



**TECNOLÓGICO
NACIONAL DE MÉXICO**



TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Apizaco

**INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN Y
COMUNICACIONES**

NOMBRE DEL PROYECTO

Evaluación de la Calidad de Componentes de Software de uso libre
Mediante Herramientas para el Análisis Estático

NOMBRE DE LA EMPRESA

Instituto Tecnológico de Apizaco

ALUMNO

Pedro Cocoltzi Hernández

ASESOR INTERNO

Dra. María Guadalupe Medina Barrera

ASESOR EXTERNO

Dr. José Juan Hernández Mora



Educación
Secretaría de Educación Pública



TECNOLÓGICO
NACIONAL DE MÉXICO



Instituto Tecnológico de Apizaco
Sistemas y Computación

Tzompantepec, Tlaxcala, 07/febrero/2025

OFICIO No: SC/OPV/0017/25

ASUNTO: Liberación de Proyecto
para Titulación Integral.

ISRAEL JORGE SÁNCHEZ SÁNCHEZ
JEFE DE LA DIVISIÓN DE ESTUDIOS PROFESIONALES
P R E S E N T E.

POR ESTE MEDIO LE INFORMO QUE HA SIDO LIBERADO EL SIGUIENTE PROYECTO PARA LA TITULACIÓN INTEGRAL:

NOMBRE DEL EGRESADO:	COCOLETZI HERNÁNDEZ PEDRO
CARRERA:	INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIONES
NO. DE CONTROL:	19370985
NOMBRE DEL PROYECTO:	Evaluación de la Calidad de Componentes de Software de uso libre Mediante Herramientas para el Análisis Estático.
PRODUCTO U OPCIÓN:	TITULACIÓN INTEGRAL INFORME DE RESIDENCIA PROFESIONAL

AGRADEZCO DE ANTEMANO SU VALIOSO APOYO EN ESTA IMPORTANTE ACTIVIDAD PARA LA FORMACIÓN PROFESIONAL DE NUESTROS EGRESADOS.

ATENTAMENTE

Excelencia en Educación Tecnológica®
Pensar para Servir, Servir para Triunfar®

MIQUELINA SÁNCHEZ PULIDO
JEFA DEL DEPARTAMENTO SISTEMAS Y COMPUTACIÓN



SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL
DE MÉXICO
INSTITUTO TECNOLÓGICO DE APIZACO
DEPARTAMENTO DE
SISTEMAS Y COMPUTACIÓN

MARÍA GUADALUPE MEDINA BARRERA
ASESORA INTERNO

c.c.p. División de Estudios Profesionales.
c.c.p. Depto. De Servicios Escolares
c.c.p. Archivo
MSP/mrol*



2025
Año de
La Mujer
Indígena

Av. Instituto Tecnológico No. 418, San Andrés Ahuashuatepec, Municipio de
Tzompantepec, Tlaxcala, Mex. C.P. 90491 Tel. (241) 4172010 Ext. 115, e-mail:
sistemas@apizaco.tecnm.mx | apizaco.tecnm.mx





Educación
Secretaría de Educación Pública



TECNOLÓGICO
NACIONAL DE MÉXICO



Instituto Tecnológico de Apizaco
División de Estudios Profesionales

Tzompantepec, Tlaxcala, 10/abril/2025

COCOLETZI HERNÁNDEZ PEDRO
EGRESADO DE LA CARRERA DE INGENIERÍA
EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIONES.
CON NÚMERO DE CONTROL 19370985
P R E S E N T E.

Por este medio, me permito informar a usted que por aprobación de la comisión revisora asignada para valorar el trabajo que mediante la opción **TITULACIÓN INTEGRAL INFORME DE RESIDENCIA PROFESIONAL** y cuyo tema es: **EVALUACIÓN DE LA CALIDAD DE COMPONENTES DE SOFTWARE DE USO LIBRE MEDIANTE HERRAMIENTAS PARA EL ANÁLISIS ESTÁTICO**, conforme a lo establecido en el procedimiento para la obtención del Título Profesional en el Instituto Tecnológico, la División de Estudios Profesionales a mi cargo le otorga la:

AUTORIZACIÓN DE IMPRESIÓN

Debiendo entregar **2 USB** del mismo, apropiadamente empastados, a la brevedad, para presentar su Acto de Recepción Profesional.

Sin otro particular por el momento me despido.

ATENTAMENTE
Excelencia en Educación Tecnológica®
Pensar para Servir, Servir para Triunfar®

SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE APIZACO
DIVISIÓN DE ESTUDIOS PROFESIONALES

ISRAEL JORGE SÁNCHEZ SÁNCHEZ
JEFE DE LA DIVISIÓN DE ESTUDIOS PROFESIONALES

ccp. Archivo
ijss/Gsc*



2025
Año de
La Mujer
Indígena

Av. Instituto Tecnológico No. 418, San Andrés Ahuashuatepec,
Municipio de Tzompantepec, Tlaxcala, Mex. C.P. 90491 Tel. (241) 4172010
Ext. 304, e-mail: dep_titulacion@apizaco.tecnm.mx | apizaco.tecnm.mx



AGRADECIMIENTOS

Agradezco a mi madre:

*A mi madre **Marcela Cocoltzi Hernández**, todo lo que me has dado ha sido más que suficiente, incluso se podría decir que es más de lo que merezco. Siempre muestras coraje, rectitud, paciencia y comprensión hacia mis errores. Eres un ejemplo que trato de seguir y no avergonzar con mis acciones. Aunque no dispongas de mucho tiempo, los momentos que pasamos juntos los atesoro. Agradezco todo el apoyo que me has dado. Te quiero mamá.*

Agradezco a mis abuelos:

*A mi abuelo **Pedro Cocoltzi Hernández** y mi abuela **Adelina Hernández Tilayatzí**, desde niño han cuidado de mí, aunque no fuera su responsabilidad, estuvieron conmigo lado a lado. Ustedes siempre me han considerado un hijo más y no un nieto. Siempre le pido a Dios que los mantenga un poco más de tiempo junto a mí, quiero que estén en mis momentos de triunfo y más. Los quiero mucho.*

Agradezco a mi hermano:

*A mi hermano **Aldo Alain Sánchez Cocoltzi**, no somos los hermanos más cercanos, pero te quiero demasiado. Cada vez que puedo busco ayudarte y ser un guía para ti. Espero que haya más oportunidades de acercarnos y disfrutar de la vida junto a mi hermano. Te quiero.*

RESUMEN

El presente proyecto tiene como objetivo evaluar la calidad de los componentes de software de uso libre mediante herramientas de análisis estático. La evaluación se centra en identificar y analizar atributos clave de calidad, como seguridad, fiabilidad y mantenibilidad, basándose en estándares internacionales como la norma ISO/IEC 9126:2001.

Para llevar a cabo este análisis, se investigaron y seleccionaron herramientas de análisis estático que permitan detectar problemas como bugs, vulnerabilidades y code smells, proporcionando métricas cuantitativas y cualitativas sobre la calidad del software. Asimismo, se diseñó un plan de pruebas que incluye la selección de herramientas, la configuración del entorno de análisis y la definición de métricas específicas para evaluar los componentes seleccionados.

Desde la investigación de herramientas de análisis estático hasta el análisis de los componentes seleccionados. Los resultados obtenidos permitirán no solo identificar áreas de mejora en los componentes evaluados, sino también proponer recomendaciones para optimizar su calidad en futuros desarrollos.

Este proyecto busca promover el uso de herramientas de análisis estático como parte integral del desarrollo y mantenimiento de software, asegurando que los componentes de software de uso libre cumplan con estándares de calidad que favorezcan su adopción y sostenibilidad en diversos entornos tecnológicos

TABLA DE CONTENIDO

<i>AGRADECIMIENTOS</i>	4
<i>RESUMEN</i>	5
Índice de ilustraciones	7
Índice de tablas	8
Índice de gráficas.....	8
1. GENERALIDADES DEL PROYECTO	9
1.1 INTRODUCCIÓN	9
1.2 DESCRIPCIÓN DE LA EMPRESA.....	10
1.3 ÁREA DE TRABAJO	11
1.4 PLANTEAMIENTO DEL PROBLEMA	11
1.5 OBJETIVO GENERAL	11
1.6 OBJETIVOS ESPECÍFICOS	12
1.7 JUSTIFICACIÓN.....	12
2. MARCO TEÓRICO.....	12
2.1 CALIDAD DEL SOFTWARE.....	12
2.1.1 Análisis estático de código.....	13
2.1.2 Herramientas de análisis estático de código	15
2.1.3 Normas y métricas de calidad	18
2.2 SOFTWARE DE USO LIBRE.....	24
2.2.1 La frontera entre software de uso libre y software privativo	24
2.2.2 Frameworks y librerías de software de uso libre para aprendizaje automático.....	25
3. DESARROLLO	27
3.1 INVESTIGACIÓN DE HERRAMIENTAS PARA ANÁLISIS ESTÁTICO	27
3.2 SELECCIÓN DE HERRAMIENTAS.....	33

3.3 BÚSQUEDA DE REPOSITORIOS CON PROYECTOS DE CÓDIGO ABIERTO.....	38
3.4 SELECCIÓN DE LOS COMPONENTES DE SOFTWARE DE USO LIBRE.....	42
3.5 DISEÑO DEL PLAN DE PRUEBAS DE ANÁLISIS ESTÁTICO.....	45
3.6 ANÁLISIS DE LOS COMPONENTES DE SOFTWARE.....	57
4. RESULTADOS.....	68
4.1 DIAGNÓSTICO DE CALIDAD DE LOS COMPONENTES SELECCIONADOS.....	68
CONCLUSIONES	83
COMPETENCIAS DESARROLLADAS	84
FUENTES DE INFORMACIÓN	86
ANEXOS	89

Índice de ilustraciones

FIGURA 1. INSTITUTO TECNOLÓGICO DE APIZACO.....	10
FIGURA 2. NORMA ISO/IEC 9126:2001	22
FIGURA 3. PÁGINA OFICIAL DE CRAN.....	39
FIGURA 4. REPOSITORIOS DE ALMACENAMIENTO DE LAS VERSIONES NNET	40
FIGURA 5. REPOSITORIO OFICIAL DE KERAS EN GITHUB.....	41
FIGURA 6. EJEMPLO: REPOSITORIO DE LA VERSIÓN 7.3.17 DE NNET EN GITHUB	42
FIGURA 7. IMPLEMENTACIÓN DE SONARQUBE COMO UN SERVICIO	58
FIGURA 8. HOME DE SONARQUBE.....	58
FIGURA 9. INTEGRACIÓN DE GITHUB CON CODACY.....	59
FIGURA 10. HOME DE CODACY.	59
FIGURA 11. REPOSITORIOS DE ALMACENAMIENTO EN GITHUB	63
FIGURA 12. RESULTADOS DE KERAS, PARTE 1	64
FIGURA 13. RESULTADOS DE KERAS, PARTE 2	65
FIGURA 14. RESULTADOS DE NNET, PARTE 1	66
FIGURA 15. RESULTADOS DE NNET, PARTE 2	66
FIGURA 16. RESULTADOS DE NNET, PARTE 3	67
FIGURA 17. RESULTADOS DE NNET, PARTE 4	67

FIGURA 18. CAUSA DOMINANTE DE LOS ASUNTOS DE NNET	81
--	-----------

Índice de tablas

TABLA 1. COMPARACIÓN DE HERRAMIENTAS DE ANÁLISIS ESTÁTICO	35
TABLA 2. VERSIONES DEL COMPONENTE KERAS SELECCIONADAS.....	43
TABLA 3. VERSIONES DEL COMPONENTE NNET SELECCIONADAS	44
TABLA 4. CLASIFICACIÓN: PRUEBAS DENTRO Y FUERA DE ALCANCE	46
TABLA 5. GLOSARIO: DEFINICIONES, ACRÓNIMOS Y ABREVIATURAS RELEVANTES	47
TABLA 6. CRONOGRAMA DEL DISEÑO DEL PLAN DE PRUEBAS	48
TABLA 7. ELEMENTOS CLAVE EN EL ANÁLISIS ESTÁTICO.....	51
TABLA 8. TÉCNICAS DE EVALUACIÓN APLICADAS	52
TABLA 9. ROLES Y RESPONSABILIDADES EN EL PLAN DE PRUEBAS.....	52
TABLA 10. FORMATO 1: SEGUIMIENTO PARA CASOS DE PRUEBA	53
TABLA 11. FORMATO 2: SEGUIMIENTO DE DEFECTOS ENCONTRADOS	54
TABLA 12. ANÁLISIS DE RIESGOS Y ESTRATEGIAS DE MITIGACIÓN.....	56

Índice de gráficas

GRÁFICA 1. INCREMENTO DE PROBLEMAS RELACIONADOS CON LA MANTENIBILIDAD EN KERAS	70
GRÁFICA 2. PROBLEMAS DE CONFIABILIDAD	73
GRÁFICA 3. DUPLICACIÓN DE CÓDIGO EN KERAS	78
GRÁFICA 4. INCREMENTO DE LA COMPLEJIDAD CICLOMÁTICA DE KERAS.....	79
GRÁFICA 5. ASUNTOS DE NNET	80
GRÁFICA 6. PROBLEMAS DE NNET	82

1. GENERALIDADES DEL PROYECTO

1.1 INTRODUCCIÓN

Hoy en día la integración de componentes de software de uso libre se ha convertido en una práctica habitual dentro del desarrollo de aplicaciones modernas, ya que permite a los desarrolladores aprovechar soluciones preexistentes, optimizando tiempo y recursos. Sin embargo, el desconocimiento sobre la calidad de estos componentes representa un desafío significativo, ya que muchos de ellos pueden estar incompletos o no haber pasado por procesos adecuados de prueba y validación. Esta situación puede derivar en problemas de fiabilidad, mantenibilidad o seguridad al momento de integrarlos en proyectos de software.

Por ello, surge la necesidad de evaluar previamente la calidad de estos componentes, asegurando que cumplen con los estándares necesarios para ser utilizados de forma confiable en sistemas de software. Para abordar este problema, este proyecto tiene como objetivo evaluar la calidad de componentes de software de uso libre mediante herramientas de análisis estático, las cuales permiten identificar problemas potenciales en aspectos de errores de codificación, vulnerabilidades y buenas prácticas de desarrollo.

Este proyecto presenta la investigación y se centra en el análisis de herramientas de análisis estático aplicadas a componentes de software disponibles en repositorios de uso libre. Con ello, se pretende establecer una base para decisiones informadas en la selección e integración de estos componentes, favoreciendo el desarrollo de aplicaciones más confiables y de calidad.

1.2 DESCRIPCIÓN DE LA EMPRESA

Tecnológico Nacional de México, Campus Apizaco



Figura 1. Instituto Tecnológico de Apizaco

Dirección: Av. Instituto Tecnológico No. 418, San Andrés Ahuashuatepec, Municipio de Tzompantepec, Tlaxcala, Méx. C.P. 90491.

Misión

Formar profesionistas que sean líderes visionarios, emprendedores, con capacidad de generar soluciones disruptivas, innovando en la ciencia y la tecnología, comprometidos con el desarrollo económico, responsabilidad social teniendo una visión sostenible y sustentable.

Visión

Ser una Institución de educación superior tecnológica de vanguardia, formadora de ciudadanos del mundo, con conocimientos intercultural, respetando la diversidad social.

Valores

- **Honestidad:** Se demuestra día a día a través de la transparencia con la ejecución de las actividades laborales cotidianas.
- **Responsabilidad:** Es la actitud que caracteriza al personal en el cumplimiento de su tarea con eficacia y eficiencia en beneficio de la comunidad.

- Lealtad: Es la actitud que caracteriza al personal del Instituto Tecnológico de Apizaco, aplica sus conocimientos y habilidades en el logro de las metas institucionales, sin olvidar en ningún momento el lema.

1.3 ÁREA DE TRABAJO

Departamento de División de Estudios de Posgrado e Investigación

1.4 PLANTEAMIENTO DEL PROBLEMA

La integración de componentes de software de uso libre es una práctica creciente debido a que representan soluciones rápidas para construir aplicaciones. Sin embargo, los desarrolladores de software desconocen la calidad de tales componentes de uso libre, ya que pueden estar incompletos o bien, no haber pasado por un proceso de testing. Por ello, se requiere evaluarlos antes de ser incorporados.

Actualmente, existen varias herramientas gratuitas para el análisis estático que permite testear software. El análisis estático permite hacer pruebas al software, sin necesidad de observar su funcionamiento en ejecución. Un diagnóstico de análisis estático puede apoyar al desarrollador de software para identificar posibles anomalías en los componentes de software, antes de ser incorporados. De esta manera, el desarrollador puede tomar decisiones para evitar fallas en sus aplicaciones.

1.5 OBJETIVO GENERAL

Evaluar la calidad de componentes de software mediante el uso de herramientas gratuitas para el análisis estático en repositorios de software de uso libre.

1.6 OBJETIVOS ESPECÍFICOS

- Realizar una revisión de las herramientas gratuitas que existen para el análisis estático.
- Seleccionar la herramienta gratuita para el análisis estático que ofrezca el diagnóstico más adecuado.
- Seleccionar el conjunto de componentes de software de uso libre a evaluar desde repositorios de proyectos de código abierto.
- Diseñar un plan de pruebas para el análisis estático de los componentes en el repositorio.
- Obtener un diagnóstico con las anomalías encontradas.
- Integrar un informe de resultados.

1.7 JUSTIFICACIÓN

Los desarrolladores de software requieren conocer herramientas para diagnosticar la calidad del software. Esto les permite tomar decisiones durante los procesos de desarrollo que van desde la selección de la herramienta más adecuada, hasta el planteamiento e instrumentación de pruebas para mejorar la calidad del software.

2. MARCO TEÓRICO

2.1 CALIDAD DEL SOFTWARE

El software se puede definir como un conjunto de programas intangibles que reciben ordenes, procesan datos y reflejan resultados, estas características hacen que el software sea muy demandado puesto que complementa o automatiza tareas y procesos que se llevan a cabo día a día (electrónica, 2021).

En cuanto a la calidad, se refiere al grado que un producto o servicio cumple con requisitos y expectativas por los clientes o usuarios finales, lo que implica la conformidad con especificaciones y la capacidad de satisfacer necesidades de manera consistente y confiable.

El impulso por la calidad de software comenzó en la década de 1980 debido a la rápida evolución tecnológica y la creciente dependencia de sistemas críticos resaltaron la urgencia de desarrollar métodos estandarizados para garantizar la calidad en el desarrollo de software (Román, 2024).

Convirtiéndose en un pilar fundamental en el desarrollo de productos tecnológicos de los últimos años, la calidad de software ayuda a cumplir las expectativas de los clientes a través de la fiabilidad y eficiencia de estos productos. La calidad de software depende de aplicar técnicas de gestión y control de producción, asegurando que el proyecto sea tratado como un proceso de ingeniería rigurosa.

Para lograr una alta calidad en proyectos de desarrollo, la calidad de software se compone de elementos clave que aseguran que el producto final cumpla con los requisitos y expectativas del cliente. La gestión de calidad coordina todas las actividades y recursos relacionados con la calidad de software. El ciclo de vida del software abarca todas las fases que un software atraviesa en su desarrollo.

