidad de ser Modificado” de los productos de software desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva teniendo como referencia el modelo Square ISO/IEC 25000 cuyo valor medio fue de 92.51% y un falta de Mantenibilidad de 7.49%, donde la propiedad de calidad duplicaciones tiene que ser tomado más en cuenta.

Se logró determinar el grado de “Capacidad de ser Probado” de los productos de software desarrollados por los egresados de la Facultad de Ingeniería en Informática y Sistemas de la Universidad Nacional Agraria de la Selva teniendo como referencia el modelo Square ISO/IEC 25000 cuyo valor medio fue de 74.70% y un falta de Mantenibilidad de 25.3%, donde la propiedad de calidad preocupante viene a ser la cobertura.

RECOMENDACIONES

A la academia reforzar sus contenidos en la formación profesional incluyendo habilidades, técnicas y buenas prácticas en cuanto a los procesos de desarrollo de software poniendo énfasis en la realización de pruebas unitarias y en evitar duplicaciones de código fuente mediante prácticas de reutilización y extensibilidad.

A las empresas de la industria desarrollo de software que implementen es sus procesos, actividades de medición de la mantenibilidad de sus productos de software para evitar costos de no calidad.

A las entidades académicas y empresas desarrolladoras de software, la incorporación de herramientas de análisis estático y herramientas “User Friendly” en su proceso medición de mantenibilidad.

Seguir investigando sobre modelos de medición de la mantenibilidad y las métricas y/o propiedades de calidad que cobertura esta característica. También realizar investigaciones con las demás características que propone el modelo Square ISO/IEC 25000.

Realizar evaluación la mantenibilidad con el modelo de medición utilizado con productos de software desarrollados en lenguajes de programación como php, c#, Python, etc.

REFERENCIAS BIBILIOGRAFICAS

Acosta, N. J., Espinel, L. A., y Garcia, J. L. (2017). Estándares para la calidad de software. *Tecnología Investigación y Academia*, *5*(1), 75-84. Recuperado a partir de http://revistas.udistrital.edu.co/ojs/index.php/tia/article/view/8388

Balseca Chisaguano, E. A. (2009). *Escuela Politécnica Nacional*. Escuela Politecnica Nacional.

Bayona Zubieta, J. C., Pineda Samaca, O. L., & Pardo Mahecha, O. D. (2016). El papel de la Ingeniería de Software en el desarrollo de aplicaciones. *Tecnología Investigación y Academia*, *4*(1), 3-14. Recuperado a partir de http://revistas.udistrital.edu.co/ojs/index.php/tia/article/view/5740

Chawla, M. K. (2015). SQMMA : Software Quality Model for Maintainability Analysis.

Erazo, J., Florez, A., y Pino, F. (2016). Análisis y clasificación de atributos de mantenibilidad del software: una revisión comparativa desde el estado del arte, (19), 40-49.

Espinoza Montes, C. (2014). *Metodología de Investigación Tecnológica*. (C. Espinoza Montes, Ed.). Huancayo. Recuperado a partir de http://repositorio.uncp.edu.pe/handle/UNCP/1148

Galán Pausic, M. V. (2017). *Estudio de la plataforma SonarQube para la implantación de ISO / IEC 25000*. Universidad Pontificia de Salamanca.

Hernandez Sampieri, R., Fernandez Collado, C., & Baptista Lucio, M. del P. (2010). *Metodología de la investigación*. *Metodología de la investigación*. Mexico: McGRAW-HILL. https://doi.org/- ISBN 978-92-75-32913-9

INTECO. (2009). INGENIERÍA DEL SOFTWARE : METODOLOGÍAS Y CICLOS VIDA. ESPAÑA: Laboratorio Nacional de Calidad del Software - INTECO.

Irrazábal, E. (2012). *Construcción de un Entorno para la Medición Automatizada de la Calidad de los Productos Software*. Universidad Rey Juan Carlos.

ISO 25000. (2017). Portal ISO 25000. Recuperado a partir de

http://iso25000.com/

Marshall, M. N. (1996). Sampling for qualitative research Sample size. *Family Practice*, *13*(6), 522-525. https://doi.org/10.1093/fampra/13.6.522

Pérez-gonzález, H. G., Martínez-pérez, F. E., Nava-muñoz, S. E., Núñez-varela, A. S., Escalante, M. V., Saucedo, J. A. F., … Luis, S. (2015). Analizando la Mantenibilidad de Software Desarrollado Durante la Formación Universitaria, *3*(6), 231-236.

Pressman, R. S. (2010). *Ingenieria Del Software* (SÉPTIMA ED). Mexico: McGRAW-HILL. https://doi.org/10.1017/CBO9781107415324.004

Roa, P., Morales, C., y Gutierrez, P. (2015). Norma ISO / IEC 25000, *3*(2).

Rodríguez, M., Fernández, C. M., y Pedreira, O. (2015). Certificación de la Mantenibilidad del Producto Software : Un Caso Práctico, *3*(3), 127-134.

Rodríguez, M., & Piattini, M. (2014). Entorno para la Evaluación y Certificación de la Calidad del Producto Software, 163-176.

SonarSource. (2018). Sonarqube. Recuperado a partir de https://docs.sonarqube.org/

Valdivia Huamán, D. A. (2017). *Facultad de ingeniería*. Universidad Privada del Norte.

Valenciano López, J. (2015). *AUDITORÍA MANTENIBILIDAD APLICACIONES SEGÚN LA ISO/IEC 25000*. UNIVERSIDAD COMPLUTENSE DE MADRID. Recuperado a partir de http://eprints.ucm.es/37485/1/AUDITORÍA MANTENIBILIDAD APLICACIONES SEGÚN LA ISO_IEC 25000.pdf

ANEXOS

ANEXO I. CONVENIO DE USO DE ACTIVOS DE SOFTWARE CON FINES DE INVESTIGACIÓN

Universidad Nacional Agraria de la Selva | Facultad de Ingeniería en Informática y Sistemas | Departamento Académico de Ciencias en Informática y Sistemas | Laboratorio de Ingeniería de software (LINSOFT)

CONVENIO DE USO DE ACTIVOS DE SOFTWARE CON FINES DE INVESTIGACIÓN

1. **TRABAJO DE INVESTIGACIÓN:** MANTENIBILIDAD DE PRODUCTOS DE SOFTWARE SEGÚN EL MODELO SQUARE ISO/IEC 25000.
2. **TIPO DE TRABAJO:** TESIS PREGRADO
3. **PROPIETARIO DE PRODUCTO:**
   JOHN DEIVIS MARTIN PARDO
4. **RESPONSABLES DE LA INVESTIGACIÓN:**
   RONALD EDUARDO IBARRA ZAPATA – ASESOR
   BRIAN CESAR PANDO SOTO – ASESOR
   SAMUEL RICARDO PARDO MESIAS - TESISTA
5. **CONSENTIMIENTOS Y COMPROMISOS:**

   Conste por el presente convenio de uso de activos de software, el consentimiento otorgado por el propietario en favor de los responsables de la investigación para hacer uso de los activos del producto de software descrito en la ficha de la sección 6, con fines de investigación.

   Conste también el compromiso de los responsables de la investigación para usar los activos del producto de software única y exclusivamente con fines investigativos, quedando totalmente prohibido cualquier tipo de explotación comercial y uso no autorizado.

