Machine learning para el testeo de calidad del software: caso de estudio de Sonarqube en Cobistopaz

Laura Veronica Ocampo Betancourt

Asesor

Joel Caroll Vargas

Universidad Nacional Abierta y a Distancia (UNAD)

Escuela de Ciencias Básicas, Tecnología y e Ingeniería

Ingeniería en Sistemas

2024

Resumen

El actual estudio buscara explorar el estado actual del uso de machine learning (ML) en el proceso de pruebas de software, analizando el impacto que este ha tenido en la calidad del software, automatización de tareas, detección inteligente de errores y en el ciclo de pruebas en general. Exploraremos como están siendo usados los distintos modelos y técnicas de machine learning para la automatización de casos de pruebas, cobertura de pruebas, optimización de código fuente, corrección de errores e identificación de patrones en datos y comportamientos de las aplicaciones. Por último, examinaremos varios casos de éxito en la industria que muestran la implementación del aprendizaje automático en el proceso de pruebas y su impacto en la calidad del software. Buscando identificar mejores prácticas y lecciones aprendidas que puedan orientar e inspirar la adopción efectiva del aprendizaje automático en los procesos de pruebas de software en diferentes sectores e industrias.

El estudio se complementa con con la presentación de un caso práctico en la empresa Cobis-Topaz, donde se implementa la herramienta SonarQube, una solución basada en Machine Learning, para evaluar la calidad del código de software. Este caso de estudio ilustra cómo una herramienta de este tipo puede ser aplicada en un entorno empresarial para mejorar los procesos de testeo y asegurar estándares de calidad más elevados.

Palabras Claves: Aprendizaje automático, Prueba De Software, Calidad, Modelo.

Abstract

The current study will seek to explore the current state of the use of machine learning (ML) in the software testing process, analyzing the impact it has had on software quality, task automation, intelligent error detection and the development cycle. tests in general. We will explore how different machine learning models and techniques are being used for test case automation, test coverage, source code optimization, error correction, and identifying patterns in application data and behaviors. Finally, we will examine several industry success stories that show the implementation of machine learning in the testing process and its impact on software quality. Seeking to identify best practices and lessons learned that can guide and inspire the effective adoption of machine learning in software testing processes in different sectors and industries.

The study is complemented by the presentation of a practical case in the Cobis-Topaz company, where the SonarQube tool, a solution based on Machine Learning, is implemented to evaluate the quality of the software code. This case study illustrates how such a tool can be applied in a business environment to improve testing processes and ensure higher quality standards.

Keywords: Machine Learning, Software Testing, Quality, Model.

Tabla de Contenido

Lista de Tablas

Lista de Figuras

Introducción

En el ámbito del desarrollo de software, según Julián Andrés Mera-Paz, la implementación de un esquema de pruebas adecuado es una parte integral y fundamental para garantizar la calidad del producto final (Mera Paz, 2016). Igual que para garantizar el buen funcionamiento de la aplicación y contemplar los casos de uso potenciales por parte del usuario final, ofrecer un óptimo rendimiento y permite controlar y seguir los defectos, identificar problemas tempranamente, reducir riesgos, facilitar la mejora continua y mejorar la imagen corporativa, mitigar funcionamientos incorrectos de la aplicación previo a la implementación o puesta en producción de esta, reduce costos de mantenimiento e incumplimientos de requisitos y normativas, aumenta la eficiencia operacional y beneficia la experiencia de usuario y fiabilidad. En tiempos recientes, el uso de inteligencia artificial (IA) y principalmente de una de sus ramas más importantes, el machine learning (ML) ha supuesto un innovador e interesante agregado para mejorar el proceso de testeo de calidad de software, el objetivo del actual documento busca explorar y determinar las técnicas y herramientas de machine learning más convenientes para este propósito, así como realizar una revisión del estado actual de la implementación del aprendizaje automático y casos de éxito donde se ha implementado el machine learning para el proceso de testeo de calidad de software.

El estudio realizado por Vinicius Durelli resalta varios beneficios del uso de algoritmos de Machine Learning (ML) para automatizar y simplificar las actividades de prueba de software, como lo son: Escalabilidad, reducción de la carga de trabajo manual, mejor comprensión del sistema y predicción de métricas clave. Los resultados indican que los enfoques basados en Machine Learning suelen ser altamente escalables, haciendo que se afronten a la creciente