Se le da seguimiento a través del aseguramiento de la calidad en cuestión del cumplimiento de los procesos de desarrollo de software. El control de calidad comprende, documenta y valida con el cliente los requerimientos funcionales y no funcionales del producto solicitado y, finalmente, las pruebas o testing deben diseñarse de forma óptima y aplicarse para comparar los resultados obtenidos contra los requerimientos del cliente, para ser compartidos y emitirse evaluaciones de cumplimiento o defectos a corregir.

2.1.1 Análisis estático de código

El análisis de código estático se define como una opción de prueba para el software de las aplicaciones o programas que usa métodos de revisión que detectan los potenciales fallos, como errores de funcionamiento, bugs o riesgos del sistema (Mallón, 2024).

A diferencia del análisis dinámico, sin ejecutar el código, mediante una serie de pruebas y análisis ejecutados sobre el código fuente de una aplicación puede detectar problemas potenciales. Lo que permite a equipos de desarrolladores

tener una capa más control de calidad que facilita la detección de los problemas más difíciles de encontrar mediante otros procedimientos.

Al identificar malas prácticas de programación, el análisis estático puede ayudar a mejorar la calidad del código y la habilidad de los desarrolladores. Al integrarlo en el ciclo de vida del desarrollo de software para asegurar desde el principio, reduciendo el costo y el tiempo de corrección de errores. Suele realizarse en las primeras etapas del desarrollo, como el caso de organizaciones que practican DevOps, este se lleva a cabo durante la fase de creación.

Ventajas del análisis estático del código

- **Detección precoz de errores**
Permite al equipo de desarrollo y pruebas detectar errores en una fase temprana del desarrollo. Gracias a ello, se puede corregir fácilmente los errores del código antes de añadir más dependencias.
- **Mejora de la calidad del código**
Un código con una alta calidad significa tener un software fiable, sofisticado y seguro. Por ello, el análisis estático permite eliminar código obsoleto o deudas técnicas para que el software funcione mejor y hace que sea más fácil de mantener.
- **Mayor seguridad**
Ante malos actores u otros atacantes que buscan vulnerabilidades en el código fuente, realizar un análisis estático permite adoptar un enfoque proactivo de la seguridad y a su vez tomar las medidas necesarias que permiten ahorrar costos y ayudar a mantener el software.

Tipos de análisis de código estático

En primer lugar, encontramos el análisis de código estático basado en patrones. Esta aproximación está programada mediante reglas analíticas que permiten detectar casos indeseables, como puede ser un error en un método constructor que ocasionaría un error en tiempo de ejecución.

Por otra parte, el análisis de código estático basado en flujos puede detectar posibles problemas de ejecución. Detecta errores complejos relacionados con la interacción entre diferentes partes del código, como la creación, modificación y uso de variables en el código (Go4IT Solutions, 2024).

Análisis estático vs análisis dinámico

El análisis estático identifica defectos antes de ejecutar un programa, mientras que un análisis de código dinámico lo hace después de ejecutar un programa. Sin embargo, es posible que algunos errores de codificación no se detecten durante las pruebas unitarias. Por lo que existen defectos que las pruebas dinámicas pueden pasar por alto y que el análisis estático si puede detectar (Foster, 2023).

Ambas técnicas son esenciales y complementarios para garantizar la calidad del software puesto que el análisis estático garantiza que el código sea legible y de calidad el análisis dinámico asegura que funcione correctamente en condiciones reales. Juntos forman una estrategia integral para desarrollar software robusto y confiable.

2.1.2 Herramientas de análisis estático de código

Las herramientas de análisis estático de código son programas que examinan el código fuente sin ejecutarlo, buscando problemas potenciales o áreas de mejora. Actúan como guardianes, detectando errores de sintaxis, vulnerabilidades de seguridad, complejidad excesiva y otras deficiencias que podrían afectar la calidad del software (C, 2024).

Importancia de las herramientas de análisis estático de código

Las herramientas de análisis estático desempeñan un papel crucial ante el desarrollo de software por varias razones como:

- Mejora de la calidad del código: Identificando problemas en el código fuente antes de la ejecución del programa lo que ahorra tiempo y esfuerzo a largo plazo.
- Aumenta la seguridad: Puede detectar vulnerabilidades de seguridad antes de que sean explotadas, lo que ayuda a proteger los proyectos y datos.
- Facilita la colaboración: En trabajos en equipo, las herramientas de análisis estático de código ayudan a mantener un estándar de codificación común y facilitan la colaboración entre miembros.

- Ahorra costos: Identificar y solucionar problemas en una etapa temprana del desarrollo ahorra costes en comparación con la corrección de errores en etapas avanzadas del proyecto.

Se busca promover mejores prácticas de desarrollo al proporcionar retroalimentación automática y objetiva. Su uso mejora la mantenibilidad del software.

Funcionamiento de las herramientas de análisis estático de código

En primer lugar, antes de que se pueda realizar cualquier análisis significativo, la herramienta analiza el código. El código fuente debe traducirse a un formato que la herramienta pueda analizar con mayor facilidad. Esto implica convertir el código en un árbol de sintaxis abstracta u otra representación intermedia.

Estas herramientas utilizan un conjunto de reglas, patrones y algoritmos para examinar el código fuente en busca de problemas. Estas reglas pueden abarcar desde la detección de errores de sintaxis hasta la identificación de patrones de código que no cumplen con las mejores prácticas de programación.

Cuando se ejecuta una herramienta de análisis estático, escanea el código fuente y genera un informe que enumera los problemas encontrados junto con recomendaciones para su corrección. Estos informes detallan y proporcionan información útil para los desarrolladores, como la ubicación exacta de los problemas en el código fuente (Casero, 2024).

Principales características de las herramientas de análisis estático de código

Estas herramientas tienen funciones diseñadas para proporcionar evaluaciones exhaustivas del código y agilizar el proceso de desarrollo. Si bien las funciones específicas pueden variar según la herramienta, varias funcionalidades básicas son estándar en muchas plataformas de análisis de código estático.

Análisis basado en reglas

- Reglas predefinidas: Contienen reglas integradas que detectan errores de codificación comunes, vulnerabilidades o desviaciones de los estándares de codificación.

- Niveles de gravedad: Los problemas detectados son clasificados por niveles de gravedad, lo que permite a los desarrolladores priorizar las correcciones.

Reglas personalizadas

- Personalización: Los equipos pueden tener necesidades específicas. Muchas herramientas permiten a los usuarios definir reglas personalizadas adaptadas a los requisitos de un proyecto o una organización.
- Gestión de reglas: Las herramientas proporcionan interfaces o configuraciones para administrar, habilitar o deshabilitar reglas específicas según las necesidades del proyecto.

Reportando

- Informes detallados: Las herramientas generan reportes detallados que describen los problemas detectados, sus ubicaciones y, a menudo, soluciones o referencias sugeridas para comprender el problema.
- Visualización: Algunas herramientas proporcionan paneles o funciones de visualización que ofrecen una descripción general del estado del código, las tendencias a lo largo del tiempo o las áreas que necesitan atención.

Integraciones

- Pipelines de CI/CD: Las herramientas de análisis de código pueden integrarse sin problemas en pipelines de integración continua, lo que garantiza las verificaciones de código se realicen automáticamente durante las etapas de compilación o implementación.
- Sistemas de control de versiones: La integración con plataformas como GitHub, GitLab o Bitbucket permite que el análisis de código sea parte de las verificaciones de solicitudes de extracción o fusión.

Si bien el análisis estático, con sus crecientes capacidades y características, seguirá desempeñando un papel fundamental para garantizar que el código funcione como se espera y resista la prueba del tiempo en términos de confiabilidad y seguridad (Codacy, 2023).

2.1.3 Normas y métricas de calidad

Las métricas de calidad del software son criterios y medidas utilizadas para evaluar qué tan bien se desarrolla una solución de software por un grupo. Son una brújula que nos ayuda a entender la eficiencia, confiabilidad y rendimiento del software, asegurando que cumpla con los estándares y requisitos establecidos (Navarro, 2024).

Importancia de las métricas de calidad de software

- **Evaluación imparcial.** Los desarrolladores y los QA pueden evaluar si sus herramientas y el sistema de pruebas cumplen todas las normas exigidas o si necesitan modificarlo para aumentar la eficacia de las pruebas.
- **Mejor toma de decisiones.** Las métricas de pruebas de software ayudan a los equipos de entrega a tomar decisiones más informadas sobre el proceso de pruebas. Por ejemplo, las métricas de pruebas de automatización revelarían si una prueba automatizada requeriría más tiempo del presupuestado.
- **Visibilidad mejorada.** El equipo de entrega puede determinar qué cambios de control de calidad deben realizar para garantizar que se detectan los errores críticos de forma coherente y que el producto cumple las normas. Las métricas de las pruebas ayudan a detectar las áreas más propensas a errores y a conectar los puntos entre determinadas acciones de los usuarios y los fallos de funcionamiento de la aplicación.

La mayoría de empresas que buscan realizar una transformación digital de sus servicios o su plan de negocios y pretenden evolucionar sus procesos más complejos a través de automatizaciones o desarrollos nuevos, quieren que la calidad se garantice en todo momento.

La gestión de calidad de software unida a términos como QA control, QA assurance o QA management se plantea como un requisito indispensable en la construcción de software y junto a los estándares de calidad de software se

busca garantizar la calidad constante de los procesos y productos (ICARIA Technology, 2023).

Principales métricas de calidad en el análisis estático de código

En el panorama actual de desarrollo de software la calidad del código afecta no sólo el rendimiento y la funcionalidad del software, sino también la facilidad con la que los desarrolladores pueden ampliar y mantener el código a lo largo del tiempo (IN-COM, 2024).

A continuación, se exploran métricas clave para medir la calidad del código que ayudan a los desarrolladores a priorizar la calidad en sus flujos de trabajo.

❖ Bugs

➤ Categoría: Confiabilidad

- Los bugs se relacionan con errores de programación que pueden provocar fallos en el comportamiento del software.

❖ Vulnerabilidades

➤ Categoría: Seguridad

- Representan puntos débiles en el código que podrían ser explotados por atacantes para comprometer la seguridad del sistema.

❖ Code smells

➤ Categoría: Mantenibilidad

- Indican problemas en el diseño o estructura en el código que dificultan su mantenimiento, pero no afectan directamente su funcionalidad.

❖ Duplicación de código

➤ Categoría: Mantenibilidad

- La duplicación de código aumenta la complejidad de mantenimiento y la probabilidad de errores si las duplicaciones no se actualizan de manera consistente.

❖ Puntos de acceso de seguridad

➤ Categoría: Seguridad

- Son partes del código donde se gestionan recursos críticos, como datos sensibles, y requieren especial atención para evitar vulnerabilidades.

❖ Complejidad ciclomática

➤ Categoría: Mantenibilidad

- Mide el número de caminos independientes a través del código, lo que refleja su dificultad de comprensión, prueba y mantenimiento.
- ❖ Complejidad cognitiva
 - Categoría: Mantenibilidad
 - Evalúa la dificultad mental que requiere entender el flujo lógico del código, siendo un indicador clave para la legibilidad y simplicidad.

Estándares de calidad de software

Normas enfocadas de forma global a la Gestión de la Calidad

Estándares internacionales diseñados para establecer principios, prácticas y requisitos que garanticen la calidad de productos, servicios y procesos en diversas industrias. Promueven la consistencia, la mejora continua y la satisfacción al cliente mediante un enfoque estructurado en la planificación, implementación, control y evaluación de la calidad (Medina, 2022).

Ejemplos:

- ISO 9001 – Gestión de la Calidad
- ISO 10005:2018 Sistemas de Gestión de la Calidad. Directrices para los planes de la calidad

Normas enfocadas al Aseguramiento de la Calidad

Estándares que establecen directrices, procedimientos y requisitos para garantizar que los procesos y productos cumplan los criterios de calidad definidos. Estas normas se centran en prevenir errores o defectos durante las etapas de desarrollo y producción, promoviendo la implementación de controles sistemáticos que aseguren la conformidad con los estándares especificados.

Ejemplos:

- ISO 33000 Calidad de los procesos de desarrollo de software
- ISO 12207 Modelos de Ciclos de Vida del Software
- ISO IEC IEEE 12207 Procesos de ciclo de vida de software
- IEEE 730 – 2002 Standard for Software Quality Assurance

Normas enfocadas al Control de la Calidad

Estándares que establecen los métodos y procedimientos para inspeccionar, medir y verificar la calidad de productos o procesos, asegurando que cumplan con los requisitos establecidos. Estas normas se aplican durante las etapas de producción o desarrollo para identificar y corregir defectos antes de que los productos lleguen al usuario final.

Ejemplos:

- ISO IEC 25000
- ISO 5055 Software Quality Standards

Normas enfocadas a las Pruebas o al Testing

Estándares diseñados para proporcionar directrices y procedimientos que aseguren la planificación, ejecución y documentación adecuada de las pruebas de software. Estas normas buscan garantizar que los productos sean funcionales, confiables y cumplan con los requisitos especificados antes de su implementación o liberación.

Ejemplos:

- ISO IEC IEE 29119 Norma para la documentación de prueba de software.
Standard for Software Test Documentation
- ISO IEC 20246 Ingeniería de Software

Uso de la norma ISO/IEC 9126:2001

Este estándar presenta un marco conceptual para el modelo de calidad y define un conjunto de características y subcaracterísticas que debe cumplir todo producto de software. Establece que cualquier componente de la calidad de software puede ser descrito en términos de alguno de sus características básicas: funcionalidad, confiabilidad, usabilidad, eficiencia, mantenibilidad, portabilidad y eficiencia (Verity, 2022).

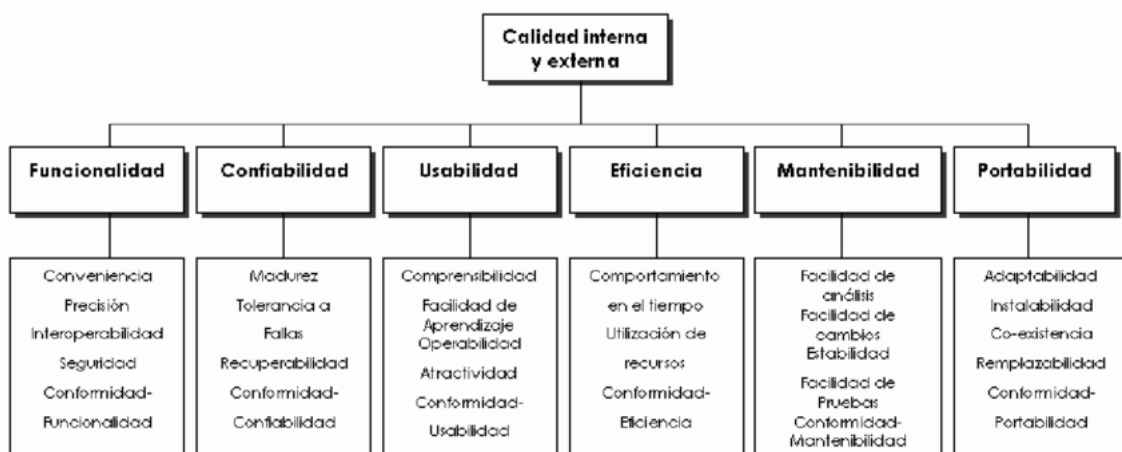


Figura 2. Norma ISO/IEC 9126:2001

Fuente: Castillo, I. (2006). Modelo para la calidad interna y externa según ISO/IEC 9126:2001. [Figura]. Recuperado de <https://www.researchgate.net>

En la figura 2 se muestra el modelo el cual su importancia radica en proporcionar un marco estandarizado para evaluar la calidad del software de manera objetiva, sistemática y medible. Sus características y subcaracterísticas abarcan aspectos críticos del desempeño y comportamiento del software. Resultando esencial para los desarrolladores, evaluadores y clientes que buscan asegurar la calidad y efectividad de los sistemas de software.

Considerando dos perspectivas principales, tenemos que la calidad interna se relaciona con las métricas que se pueden medir directamente en el código fuente, como la estructura y la coherencia del diseño. Mientras que la calidad externa, se refiere a cómo el software se comporta durante su ejecución y a su capacidad para satisfacer los requisitos del usuario.

Características de calidad de un software según la ISO/IEC 9126: 2001

Según (Manzanelli, 2023), las características que aborda la norma ISO/IEC 9126:2001 son las siguientes:

- **Funcionalidad:** se evalúa la adecuación, el cumplimiento funcional, idoneidad, corrección, interoperabilidad, conformidad y seguridad de acceso. Por lo que es posible afirmar que la funcionalidad determina la capacidad del software de funcionar en términos de lo que el usuario necesita, de interactuar con otros sistemas y que permita el acceso de

diferentes personas pero que cumpla con las regulaciones de las leyes de protección de datos.

- **Confiabilidad:** se tienen en cuenta aspectos como la capacidad y facilidad de recuperación, la mitigación de fallos, cantidad de tiempo que el software está disponible para su uso y la tolerancia.
- **Usabilidad:** mide el grado en que el software es fácil de usar, qué tan intuitivo es, el manejo que el usuario le da al sistema y si este presenta menús sencillos, lectura de textos ágil, cuenta con funciones de forma clara y puntual, entre otros.
- **Eficiencia:** analiza y mide la capacidad en que el software hace óptimo el uso de los recursos del sistema, en términos de tiempo de uso y recursos de los cuales dispone.
- **Mantenibilidad:** la facilidad con que una modificación puede ser realizada, la capacidad para hacerle pruebas de rendimiento, regresión, accesibilidad, inspección de código y toda la ingeniería de requerimientos. Esto hace que el producto de software sea escalable ya que es posible hacerle mejoras constantes sin que este no tenga ningún problema al querer realizarle alguna modificación o incorporación.
- **Portabilidad:** la facilidad con que el software puede ser llevado de un entorno a otro. Proporciona facilidad de instalación, facilidad de ajuste, facilidad de adaptación al cambio y otros aspectos que lo hacen un sistema que garantiza portabilidad.
- **Eficiencia:** se tiene en cuenta el cumplimiento de todos los requerimientos para los que fue desarrollado el software, basado en las expectativas del cliente final. Estas expectativas se definen de acuerdo con las necesidades, y estas se basan en la eficiencia y efectividad en coherencia para lo que fue hecho el producto.

2.2 SOFTWARE DE USO LIBRE

El software libre es todo aquel software cuyo código fuente se puede estudiar, modificar y utilizar libremente con cualquier fin, e incluso, copiar y redistribuir el programa con cambios o sin ellos (Souza, 2019).

Un programa es software libre si se tiene la libertad de:

- Ejecutar el programa como se desee, con cualquier propósito.
- Estudiar cómo funciona el programa, y cambiarlo para que haga lo que se desee. El acceso al código fuente es una condición necesaria para ello.
- Redistribuir copias para ayudar a otros.
- Distribuir copias de sus versiones modificadas a terceros. Este permite ofrecer a toda la comunidad la oportunidad de beneficiarse de las modificaciones. El acceso al código fuente es una condición necesaria.

2.2.1 La frontera entre software de uso libre y software privativo

El alcance que debe tener cada una de las libertades antes mencionadas se describen con más precisión en diversos asuntos, para que un programa sea libre.