6. **FICHA DE DESCRIPCION DEL ACTIVO DE SOFTWARE**

Nombre de producto de software:	✓ Sistema de Gestión Administrativa y Académica (SYS-EPG V2.0).
Descripción de producto de software:	✓ Registro de Datos Docentes Alumnos ✓ Registros Académicos ✓ Registros de Movimientos Pagos ✓ Reportes.

Dominio de negocio	▪ Comercial ( ) ▪ Académico (X) ▪ Administrativo ▪ Bancario ▪ Educación ( ) ▪ Otros ( )	
Tecnología	Lenguaje programación	Java
	Framework	Spring
	Framework JavaScript	JS.
	Motor de Base Datos	PostgreSQL.
Costo de producción	▪ 0 – 5000.00 ( ) ▪ 5001.00- 10000.00 ( ) ▪ 10001.00 – 20000.00 (X) ▪ 20001.00 a más. ( )	
Recursos brindados:		
Código fuente Base datos Manuales de instalación	▪ (X) ▪ (X) ▪ ( )	

Ronald Eduardo Ibarra Zapata
Responsable

Brian Cesar Pando Soto
Responsable

Samuel Ricardo Pardo Mesias
Tesista

John Deivis Martin Pardo
Propietario

CONVENIO DE USO DE ACTIVOS DE SOFTWARE CON FINES DE INVESTIGACIÓN

1. **TRABAJO DE INVESTIGACIÓN:** MANTENIBILIDAD DE PRODUCTOS DE SOFTWARE SEGÚN EL MODELO SQUARE ISO/IEC 25000.
2. **TIPO DE TRABAJO:** TESIS PREGRADO
3. **PROPIETARIO DE PRODUCTO:**
   ELVIS RAUL MENDOZA ATERO
4. **RESPONSABLES DE LA INVESTIGACIÓN:**
   RONALD EDUARDO IBARRA ZAPATA – ASESOR
   BRIAN CESAR PANDO SOTO – ASESOR
   SAMUEL RICARDO PARDO MESIAS - TESISTA
5. **CONSENTIMIENTOS Y COMPROMISOS:**

   Conste por el presente convenio de uso de activos de software, el consentimiento otorgado por el propietario en favor de los responsables de la investigación para hacer uso de los activos del producto de software descrito en la ficha de la sección 6, con fines de investigación.

   Conste también el compromiso de los responsables de la investigación para usar los activos del producto de software única y exclusivamente con fines investigativos, quedando totalmente prohibido cualquier tipo de explotación comercial y uso no autorizado.

6. **FICHA DE DESCRIPCION DEL ACTIVO DE SOFTWARE**

Nombre de producto de software:	SISTEMA DE GESTION DE DOCUMENTOS TECNICOS Y CIENTIFICOS DE LA UNAS - SYSCIUNAS-
Descripción de producto de software:	Realizar el seguimiento y Control de la ejecución de los proyectos de Investigación de Docentes y Estudiantes de la UNAS.

Dominio de negocio	▪ Comercial ( ) ▪ Académico ( ) ▪ Administrativo (x) ▪ Bancario ▪ Educación ( ) ▪ Otros ( )	
Tecnología	Lenguaje programación	Java
	Framework	ORM - HIBERNATE
	Framework JavaScript	-
	Motor de Base Datos	Postgres SQL
Costo de producción	▪ 0 – 5000.00 ( ) ▪ 5001.00- 10000.00 ( ) ▪ 10001.00 – 20000.00 (x) ▪ 20001.00 a más. ( )	
Recursos brindados:		
Código fuente Base datos Manuales de instalación	▪ (x) ▪ (x) ▪ ( )	

Ronald Eduardo Ibarra Zapata
Responsable

Brian Cesar Pando Soto
Responsable

Samuel Ricardo Pardo Mesias
Tesista

Elvis Raul Mendoza Atero
Propietario

CONVENIO DE USO DE ACTIVOS DE SOFTWARE CON FINES DE INVESTIGACIÓN

1. **TRABAJO DE INVESTIGACIÓN:** MANTENIBILIDAD DE PRODUCTOS DE SOFTWARE SEGÚN EL MODELO SQUARE ISO/IEC 25000.
2. **TIPO DE TRABAJO:** TESIS PREGRADO
3. **PROPIETARIO DE PRODUCTO:**
   YELSIN RAMIREZ LUJAN
4. **RESPONSABLES DE LA INVESTIGACIÓN:**
   RONALD EDUARDO IBARRA ZAPATA – ASESOR
   BRIAN CESAR PANDO SOTO – ASESOR
   SAMUEL RICARDO PARDO MESIAS - TESISTA
5. **CONSENTIMIENTOS Y COMPROMISOS:**

   Conste por el presente convenio de uso de activos de software, el consentimiento otorgado por el propietario en favor de los responsables de la investigación para hacer uso de los activos del producto de software descrito en la ficha de la sección 6, con fines de investigación.

   Conste también el compromiso de los responsables de la investigación para usar los activos del producto de software única y exclusivamente con fines investigativos, quedando totalmente prohibido cualquier tipo de explotación comercial y uso no autorizado.

6. **FICHA DE DESCRIPCION DEL ACTIVO DE SOFTWARE**

Nombre de producto de software:	SIMBA
Descripción de producto de software:	Es un mini ERP, que contiene: - Cotizaciones - Ventas - Pedidos - Guías de remisión - Apertura y Cierre Caja - Almacen - Manejo de reportes

Dominio de negocio	▪ Comercial (X) ▪ Académico ( ) ▪ Administrativo ▪ Bancario ▪ Educación ( ) ▪ Otros ( )	
Tecnología	Lenguaje programación	Java
	Framework	JSF y Primefaces
	Framework JavaScript	—
	Motor de Base Datos	Postgres Sql
Costo de producción	▪ 0 – 5000.00 ( ) ▪ 5001.00- 10000.00 ( ) ▪ 10001.00 – 20000.00 (X) ▪ 20001.00 a más. ( )	
Recursos brindados:		
Código fuente Base datos Manuales de instalación	▪ (X) ▪ ( ) ▪ ( )	

Ronald Eduardo Ibarra Zapata
Responsable

Brian Cesar Pando Soto
Responsable

Samuel Ricardo Pardo Mesias
Tesista

Yelsin Ramirez Lujan
Propietario

CONVENIO DE USO DE ACTIVOS DE SOFTWARE CON FINES DE INVESTIGACIÓN

1. **TRABAJO DE INVESTIGACIÓN:** MANTENIBILIDAD DE PRODUCTOS DE SOFTWARE SEGÚN EL MODELO SQUARE ISO/IEC 25000.
2. **TIPO DE TRABAJO:** TESIS PREGRADO
3. **PROPIETARIO DE PRODUCTO:**
   DANNY MARCEL LOPEZ BENZAQUEN
4. **RESPONSABLES DE LA INVESTIGACIÓN:**
   RONALD EDUARDO IBARRA ZAPATA – ASESOR
   BRIAN CESAR PANDO SOTO – ASESOR
   SAMUEL RICARDO PARDO MESIAS - TESISTA
5. **CONSENTIMIENTOS Y COMPROMISOS:**

   Conste por el presente convenio de uso de activos de software, el consentimiento otorgado por el propietario en favor de los responsables de la investigación para hacer uso de los activos del producto de software descrito en la ficha de la sección 6, con fines de investigación.