complejidad de los sistemas de software modernos. Muchos de los enfoques basados en Machine Learning descritos requieren una intervención humana mínima, lo que los hace una alternativa interesante para automatizar tareas de prueba tediosas y propensas a errores manualmente. En cuanto a la comprensión de sistema, algunos algoritmos de Machine Learning, como los árboles de decisión, generan modelos interpretables que pueden ayudar a los probadores a comprender mejor el comportamiento del sistema bajo prueba. Adicional resalta que los enfoques basados en Machine Learning han demostrado ser efectivos y traer grandes beneficios en términos de escalabilidad, automatización, comprensión del sistema y medición de métricas relacionadas al proceso de pruebas, Tales como el esfuerzo de ejecución de las pruebas o a tasa de detección de errores, lo que puede ayudar a los analistas funcionales a estimar y planificar mejor los esfuerzos de prueba, traduciéndose en una mayor eficiencia y efectividad en la ejecución de los planes de pruebas planteados. (Durelli, y otros, 2019). Por su parte Mahdi Noorian sostiene que el uso de técnicas de Aprendizaje de Máquina (ML) en las pruebas de software ofrece automatizar diferentes aspectos de las pruebas de software, lo que reduce el costo y el tiempo de este proceso, adicional de mejorar significativamente el desempeño de del proceso de pruebas (Noorian, Bagheri, & Du, 2011). En la misma línea Noorian, indica en su estudio que el proceso de pruebas de software manual es intensivo en mano de obra y costoso, adicional puede requerir hasta el 50% de recursos de los recursos de desarrollo.

El actual análisis se enfocará en analizar en detalle como el aprendizaje automático o machine learning (ML) puede integrarse y contribuir a mejorar, automatizar y optimizar diferentes aspectos y tareas realizadas durante el proceso de testeo de calidad, tales como la generación automática de casos y guiones de prueba, la detección de defectos y anomalías, la evaluación de escenarios de pruebas y la mejora de la eficiencia en la identificación de

problemas y comparativas en el rendimiento del sistema. Se realizo una revisión integral de la literatura existente, examinando ensayos, investigaciones, libros, artículos, revistas y aplicaciones prácticas que nos permitan evidenciar el impacto del machine learning en el testeo de calidad de software. Al igual, que experiencias, pilotos y casos de éxito realizadas en la industria que nos permita identificar tendencias, desafíos y áreas de oportunidad en este campo emergente.

Con el presente documento, se espera ofrecer una visión integral y actualizada de cómo el machine learning está transformando el testeo de calidad de software, proporcionando recomendaciones claras sobre las técnicas más efectivas y las mejores prácticas para su implementación en entornos de desarrollo de software. Al igual, que lecciones aprendidas y experiencias resultantes de la implementación del machine learning en la industria. Por último, Este trabajo busca establecer una base y establecer nuevas perspectivas para futuras investigaciones y desarrollos en este campo innovador y prometedor.

Planteamiento del Problema

La fase de pruebas o testing dentro del proceso de desarrollo de software es un proceso critico en el ciclo de vida del desarrollo de aplicaciones y el cual cada día toma más protagonismo, dado que los tiempos exigidos por el mercado e industria son cada vez más reducidos tanto para la creación de nuevos aplicativos como para la actualización y despliegue de nuevas funcionalidades. al igual, que los niveles de exigencia y los estándares de calidad cada día son más estrictos y rigurosos. Lo cual nos conlleva a evolucionar y optimizar los esquemas de pruebas, en busca de identificar y corregir defectos que podrían afectar la funcionalidad, seguridad o experiencia que el usuario final pueda tener en el uso del aplicativo. Generando un gran paradigma porque los métodos tradicionales de pruebas y automatización se enfrentan a grandes desafíos para adaptarse a estas nuevas exigencias, ya que a menudo quedan cortos en cuanto a eficacia, saturación, planteamiento de escenarios de pruebas o en el registro de tiempos óptimos, en la culminación de los casos planteados.

En este contexto, es donde el uso de los distintos modelos y técnicas de aprendizaje automático o Machine Learning (ML) han surgido como un parteaguas y opción revolucionaria para mejorar la efectividad y eficiencia del testeo de calidad de software. Sin embargo, pese a las grandes bondades y ventajas que nos puede brindar la integración de Machine Learning a los procesos de pruebas, existen aún varios desafíos y preguntas clave que requieren un análisis más profundo, tal como son las planteadas a continuación:

- **Selección de Técnicas Adecuadas:** Identificar las adecuadas y más eficaces técnicas de Machine Learning para integrar a las diferentes fases del testeo de calidad, como la

generación automática de casos y guiones de prueba, la detección de defectos y la evaluación de los posibles escenarios de pruebas.

- **Análisis de Resultados:** Evaluar la optimización de los diferentes modelos de aprendizaje para garantizar una detección precisa de defectos, análisis de patrones en los resultados obtenidos y una mayor cobertura de casos de prueba, minimizando falsos positivos y falsos negativos.

- **Adaptabilidad:** Analizar la capacidad de los modelos de Machine Learning para generar nuevos procesos de aprendizaje y adaptarse a cambios en los requisitos y funcionalidades del software y en el entorno de desarrollo.

- **Estado del Arte y Buenas Prácticas:** Realizar una revisión sistemática del estado del arte del uso de Machine Learning en el testeo de calidad de software, identificando tendencias, áreas de mejora y retos por superar en la actualidad.