- La libertad de ejecutar el programa como se desee: La libertad de ejecutar el programa permite a cualquier persona usarlo para cualquier propósito, sin necesidad de informar al programador o a otra entidad. Lo que importa es el propósito del usuario, no el del programador.
- La libertad de estudiar el código fuente y modificarlo: El código fuente debe ser accesible y no ofuscado. Se incluye la posibilidad de usar versiones modificadas, pero si el software bloquea estas versiones, como en la "tivoización" (proceso de restringir las libertades del software libre mediante medidas de hardware que limitan su modificación y redistribución), no se considera libre.
- La libertad de redistribuir copias su así lo desea: Se tiene la libertad de modificar y usar el software en privado, sin obligación de notificar

cambios. Publicar versiones modificadas como software libre es permitido, pero una licencia que prohíba esto no es libre.

- Copyleft: Regla que impide agregar restricciones que nieguen las libertades principales al redistribuir software, protegiendo esas libertades.
- Reglas acerca del empaquetamiento y la distribución: Las reglas sobre cómo empaquetar una versión modificada son aceptables si no limitan significativamente la libertad de publicar o usar versiones modificadas.
- Normas de exportación: Las licencias de software libre no deben imponer normas de exportación como condición de uso, ya que esto restringe las libertades esenciales.
- Consideraciones legales: Las libertades deben ser permanentes y no revocables arbitrariamente. Una licencia libre no puede exigir cumplir condiciones de programas no libres.
- Licencias basadas en contrato: La mayoría de las licencias de software libre se basan en el copyright y respetan la libertad del usuario, lo que reduce problemas imprevistos.

2.2.2 Frameworks y librerías de software de uso libre para aprendizaje automático

Un framework se trata de una plantilla o esquema conceptual que sirve para organizar y desarrollar un software determinado. Su objetivo es hacer más sencilla la programación ofreciendo grandes ventajas para los programadores ya que permite automatizar multitud de procesos (VIEWNEXT, 2021).

En el campo de aprendizaje automático sirven para construir, entrenar, y desplegar modelos de aprendizaje automático y redes neuronales. Incorporan estructuras modulares, optimizaciones y soporte para hardware especializado.

Estos son algunos ejemplos:

- TensorFlow
- PyTorch
- Apache MXNet

Por otro lado, una librería se trata de un fragmento de código elaborado por un tercero y que sirve básicamente para solucionar un problema concreto con el objetivo de ampliar las funcionalidades del código (VIEWNEXT, 2021).

En tareas específicas en el campo del aprendizaje automático, se suelen enfocar en preprocesamiento de datos, implementación de algoritmos o visualización de resultados.

Algunos ejemplos:

- Scikit-learn
- Keras
- XGBoost

La importancia de ambos en el aprendizaje automático se debe a que nos permite acceso y colaboración entre desarrolladores e investigadores, rapidez y eficiencia en el desarrollo y experimentación de modelos, y crear ecosistemas amplios por medio de comunidades activas para resolver problemas.

La diferencia entre estas son que los frameworks proporcionan un marco de trabajo para desarrollar aplicaciones, mientras que las librerías únicamente solucionan un problema concreto, haciendo el código más fácil de leer.

nnet

Librería de software para el lenguaje de programación R, diseñada específicamente para trabajar redes neuronales feed – forward con una sola capa oculta y construir modelos log – lineales multinomiales. Es una herramienta compacta y sencilla que permite a los usuarios explorar conceptos básicos de aprendizaje automático con redes neuronales en un entorno estadístico.

Keras

Es una API de alto nivel para redes neuronales que facilita la construcción, entrenamiento y evaluación de modelos de aprendizaje profundo.

3. DESARROLLO

3.1 INVESTIGACIÓN DE HERRAMIENTAS PARA ANÁLISIS ESTÁTICO

En el ámbito de desarrollo de software, es crucial garantizar la calidad del código fuente para prevenir errores, mejorar el rendimiento y simplificar el proceso de mantenimiento. A pesar de ello, alcanzar este grado de calidad puede resultar todo un desafío, sobre todo en proyectos de gran envergadura o con varios múltiples desarrolladores. Las herramientas de análisis estático de código juegan un papel fundamental en este punto, ya que permiten detectar problemas de manera automática sin necesidad de ejecutar el programa.

A pesar de la disponibilidad de numerosas herramientas de análisis estático elegir la más apropiada para un proyecto específico puede resultar complicado. Es indispensable realizar un análisis minucioso de las diferencias en características, compatibilidad con lenguajes de programación, escalabilidad y soporte técnico antes de tomar decisiones informadas. La ausencia de una evaluación sistemática de estas herramientas puede resultar en inversiones poco eficientes en cuanto a tiempo y recursos.

A continuación, se presenta y analizan varias herramientas de evaluación de calidad de software, destacando sus principales características e importancia en la evaluación del análisis de código.

SonarQube

SonarQube es una herramienta de análisis estático de código ampliamente utilizada para evaluar la calidad del software en múltiples lenguajes de programación. Con el propósito de identificar problemas relacionados con la mantenibilidad, la seguridad, el rendimiento y la conformidad con estándares de codificación.

Funcionalidades principales

- Realiza revisiones automáticas con análisis de código estático.

- Facilita informes ofreciendo información objetiva de la calidad actual de los proyectos utilizando métricas y gráficos de prueba de calidad avanzados.
- Es ampliable con el uso de complementos.
- Permite la creación de paneles y filtros personalizables para centrarse en áreas clave y entregar productos de calidad a tiempo.

Lenguajes soportados

- Java, JavaScript
- C, C++, C#
- VB.net
- PHP
- Python
- PL/SQL
- COBOL

Integración con IDEs y otras herramientas

- GitHub Actions
- GitLab CI/CD
- Azure Pipelines
- Bitbucket Pipelines
- Jenkins
- SonarLint, extensión para Visual Studio Code

Licencia

- Edición gratuita para la comunidad
- Edición Developer, \$ 160
- Edición Enterprise, hablar con ventas
- Edición Data Center, hablar con ventas

Comunidad y soporte

- Documentación oficial de la herramienta
- Guías para desarrolladores
- Comunidad Sonar
- Soporte técnico disponible para todas las ediciones, el soporte aumenta de nivel conforme la edición adquirida.

Codacy

Codacy es una herramienta de análisis de código estático basada en la nube enfocada en automatizar revisiones de código para mejorar la calidad y mantener estándares para proyectos de desarrollo de software. Codacy es compatible con múltiples lenguajes de programación y puede analizar métricas como mantenibilidad, cobertura de pruebas, calidad del código, duplicación y cumplimiento de estándares de seguridad.

Funcionalidades principales

- Fácil integración
- Escaneos de código sin canalización
- Escaneos de código basados en solicitudes de extracción fuera de su CI/CD
- Análisis de seguridad integral con SAST, SCA, DAST y análisis de pentest

Lenguajes soportados

Codacy proporciona análisis estático, así como duplicación de código, complejidad de código, detección de secretos, escaneo de vulnerabilidad de dependencia y métricas de cobertura de código para lenguajes clave.

- C, C++, C#
- CSS
- Dart
- Go

- Java, JavaScript
- JSON
- Kotlin
- Objective-C
- PHP
- PL/SQL, PostgreSQL
- Python
- Ruby, Rust
- TypeScript
- XML

Integración con IDEs y otras herramientas

Codacy proporciona análisis estático y detección de secretos para aplicar las mejores prácticas de seguridad y cumplimiento.

- GitHub
- GitLab
- Bitbucket
- Docker
- Jira

Licencia

- Edición gratuita, repositorios públicos y privados en la nube limitados
- Edición Pro, \$15
- Edición para negocios, contactar a ventas

Comunidad y soporte

- Documentación oficial
- Guías de configuración
- Comunidad activa

CodeScene

CodeScene es una herramienta de análisis y visualización de código, obtiene información útil para reducir eficazmente la deuda técnica y entregar código limpio mediante la comparación de factores contextuales como la calidad del código, la dinámica del equipo y el resultado de entrega para la toma de decisiones.

Funcionalidades principales

- Calcular los costes de desarrollo.
- Análisis basado en evolución, analiza cambios históricos en los repositorios
- Cuantificación del coste de la deuda técnica
- Cobertura de código
- Configuración de métricas de cobertura

Lenguajes soportados

- Java, JavaScript
- Python
- C#
- PHP

Integración con IDEs y otras herramientas

- GitHub
- Jira
- Trello
- Bitbucket
- Jenkins
- Azure DevOps
- Extensión para Visual Studio Code

Licencia

- Versión gratuita para proyectos de código abierto
- Versión Estándar, 18 euros
- Versión Pro, 27 euros
- Versión Empresa, precio personalizado

Comunidad y soporte

- Centro de desarrolladores
- Documentación oficial
- Tutoriales para desarrolladores
- Casos de uso
- Soporte técnico para la versión empresarial

DeepSource

Herramienta de análisis estático de código diseñada para ayudar a los desarrolladores a identificar y solucionar problemas relacionados con la calidad, seguridad y estándares del código de manera automatizada. Es una herramienta que simplifica el proceso de revisión y mantenimiento de código, proporcionando análisis detallados y sugerencias prácticas directamente desde el repositorio del proyecto.

Funcionalidades principales

- Cobertura del código.
- Análisis de IaC.
- Hospedaje propio
- Reparación automática
- Informes detallados

Lenguajes soportados

- PHP
- Java, JavaScript
- TypeScript

- Python
- C#, C++
- Go
- Rust
- SQL
- Scal

Integración con IDEs y otras herramientas

- GitHub, GitLab
- Bitbucket
- Repositorios de fuentes de google
- Azure DevOps

Licencia

- Edición gratuita
- Edición Starter, \$8
- Edición Business, \$24
- Edición Enterprise, contactarse con ventas.

Comunidad y soporte

- Documentación oficial
- Foro oficial de la comunidad
- Apoyo comunitario

3.2 SELECCIÓN DE HERRAMIENTAS

La selección de herramientas de análisis estático es un proceso clave para garantizar la efectividad en la evaluación de componentes de software. Este proceso se fundamenta en una serie de criterios de selección que permiten

identificar las herramientas más adecuadas de acuerdo con las necesidades específicas del proyecto.

A continuación, se detallan los principales criterios considerados, y su aplicación a las herramientas antes vistas, es decir, SonarQube, Codacy, CodeScene y DeepSource.

Criterios de selección

La selección de herramientas de análisis estático de código debe basarse en criterios fundamentales que permitan evaluar su eficacia y adaptabilidad en distintos contextos de desarrollo.

Según Mobilvendedor (2024), se describen dichos criterios y la consideración de aspectos con el objetivo de determinar su idoneidad para satisfacer las necesidades de este proyecto.

1. Funcionalidad y características

Se evalúa las capacidades técnicas de las herramientas en relación con el análisis estático. Se consideran aspectos como:

- Cobertura de lenguaje: Compatibilidad con los lenguajes de programación utilizados en el proyecto.
- Tipos de análisis ofrecidos.
- Facilidad de configuración: Como las opciones intuitivas para personalizar las reglas de análisis.

2. Escalabilidad y rendimiento

Se evalúa que la herramienta sea capaz de manejar proyectos de distintos tamaños sin comprometer su eficiencia. Se consideran aspectos como:

- Velocidad de análisis.
- Capacidad para proyectos grandes.
- Consumo de recursos.

3. Integración y compatibilidad

Se evalúa que la herramienta debe integrarse sin dificultades en el entorno de desarrollo existente. Se consideran aspectos como:

- Compatibilidad con IDEs y CI/CD: Soporte para integrarse con plataformas de desarrollo y sistemas de integración continua.

- Formatos de reporte.
 - Interoperabilidad: Compatibilidad con otras herramientas de análisis o gestión de proyectos.
4. Comunidad y soporte
- Se evalúa que la herramienta cuente con un fuerte respaldo por parte de la comunidad y un soporte técnico sólido que garantice la sostenibilidad del uso de la herramienta. Se consideran aspectos como:
- Documentación disponible.
 - Foros y comunidades activas.
 - Frecuencia de actualizaciones.
5. Licencia y costos
- Se evalúa el modelo de licencia y los costos asociados a los proyectos con el fin de minimizar gastos. Se consideran aspectos como:
- Gratuidad o costo accesible.
 - Restricciones de uso.
 - Costos asociados: Gastos adicionales en caso de requerir módulos premium.

Evaluación y comparación entre herramientas

Tabla 1. Comparación de herramientas de análisis estático

Herramientas de análisis estático de código

Criterios de selección	SonarQube	Codacy	CodeScene	DeepSource
Funcionalidad y características	Detección de bugs, vulnerabilidades y code smells; soporte para múltiples lenguajes; análisis incremental.	Detección de problemas de cobertura de código, integración con repositorios.	Enfoque en análisis basado en cambios, visualización de problemas de deuda técnica y patrones de equipo.	Identificación automática de errores, soporte para múltiples lenguajes, recomendaciones personalizadas.

Escalabilidad y rendimiento	Alta escalabilidad en implementaciones locales y en la nube.	Eficiente para proyectos pequeños y medianos; óptimo para grandes bases de código.	Adaptado para proyectos dinámicos con análisis focalizado en la evolución del código.	Escalabilidad alta en la nube con capacidad para manejar grandes repositorios.
Integración y compatibilidad	Integración con CI/CD, herramientas como Jenkins y GitLab.	Compatibilizada con plataformas como GitHub, GitLab, Bitbucket.	Soporte para CI/CD, integración con repositorios Git.	Integración con GitHub, GitLab, Bitbucket, soporte para CI/CD.
Comunidad y soporte	Amplia comunidad, extensa documentación y soporte comercial.	Comunidad en crecimiento, buen soporte técnico y foros de ayuda.	Comunidad más pequeña, pero especializada, soporte profesional disponible.	Comunidad activa con soporte técnico rápido y foros en línea.
Licencia y costos	Edición gratuita (Comunidad) y versiones de pago.	Modelos de pago escalables, planes para uso gratuito disponibles.	Licencias comerciales, costos relacionados con funcionalidades avanzadas.	Planes gratuitos y de pago, costos según el tamaño del equipo y la personalización.

Fuente: Elaboración propia

Como se muestra en la tabla 1 se realizó la comparación de las cuatro herramientas considerando los criterios descritos anteriormente en la selección de herramientas. Estas herramientas constan de SonarQube, Codacy, CodeScene y DeepSource.

Conclusión del proceso de selección de herramientas

Después de realizar un análisis detallado y comparativo de las herramientas de análisis estático de código consideradas en este proyecto, se ha concluido que SonarQube y Codacy serán las herramientas utilizadas durante su desarrollo.

Esta decisión se fundamenta en los siguientes puntos clave:

1. Amplia funcionalidad y características

Ambas herramientas ofrecen una cobertura robusta para detectar problemas comunes, como bugs, vulnerabilidades y code smells. Además, permiten evaluar múltiples lenguajes de programación, lo que se alinea con las necesidades de este proyecto.

2. Escalabilidad y rendimiento confiables

Las capacidades de estas herramientas respecto a la implementación local y en la nube, así como la velocidad en las pruebas iniciales y análisis rápidos las hacen adecuadas para el desarrollo de este proyecto.

3. Integración y compatibilidad

Tanto Codacy como SonarQube se integran de manera eficiente con herramientas CI/CD, lo que garantiza que el flujo de trabajo del proyecto sea continuo y sin interrupciones.

4. Comunidad activa y soporte sólido

La amplia comunidad y documentación de ambas herramientas aseguran un respaldo confiable durante su uso. Además, las comunidades y el soporte técnico hacen que la resolución de dudas sea más ágil.

5. Flexibilidad de costos y licencias

Ambas herramientas ofrecen versiones gratuitas que proporcionan las funcionalidades esenciales necesarias para cumplir con el objetivo de este proyecto, lo que permite maximizar los recursos sin comprometer la calidad del análisis.

3.3 BÚSQUEDA DE REPOSITORIOS CON PROYECTOS DE CÓDIGO ABIERTO

Un repositorio o también solo llamado repo, es un tipo de almacenamiento digital centralizado que los desarrolladores utilizan para realizar y administrar cambios en el código fuente de una aplicación, ya que los desarrolladores tienen que almacenar y compartir carpetas, archivos de texto y otros tipos de documentos al desarrollar software.

El uso de repositorios permite a los equipos de desarrollo de software aplicar múltiples cambios al código de un programa si comprometer el código fuente principal. Ya que en lugar de aplicar los cambios directamente, utilizan características de un repositorio para editar y revisar los cambios.

Uso de repositorios de GitHub

GitHub es un repositorio basado en la nube que permite almacenar y trabajar en los códigos de los proyectos de forma organizada, GitHub está basado en Git, un sistema de control de versiones, e incluye características que mejoran la colaboración entre desarrolladores.

Dado que GitHub es de los repositorios más populares y utilizado entre las comunidades de desarrolladores, la búsqueda de repositorios de proyectos de código abierto se centra en los repositorios de GitHub.

Selección de repositorios

Los proyectos por seleccionar deben cumplir con los siguientes puntos:

- Estar relacionados con redes neuronales y aprendizaje automático, ya que son el enfoque principal del análisis.
- Disponibilidad del código fuente para su descarga y modificación.
- Buena documentación que facilite el entendimiento del código y su propósito.
- Actividad alta y relevancia entre la comunidad.

Proceso de búsqueda

1. Entrar a la plataforma GitHub.

2. Uso de palabras clave, como “Keras”, “deep learning”, “neural networks”, “log-linear models”, “feed-forward neural networks”, para localizar repositorios relacionados.

3. Caso de nnet

Para el uso de nnet que no se encontraron repositorios adecuados, se tomó la decisión de descargar las versiones desde la fuente original y crear repositorios en GitHub para poder gestionarlas.