   Conste también el compromiso de los responsables de la investigación para usar los activos del producto de software única y exclusivamente con fines investigativos, quedando totalmente prohibido cualquier tipo de explotación comercial y uso no autorizado.

6. **FICHA DE DESCRIPCION DEL ACTIVO DE SOFTWARE**

Nombre de producto de software:	SISTEMAS DE PRACTICAS PRE PROFESIONALES (SIS - SPPP)
Descripción de producto de software:	Registros de practicas, gestiona procesos, de aprobación, seguimiento hasta su sustentación del alumno practicante. de la Facultad de Ingeniería e Informática y sistemas.

Dominio de negocio	▪ Comercial ( ) ▪ Académico ( ) ▪ Administrativo (x) ▪ Bancario ▪ Educación ( ) ▪ Otros ( )	
Tecnología	Lenguaje programación	Java
	Framework	JSF
	Framework JavaScript	Spring MVC JSF, JQUERY JS
	Motor de Base Datos	Postgres
Costo de producción	▪ 0 – 5000.00 ( ) ▪ 5001.00- 10000.00 ( ) ▪ 10001.00 – 20000.00 (x) ▪ 20001.00 a más. ( )	
Recursos brindados:		
Código fuente Base datos Manuales de instalación	▪ (x) ▪ ( ) ▪ ( )	

Ronald Eduardo Ibarra Zapata
Responsable

Brian Cesar Pando Soto
Responsable

Samuel Ricardo Pardo Mesias
Tesista

Danny Marcel Lopez Benzaquen
Propietario

CONVENIO DE USO DE ACTIVOS DE SOFTWARE CON FINES DE INVESTIGACIÓN

1. **TRABAJO DE INVESTIGACIÓN:** MANTENIBILIDAD DE PRODUCTOS DE SOFTWARE SEGÚN EL MODELO SQUARE ISO/IEC 25000.
2. **TIPO DE TRABAJO:** TESIS PREGRADO
3. **PROPIETARIO DE PRODUCTO:**
   SAMUEL RICARDO PARDO MESIAS
4. **RESPONSABLES DE LA INVESTIGACIÓN:**
   RONALD EDUARDO IBARRA ZAPATA – ASESOR
   BRIAN CESAR PANDO SOTO – ASESOR
   SAMUEL RICARDO PARDO MESIAS - TESISTA
5. **CONSENTIMIENTOS Y COMPROMISOS:**

   Conste por el presente convenio de uso de activos de software, el consentimiento otorgado por el propietario en favor de los responsables de la investigación para hacer uso de los activos del producto de software descrito en la ficha de la sección 6, con fines de investigación.

   Conste también el compromiso de los responsables de la investigación para usar los activos del producto de software única y exclusivamente con fines investigativos, quedando totalmente prohibido cualquier tipo de explotación comercial y uso no autorizado.

6. **FICHA DE DESCRIPCION DEL ACTIVO DE SOFTWARE**

Nombre de producto de software:	SYSCOIUNAS
Descripción de producto de software:	Realizar control y seguimiento de alumnos, docentes, personal de la Institución, Gestion de notas y pagos, y emisión de certificados de estudios Control de aulas y de idiomas a doctorat.

Dominio de negocio	▪ Comercial ( ) ▪ Académico ( ) ▪ Administrativo (X) ▪ Bancario ▪ Educación ( ) ▪ Otros ( )	
Tecnología	Lenguaje programación	Java
	Framework	Phisfaces y JSF
	Framework JavaScript	JQ
	Motor de Base Datos	PostgreSQL
Costo de producción	▪ 0 – 5000.00 ( ) ▪ 5001.00- 10000.00 (X) ▪ 10001.00 – 20000.00 ( ) ▪ 20001.00 a más. ( )	
Recursos brindados:		
Código fuente Base datos Manuales de instalación	▪ (X) ▪ ( ) ▪ ( )	

Ronald Eduardo Ibarra Zapata
Responsable

Brian Cesar Pando Soto
Responsable

Samuel Ricardo Pardo Mesias
Tesista y Propietario

ANEXO II. DESCRIPCIÓN DE LAS PROPIEDADES DE CALIDAD

Nombre	Mala Practica	
Descripción	Es la codificación que complica la capacidad de lectura del código fuente	
Función de medición	R=LC/Vpropiedad	
Elementos de medición	R=Resultado Líneas de código=LC Vpropiedad = Valor de Propiedad	
Reglas de Codificación		
N°	Nombre	Descripción
1	La ejecución del recolector de basura debe ser activada solo por la JVM	Llamar System.gc()o Runtime.getRuntime().gc() es una mala idea
2	Las constantes no deberían definirse en las interfaces	El patrón de interfaz constante es un uso deficiente de las interfaces.
3	"Thread.sleep" no debe usarse en pruebas	Usar Thread.sleepen una prueba es generalmente una mala idea. Crea pruebas frágiles que pueden fallar impredeciblemente en función del entorno
4	Throwable and Error no debe ser atrapado	Throwable es la superclase de todos los errores y excepciones en Java.
5	"@Override" se debe utilizar en la anulación y la implementación de métodos	@Override Es util por lo siguiente: Se obtiene una advertencia del compilador si el método anotado no anula nada, como en el caso de un error ortográfico y mejora la legibilidad del código fuente al hacer obvio que los métodos están anulados.
6	Los elementos obsoletos deberían tener tanto la anotación como la etiqueta Javadoc	La depreciación debe marcarse con la @Deprecated anotación y la etiqueta @deprecated Javadoc
7	Las salidas estándar no deben usarse directamente para registrar nada	Al registrar un mensaje hay varios requisitos importantes que deben cumplirse: El usuario debe ser capaz de recuperar fácilmente los registros, Los datos registrados deben ser grabados, Los datos confidenciales solo deben registrarse de forma segura.

8	Los campos "enum" no deben ser públicamente mutables	Enum son generalmente considerados como constante, sino una enum con un public campo o public colocador no sólo no es constante, pero también vulnerable a código malicioso
9	Las declaraciones deben utilizar interfaces de recopilación de Java, como "Lista", en lugar de clases de implementación específicas, como "LinkedList".	El objetivo de la API de colecciones de Java es proporcionar una jerarquía de interfaces bien definida para ocultar los detalles de implementación.
10	Las declaraciones de "switch“ deben tener al menos 3 cláusulas de "case"	las declaraciones switch son útiles cuando hay muchos casos diferentes según el valor de la misma expresión.
11	Los bloques de códigos anidados no deben usarse	Los bloques de código anidados se pueden usar para crear un nuevo alcance y restringir la visibilidad de las variables definidas dentro de él
12	Seguimiento de los usos de los comentarios "NOSONAR"	Cualquier problema con la regla de calidad se puede desactivar con el NOSONAR marcador.
13	Seguimiento de los usos de los comentarios de supresión "CHECKSTYLE: OFF"	Esta regla le permite rastrear el uso del mecanismo de comentario de supresión Checkstyle.
14	Seguimiento de los usos de los comentarios de supresión "NOPMD"	Esta regla le permite rastrear el uso del mecanismo de comentario de supresión de PMD.
15	Los literales de cadenas deben colocarse en el lado izquierdo cuando se busca la igualdad	Es preferible colocar literales de cadena en el lado izquierdo de una llamada de método equals() o equalsIgnoreCase().