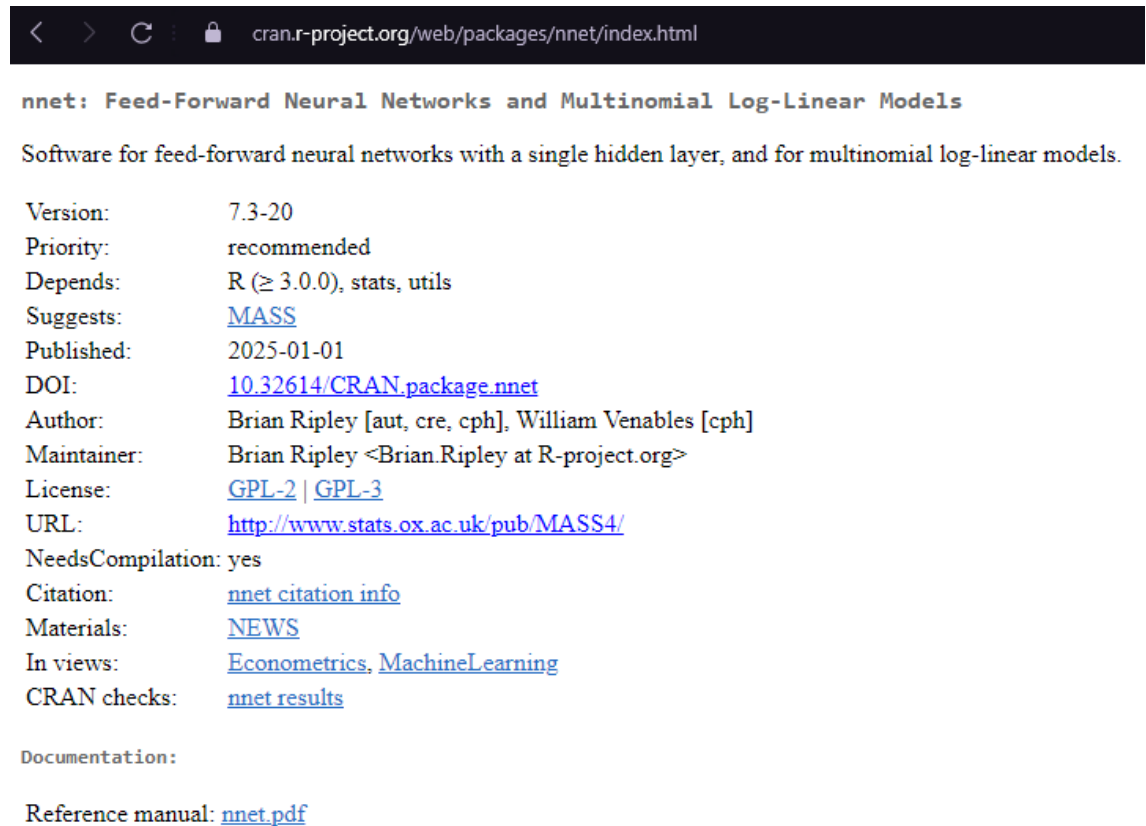


Figura 3. Página oficial de CRAN.

Fuente: Elaboración propia

Se accedió a la página oficial de CRAN donde se realizó la búsqueda del componente nnet. Una vez localizado, como se muestra en la figura 3, se procedió a una revisión manual de las versiones existentes.

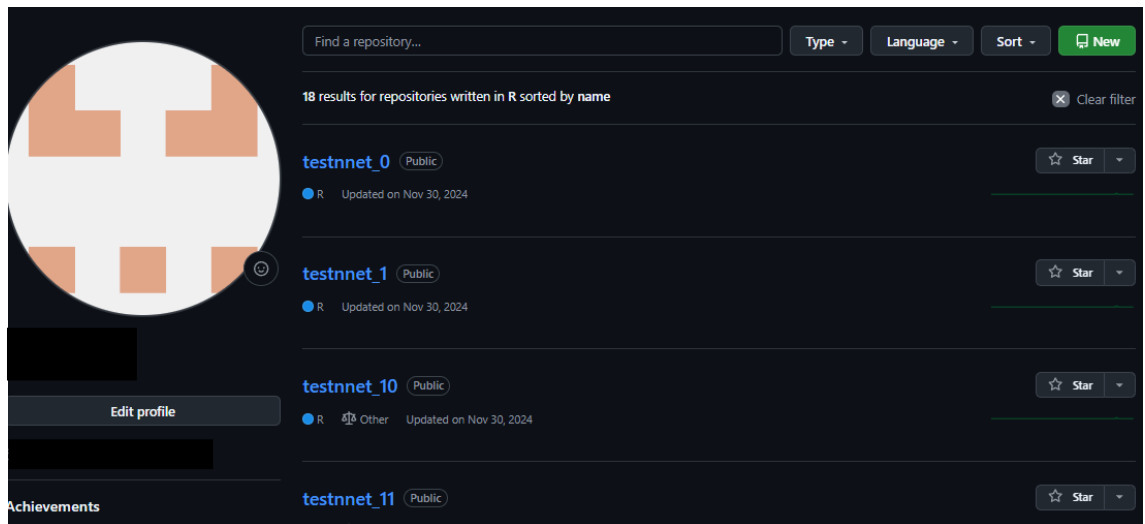


Figura 4. Repositorios de almacenamiento de las versiones nnet

Fuente: Elaboración propia.

Como se muestra en la figura 4, se crearon repositorios de almacenamiento para las versiones del componente nnet utilizando una cuenta personal de GitHub.

4. Revisión manual

Se inspeccionaron manualmente los repositorios para verificar que cumplan con los criterios establecidos.

Resultados de la búsqueda

Caso Keras

Como una biblioteca de alto nivel ampliamente utilizada en la comunidad de aprendizaje automático, la búsqueda de un repositorio adecuado en GitHub fue sencilla debido a su popularidad y soporte oficial.

- Se localizó el repositorio oficial alojado en GitHub, el cual es mantenido por el equipo de desarrollo de Keras y tiene una comunidad activa.
- El repositorio cuenta con una documentación extensa, guías de uso, y explicaciones de las funcionalidades principales.
- El código está bien organizado en carpetas y módulos, facilitando el análisis.

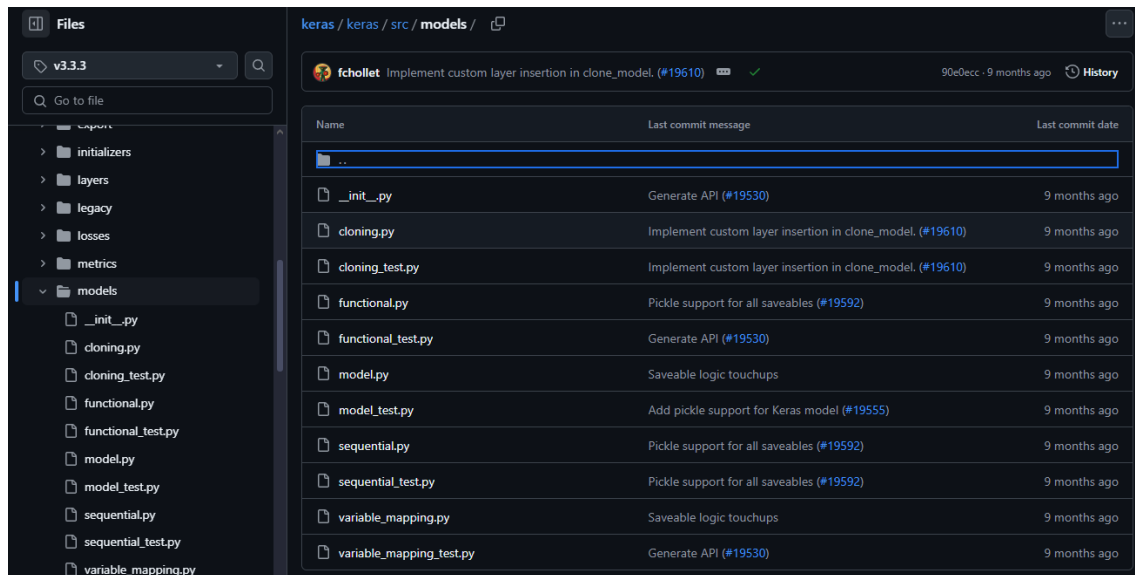


Figura 5. Repositorio oficial de Keras en GitHub

Fuente: Elaboración propia

Como se muestra en la figura 5, este repositorio cumple con los puntos previamente definidos. Su relevancia, disponibilidad, y la claridad del código lo convierten en un candidato para el análisis estático de calidad.

Caso nnet

El paquete nnet diseñado para redes neuronales feed-forward con una capa oculta y modelos log-lineales multinomiales, es ampliamente utilizado en entornos académicos y estadísticos. Sin embargo, la búsqueda de un repositorio adecuado en GitHub presentó desafíos.

- No se encontró un repositorio oficial o comunitario actualizado y bien estructurado para nnet en GitHub. La mayoría de las referencias están vinculadas a su distribución oficial a través de CRAN, donde se proporciona el código fuente y documentación básica.

Se tomó la decisión de descargar las versiones relevantes, crear repositorios propios para facilitar el análisis estático donde cada versión fue etiqueta correctamente para diferenciarla y permitir un análisis ordenado, ver figura 6. Además, los repositorios fueron organizados con las carpetas necesarias para contener el código fuente, la documentación y cualquier otro archivo adicional requerido para el análisis.

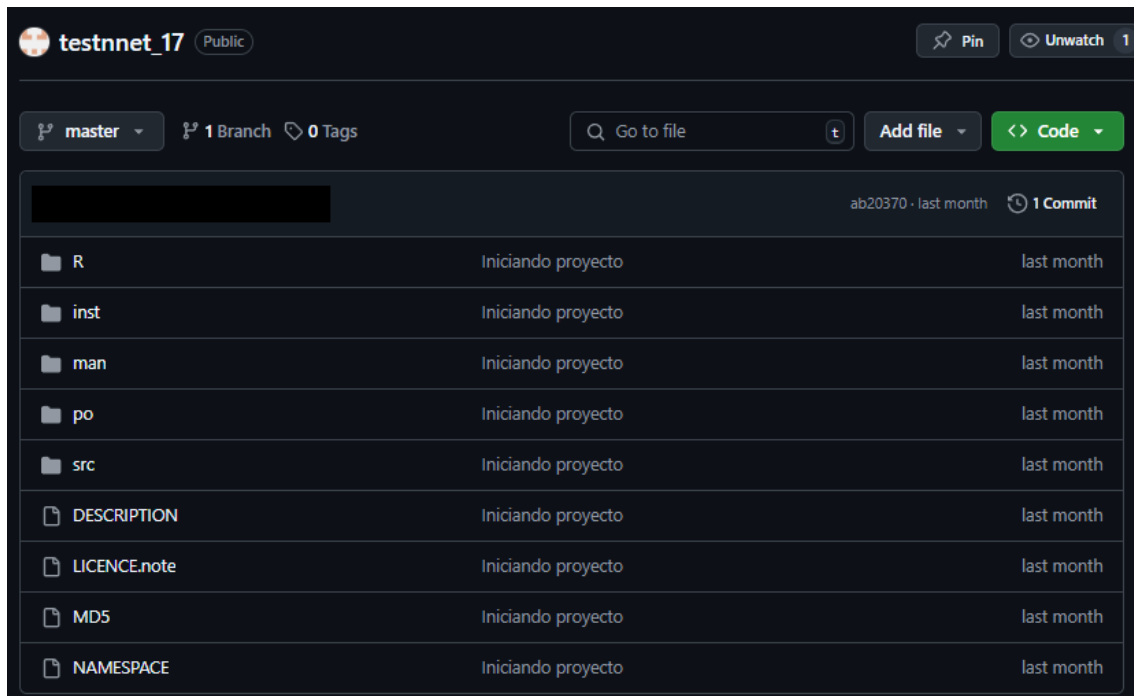


Figura 6 .Ejemplo: Repositorio de la versión 7.3.17 de nnet en GitHub

Fuente: Elaboración propia

En resumen, la búsqueda permitió identificar y preparar los componentes para el análisis, lo que se detallará más adelante en la selección de versiones específicas.

3.4 SELECCIÓN DE LOS COMPONENTES DE SOFTWARE DE USO LIBRE

A continuación, se detalla el proceso de selección de componentes de software de uso libre para el análisis estático de calidad. Los componentes seleccionados son Keras y nnet, herramientas representativas en el ámbito de aprendizaje automático y las redes neuronales. Según Talavera (2023), la selección se realizó con base en criterios técnicos, disponibilidad y relevancia para los objetivos de este proyecto.

Criterios de selección

- Relevancia técnica: Se seleccionaron versiones que representaran cambios significativos en funcionalidad o arquitectura.

- Disponibilidad: Las versiones seleccionadas debían ser accesibles en repositorios oficiales o en fuentes confiables.
- Compatibilidad: Las versiones elegidas deben ser compatibles con las herramientas de análisis estático y con los entornos utilizados en el proyecto.
- Documentación: Se priorizaron versiones con documentación suficiente para facilitar su análisis y comprensión.
- Mantenibilidad: Se consideraron versiones que mostraran diferencias significativas en la calidad de código, según los objetivos del proyecto.

Proceso de selección

- Keras

Al contar con un repositorio oficial en GitHub, permitió una selección ordenada de versiones, donde se revisaron las versiones disponibles en la lista del repositorio oficial. Donde se eligieron versiones que marcaran hitos en el desarrollo, como la transición a nuevas arquitecturas o la inclusión de funcionalidades relevantes.

- nnet

Dado que no se encontró un repositorio oficial en GitHub, el proceso de selección de versiones de nnet incluyó:

- Descargas desde CRAN: Se obtuvieron las versiones del sitio oficial de CRAN, considerando su historial de cambios, de las cuales presentaran cambios significativos, como mejoras en algoritmos o corrección de errores.

Componentes seleccionados

Keras

Tabla 2. Versiones del componente Keras seleccionadas

Versión	Fuente
1.0.0	Repositorio oficial de GitHub
2.0.0	Repositorio oficial de GitHub
3.0.0	Repositorio oficial de GitHub

3.4.0	Repositorio oficial de GitHub
3.5.0	Repositorio oficial de GitHub
3.6.0	Repositorio oficial de GitHub
3.7.0	Repositorio oficial de GitHub

Fuente: Elaboración propia

La tabla 2 muestra las versiones del componente Keras que han sido seleccionadas para el análisis estático de código. Todas estas versiones tienen origen del repositorio oficial de Keras en GitHub.

nnet

Tabla 3. Versiones del componente nnet seleccionadas

Versión	Fuente
7.3.0	CRAN
7.3.1	CRAN
7.3.5	CRAN
7.3.7	CRAN
7.3.8	CRAN
7.3.10	CRAN
7.3.12	CRAN
7.3.13	CRAN
7.3.15	CRAN
7.3.16	CRAN
7.3.18	CRAN
7.3.19	CRAN

Fuente: Elaboración propia

La tabla 3 muestra las versiones del componente nnet que han sido seleccionadas para el análisis estático de código. Todas estas versiones tienen origen de la página oficial de CRAN.

Justificación de la selección

La selección de estas versiones responde a la necesidad de analizar diferencias significativas en calidad de código y funcionalidad entre iteraciones clave. Esto permite realizar comparaciones y obtener resultados representativos en términos de mantenibilidad, seguridad y confiabilidad.

3.5 DISEÑO DEL PLAN DE PRUEBAS DE ANÁLISIS ESTÁTICO

Plan de pruebas

1. Introducción

Este documento detalla el plan de pruebas de análisis estático para los componentes de software Keras y nnet, utilizando SonarQube y Codacy como principales herramientas de análisis estático de código. Se busca garantizar la calidad del código en términos de mantenibilidad, seguridad y confiabilidad de la norma ISO/IEC 9126:2001, siguiendo las mejores prácticas de análisis estático.

1.1. Propósito

El propósito de este plan es establecer los pasos necesarios para evaluar los componentes Keras y nnet seleccionados mediante análisis estático, identificando defectos y áreas de mejora en el código fuente.

2. Alcance

El alcance de este plan se limita al análisis de calidad del código fuente de versiones seleccionadas de los componentes Keras y nnet que cumplen con criterios de selección de componentes anteriormente establecidos. Se excluye realizar un análisis dinámico que conlleve a pruebas de rendimiento o la evaluación de la funcionalidad de los componentes de software.

Tabla 4. Clasificación: Pruebas dentro y fuera de alcance

Dentro del alcance	Fuera de alcance
Detección de errores de sintaxis	Pruebas de integración continua
Análisis de vulnerabilidades de seguridad	Pruebas basadas en el comportamiento del sistema
Evaluación de mantenibilidad y calidad de código	Pruebas de carga y estrés
Revisión de estándares y convenciones de estilo	Evaluación de la experiencia de usuario o interfaz gráfica
Identificación de duplicación de código	
Análisis de modularidad y acoplamiento	

Fuente: Elaboración propia

La tabla 4 refleja claramente la categorización de las pruebas según su pertinencia, se detalla los tipos de prueba considerados dentro y fuera del alcance de un análisis estático.

2.1. Objetivos

- Evaluar la efectividad de las herramientas SonarQube y Codacy en el análisis de los componentes de software seleccionados y su capacidad para medir métricas clave de calidad.
- Evaluar la calidad del código con base en cualidades de software como mantenibilidad, seguridad y confiabilidad, según la norma ISO/IEC 9126:2001.
- Comparar las métricas clave de calidad de código entre las versiones seleccionadas de los componentes de software, destacando mejoras o deterioros identificados.
- Identificar vulnerabilidades críticas y áreas de mejora en el código fuente de los componentes seleccionados.

2.2. Definiciones, Acrónimos y Abreviaturas

Tabla 5. Glosario: Definiciones, Acrónimos y Abreviaturas relevantes

Análisis estático	Evaluación automatizada del código fuente sin ejecutarlo.
Keras	Biblioteca de alto nivel en Python para la creación y entrenamiento de modelos de aprendizaje profundo.
Nnet	Librería en R para la implementación de redes neuronales feed-forward y modelos log-lineales multinomiales.
Componente de software	Unidad funcional que forma parte de un sistema más grande, evaluada por su calidad, funcionalidad y mantenibilidad.
API (Application Programming Interface)	Conjunto de funciones y procedimientos que permiten la interacción entre componentes de software.
CI/CD (Continuos Integration/Continuos Deployment)	Práctica de desarrollo que implica integración y entrega continuas de software.
Duplicaciones de código	Fragmentos de código idénticos o muy similares presentes en un programa, que pueden ocultar la mantenibilidad y aumentar el riesgo de errores.
Code smells	Indicios de código que sugieren problemas potenciales en diseño o implementación, los cuales podrían dificultar la extensión, el mantenimiento o la comprensión del software.
Vulnerabilidades	Debilidades en el código que pueden ser explotadas por atacantes para comprometer la seguridad, como

	accesos no autorizados o interrupción de servicios.
Deuda técnica	Describe las consecuencias a largo plazo de decisiones rápidas o subóptimas en el desarrollo de software, como soluciones provisionales o compromisos de calidad, que pueden dificultar el mantenimiento, la escalabilidad y la extensibilidad del sistema si no se abordan.

Fuente: Elaboración propia

Como se muestra en la tabla 5, se desglosan definiciones, acrónimos y abreviaturas consideradas a lo largo del diseño del plan de pruebas para comprender el contenido sin ambigüedades.

3. Responsabilidades del proyecto

- Planificar, supervisar y coordinar todas las actividades relacionadas con el proyecto.
- Seleccionar las herramientas de análisis estático y definir criterios de calidad.
- Configurar las herramientas, ejecutar las pruebas y recopilar resultados.
- Realizar la configuración y análisis de las versiones de los componentes de software seleccionados.
- Redactar informes de resultados, actualizar el historial de cambios y generar la documentación final.

4. Cronograma

Tabla 6. Cronograma del Diseño del Plan de Pruebas

Fase	Descripción	Duración
<i>Fase1: Preparación</i>	Configuración de herramientas, definición	3 semanas

	de criterios y selección de componentes.	
<i>Fase 2: Ejecución</i>	Análisis de calidad en Keras y nnet utilizando SonarQube y Codacy.	4 semanas
• <i>Primera iteración de pruebas</i>	Análisis inicial de Keras y nnet. Revisión y categorización de hallazgos.	2 semanas
• <i>Segunda iteración de pruebas</i>	Aplicación de ajustes. Reanálisis de Keras y nnet. Consolidación de resultados.	2 semana
<i>Fase 3: Evaluación</i>	Clasificación de resultados, identificación de hallazgos y evaluación frente a métricas de calidad.	2 semanas
<i>Fase 4: Informe</i>	Elaboración de documentación y entrega del informe final.	3 semanas

Fuente: Elaboración propia

La tabla 6, que trata del cronograma del plan de pruebas engloba todas las fases necesarias, que en este caso consta de cuatro fases, así como su descripción y duración estimada de cada una.

Primera iteración de pruebas

Esta primera iteración de pruebas abarca los primeros análisis de las versiones Keras y nnet. Durante esta etapa, se evaluaron las versiones seleccionadas de

ambos componentes, identificando las métricas clave. Además, los hallazgos generados por las herramientas fueron revidados y categorizados.

Este proceso permitió obtener una visión preliminar del estado de calidad del código, sentando las bases para priorizar áreas de mejora y definir los enfoques de análisis más detallados en iteraciones posteriores.

Segunda iteración de pruebas

Esta segunda iteración se centró en la aplicación de ajustes derivados de la primera iteración. Se realizó una depuración de las versiones seleccionadas de elementos no relevantes con el funcionamiento de software.

El enfoque se dirigió a carpetas clave, como aquellas que contienen los modelos de aprendizaje y otros componentes esenciales. Posteriormente, se llevó a cabo un reanálisis, evaluando nuevamente las métricas de calidad escogidas. Finalmente, los resultados obtenidos fueron consolidados, permitiendo una comparación más precisa entre versiones y una evaluación más enfocada en los aspectos críticos del software.

5. Criterios de entrada y salida de pruebas

Para asegurar que las pruebas sean efectivas, eficientes y que se detengan solo cuando se cumplan las condiciones necesarias se debe cumplir lo siguiente.

Para iniciar las pruebas:

- Disponibilidad del código fuente.
- Configuración de herramientas.
- Requisitos del entorno.
- Definición de criterios de calidad.

Para dar por finalizada la fase de pruebas:

- Cobertura de pruebas completa.
- Informe de resultados generado.
- Revisión y validación de resultados.

6. Pruebas a los componentes

Tabla 7. Elementos clave en el análisis estático

Elemento	Descripción
<i>Configuración de herramientas</i>	SonarQube y Codacy se configurarán para analizar repositorios de Keras y nnet de acuerdo con las reglas y criterios definidos anteriormente.
<i>Integración con GitHub</i>	Los repositorios de Keras y nnet estarán integrados con SonarQube y Codacy para realizar análisis automáticos cada vez que se realicen cambios o actualizaciones en el código.
<i>Selección de métricas de calidad</i>	Se definirán las métricas de calidad a evaluar, como duplicación de código, vulnerabilidades, code smells, y complejidad cíclica.
<i>Frecuencia de pruebas</i>	Las pruebas se ejecutarán de manera periódica, dependiendo del flujo de trabajo del proyecto.
<i>Manejo de resultados</i>	Los resultados de las pruebas se generarán automáticamente y se almacenarán en un informe para su revisión.
<i>Generación de informes</i>	El informe de resultados será revisado y utilizado para identificar las áreas de mejora y obtener un diagnóstico de las anomalías encontradas.

Fuente: Elaboración propia

En la tabla 7, se detallan los principales elementos considerados durante las pruebas de análisis estático realizadas sobre los componentes de software Keras y nnet. Cada elemento está acompañado de una descripción que explica su relevancia dentro del proceso.

6.1. Técnicas de evaluación

Se utilizaron distintos enfoques para validar y medir los resultados obtenidos de las pruebas.

Tabla 8. Técnicas de evaluación aplicadas

<i>Pruebas de exploración</i>	Implica la revisión detallados de los resultados obtenidos, explorando los informes generados para identificar patrones y problemas emergentes en el código.
<i>Técnicas de comparación</i>	Comparar las métricas de calidad obtenidas en diferentes versiones del código. Esto ayuda a identificar si la calidad del código ha mejorado a lo largo del tiempo.
<i>Análisis de tendencias</i>	Observación de las tendencias de calidad en el código a lo largo de varias iteraciones de pruebas.

Fuente: Elaboración propia

Como se muestra en la tabla 8, se presentan las técnicas utilizadas para evaluar los resultados del análisis estático como la exploración detallada de los informes generados, la comparación de métricas y el análisis de tendencias. Esto permite identificar los patrones, mejorar la calidad y tomar decisiones informadas.

7. Participación en las pruebas

La participación en las pruebas será sumida íntegramente por el desarrollador, quien desempeñará los roles necesarios para garantizar la correcta ejecución del plan de pruebas.

Tabla 9. Roles y responsabilidades en el plan de pruebas

Rol/Responsabilidad	Descripción
<i>Planificador de pruebas</i>	Diseñar y estructurar el plan de pruebas, definir el cronograma, establecer los criterios de entrada y

	salida, y garantizar que las pruebas cumplan los objetivos establecidos.
<i>Analista de resultados</i>	Revisar los informes generados por las herramientas de análisis estático seleccionadas para analizar los componentes de software.
<i>Gestor de defectos</i>	Documentar, priorizar y dar seguimiento a los defectos encontrados durante las pruebas.
<i>Generador de informes</i>	Crear informes detallados sobre los resultados de las pruebas, los problemas detectados y las acciones tomadas.

Fuente: Elaboración propia

En la tabla 9, se detallan los roles involucrados en el proceso del plan de pruebas. Cada rol tiene sus funciones específicas para garantizar la eficacia y calidad del análisis estático.

8. Seguimiento de los casos de prueba

Mediante una documentación estructurada y sistemática queda registro de cada caso de prueba ejecutado, sus resultados y el estado final tras ejecución. Dado que las herramientas de análisis estático generan reportes automáticos, se tomará como base la información proporcionada por estas herramientas y se complementará con observaciones propias.

Cada caso de prueba será registrado en un formato como el siguiente:

Tabla 10. *Formato 1: Seguimiento para casos de prueba*

<i>ID Caso de prueba</i>	Descripción	Componente analizado	Herramienta utilizada	Resultado esperado	Resultado obtenido
CD-001	Detección de vulnerabilidades	Keras versión 3.0.0.	SonarQube	No se encontraron vulnerabilidades	Vulnerabilidades de nivel medio encontradas

Fuente: Elaboración propia

La tabla 10 muestra el formato utilizado para dar seguimiento a los casos de prueba hechos durante el análisis. Este formato está compuesto por un ID de

identificación, una descripción del caso de prueba, el componente analizado, la herramienta utilizada, y los resultados esperados y obtenidos.

Seguimiento

- Ejecución del caso de prueba: Configuración y ejecución del análisis en la herramienta correspondiente.
- Registro de resultados: Documentación de los hallazgos en la tabla de seguimiento.
- Validación: Confirmar los resultados obtenidos con los esperados.
- Actualización del estado: Marcar el estado del caso de prueba según corresponda.

8.1. Seguimiento de defectos

El seguimiento de defectos garantiza que cada defecto será registrado y clasificado. Se utilizará el siguiente formato.

Tabla 11. Formato 2: Seguimiento de defectos encontrados

ID del defecto	DF-001
Descripción del defecto	Vulnerabilidad de inyección SQL
Tipo	Vulnerabilidad
Gravedad	Alta
Componente afectado	Keras versión 3.3.0
Estado (abierto/cerrado)	Abierto
Fecha de detección	10/12/2024
Acción tomada	Pendiente de revisión

Fuente: Elaboración propia

La tabla 11 muestra un formato que ayuda a dar seguimiento a los defectos encontrados durante las pruebas. Se compone de un ID de identificación, la descripción del defecto encontrado, el tipo de defecto, su nivel de gravedad, el componente donde se encuentra y por lo cual afecta, el estado de seguimiento, la fecha en la cual se detectó y la acción tomada como resolución.

Aunque el seguimiento será manual, las herramientas de análisis estático también generarán reportes que facilitarán la identificación y registro inicial de defectos.

9. Ciclo de pruebas

El ciclo está diseñado para garantizar una evaluación sistemática y detallada de los componentes de software.

Fases del ciclo de pruebas

1. Configuración inicial:
 - Fecha de inicio: 21/10/2024
 - Actividades
 - Configurar las herramientas de análisis seleccionadas.
 - Preparar los repositorios de las versiones seleccionadas de Keras y nnet para análisis.
2. Primera iteración de pruebas
 - Fecha: 28/10/2024
 - Actividades:
 - Realizar el primer análisis de cada componente.
 - Documentar resultados y casos de prueba usando el Formato 1 de Seguimiento de casos de prueba, ver tabla 10.
 - Identificar defectos y dar seguimiento usando el Formato 2 Seguimiento de defectos encontrados, ver tabla 11.
3. Revisión de resultados
 - Fecha: 04/11/2024
 - Actividades:
 - Analizar los reportes generados por SonarQube y Codacy.
 - Validar los hallazgos y confirmar su relevancia.
4. Segunda iteración de pruebas
 - Fecha: 11/11/2024
 - Actividades:
 - Realizar un segundo análisis tras la implementación de ajuste o cambios.
 - Validar los resultados obtenidos de la primera iteración de pruebas.
5. Generación de reportes finales
 - Fecha: 25/11/2024

- Actividades:
 - Consolidar los resultados finales de ambas iteraciones.
 - Preparar los reportes detallados de los hallazgos de cada componente.

El ciclo de pruebas se llevará a cabo en los siguientes horarios:

- Lunes a viernes: 10:00 am – 2:00 pm

El ciclo de pruebas concluirá en la fecha 16/12/2024, tras la revisión y aprobación de los reportes finales.

10. Gestión de riesgos

Ante posibles riesgos que pueden afectar el proceso o los resultados, la gestión de riesgos anticipa, identifica y mitiga problemas potenciales.

Tabla 12. Análisis de riesgos y estrategias de mitigación

Riesgo	Impacto	Probabilidad	Estrategia de mitigación
<i>Configuración incorrecta de las herramientas</i>	Alto	Media	Validar las configuraciones antes de ejecutar el análisis.
<i>Limitaciones en el análisis automatizado</i>	Alto	Alta	Complementar con revisiones manuales si es necesario.
<i>Problemas de compatibilidad de versiones</i>	Medio	Baja	Asegurarse de que las versiones seleccionadas sean compatibles con las herramientas.
<i>Plazos ajustados</i>	Medio	Media	Priorizar actividades críticas y ajustar el cronograma si es necesario.

Fuente: Elaboración propia

Como se muestra en la tabla 12, se identifican diversos riesgos potenciales que podrían afectar el éxito del análisis estático de código, así como su impacto, probabilidad y las estrategias para mitigar cada uno de ellos. Este análisis permite anticipar problemas y definir acciones preventivas claras.

3.6 ANÁLISIS DE LOS COMPONENTES DE SOFTWARE

Los componentes de software seleccionados que son Keras y nnet representan ejemplos de software de libre significativos en el ámbito del aprendizaje automático y redes neuronales.

El propósito principal de este análisis es evaluar la calidad del código de estos componentes mediante el uso de herramientas de análisis estático, específicamente SonarQube y Codacy. Estas herramientas automatizan el proceso de detección de deudas técnicas, como vulnerabilidades, duplicación de código, entre otros.

Por un lado, Keras es una biblioteca de alto nivel diseñada para la construcción y entrenamiento de aprendizaje profundo, ampliamente utilizada. Por otro lado, nnet, una biblioteca más compacta y enfocada en redes neuronales feed-forward y modelos log-lineales,

Proceso de configuración

SonarQube

1. A través de una búsqueda en su página oficial, se descargó la versión comunitaria para utilizarla como herramienta de análisis.
2. La instalación y configuración de la herramienta se realizó utilizando la documentación proporcionada por la página oficial.
3. Siguiendo las indicaciones de la documentación, SonarQube fue instalado como un servicio local, como se muestra en la figura 7.

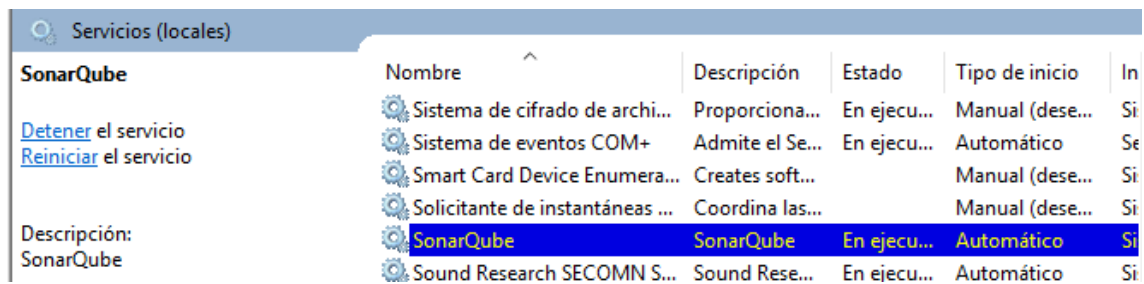


Figura 7. Implementación de SonarQube como un servicio

Fuente: Elaboración propia

- Realizada la instalación, se realizó la configuración de credenciales del usuario para poder acceder a la plataforma de SonarQube.
- Una vez configuradas las credenciales de acceso. Entramos a la página de inicio de SonarQube, ver figura 8.

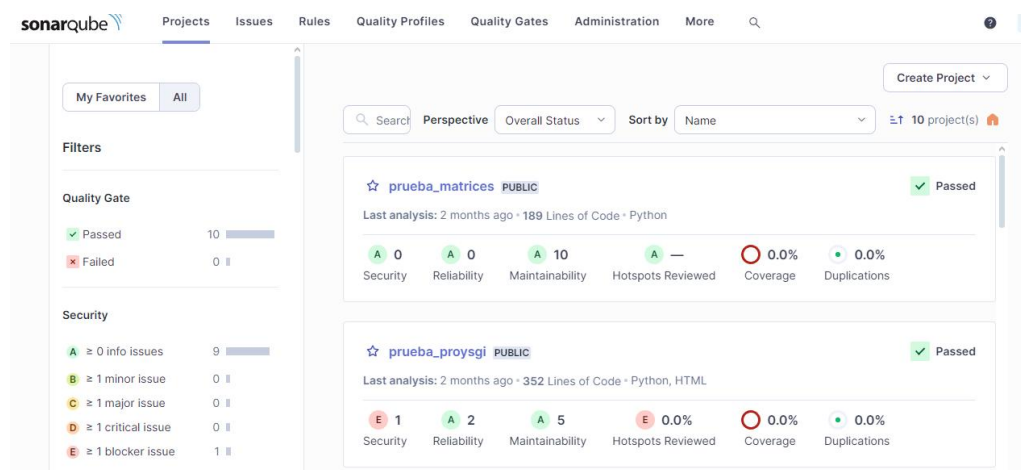


Figura 8. Home de SonarQube

Fuente: SonarQube (2024). SonarQube (Versión 25.1) [Software]. SonarSource, <https://www.sonarsource.com>

Codacy

- Ingresando a la página oficial de Codacy, donde iniciaremos sesión usando una cuneta GitHub personal que nos ayudará a integrar los

repositorios que contienen las versiones de los componentes Keras y nnet que hemos seleccionado anteriormente.

2. Ingresando las credenciales y pasando los métodos de seguridad de la plataforma GitHub ya podemos ingresar a la plataforma, ya que GitHub se integra fácilmente con Codacy, ver figura 9.

Organizations

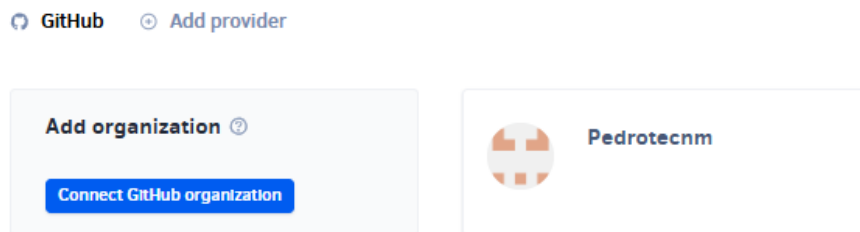


Figura 9. Integración de GitHub con Codacy

Fuente: Elaboración propia



Figura 10. Home de Codacy.

Fuente: Codacy (2024). Codacy (Versión 14.0) [Software]. Codacy, <https://www.codacy.com/>

Hecho los dos pasos anteriores podemos ingresar a Codacy, ver figura 10, y ahora podemos administrar los repositorios que almacenan las versiones de los componentes Keras y nnet.

Componentes analizados

Tabla 13. Versiones analizadas de Keras y nnet.

Componente	Versiones
<i>Keras</i>	1.0.0
	2.0.0
	3.0.0
	3.3.0
	3.4.0
	3.5.0
	3.6.0
	3.7.0
<i>nnet</i>	7.3.0
	7.3.1
	7.3.5
	7.3.7
	7.3.8
	7.3.10
	7.3.12
	7.3.13
	7.3.15
	7.3.16
	7.3.18
	7.3.19

Fuente: Elaboración propia

Como se muestra en la tabla 13, estas son las versiones de Keras y nnet que se analizarán más adelante.

Aspectos clave

El código limpio genera un software seguro, confiable y fácil de mantener. En lo que respecta a tres aspectos, como seguridad, confiabilidad y facilidad de mantenimiento, se denominan cualidades del software y contribuyen al valor a largo plazo de su software.

Los aspectos que se considerarán son los siguientes:

- Code smells
- Vulnerabilidades
- Bugs
- Duplicación de código
- Complejidad ciclomática
- Complejidad cognitiva
- Puntos de acceso de seguridad

También se toma en consideración los atributos de código limpio relacionados con las principales cualidades de evaluación de software, como mantenibilidad, seguridad y confiabilidad, como se muestra en la tabla 14.

Tabla 14. Atributos de código limpio

Atributos de código limpio

<i>Consistencia</i>	El código sigue estándares uniformes de estilo y estructura.
<i>Intencionalidad</i>	El código comunica claramente su propósito sin ambigüedad.
<i>Adaptabilidad</i>	Facilidad de modificar o extender el código sin comprometer su funcionalidad.
<i>Responsabilidad</i>	Cada unidad de código tiene una función específica y no mezcla responsabilidades.

Fuente: Elaboración propia

Severidad a nivel de calidad

Se consideran los niveles de severidad que las herramientas consideran en sus análisis.

Tabla 15. *Niveles de severidad*

Gravedad	Definición
<i>Bloqueador</i>	Error con alta probabilidad de impactar el comportamiento de la aplicación en producción.
<i>Alto</i>	Error de baja probabilidad de afecta el comportamiento de la aplicación en producción o de un problema que representa una falla de seguridad.
<i>Medio</i>	Defecto de calidad que puede afectar enormemente la productividad del desarrollador
<i>Bajo</i>	Defecto de calidad que puede afectar levemente la productividad del desarrollador.
<i>Información</i>	Ni un error ni un defecto de calidad, solo un hallazgo.

Fuente: *Elaboración propia*

Como se muestra en la tabla 15, estos son los niveles que las herramientas como SonarQube y Codacy utilizan para clasificar la gravedad de los problemas encontrados al realizar un análisis estático de código.