Nombre	Redundante
Descripción	Código redundante en un punto del programa si se ha calculado previamente y se garantiza su resultado estar disponible en ese punto. Si tales cálculos pueden ser identificados, pueden obviamente ser eliminado sin afectar el comportamiento del programa.
Función de medición	R=LC/Vpropiedad
Elementos de medición	R=Resultado Líneas de código=LC Vpropiedad = Valor de Propiedad

Reglas de Codificación

N°	Nombre	Descripción
1	Las asignaciones no deberían ser redundantes	Esta regla plantea un problema cuando una asignación es inútil porque la variable asignada ya tiene el valor en todas las rutas de ejecución.
2	las declaraciones "throws" no deberían ser superfluas	Una excepción en una declaración en Java es superflua si es: enumerado varias veces, una subclase de otra excepción enumerada, completamente innecesario porque el tipo de excepción declarado no se puede lanzar realmente
3	Los casts redundantes no deben usarse	Las expresiones de conversión innecesarias hacen que el código sea más difícil de leer y comprender.
4	Se deben usar nombres de clase simples	El mecanismo import de Java permite el uso de nombres de clases simples. Por lo tanto, utilizar un nombre completo de clase en un archivo que import la clase es redundante y confusa.
5	Las llamadas "close()" no deberían ser redundantes.	La estructura try-with-resources de Java 7 maneja automáticamente el cierre de los recursos que abre e try. Por lo tanto, agregar un explícito. La llamada close( ) es redundante y potencialmente confusa
6	Los métodos de anulación deberían hacer más que simplemente llamar al mismo método en la clase superior	Anular un método solo para llamar al mismo método desde la súper clase sin realizar ninguna otra acción es inútil y engañoso.
7	Las expresiones booleanas no deberían ser gratuitas.	Si una expresión booleana no cambia la evaluación de la condición, entonces es completamente innecesaria y puede eliminarse. Si es gratuito porque no coincide con la intención del programador, entonces es un error y la expresión debe ser corregida.

Nombre	Complejidad cognitiva
Descripción	Esta propiedad está asociada a la complejidad cognitiva, dificultad de mantenibilidad ya que es difícil entender el flujo de control.
Función de medición	R=LC/Vpropiedad
Elementos de medición	R=Resultado Líneas de código=LC Vpropiedad = Valor de Propiedad

Reglas de Codificación

N°	Nombre	Descripción
1	La complejidad cognitiva de los métodos no debe ser demasiado alta	La complejidad cognitiva es una medida de cuán difícil es comprender el flujo de control de un método. Los métodos con alta complejidad cognitiva serán difíciles de mantener.
2	Un campo no debe duplicar el nombre de su clase contenedora	Es confuso tener un miembro de la clase con el mismo nombre (aparte de las diferencias de caso) como su clase adjunta. Esto es particularmente así cuando se considera la práctica común de nombrar una instancia de clase para la clase misma.
3	Las declaraciones de "cambio" no deben tener demasiadas cláusulas de "caso".	Cuando las declaraciones switch tienen grandes conjuntos de case cláusulas, generalmente es un intento de mapear dos conjuntos de datos. Una estructura de mapa real sería más legible y mantenible, y debería utilizarse en su lugar.
4	Los métodos no deberían tener demasiados parámetros	Una larga lista de parámetros puede indicar que se debe crear una nueva estructura para ajustar los numerosos parámetros o que la función está haciendo demasiadas cosas
5	Los bucles no deben contener más de una declaración de "break" o "continue"	Restringir el número de break y las continue instrucciones en un bucle se realiza en interés de una buena programación estructurada.
6	Los métodos no deberían ser demasiado complejos	La complejidad ciclomatica de los métodos no debe superar un umbral definido
7	Las sentencias de control de flujo "if", "for", "while", "swich" y "try" no deben anidarse demasiado profundamente	Tener anidado if, for, while, switch, y try declaraciones son ingredientes clave para hacer lo que se conoce como "código espagueti". Tal código es difícil de leer, refactorizar y, por lo tanto, mantener.

8	Las expresiones no deberían ser demasiado complejas	La complejidad de la expresión se define por el número de &&, || y condition? if True: if False operadores que contiene.
9	Las clases no deberían tener demasiadas importaciones "static"	Importar una clase estáticamente le permite usar sus public static miembros sin calificarlos con el nombre de la clase. Eso puede ser útil, pero sí importa demasiadas clases estáticamente, su código puede volverse confuso y difícil de mantener.
10	Las clases internas no deberían tener demasiadas líneas de código	Las clases internas deben ser cortas y dulces, para administrar la complejidad en el archivo general. Una clase interna que ha crecido más de un cierto umbral probablemente debería externalizarse a su propio archivo.
11	Los archivos deben contener solo una clase o interfaz de nivel superior	Un archivo que crece demasiado tiende a agregar demasiadas responsabilidades e inevitablemente se vuelve más difícil de entender y, por lo tanto, de mantener.
12	Las clases no deberían tener demasiados campos	Una clase que crece demasiado tiende a agregar demasiadas responsabilidades e inevitablemente se vuelve más difícil de entender y, por lo tanto, de mantener
13	El operador ternario no debe ser usado	Si bien el operador ternario es agradablemente compacto, su uso puede hacer que el código sea más difícil de leer. Por lo tanto, debe evitarse en favor de la estructura if más detallado que else.
14	Las clases no deben tener demasiados métodos	Una clase que crece demasiado tiende a agregar demasiadas responsabilidades e inevitablemente se vuelve más difícil de entender y, por lo tanto, de mantener
15	Los métodos no deberían tener demasiadas líneas	Un método que crece demasiado tiende a agregar demasiadas responsabilidades. Tal método inevitablemente se vuelve más difícil de comprender y, por lo tanto, más difícil de mantener.
16	Las clases no deben acoplarse a muchas otras clases (Principio de Responsabilidad Individual)	Las clases que dependen de muchas otras clases tienden a agregar demasiadas responsabilidades y deben dividirse en varias más pequeñas.
17	Las cláusulas "mayúsculas y minúsculas" no deberían tener demasiadas líneas de código	La switch declaración debe usarse solo para definir claramente algunas ramas nuevas en el flujo de control. Tan pronto como una case cláusula contiene demasiados enunciados, esto disminuye

		enormemente la legibilidad del enunciado de flujo de control general.
18	Los métodos no deberían tener demasiadas declaraciones de retorno.	Tener demasiadas instrucciones de devolución en un método aumenta la complejidad esencial del método porque el flujo de ejecución se rompe cada vez que se encuentra una declaración de devolución. Esto hace que sea más difícil leer y comprender la lógica del método.
19	Los números mágicos no deben ser utilizados	Usar números mágicos puede parecer obvio y sencillo cuando estás escribiendo un fragmento de código, pero son mucho menos obvios y sencillos en el momento de la depuración
20	Los archivos no deben tener demasiadas líneas de código	Un archivo fuente que crece demasiado tiende a agregar demasiadas responsabilidades e inevitablemente se vuelve más difícil de entender y, por lo tanto, de mantener
21	Las asignaciones de "acción" no deberían tener demasiadas entradas "hacia adelante".	Para facilitar el mantenimiento, esta regla plantea un problema cuando una action cuenta tiene más forward etiquetas que las permitidas.
22	Las variables no deben declararse antes de que sean relevantes	Las variables deben declararse lo más cerca posible de su uso. Esto es particularmente cierto cuando se consideran métodos que contienen retornos tempranos y la posibilidad de lanzar excepciones

Nombre	Obsoleto	
Descripción	Corresponde a clases e interfaces del lenguaje que han caído en desuso.	
Función de medición	R=LC/Vpropiedad	
Elementos de medición	R=Resultado Líneas de código=LC Vpropiedad = Valor de Propiedad	
Reglas de Codificación		
N°	Nombre	Descripción
1	Los métodos jQuery obsoletos no se deben usar	La depreciación es una advertencia de que un método ha sido reemplazado y eventualmente será eliminado. El período de desaprobación le permite hacer una transición transición lejos de la tecnología obsoleta, que pronto se retirará

2	"importación" se debe utilizar para incluir código externo	Proporciona mecanismos estándar de lenguaje para la gestión de módulos, import y export, y los usos más antiguos deben ser convertidos.
3	"arguments.caller" y "arguments.callee" no se deben usar	Los objetos Arguments para funciones de modo estrictas definen propiedades de acceso no configurables llamadas "caller" y "callee" que arrojan un TypeError excepción de acceso.

Nombre	Confuso
Descripción	Es la codificación confusa ya sea cómo definir variables con el mismo nombre en clases relacionadas.
Función de medición	R=LC/Vpropiedad
Elementos de medición	R=Resultado Líneas de código=LC Vpropiedad = Valor de Propiedad

Reglas de Codificación

N°	Nombre	Descripción
1	Los campos de clases secundarias no deben sombrear los campos de clases principales	Tener una variable con el mismo nombre en dos clases no relacionadas está bien, pero haga lo mismo dentro de una jerarquía de clases y obtendrá confusión en el mejor de los casos, caos en el peor.
2	Test Cases debe contener pruebas	Esta regla plantea un problema cuando los archivos en el directorio de prueba tienen "Test" en el nombre o implementación, TestCase pero no contienen ninguna prueba.
3	Los métodos y los nombres de campo no deben ser iguales o diferir solo por mayúsculas	Observar el conjunto de métodos en una clase, incluidos los métodos de superclase, y encontrar dos métodos o campos que difieren solo por mayúsculas es confuso para los usuarios de la clase. Es igualmente confuso tener un método y un campo que difieren solo en mayúsculas o un método y un campo con exactamente el mismo nombre y visibilidad.
4	Una línea individual ejecutada condicionalmente debe denotarse con sangría	En ausencia de llaves, la línea inmediatamente después de un condicional es la que se ejecuta condicionalmente. Tanto por convención como por buenas prácticas, tales líneas

		están sangradas. En ausencia de llaves y muescas, la intención del programador original no está del todo clara y tal vez no sea realmente lo que se ejecuta. Además, es muy probable que dicho código sea confuso para los mantenedores.
5	las cláusulas de incremento de bucle "for" deberían modificar los contadores de los bucles	Puede ser extremadamente confuso cuando “for” el contador de un bucle se incrementa fuera de su cláusula de incremento. En tales casos, el incremento debe moverse a la cláusula de incremento del bucle, si es posible.
6	Los métodos no deberían tener implementaciones idénticas	Cuando dos métodos tienen la misma implementación, o bien fue un error - se pretendía algo más - o la duplicación fue intencional, pero puede ser confusa para los mantenedores
7	Las cadenas de formato de estilo Printf deben usarse correctamente	Debido a que “printflas” cadenas de formato de estilo se interpretan en tiempo de ejecución, en lugar de ser validadas por el compilador, pueden contener errores que provocan la creación de cadenas incorrecta.
8	Los operadores ternarios no deben estar anidados	Anidar operadores ternarios da como resultado el tipo de código que puede parecer claro como el día en que lo escribes, pero seis meses más tarde dejará a los mantenedores (o peor, a futuro) rascándose la cabeza y maldiciendo.
9	"writeObject" no debería ser el único código "sincronizado" en una clase	El objetivo de la sincronización es garantizar que solo un subproceso ejecute un bloque de código dado a la vez. No hay ningún problema real con el marcado writeObject synchronized, pero es muy sospechoso si este método relacionado con la serialización es el único synchronizedcódigo en a class
10	"readObject" no debe estar "sincronizado"	Se readObjectescribe un método cuando un Serializableobjeto necesita un manejo especial para rehidratarse del archivo. Debería darse el caso de que el objeto que se está creando readObjectsolo sea visible para el hilo que invocó el método, y la synchronizedpalabra clave no es necesaria, y el uso de synchronizedtodos modos es confuso

11	Los bloques Try-catch no deben estar anidados	El anidamiento “try” / “catch” bloques afecta severamente la legibilidad del código fuente porque hace que sea muy difícil de entender qué bloque capturará qué excepción.
12	Las etiquetas no deben ser usadas	Las etiquetas no se usan comúnmente en Java, y muchos desarrolladores no entienden cómo funcionan. Además, su uso hace que el control sea más difícil de seguir, lo que reduce la legibilidad del código.
13	Los pares redundantes de paréntesis deben eliminarse	El uso de paréntesis, incluso aquellos que no son necesarios para hacer cumplir un orden de operaciones deseado, puede aclarar la intención detrás de un fragmento de código. Pero los pares redundantes de paréntesis podrían ser engañosos y deberían eliminarse.
14	Los métodos no deben devolver constantes	Esta regla plantea un problema si se trata de métodos que contienen solo una declaración: el “return” de un valor constante.
15	los métodos "private" llamados solo por clases internas deberían moverse a esas clases	Cuando un método private solo es invocado por una clase interna, no hay razón para no moverlo a esa clase. Todavía tendrá el mismo acceso a los miembros de la clase externa, pero la clase externa será más clara y menos abarrotada.
16	Los métodos abstractos no deberían ser redundantes	No tiene sentido en la definición redundante de un método abstract con la misma firma que un método en una interface clase implements
17	Las comprobaciones de "indexOf" deben usar una posición inicial	Una cosa que hace que un buen código sea bueno es la claridad con la que transmite la intención del programador original a los mantenedores, y la elección correcta de los métodos indexOf puede ayudar a que el código pase de confuso a transparente.
18	Los caracteres Unicode escapados no deben ser utilizados	El uso de secuencias de escape Unicode debe reservarse para caracteres que de otro modo serían ambiguos, como caracteres no imprimibles.

19	las clases "finales" no deberían tener miembros "protegidos"	La diferencia entre private y protected visibilidad es que las clases de hijas pueden ver y usar protected miembros, pero no pueden verlos private. Como una finalclase no tendrá hijos, marcar a los miembros de una clase final protectedes confusamente inútil.
20	Se deben usar nombres de clase simples	El import mecanismo de Java permite el uso de nombres de clase simples. Por lo tanto, usar un nombre completamente calificado de clase en un archivo que importes la clase es redundante y confuso.