Preparación de los repositorios

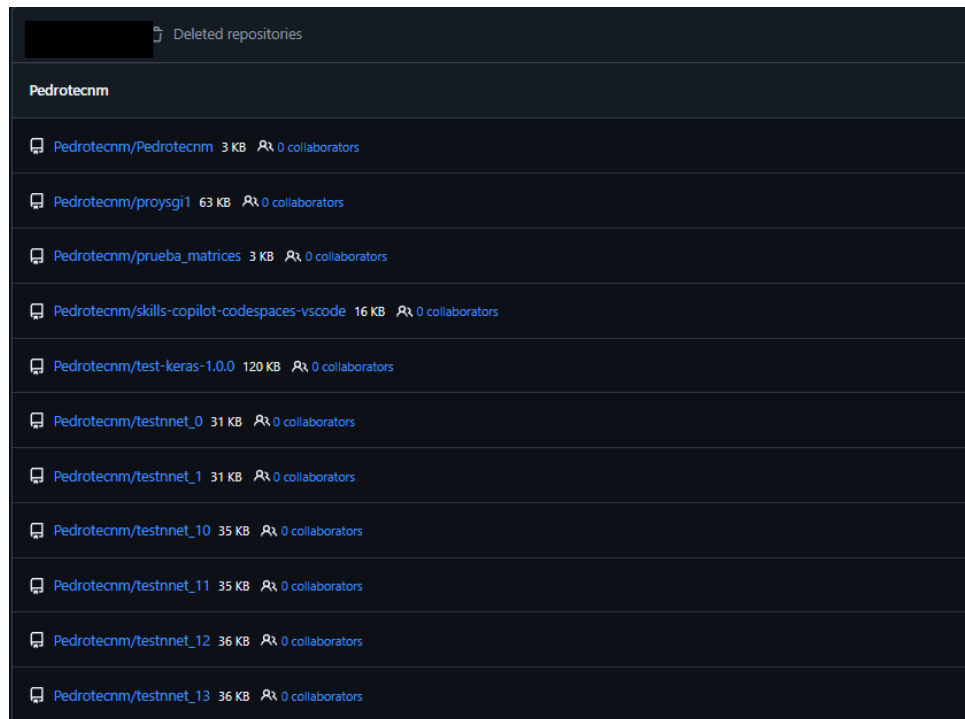


Figura 11. Repositorios de almacenamiento en GitHub

Fuente: Elaboración propia

Usando la plataforma GitHub se almaceno y gestiono las versiones seleccionadas de los componentes de software Keras y nnet, lo cual garantiza que estas versiones estén organizadas y disponibles en cualquier momento y desde cualquier lugar, ver figura 11.

Subir las versiones seleccionadas a los repositorios proporciona transparencia al proyecto. Además, la integración con las herramientas de análisis estático como SonarQube y Codacy facilitan la automatización de análisis y obtener reportes directamente desde la herramienta, simplificando el flujo de trabajo.

Análisis del componente Keras

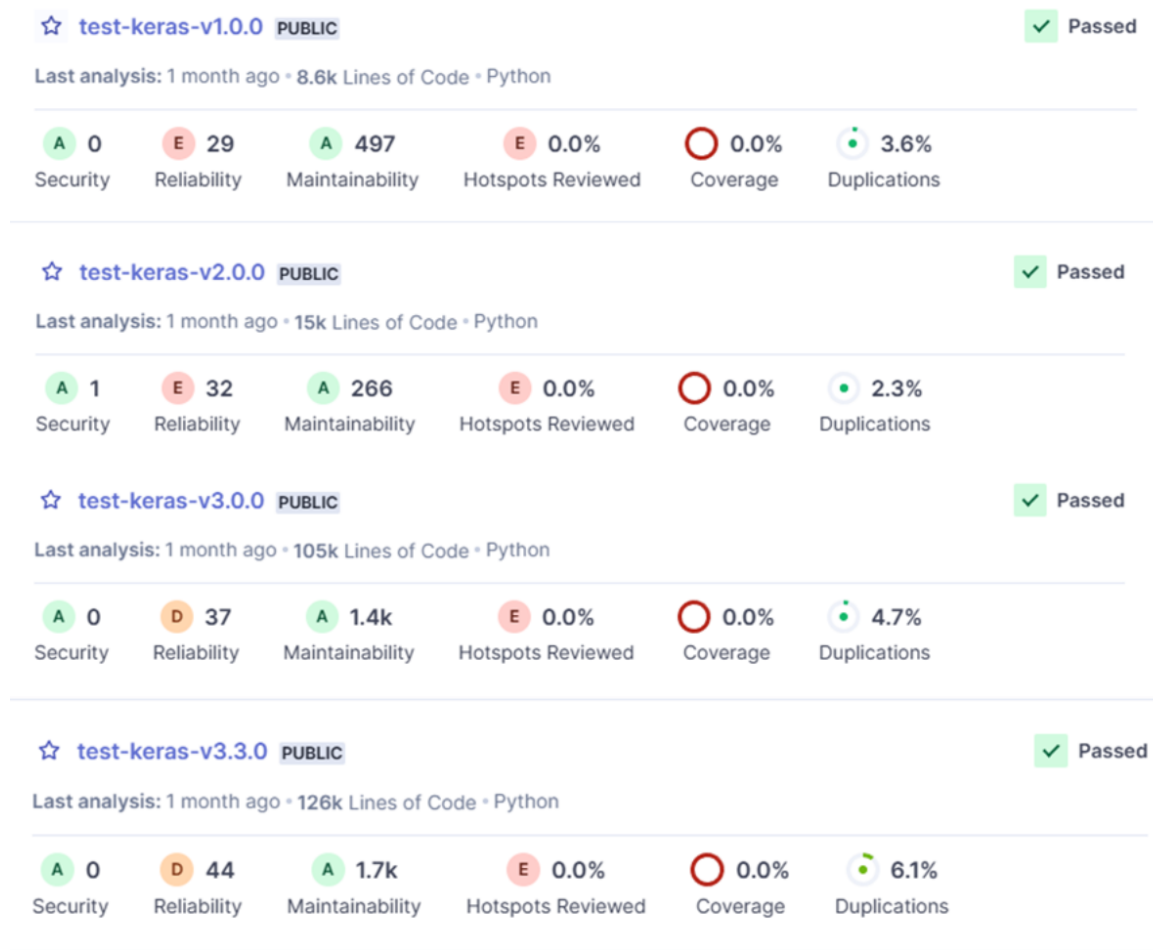


Figura 12, Resultados de Keras, parte 1

Fuente: Elaboración propia

La figura 12 muestra la primera parte de los resultados de los análisis a las versiones Keras. Esta primera parte abarca las primeras cuatro versiones analizadas, donde muestra sus resultados obtenidos en cuestión de las cualidades y métricas de calidad evaluadas.

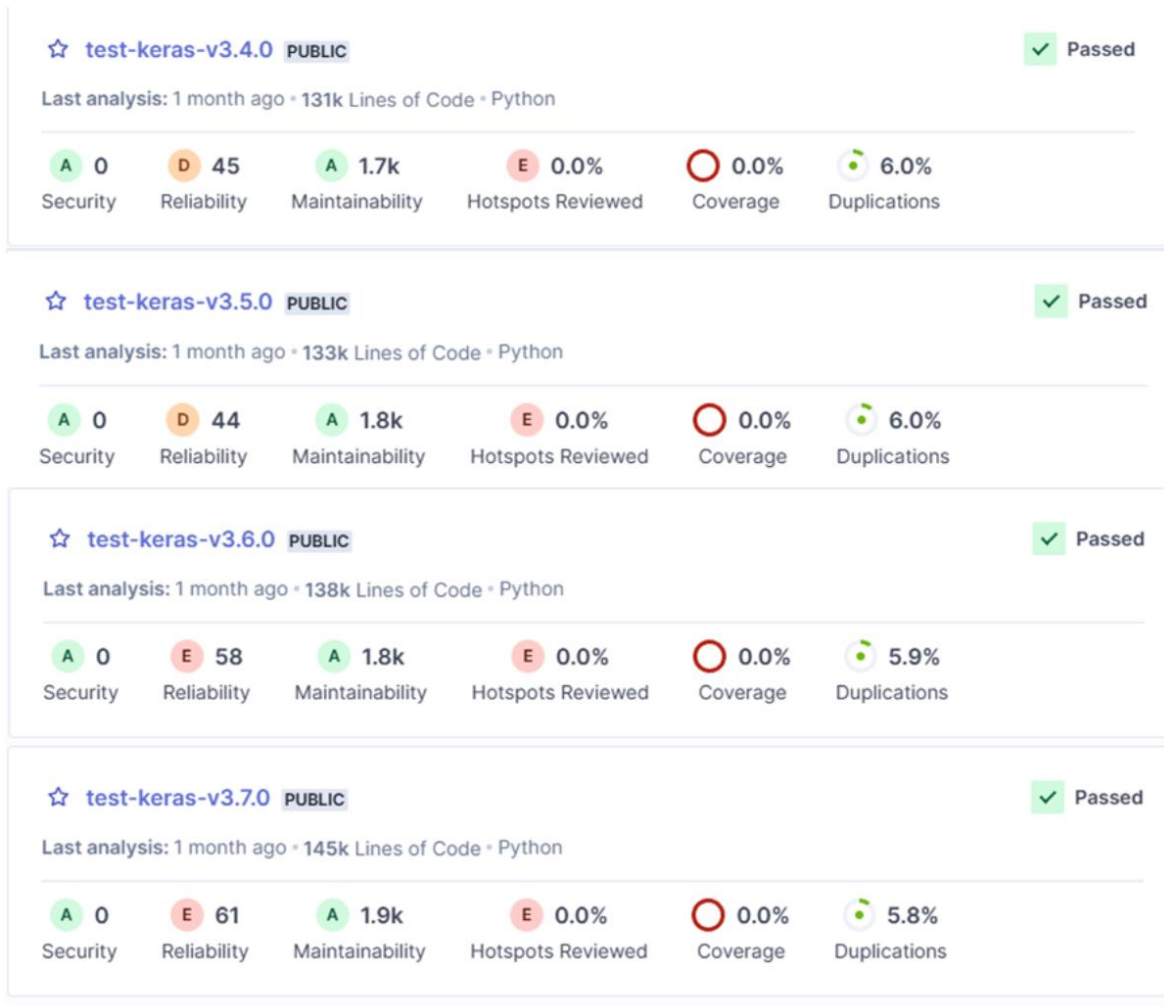


Figura 13. Resultados de Keras, parte 2

Fuente: Elaboración propia

La figura 13 muestra la segunda parte de los resultados, que complementan las versiones analizadas de Keras, y se muestra sus resultados obtenidos en cuestión de las cualidades y métricas de calidad evaluadas.

Análisis del componente nnet

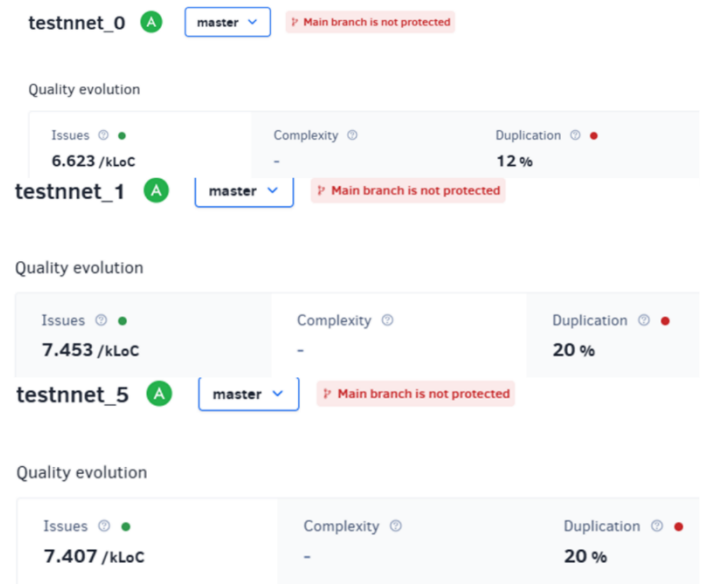


Figura 14. Resultados de nnet, parte 1

Fuente: Elaboración propia

Los primeros resultados de los análisis al componente nnet se muestran en la figura 14, que demuestra las primeras 3 versiones analizadas.

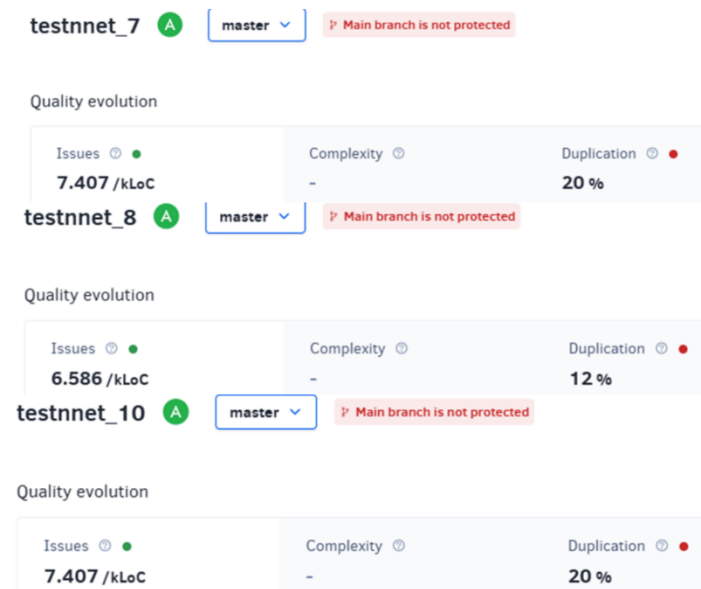


Figura 15. Resultados de nnet, parte 2

Fuente: Elaboración propia

Segundos resultados de los análisis al componente nnet siguiendo el orden de selección de las versiones de nnet, ver figura 15.

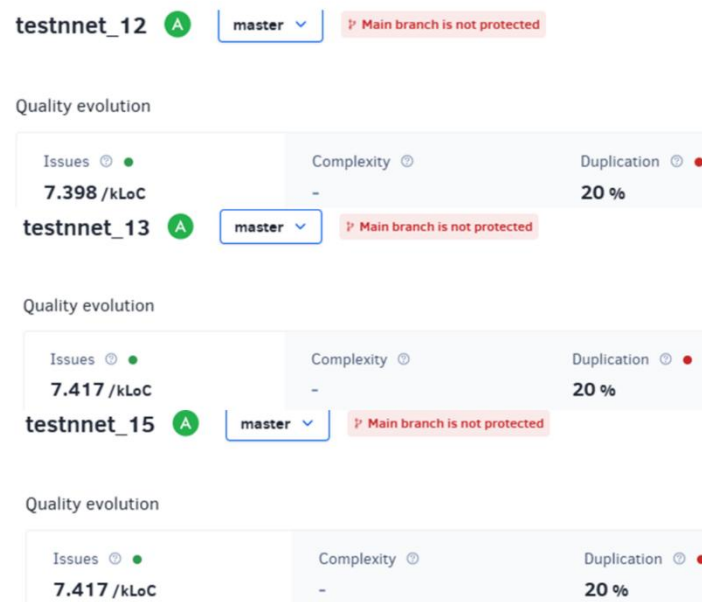


Figura 16. Resultados de nnet, parte 3

Fuente: Elaboración propia

Tercer resultado de los análisis al componente nnet siguiendo el orden de selección de las versiones de nnet, ver figura 16.

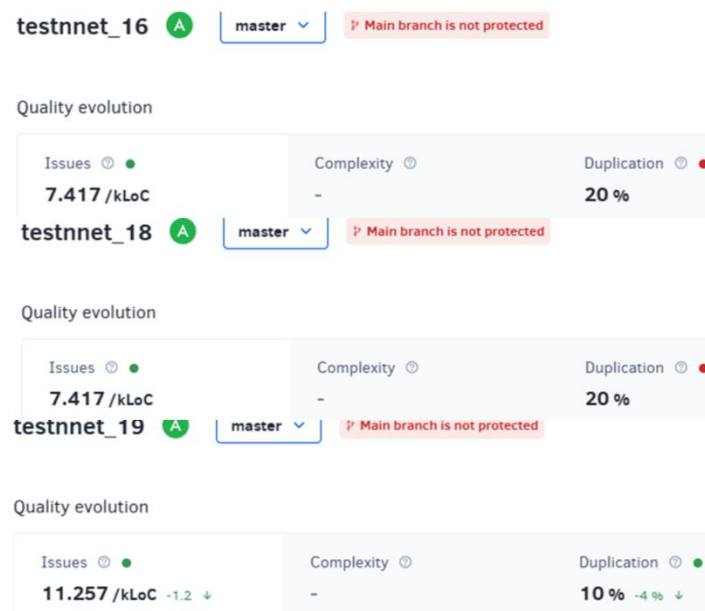


Figura 17. Resultados de nnet, parte 4

Fuente: Elaboración propia

Resultados finales de los análisis al componente nnet siguiendo el orden de selección de las versiones de nnet, ver figura 17.

Observaciones

- Cobertura de las herramientas
SonarQube y Codacy tienen un alcance limitado en algunos aspectos, como métricas avanzadas que no pueden ser capturadas.
- Características de los componentes
La comparación entre Keras y nnet es desigual debido a la diferencia en tamaño y complejidad de los proyectos.
- Interpretación
La interpretación de métricas depende del contexto y experiencia del desarrollador, lo que puede llevar a diferentes conclusiones según el enfoque del análisis.
- Limitaciones técnicas
El análisis se basa exclusivamente de reportes generados sin pruebas dinámicas. No se identifican problemas de rendimiento o comportamientos en tiempo de ejecución.

4. RESULTADOS

4.1 DIAGNÓSTICO DE CALIDAD DE LOS COMPONENTES SELECCIONADOS

El diagnostico de calidad de los componentes seleccionados tiene como objetivo evaluar y comparar las métricas clave obtenidas mediante herramientas de análisis estático. Este análisis se centra en identificar áreas críticas que puedan afectar la calidad del software. Se presentan los resultados obtenidos.

Resultados Keras

Tabla 16. Resultados de las versiones Keras analizadas.

DESCRIPCIÓN GENERAL						
Versión	Seguridad	Confiabilidad	Mantenibilidad	Duplicaciones (%)	Puntos de acceso de seguridad	Líneas de código (K)
1.0.0	A	E	A	3.6	E	8.6
2.0.0	A	E	A	2.3	E	15
3.0.0	A	D	A	4.7	E	105
3.3.0	A	D	A	6.1	E	126
3.4.0	A	D	A	6	E	131
3.5.0	A	D	A	6	E	133
3.6.0	A	E	A	5.9	E	138
3.7.0	A	E	A	5.8	E	145

Fuente: Elaboración propia a partir de datos obtenidos de SonarQube

Como se muestra en la tabla 16, los resultados muestran tres criterios de calidad de software evaluados considerando la norma ISO/IEC 9126:2001, así como aspectos clave que influyen en la calidad de software, como el porcentaje de duplicación de código y los puntos de acceso de seguridad, y finalmente la cantidad de líneas de código de cada versión.