Nombre	Diseño
Descripción	Es cuestionable sobre el diseño del código fuente que hacen que sea difícil de comprender.
Función de medición	R=LC/Vpropiedad
Elementos de medición	R=Resultado Líneas de código=LC Vpropiedad = Valor de Propiedad

Reglas de Codificación

N°	Nombre	Descripción
1	Los literales de cadena no deben duplicarse	Los literales de cadena duplicados hacen que el proceso de refactorización sea propenso a errores, ya que debe asegurarse de actualizar todas las instancias. Por otro lado, las constantes se pueden referenciar desde muchos lugares, pero solo necesitan actualizarse en un solo lugar.
2	Dos ramas en una estructura condicional no deberían tener exactamente la misma implementación	Tener dos cases en una switch declaración o dos ramas en una if cadena con la misma implementación es, en el mejor de los casos, código duplicado y, en el peor, un error de codificación. Si realmente se necesita la misma lógica para ambas instancias, entonces en una if cadena deben combinarse, o para una switch, uno debe pasar al otro.
3	Las clases de utilidad no deberían tener constructores públicos	Las clases de utilidad, que son colecciones de static miembros, no están destinadas a crear

		instancias. Incluso las clases de utilidad abstractas, que pueden ampliarse, no deberían tener constructores públicos.
4	El árbol de clases de herencia no debe ser demasiado profundo	La herencia es sin duda uno de los conceptos más valiosos en la programación orientada a objetos. Es una forma de compartimentar y reutilizar código creando colecciones de atributos y comportamientos llamados clases que pueden basarse en clases creadas previamente. Pero abusar de este concepto al crear un árbol de herencia profunda puede conducir a un código fuente muy compleja e inmanejable.
5	Se debe usar la inyección de constructor en lugar de la inyección de campo	Esta regla plantea un problema cuando las clases con no private constructores (incluido el constructor predeterminado) usan la inyección de campo.
6	Los componentes de resorte deben usar inyección de constructor	Esta regla plantea un problema cuando cualquier static no miembro de un componente de Spring tiene una anotación de inyección, o si el constructor del componente de Spring no tiene una anotación de inyección.
7	Los métodos públicos no deben contener argumentos de selector.	Esta regla encuentra métodos con a boolean que se usan para determinar qué camino tomar a través del método.
8	Las clases sin constructores "public" deben ser "final"	Las clases con solo private constructores deben marcarse finalpara evitar intentos de extensión erróneos.

Nombre	Dificultad encontrada
Descripción	Es el código tramposo. Si bien es técnicamente correcto reduce la capacidad de ser leído y es confuso entender lo que hace.
Función de medición	R=LC/Vpropiedad
Elementos de medición	R=Resultado Líneas de código=LC Vpropiedad = Valor de Propiedad

Reglas de Codificación

N°	Nombre	Descripción
1	Las cadenas de formato de estilo de impresión no	Debido a que “printflas” cadenas de formato de estilo se interpretan en tiempo de ejecución, en lugar de ser validadas por el compilador de Java, pueden contener

	deben generar un comportamiento inesperado durante el tiempo de ejecución	errores que conducen a comportamientos inesperados o errores de tiempo de ejecución.
2	Las palabras clave futuras no deben usarse como nombres	A través de la evolución de Java se han agregado palabras clave. Si bien el código que usa esas palabras como identificadores puede compilarse en versiones anteriores de Java, no será bajo versiones modernas.
3	Las semillas "SecureRandom" no deben ser predecibles	La java.security.SecureRandomclase proporciona un fuerte generador de números aleatorios (RNG) apropiado para la criptografía. Sin embargo, sembrarlo con una constante u otro valor predecible lo debilitará significativamente. Esta regla plantea un problema cuando SecureRandom.setSeed()o SecureRandom(byte[])se invoca una semilla que sea: constante o System.currentTimeMillis()
4	Getters y setters deben acceder a los campos esperados	Getters y setters proporcionan una forma de aplicar la encapsulación al proporcionar métodos public que dan acceso controlado a los campos private. Sin embargo, en clases con campos múltiples no es inusual que cortar y pegar se use para crear rápidamente los getters y setters necesarios, lo que puede dar como resultado que un getter o setter acceda al campo incorrecto.
5	La firma de "finalize ()" debe coincidir con la de "Object.finalize ()"	Object.finalize () es llamado por el recolector de basura en algún momento después de que el objeto no se referencia. En general, la sobrecarga Object.finalize () es una mala idea porque: La sobrecarga no puede ser invocada por el recolector de basura y No se espera que los usuarios llamen Object.finalize () y se confundirán.
6	Nulo no debe ser devuelto por un método "booleano"	Si bien null técnicamente es un Boolean valor válido, ese hecho y la distinción entre Booleany booleanes fácil de olvidar. Por lo tanto, regresar null de un Boolean método probablemente cause problemas con el código de la persona que llama.
7	Los nombres de clase no deben sombrear interfaces o superclases	Si bien es perfectamente legal dar a una clase el mismo nombre simple que una clase en otro paquete que amplíe o que implemente, es confuso y podría causar problemas en el futuro.
8	Try-with-resources debería ser usado	La declaración try-with-resources, que garantiza que el recurso en cuestión se cerrará. Como la nueva sintaxis es más cercana a la prueba de balas, debe preferirse a la versión anterior try/ catch/ finally. Esta regla verifica

		que los close recursos aptos se abren en una declaración try-with-resources.
9	Los métodos "readResolve" deben ser heredables	El método readResolve()permite ajustes finales al estado de un objeto durante la deserialización. Las clases no finales que implementan readResolve(), no deben establecer su visibilidad privateya que entonces no estará disponible para clases secundarias.
10	La declaración del paquete debe coincidir con el directorio del archivo fuente.	Por convención, una ubicación física de clase Java (directorios de origen) y su representación lógica (paquetes) deben mantenerse sincronizados
11	Los tipos comodín genéricos no se deben usar en los parámetros de devolución.	Se recomienda encarecidamente no utilizar tipos comodín como tipos de devolución. Debido a que las reglas de inferencia de tipo son bastante complejas, es poco probable que el usuario de esa API sepa cómo usarlas correctamente
12	"compareTo" no debe estar sobrecargado	Al implementar el método Comparable<T>.compareTo, el tipo de parámetro debe coincidir con el tipo utilizado en la declaración Comparable. Cuando se usa un tipo diferente, esto crea una sobrecarga en lugar de una anulación, que es poco probable que sea la intención.
13	Los bloques ejecutados condicionalme nte deben ser accesibles	Expresiones condicionales que siempre true o false pueden llevar a código muerto. Tal código siempre tiene errores y nunca debe usarse en producción.
14	Las declaraciones relacionadas "if / else if" no deberían tener la misma condición	Una cadena de if/ else if declaraciones se evalúa de arriba a abajo. Como mucho, solo se ejecutará una rama: la primera con una condición que evalúe true.
15	Los métodos no constructor no deberían tener el mismo nombre que la clase adjunta	Tener una clase y algunos de sus métodos compartiendo el mismo nombre es engañoso, y deja a otros preguntándose si se hizo de esa manera a propósito, o si se suponía que los métodos eran un constructor.
	Las clases de los paquetes "sun. *" No deben usarse	Las clases en los paquetes sun.*o com.sun.*se consideran detalles de implementación y no son parte de la API de Java.

	Los valores octanos no deben ser utilizados.	Los literales enteros que comienzan con cero son valores octales en lugar de valores decimales. Si bien el uso de valores octales es totalmente compatible, la mayoría de los desarrolladores no tienen experiencia con ellos
	Las estructuras de control deben usar llaves.	Si bien no es técnicamente incorrecto, la omisión de llaves puede ser engañosa y puede conducir a la introducción de errores durante el mantenimiento.
	Las clases y las enumeraciones con miembros privados deben tener un constructor.	No abstractos classes y enums con no static, private los miembros deben inicializar explícitamente esos miembros, ya sea en un constructor o con un valor por defecto.
	Las funciones no se deben definir con una cantidad variable de argumentos.	Los métodos de Varargs son una forma conveniente de definir métodos que requieren una cantidad variable de argumentos, pero no deben usarse en exceso. Pueden producir resultados confusos si se usan de manera inapropiada.
	Los métodos "privados" que no acceden a los datos de instancia deben ser "estáticos"	Los métodos Private que no acceden a los datos de instancia pueden ser static para evitar cualquier malentendido sobre el contrato del método.

Nombre	Falto de ingenio
Descripción	Esta propiedad está asociada a código escrito con cierta torpeza o falto de ingenio. Se deben usar pasos adicionales para lograr algo que podría hacerse de forma más clara y concisa
Función de medición	R=LC/Vpropiedad
Elementos de medición	R=Resultado Líneas de código=LC Vpropiedad = Valor de Propiedad

Reglas de Codificación

N°	Nombre	Descripción
1	El uso de la función de cadena debe optimizarse para caracteres individuales	Una llamada indexOfo una lastIndexOf llamada con una sola letra String puede hacerse más efectiva al cambiar a una llamada con un char argumento.

2	Las clases con solo métodos "estáticos" no se deben instanciar	Static puede acceder a los métodos sin una instancia de la clase adjunta, por lo que no hay ninguna razón para crear una instancia de una clase que solo tenga static métodos.
3	Tontas matemáticas no deberían ser realizadas	Ciertas operaciones matemáticas son simplemente tontas y no deben realizarse porque sus resultados son predecibles.
4	Los tipos de excepción no se deben probar utilizando "instanceof" en los bloques catch	Se deben usar múltiples bloques de captura del tipo apropiado en lugar de capturar una excepción general, y luego probar el tipo.
5	Las matrices no deberían crearse para parámetros varargs	No tiene sentido crear una matriz con el único fin de pasarla como un argumento varargs ( ); varargs es una matriz. Simplemente pase los elementos directamente. Se consolidarán en una matriz automáticamente
6	Se debe usar el operador de diamante ("<>")	EL operador de diamante (<>) para reducir la verbosidad del código de genéricos. Por ejemplo, en lugar de tener que declarar Listel tipo de un 's tanto en su declaración como en su constructor, ahora puede simplificar la declaración del constructor con <>, y el compilador inferirá el tipo.
7	Las clases no deben estar vacías	No hay una buena excusa para una clase vacía. Si se usa simplemente como un punto de extensión común, se debe reemplazar con un interface. Si fue anotado como un marcador de posición para el desarrollo futuro, debe ser desarrollado. En cualquier otro caso, debe ser eliminado.
8	"toString ()" nunca debe invocarse en un objeto String	Invocar un método diseñado para devolver una representación de cadena de un objeto que ya es una cadena es un desperdicio de las pulsaciones de teclas. Esta construcción redundante puede ser optimizada por el compilador, pero será confusa mientras tanto.
9	Se debe usar un ciclo "while" en lugar de un ciclo "for"	Cuando solo se define la expresión de condición en un for bucle y faltan las expresiones de inicialización e incremento, se whiledebe usar un bucle para aumentar la legibilidad.

10	El retorno de las expresiones booleanas no debe incluirse en una declaración "if-then-else"	El retorno de las declaraciones literales booleanas envueltas en if-then-else unas debe simplificarse.
11	Los literales booleanos no deberían ser redundantes	Los literales booleanos redundantes deben eliminarse de las expresiones para mejorar la legibilidad.
12	"Opcional" no debe usarse para parámetros	Los autores del lenguaje Java han sido bastante francos y Optional su única intención era utilizarlos como un tipo de devolución, como una forma de expresar que un método puede devolver un valor o no.
13	Las pruebas unitarias deben arrojar excepciones"	Esta regla plantea un problema cuando hay una afirmación de falla dentro de un bloque catch.
14	Los modificadores redundantes no deben ser utilizados	Los métodos declarados en las interface son por defecto. Cualquiera de las variables es automáticamente. No hay necesidad de declararlos explícitamente. public abstract public static final
15	las declaraciones "throws" no deberían ser superfluas	Una excepción en una throws declaración en Java es superfluo si es: enumerado varias veces, una subclase de otra excepción incluida , una Runtime Exception, o uno de sus descendientes y completamente innecesario porque el tipo de excepción declarado no puede ser arrojado.

Nombre	Duplicaciones
Descripción	Indica el porcentaje de código duplicado que hay en el software analizado mientras que
Función de medición	R=Vpropiedad/100
Elementos de medición	R=Resultado Vpropiedad = Valor de Propiedad

Reglas de Codificación

N°	Descripción
1	Duplicaciones (%): (Número de líneas duplicadas/Número total de líneas) *100 Archivos duplicados: Número de archivos Involucrados en una duplicación. Bloques duplicados: Número de bloques de líneas duplicadas Líneas duplicadas: Número de líneas involucradas en una duplicación

Nombre	Cobertura
Descripción	Indica el grado de cobertura que tiene el código con tests unitarios
Función de medición	R=(1-(Vpropiedad/100))
Elementos de medición	R=Resultado Vpropiedad = Valor de Propiedad

Reglas de Codificación

N°	Descripción
1	coverage = (CT + CF + LC)/(2*B + EL) Donde: CT = branches that evaluated to "true" at least once CF = branches that evaluated to "false" at least once LC = lines covered (lines_to_cover - uncovered_lines) B = total number of branches (2*B = conditions_to_cover) EL = total number of executable lines (lines_to_cover)

ANEXO III. FICHA DE RESULTADOS DE LA EVALUACIÓN DE LOS PRODUCTOS DE SOFTWARE

Nombre de producto: SIMBA

Líneas de código: 101279

Propiedades	Valores de Propiedades	Analizabilidad	Modularidad	Capacidad de ser modificado	Reusabilidad	Capacidad de ser probado
Mala practica	1590	0.015699207		0.015699207	0.01569921	
Redundante	44	0.000434443		0.000434443		0.000434443
Complejidad cognitiva	121	0.00119472				
Obsoleto	307				0.00303123	
Confuso	71	0.000701034				
Diseño	482	0.004759131	0.004759131	0.004759131	0.00475913	0.004759131
Dificultad encontrada	225	0.002221586				0.002221586
Falto ingenio	591	0.005835366				
Duplicaciones	14.7		0.147	0.147	0.147	
Cobertura	0		1			1
Proporción de falla en cada subcaracterística		0.004406498	0.38391971	0.041973195	0.04262239	0.25185379
Resultado		0.995593502	0.61608029	0.958026805	0.95737761	0.74814621
Falta de Mantenibilidad	0.144955117	14.50				
Mantenibilidad	0.855044883	85.50				

Nombre de producto: SYS-CEIUNAS

Líneas de código: 200056

Propiedades	Valores de Propiedades	Analizabilidad	Modularidad	Capacidad de ser modificado	Reusabilidad	Capacidad de ser probado
Mala practica	711	0.003554005		0.003554005	0.003554005	
Redundante	43	0.00021494		0.00021494		0.00021494
Complejidad cognitiva	63	0.000314912				
Obsoleto	59				0.000294917	
Confuso	341	0.001704523				
Diseño	96	0.000479866	0.000479866	0.000479866	0.000479866	0.000479866
Dificultad encontrada	261	0.001304635				0.001304635
Falto ingenio	276	0.001379614				
Duplicaciones	80.5		0.805	0.805	0.805	
Cobertura	0		1			1
Proporción de falla en cada subcaracterística		0.001278928	0.601826622	0.202312203	0.202332197	0.25049986
Resultado		0.998721072	0.398173378	0.797687797	0.797667803	0.74950014
Falta de Mantenibilidad	0.251649962	25.16				
Mantenibilidad	0.748350038	74.84				