Las calificaciones que asigna SonarQube son una forma de evaluar y visualizar la calidad del software en diferentes áreas clave. Estas calificaciones se expresan mediante letras de la A hasta E, donde:

- A y B: Representan el mejor desempeño o calidad, pero es recomendable monitorear y seguir buenas prácticas.
- C: Representa un nivel intermedio, el código es funcional, pero hay áreas para mejorar.
- D y E: Indican el peor desempeño, mostrando una calidad deficiente que requiere atención inmediata para evitar problemas mayores.

Mantenibilidad de las versiones Keras

La tabla 17 muestra los resultados en cuestión de mantenibilidad a lo largo de las versiones Keras. En ella se muestra la cantidad de problemas encontrados, así como su distribución a través de los atributos de código limpio considerados por la herramienta SonarQube.

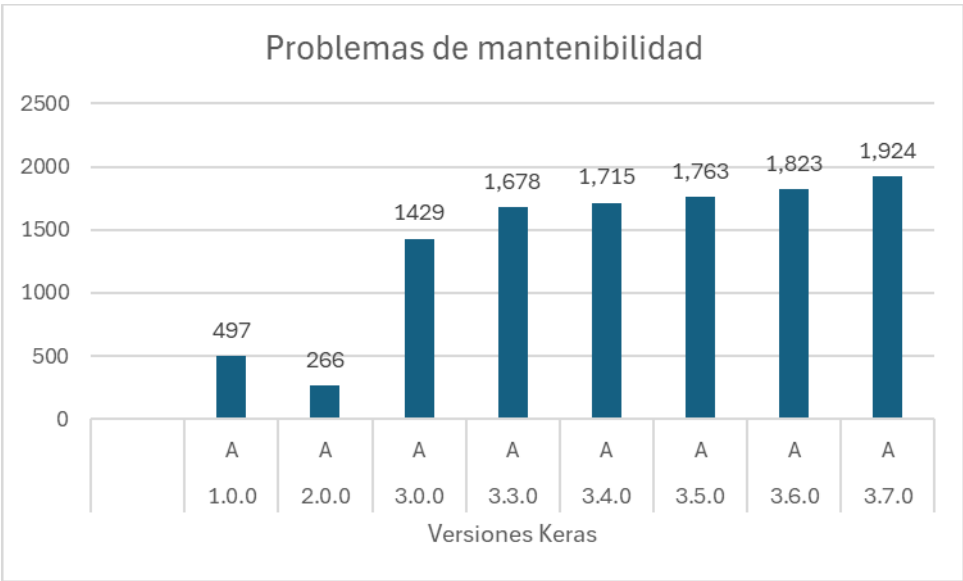
Las versiones analizadas muestran calificaciones de grado A, lo que indica que existe al menos un 5% de deuda técnica. Esto se debe a que se evalúa el

esfuerzo necesario para corregir los problemas de código técnico, es decir, la deuda técnica.

Tabla 17. *Mantenibilidad de las versiones Keras*

versión	Mantenibilidad	Problemas totales	Atributos de código limpio			
			Consistencia	Intencionalidad	Adaptabilidad	Responsabilidad
1.0.0	A	497	118	303	76	0
2.0.0	A	266	73	86	107	0
3.0.0	A	1429	710	423	296	0
3.3.0	A	1,678	866	464	348	0
3.4.0	A	1,715	877	470	368	0
3.5.0	A	1,763	905	480	378	0
3.6.0	A	1,823	936	496	391	0
3.7.0	A	1,924	1,009	511	404	0

Fuente: *Elaboración propia a partir de datos obtenidos de SonarQube*



Gráfica 1. *Incremento de problemas relacionados con la mantenibilidad en Keras*

Fuente: *Elaboración propia*

La grafica 1 muestra el incremento de deuda técnica en cuestión de mantenibilidad a lo largo de las versiones Keras.

Tabla 18. Problemas de mantenibilidad según su nivel de gravedad

versión	Problemas totales	Nivel de gravedad		
		Bajo	Medio	Alto
1.0.0	497	103	308	86
2.0.0	266	19	138	109
3.0.0	1429	235	938	256
3.3.0	1,678	244	1,131	303
3.4.0	1,715	247	1.148	320
3.5.0	1,763	248	1,183	332
3.6.0	1,823	258	1,219	346
3.7.0	1,924	258	1,303	363

Fuente: Elaboración propia a partir de datos obtenidos de SonarQube

Continuando con la mantenibilidad de las versiones Keras, la tabla 18 muestra como los problemas encontrados son distribuidos según su nivel de gravedad asignado. Donde SonarQube maneja problemas de nivel bajo, medio y alto, siendo los problemas de nivel medio con mayor cantidad y crecimiento por versión de Keras.

Tabla 19. Problemas de mantenibilidad según su tipo

versión	Problemas totales	Tipo de problema		
		Bug	Vulnerabilidad	Code Smell
1.0.0	497	0	0	497
2.0.0	266	0	0	266
3.0.0	1429	0	0	1429
3.3.0	1,678	0	0	1,678
3.4.0	1,715	0	0	1.715
3.5.0	1,763	0	0	1,763
3.6.0	1,823	0	0	1,823
3.7.0	1,924	0	0	1,924

Fuente: Elaboración propia a partir de datos obtenidos de SonarQube

Los problemas totales de cada versión son clasificados de acuerdo con el tipo de problema asignado por SonarQube, como se muestra en la tabla 19, siendo los Code smells el tipo de problema dominante.

Tabla 20. Tiempo de esfuerzo para los problemas de mantenibilidad

versión	Problemas totales	Tiempo de esfuerzo
1.0.0	497	12d
2.0.0	266	6d 1h
3.0.0	1429	21d
3.3.0	1,678	26d
3.4.0	1,715	26d
3.5.0	1,763	27d
3.6.0	1,823	28d
3.7.0	1,924	29d

Fuente: Elaboración propia a partir de datos obtenidos de SonarQube

Finalmente, dentro de la cualidad de mantenibilidad, la herramienta calculo el tiempo de esfuerzo estimado para resolver los problemas encontrados por versión, ver tabla 20.

Confiabilidad de las versiones Keras

La tabla 21 muestra los resultados en cuestión de confiabilidad a lo largo de las versiones Keras. En ella se muestra la cantidad de problemas encontrados, así como su distribución a través de los atributos de código limpio considerados por la herramienta SonarQube.

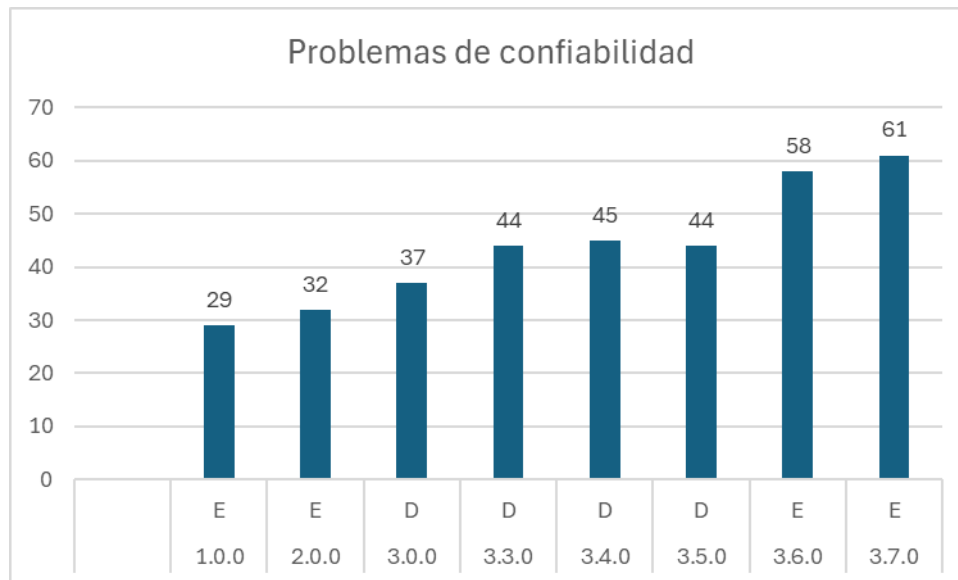
Las versiones analizadas muestran calificaciones de grado E y D lo que indica que existen fallas significativas y errores críticos que pueden comprometer completamente el sistema.

Tabla 21. Confiabilidad de las versiones Keras

versión	Confiabilidad	Problemas totales	Atributos de código limpio			
			Consistencia	Intencionalidad	Adaptabilidad	Responsabilidad
1.0.0	E	29	0	29	0	0
2.0.0	E	32	1	31	0	0
3.0.0	D	37	5	30	2	0
3.3.0	D	44	6	36	2	0
3.4.0	D	45	6	37	2	0
3.5.0	D	44	7	35	2	0
3.6.0	E	58	7	49	2	0
3.7.0	E	61	8	51	2	0

Fuente: Elaboración propia a partir de datos obtenidos de SonarQube

La gráfica 2 muestra los problemas encontrados a lo largo de las versiones Keras, y su incremento.



Gráfica 2. Problemas de confiabilidad

Fuente: Elaboración propia

Pasando ahora a la tabla 22, los problemas encontrados de confiabilidad son distribuidos según el nivel de gravedad asignado por SonarQube, demostrando pequeños cambios en las últimas versiones de Keras.

Tabla 22. Problemas de confiabilidad según su nivel de gravedad

versión	Problemas totales	Nivel de gravedad		
		Bajo	Medio	Alto
1.0.0	29	0	0	29
2.0.0	32	0	3	29
3.0.0	37	4	19	14
3.3.0	44	10	19	15
3.4.0	45	10	19	16
3.5.0	44	10	18	16
3.6.0	58	10	28	20
3.7.0	61	13	28	20

Fuente: Elaboración propia a partir de datos obtenidos de SonarQube

En la tabla 23 se muestra que la cantidad de problemas encontrados corresponden al tipo Bug y Code smells, y que no existe alguna vulnerabilidad en cuestión de confiabilidad de los componentes.

Tabla 23. Problemas de confiabilidad según su tipo

versión	Problemas totales	Tipo de problema		
		Bug	Vulnerabilidad	Code Smell
1.0.0	29	29	0	9
2.0.0	32	22	0	10
3.0.0	37	21	0	16
3.3.0	44	22	0	22
3.4.0	45	22	0	23
3.5.0	44	21	0	23
3.6.0	58	33	0	25
3.7.0	61	33	0	28

Fuente: Elaboración propia a partir de datos obtenidos de SonarQube

La tabla 24 muestra cuanto es el tiempo de esfuerzo estimado resolver los problemas de confiabilidad para cada versión de Keras.

Tabla 24. Tiempo de esfuerzo para los problemas de confiabilidad

versión	Problemas totales	Tiempo de esfuerzo
1.0.0	29	4h 5min
2.0.0	32	4h 20min
3.0.0	37	2h 52min
3.3.0	44	3h 12min
3.4.0	45	3h 17min
3.5.0	44	3h 12min
3.6.0	58	4h 32min
3.7.0	61	4h 41min

Fuente: Elaboración propia a partir de datos obtenidos de SonarQube

Seguridad en las versiones Keras

La tabla 25 muestra los resultados en cuestión de seguridad a lo largo de las versiones Keras. En ella se muestra la cantidad de problemas encontrados, así como su distribución a través de los atributos de código limpio considerados por la herramienta SonarQube.

Las versiones analizadas muestran calificaciones de grado A, que se calcula en función del tiempo requerido para solucionar todas las vulnerabilidades

detectadas. lo que indica que el tiempo estimado es menor a 1 día para solucionar los problemas.

Tabla 25. Seguridad en las versiones Keras

versión	Seguridad	Problemas totales	Atributos de código limpio			
			Consistencia	Intencionalidad	Adaptabilidad	Responsabilidad
1.0.0	A	0	0	0	0	0
2.0.0	A	1	1	0	0	0
3.0.0	A	0	0	0	0	0
3.3.0	A	0	0	0	0	0
3.4.0	A	0	0	0	0	0
3.5.0	A	0	0	0	0	0
3.6.0	A	0	0	0	0	0
3.7.0	A	0	0	0	0	0

Fuente: Elaboración propia a partir de datos obtenidos de SonarQube

Como se ve en la tabla 25 y se confirma en la tabla 26, el único problema detectado en cuestión de seguridad fue resuelto por los desarrolladores de Keras.

Tabla 26. Problemas de seguridad según su nivel de gravedad

versión	Problemas totales	Nivel de gravedad		
		Bajo	Medio	Alto
1.0.0	0	0	0	0
2.0.0	1	1	0	0
3.0.0	0	0	0	0
3.3.0	0	0	0	0
3.4.0	0	0	0	0
3.5.0	0	0	0	0
3.6.0	0	0	0	0
3.7.0	0	0	0	0

Fuente: Elaboración propia a partir de datos obtenidos de SonarQube

La tabla 27 muestra que el único problema era de tipo Code smells, se detectó en la versión 2.0.0, le dieron solución y para versiones posteriores no se volvió a detectar algún otro problema en cuestión de seguridad.

Tabla 27. Problemas de seguridad por tipo

versión	Problemas totales	Tipo de problema		
		Bug	Vulnerabilidad	Code Smell
1.0.0	0	0	0	0
2.0.0	1	0	0	1
3.0.0	0	0	0	0
3.3.0	0	0	0	0
3.4.0	0	0	0	0
3.5.0	0	0	0	0
3.6.0	0	0	0	0
3.7.0	0	0	0	0

Fuente: Elaboración propia a partir de datos obtenidos de SonarQube

El único problema detectado fue de tipo Code Smell en la versión 2.0.0 y para posteriores versiones fue resuelto.

Tabla 28. Tiempo de esfuerzo de problemas de seguridad

versión	Problemas totales	Tiempo de esfuerzo
1.0.0	0	0
2.0.0	1	5m
3.0.0	0	0
3.3.0	0	0
3.4.0	0	0
3.5.0	0	0
3.6.0	0	0
3.7.0	0	0

Fuente: Elaboración propia a partir de datos obtenidos de SonarQube

Como se muestra en la tabla 28, el problema tuvo un tiempo de esfuerzo de 5 minutos.

Pasando a otro aspecto clave de la calidad de software, tenemos todos los asuntos encontrados relacionados con los puntos de acceso de seguridad, que se describen a continuación.

Tabla 29. Asuntos de puntos de acceso de seguridad

Puntos de acceso de seguridad					
Versión	Asuntos totales	Prioridad de revisión			Tipos de problemas
		Bajo	Medio	Alto	
1.0.0	11	5	6	0	Criptografía débil, Cifrado de datos confidenciales
2.0.0	12	6	6	0	Criptografía débil, Cifrado de datos confidenciales
3.0.0	35	20	15	0	Criptografía débil, Denegación de Servicio (DoS)
3.3.0	35	20	15	0	Criptografía débil, Denegación de Servicio (DoS)
3.4.0	35	20	15	0	Criptografía débil, Denegación de Servicio (DoS)
3.5.0	36	20	16	0	Criptografía débil, Denegación de Servicio (DoS), Permisos
3.6.0	36	20	16	0	Criptografía débil, Denegación de Servicio (DoS), Permisos
3.7.0	36	20	16	0	Criptografía débil, Denegación de Servicio (DoS), Permisos

Fuente: Elaboración propia a partir de datos obtenidos de SonarQube

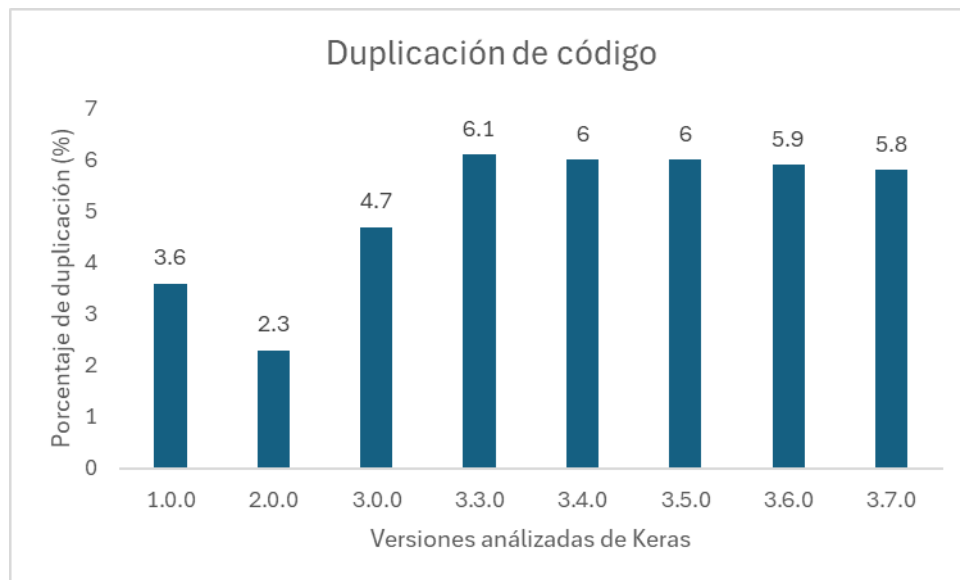
Como se muestra en la tabla 29, se presenta la cantidad de asuntos, su nivel de prioridad y los tipos de problemas encontrados en cada versión analizada de Keras.

Tabla 30. Duplicaciones de código en Keras

Duplicaciones				
Versión	Densidad (%)	Líneas duplicadas	Bloques duplicados	Archivos duplicados
1.0.0	3.6	526	50	8
2.0.0	2.3	653	43	11
3.0.0	4.7	7,387	356	116
3.3.0	6.1	11,189	474	161
3.4.0	6	11,445	515	160
3.5.0	6	11,611	525	163
3.6.0	5.9	11,765	533	166
3.7.0	5.8	12,183	572	169

Fuente: Elaboración propia a partir de datos obtenidos de SonarQube

Otro aspecto que considerar son las duplicaciones encontradas. Se consideró ya que representa un riesgo de errores y dificulta el mantenimiento. En la tabla 30 se muestra el nivel de densidad encontrado en cada versión analizada de Keras, así como la forma en que se distribuye, como la cantidad de líneas de código, bloques de código y archivos duplicados.



Gráfica 3. Duplicación de código en Keras

Fuente: Elaboración propia

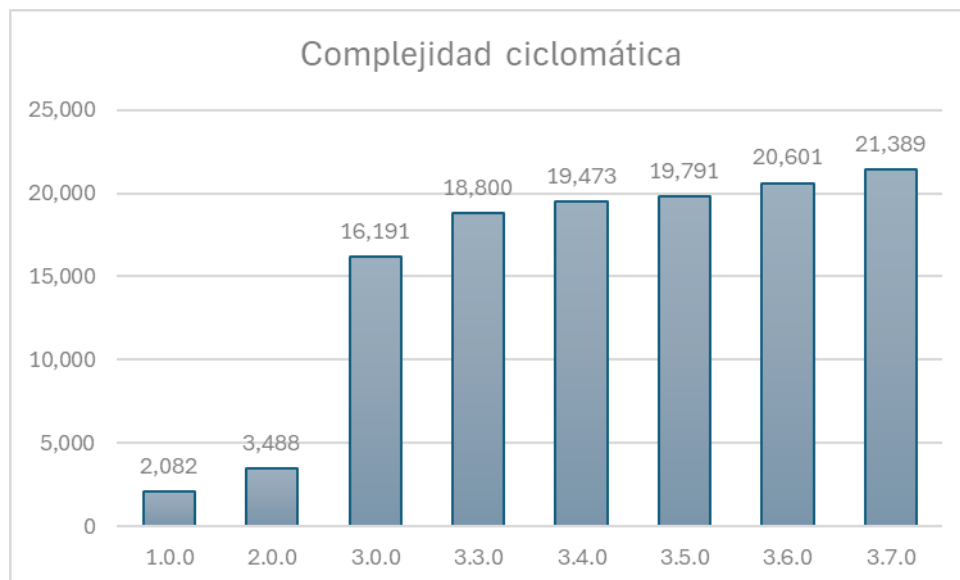
La gráfica 3 muestra el porcentaje encontrado de duplicación de Keras a lo largo de sus versiones.