Nombre de producto: SPPP

Líneas de código: 53792

Propiedades	Valores de Propiedades	Analizabilidad	Modularidad	Capacidad de ser modificado	Reusabilidad	Capacidad de ser probado
Mala practica	38	0.000706425		0.000706425	0.000706425	
Redundante	35	0.000650654		0.000650654		0.000650654
Complejidad cognitiva	9	0.000167311				
Obsoleto	51				0.000948096	
Confuso	76	0.001412849				
Diseño	106	0.001970553	0.001970553	0.001970553	0.001970553	0.001970553
Dificultad encontrada	102	0.001896193				0.001896193
Falto ingenio	68	0.001264128				
Duplicaciones	11.9		0.119	0.119	0.119	
Cobertura	0		1			1
Proporción de falla en cada subcaracterística		0.001152588	0.373656851	0.030581908	0.030656269	0.25112935
Resultado		0.998847412	0.626343149	0.969418092	0.969343731	0.74887065
Falta de Mantenibilidad	0.137435393	13.74				
Mantenibilidad	0.862564607	86.26				

Nombre de producto: SYS-EPG

Líneas de código: 40614

Propiedades	Valores de Propiedades	Analizabilidad	Modularidad	Capacidad de ser modificado	Reusabilidad	Capacidad de ser probado
Mala practica	412	0.010144285		0.010144285	0.010144285	
Redundante	70	0.001723544		0.001723544		0.001723544
Complejidad cognitiva	48	0.001181858				
Obsoleto	1				2.46221E-05	
Confuso	237	0.005835426				
Diseño	286	0.007041907	0.007041907	0.007041907	0.007041907	0.007041907
Dificultad encontrada	171	0.004210371				0.004210371
Falto ingenio	214	0.005269119				
Duplicaciones	28		0.28	0.28	0.28	
Cobertura	0		1			1
Proporción de falla en cada subcaracterística		0.005058073	0.429013969	0.074727434	0.074302704	0.253243955
Resultado		0.994941927	0.570986031	0.925272566	0.925697296	0.746756045
Falta de Mantenibilidad	0.167269227	16.73				
Mantenibilidad	0.832730773	83.27				

Nombre de producto: SYS-CIUNAS

Líneas de código: 27762

Propiedades	Valores de Propiedades	Analizabilidad	Modularidad	Capacidad de ser modificado	Reusabilidad	Capacidad de ser probado
Mala practica	116	0.004178373		0.004178373	0.004178373	
Redundante	0	0		0		0
Complejidad cognitiva	26	0.000936532				
Obsoleto	0				0	
Confuso	78	0.002809596				
Diseño	338	0.012174915	0.012174915	0.012174915	0.012174915	0.012174915
Dificultad encontrada	581	0.020927887				0.020927887
Falto ingenio	111	0.003998271				
Duplicaciones	15.5		0.155	0.155	0.155	
Cobertura	0		1			1
Proporción de falla en cada subcaracterística		0.006432225	0.389058305	0.042838322	0.042838322	0.258275701
Resultado		0.993567775	0.610941695	0.957161678	0.957161678	0.741724299
Falta de Mantenibilidad	0.147888575	14.79				
Mantenibilidad	0.852111425	85.21				

ANEXO IV. FUNCION DE LA MANTENIBILIDAD Y SUBCARACTERISTICAS SEGÚN MODELO SELECIONADO

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

Donde:

A= Analizabilidad

M= Modularidad

R= Reusabilidad

CM= Capacidad de ser Modificado

CP= Capacidad de ser Probado

Analizabilidad (A)

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

Donde:

MP= Mala Práctica.

R= Redundante

CC=Complejidad Cognitiva

C= Confuso

D= Diseño

DE=Dificultad Encontrada

FI=Falto Ingenio

Modularidad (M)

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

Donde:

DE=Dificultad Encontrada

DP=Duplicaciones

CB=Cobertura

Reusabilidad(R)

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

Donde:

MP= Mala Práctica.

O= Obsoleto

D= Diseño

DP=Duplicaciones

Capacidad de ser Modificado (CM)

$$\boldsymbol{Capacidad\ de\ ser\ Modificado} = \frac{\boldsymbol{MP + R + D + DP}}{\boldsymbol{4}}$$

Donde:

MP= Mala Práctica.

R= Redundante

D= Diseño

DP=Duplicaciones

Capacidad de Probado (CP)

$$\boldsymbol{Capacidad\ de\ ser\ Probado} = \frac{\boldsymbol{R + D + DE + CB}}{\boldsymbol{4}}$$

Donde:

R= Redundante

D= Diseño

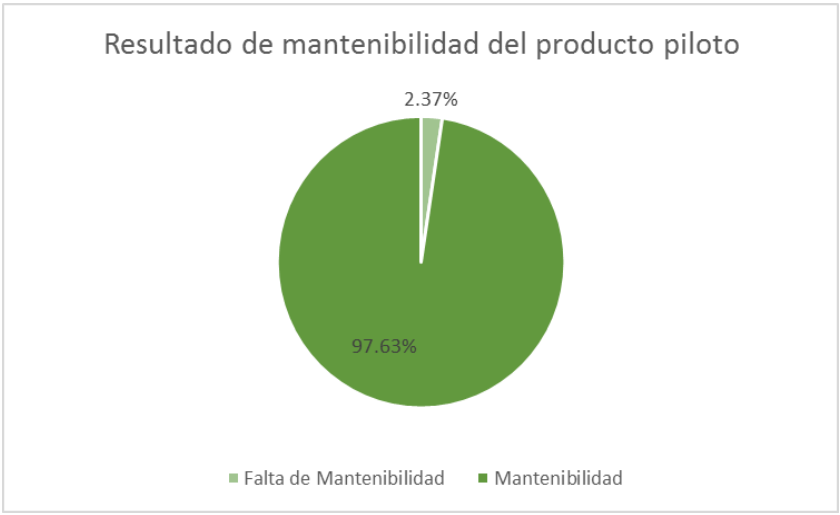
DE=Dificultad Encontrada

CB=Cobertura

ANEXO V. HERRAMIENTA USER FRIENDLY

ANEXO VI. VALIDACION DE MODELO DE MEDICION SELECCIONADO

Líneas de Código	173					
Propiedades	Valores de Propiedades	Analizabilidad	Modularidad	Capacidad de ser modificado	Reusabilidad	Capacidad de ser probado
Mala practica	2	0.01156069		0.01156069	0.01156069	
Redundante	0	0		0		0
Complejidad cognitiva	4	0.02312139				
Obsoleto	0				0	
Confuso	4	0.02312139				
Diseño	0	0	0	0	0	0
Dificultad encontrada	0	0				0
Falto ingenio	0	0				
Duplicaciones	0		0	0	0	
Cobertura	82.1		0.179			0.179
Proporción de falla en cada subcaracterística		0.83%	5.97%	0.29%	0.29%	4.48%
Resultado		99.17%	94.03%	99.97%	99.97%	95.53
Falta de Mantenibilidad	2.37%					
Mantenibilidad	97.63%					



ANEXO VII. EVALUACION CON LA HERRAMIENTA DE ANALIS ESTATICO SONARQUBE