Tabla 31. Complejidad de Keras

Complejidad		
Version	Complejidad ciclomática	Complejidad cognitiva
1.0.0	2,082	2,519
2.0.0	3,488	4,553
3.0.0	16,191	13,626
3.3.0	18,800	15,858
3.4.0	19,473	16,274
3.5.0	19,791	16,457
3.6.0	20,601	17,166
3.7.0	21,389	17,833

Fuente: Elaboración propia a partir de datos obtenidos de SonarQube

Se consideran aspectos como la complejidad ciclomática y cognitiva ya que son indicadores del nivel de complejidad que enfrenta el desarrollador para poder entender el código, así como representan cuan difícil será mantener o modificar el código fuente sin introducir errores, ver tabla 31.



Gráfica 4. Incremento de la complejidad ciclomática de Keras

Fuente: Elaboración propia

La gráfica 4 muestra el aumento de la complejidad ciclomática de Keras a lo largo de sus versiones.

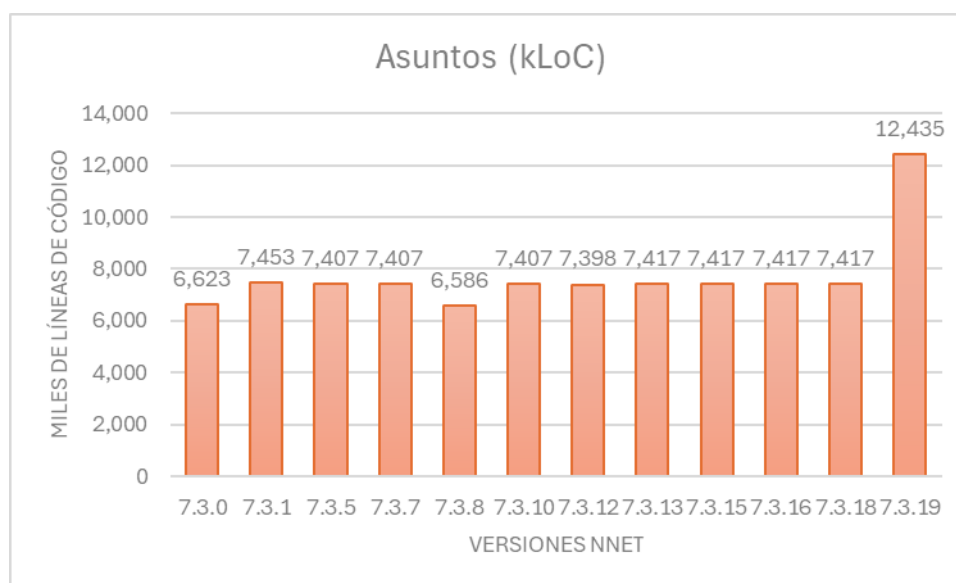
Resultados nnet

Tabla 32. Problemas de nnet

Resultados de nnet					
Version	Asuntos (kLoC)	Problemas totales	Complejidad	Duplicación (%)	Calificación
7.3.0	6,623	6	0	12	A
7.3.1	7,453	6	0	20	A
7.3.5	7,407	6	0	20	A
7.3.7	7,407	6	0	20	A
7.3.8	6,586	6	0	12	A
7.3.10	7,407	6	0	20	A
7.3.12	7,398	6	0	20	A
7.3.13	7,417	6	0	20	A
7.3.15	7,417	6	0	20	A
7.3.16	7,417	6	0	20	A
7.3.18	7,417	6	0	20	A
7.3.19	12,435	12	0	14	A

Fuente: Elaboración propia a partir de datos obtenidos de Codacy

La tabla 32 se muestran los resultados obtenidos de las versiones analizadas de nnet, donde se consideran los asuntos normalizados por cada kilo de líneas de código (kLoC), los problemas totales, la complejidad, el nivel de porcentaje de duplicación de código. Finalmente, la calificación obtenida en cuestión de calidad de software resulta aprobatoria.



Gráfica 5. Asuntos de nnet

Fuente: Elaboración propia

La gráfica 5 muestra los asuntos por cada mil líneas de código de los componentes nnet analizados.

La tabla 33 muestra que en las versiones de nnet existen pequeñas diferencias y cambios significativos relacionados con los asuntos de calidad. Desde las primeras versiones se muestra que los problemas son repetitivos a lo largo de nnet, siendo estos de gravedad menor, por lo que, los desarrolladores de nnet no le dieran importancia.

Tabla 33. Asuntos encontrados en nnet

Asuntos encontrados			
Versión	Idioma	Gravedad	Categoría
7.3.0	C	Menor	Code Style
7.3.1	C	Menor	Code Style
7.3.5	C	Menor	Code Style
7.3.7	C	Menor	Code Style
7.3.8	C	Menor	Code Style
7.3.10	C	Menor	Code Style
7.3.12	C	Menor	Code Style
7.3.13	C	Menor	Code Style
7.3.15	C	Menor	Code Style
7.3.16	C	Menor	Code Style
7.3.18	C	Menor	Code Style
7.3.19	C, CSS	Menor	Code Style

Fuente: Elaboración propia a partir de datos obtenidos de Codacy

Más a detalle respecto a los asuntos encontrados en nnet, no presentan grandes problemas, más bien, los problemas que existen desde las primeras versiones de nnet persisten hasta la más reciente donde se han ingresado más errores. Dichos errores no representan un problema significativo al ser considerados de gravedad menor.

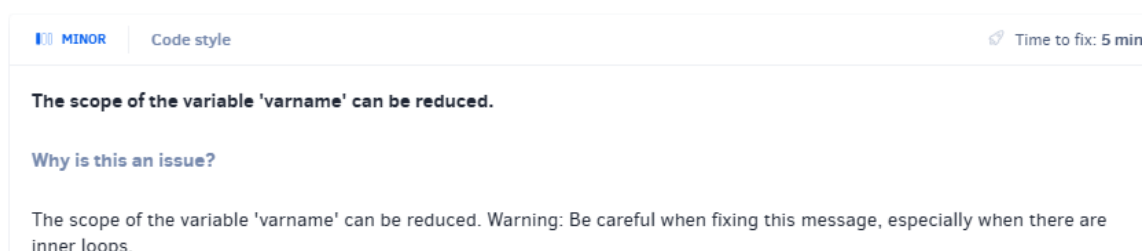
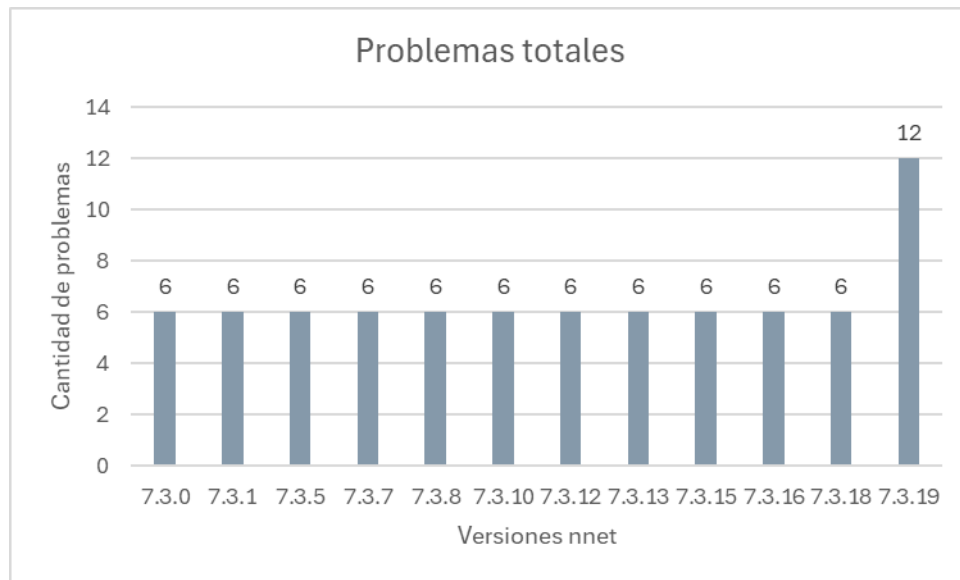


Figura 18. Causa dominante de los asuntos de nnet

Fuente: Elaboración propia

Los errores persistentes se centran en variables que tienen un alcance más de lo esperado dentro de funciones del código fuente. Se recomienda reducir el alcance de las variables para evitar afectar la calidad y la capacidad de mantenimiento del código. Ya que, al tener un alcance mayor de lo necesario, puede generar confusión, posibles conflictos y efectos secundarios no deseados.



Gráfica 6. Problemas de nnet

Fuente: Elaboración propia

Como se muestra en la gráfica 6, estos errores se encuentran desde la versión 7.3.0 y persisten hasta la versión más reciente de nnet. Por lo que reducir el alcance de estas variables que ocasionan conflictos en el código puede mejorar la legibilidad del código, facilitar su comprensión, y reducir el riesgo de introducir errores debido a modificaciones no deseadas de las variables.

Tabla 34. Duplicación de código en nnet

Duplicación de código		
Versión	Clones	Líneas de código
7.3.0	9	76
7.3.1	9	76
7.3.5	9	76
7.3.7	9	76
7.3.8	9	76
7.3.10	9	76
7.3.12	9	76
7.3.13	9	76
7.3.15	9	76
7.3.16	9	76
7.3.18	9	76
7.3.19	9	76

Fuente: Elaboración propia a partir de datos obtenidos de Codacy

Otro aspecto clave en la calidad del componente nnet es el porcentaje de duplicación. A través de las versiones ha persistido la cantidad de bloques de código y líneas de código, ver tabla 34.

CONCLUSIONES

El proyecto ha permitido evaluar la calidad de los componentes de software seleccionados, Keras y nnet, utilizando herramientas de análisis estático como SonarQube y Codacy. A través del análisis, se identificaron fortalezas y debilidades en áreas clave de la calidad de software, como mantenibilidad, confiabilidad y seguridad, lo que ofrece una visión integral sobre la calidad del código fuente de dichos componentes.

No solo se contribuye al uso y diagnóstico de calidad de componentes de software, sino que también establece una metodología para futuros análisis de calidad de software utilizando herramientas automatizadas. El uso de estas herramientas no solo mejora la calidad del código, optimiza el proceso de desarrollo al reducir el tiempo dedicado a detectar errores en etapas posteriores del ciclo de vida del software.

En conclusión, las herramientas de análisis estático de código son fundamentales para la evaluación de componentes de software, al permitir identificar áreas para mejorar la calidad y asegurar la sostenibilidad del software a largo plazo. Su implementación dentro del proceso de desarrollo de software brinda a los desarrolladores una herramienta confiable, segura y fácil de mantener, contribuyendo al éxito de los proyectos tecnológicos.

COMPETENCIAS DESARROLLADAS

Durante la investigación y desarrollo de la evaluación de calidad de componentes de software, se desarrollaron una serie de competencias clave aplicadas al proyecto.

1. Análisis estático de código

Habilidad para utilizar herramientas de análisis estático como Codacy y SonarQube para evaluar la calidad del software, considerando aspectos como mantenibilidad, confiabilidad y seguridad.

2. Gestión de proyectos

Capacidad para planificar, estructurar y llevar a cabo un proyecto de software, abarcando desde la búsqueda de componentes hasta la ejecución de pruebas y análisis.

3. Calidad de código

Desarrollo de conocimientos sobre métricas de calidad de código, como complejidad ciclomática, duplicación de código, y uso de prácticas de código limpio.

4. Diagnóstico y mejora continua

Habilidad para analizar y diagnosticar la calidad de componentes de software, así como proponer recomendaciones basadas en los resultados de análisis.

5. Documentación técnica

Redacción de informes claros y bien estructurados, incluyendo la definición de criterios de pruebas, métricas de calidad y conclusiones de análisis.

6. Toma de decisiones

Desarrollo de habilidades en la toma de decisiones técnicas, como elegir diferentes versiones de componentes y herramientas.

7. Aprendizaje autónomo

Capacidad de aprender nuevas herramientas y tecnologías de manera independiente para aplicarlas en el proyecto.

FUENTES DE INFORMACIÓN

- C, G. (13 de Marzo de 2024). *¿Qué son las herramientas de análisis estático de código y cómo ayudan a mejorar su calidad?* Obtenido de <https://es.linkedin.com/pulse/qué-son-las-herramientas-de-análisis-estático-código-y-cuesta-arza-7cvhf>
- Casero, A. (7 de Agosto de 2024). *¿Qué son las herramientas de análisis estático del código?* Obtenido de https://keepcoding.io/blog/herramientas-de-analisis-estatico-del-codigo/#¿Por_que_son_importantes_las_herramientas_de_analisis_estatico_del_codigo
- Castillo, I. (Agosto de 2006). Obtenido de ResearchGate: https://www.researchgate.net/figure/Figura-2-Modelo-para-la-calidad-interna-y-externa-segun-ISO-9126-1-6_fig2_271643473
- Codacy. (20 de Octubre de 2023). *Una mirada profunda a las herramientas de análisis de código estático.* Obtenido de https://blog-codacy-com.translate.goog/static-code-analysis-tools?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=tc
- electrónica, S. (1 de Agosto de 2021). *Calidad del software.* Obtenido de <https://saludelectronica.com/calidad-del-software/>
- Foster, S. (7 de Junio de 2023). *¿Qué es el análisis estático? Herramientas de análisis estático + Descripción general de los analizadores de código estático.* Obtenido de https://www-perforce-com.translate.goog/blog/sca/what-static-analysis?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=rq#stage-01
- Go4IT Solutions. (20 de Diciembre de 2024). *¿Qué es el análisis de código estático?* Obtenido de <https://go4it.solutions/es/blog/que-es-el-analisis-de-codigo-estatico>
- ICARIA Technology. (26 de Abril de 2023). *La calidad del software y sus estándares más importantes.* Obtenido de

<https://icariatechnology.com/la-calidad-del-software-y-sus-estandares-mas-importantes/>

IN-COM. (10 de Julio de 2024). *El papel de la calidad del código: métricas críticas y su impacto*. Obtenido de <https://in-com.com/es/blog/the-role-of-code-quality-critical-metrics-and-their-impact/>

Khan, A. (12 de Diciembre de 2024). *¿Qué es el análisis estático del código?* Obtenido de <https://www.kiuwan.com/es/static-code-analysis/>

Mallón, X. (24 de Abril de 2024). *¿Qué es el análisis de código estático?* Obtenido de <https://keepcoding.io/blog/que-es-el-analisis-de-codigo-estatico/>

Manzanelli. (30 de Mayo de 2023). *Norma ISO 9126*. Obtenido de NormalSO: <https://normasiso.org/norma-iso-9126/>

Medina, I. F. (5 de Diciembre de 2022). *Los estándares de calidad del software más importantes*. Obtenido de <https://www.hiberus.com/crecemos-contigo/los-estandares-de-calidad-del-software-mas-importantes/>

Mobilvendedor. (14 de Diciembre de 2024). *Guía para seleccionar herramientas para análisis de datos*. Obtenido de <https://mobilvendedor.com/herramientas-para-analisis-de-datos/>

Navarro, A. (30 de Enero de 2024). *10 Métricas de Calidad del Software para Mejorar la Eficiencia*. Obtenido de <https://blog.innevo.com/metricas-de-calidad-del-software#:~:text=Las%20métricas%20de%20calidad%20del%20software%20son%20criterios%20y%20medidas,los%20estándares%20y%20requisitos%20establecidos.>

Román, E. (30 de Agosto de 2024). *Entendiendo la Calidad del Software: qué es, qué la compone y estándares principales*. Obtenido de <https://blog.innevo.com/calidad-en-el-desarrollo-de-software>

Souza, I. d. (23 de Noviembre de 2019). *Conoce qué es un software libre y sus características elementales*. Obtenido de <https://rockcontent.com/es/blog/software-libre/>

- Talavera, H. (12 de Mayo de 2023). *7 criterios de evaluación de software*. Obtenido de Asesora PYME: <https://asesorapyme.org/2023/05/12/7-criterios-de-evaluacion-de-software/>
- Verity. (28 de julio de 2022). *La ISO/IEC 9126: 2001: Características de la calidad de software*. Obtenido de <https://www.verity.cl/que-es-norma-iso-iec-9126-2001/>
- VIEWNEXT. (11 de Febrero de 2021). *Diferencisas entre Librerías y Frameworks*. Obtenido de <https://www.viewnext.com/diferencias-entre-librerias-y-frameworks/>

ANEXOS

Carta de no Inconveniencia



Educación
Secretaría de Educación Pública



TECNOLÓGICO
NACIONAL DE MÉXICO



Instituto Tecnológico de Apizaco
División de Estudios de Posgrado e Investigación

Tzompantepec, Tlaxcala, 30/enero/2025

Depto.: División de Estudios de Posgrado e Investigación
Oficio No.: DEPI/064/2025

ASUNTO: Carta de No Inconveniencia.

YESICA IMELDA SAAVEDRA BENÍTEZ
DIRECTORA DEL INSTITUTO TECNOLÓGICO DE APIZACO
PRESENTE

La que suscribe Blanca Estela Pedroza Méndez, Jefa de la División de Estudios de Posgrado e Investigación, por este medio tengo a bien manifestar a Usted, que el **C. PEDRO COCOLETZI HERNÁNDEZ** de la carrera de Ingeniería en Tecnologías de la Información y Comunicaciones, con número de control **19370985**, trabajó en la realización del proyecto **“EVALUACIÓN DE LA CALIDAD DE COMPONENTES DE SOFTWARE DE USO LIBRE MEDIANTE HERRAMIENTAS PARA EL ANÁLISIS ESTÁTICO”**, durante el periodo del **02 de septiembre del 2024 al 31 de enero de 2025**, teniendo como asesor externo al Dr. José Juan Hernández Mora y como asesora interna a la Dra. María Guadalupe Medina Barrera, y **No existe inconveniente** alguno de utilizarlo **para fines de titulación**.

Sin otro particular por el momento, me es grato quedar de usted.

ATENTAMENTE
Excelencia en Educación Tecnológica®
Pensar para Servir, Servir para Triunfar®

BLANCA ESTELA PEDROZA MÉNDEZ
JEFA DE LA DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN



BEPM/Angy*
Ccp. Archivo



Av. Instituto Tecnológico No. 418, San Andrés Ahuashuatepec, Municipio de
Tzompantepec, Tlaxcala, Mex. C.P. 90491 Tel. (241) 4172010 Ext. 246, e-
mail: depi@apizaco.tecnm.mx | apizaco.tecnm.mx