Este planteamiento del problema busca destacar la necesidad de investigar a fondo el naciente potencial en la integración del Machine Learning para mejorar los actuales esquemas de prueba y calidad de software, al igual que encontrar y analizar los desafíos existentes para su implementación. Al abordar estas dudas y paradigmas se espera contribuir a la mejora continua de los procesos de calidad en el desarrollo de software. Al igual, que promover el uso del aprendizaje automático para garantizar la calidad y fiabilidad en los esquemas de pruebas en el desarrollo de software.

Justificación

En el actual contexto del desarrollo de software y en donde la sociedad integro el uso de las aplicaciones o herramienta en casi todos los ámbitos de su vida tanto a nivel personal como laboral, La calidad de estas es un factor determinante en el éxito o fracaso del aplicativo en el mercado (Montalván Vélez, Mogrovejo Zambrano, Romero Vitte, & Pinargote Carrera, 2024). De allí, que las pruebas de software al ser un proceso crítico para asegurar tanto la calidad como el cumplimiento de requisitos y funcionalidades que busca abarcar el software. Nos hace plantear métodos para la optimización y mejora de los esquemas tradicionales de pruebas, dado las limitaciones y errores humanos que estos pueden acarrear.

Los avances en inteligencia artificial (IA) y más puntualmente en la rama del aprendizaje automático o machine learning nos ofrece una oportunidad de revolucionar y transformar el proceso de pruebas de calidad de software, brindándonos modelos y técnicas que están siendo probadas e implementadas en sin fin de proyectos, organizaciones y áreas de estudio, obteniendo resultados sorprendentes y muy innovadores, los cuales abarcan desde el análisis de grandes volúmenes de información hasta la predicción de resultados y estadísticas futuras, pasando por la identificación de patrones de comportamiento, automatización de tareas repetitivas, optimización de código, reducción de tiempos y cobertura de escenarios de pruebas, asegurando con ello una mayor calidad en los productos finales (Chacón, García, Hervera, Barrios, & Moreno, 2022). Dado estos importantes resultados y evidenciando la eficacia y magnifica contribución que la implantación de machine learning significa para la industria del software, es que ha crecido exponencialmente la presión del mercado y en general del entorno, para integrar el aprendizaje automático tanto al desarrollo de software como al

proceso de pruebas de este. Haciéndolo un factor diferenciador entre aquellos que lo implementa y quienes no (Calvo, Guzmán, & Ramos, 2018) .

Lo que nos conlleva a realizar un estudio profundo del uso del machine learning en la industria y como estos están siendo aplicados para mejorar y optimizar el proceso de pruebas de calidad de software.

Esta monografía tiene como objetivos explorar los modelos, metodologías y técnicas de aprendizaje automatizado más usadas y con resultados innovadores, evaluando sus bondades, desafíos y el impacto potencial en la industria del desarrollo de software que estos están teniendo. Proporcionando una guía y una base de estudio, tanto para desarrolladores, tester, gerentes y expertos de inteligencia artificial sobre cómo usar e implementar el aprendizaje automatizado en los procesos de pruebas y con ello contribuir al aseguramiento de calidad y evolución de los aplicativos o herramientas implementadas a futuro.

Objetivos

Objetivo General

Investigar la aplicación de técnicas de Machine Learning en el proceso de testeo de calidad de software, a través del desarrollo de una revisión de literatura y la presentación de casos de estudio de la empresa Cobis-Topaz.

Objetivos Específicos

Realizar una revisión del estado del arte del uso de machine learning en testeo de calidad del software, con el propósito de identificar las principales tendencias y desafíos en este campo.

Determinar las técnicas y herramientas de machine learning más convenientes para el testeo de calidad de software, con el fin de establecer su aplicabilidad en diferentes escenarios del ciclo de vida del desarrollo de software.

Proporcionar un caso de estudio de la aplicación de machine learning para el testeo de calidad de software en la empresa Cobis-Topaz, para evaluar su impacto en la mejora de la calidad del código.

Marco Teórico

Aprendizaje Automatizado

El aprendizaje automático, aunque es un concepto complejo y difícil de comprender por la variedad de temas y conceptos que abarca, ya que engloba temas tan variados que van desde antropología, sociología, estudio animal, matemáticas y ciencias de la computación. Ya que para determinar la manera óptima en la que un maquina o sistema informático aprenda, nos tenemos que remontar a las maneras más efectivas de la que aprenden tanto humanos como animales. Lo que genera una simbiosis peculiar y algo curiosa, dado que mientras más se profundiza en el aprendizaje automático, más se aprende acerca del aprendiza humano y biológico (Dark, 2019). Uno de los conceptos más conocido en el aprendizaje humano es el aprender en base a la observación y el ejemplo, concepto el cual se extiende al aprendizaje automatizado dado que las maquinas aprenden con base en los datos etiquetados o de ejemplo suministrados y los cuales se denominan Datos de entrenamiento que pueden llegar a ser un grupo de muestra de una cantidad de datos mayor o bien puede ser datos históricos desde donde se pueden inferir tendencias o patrones de comportamiento (Cahuasa, 2024).

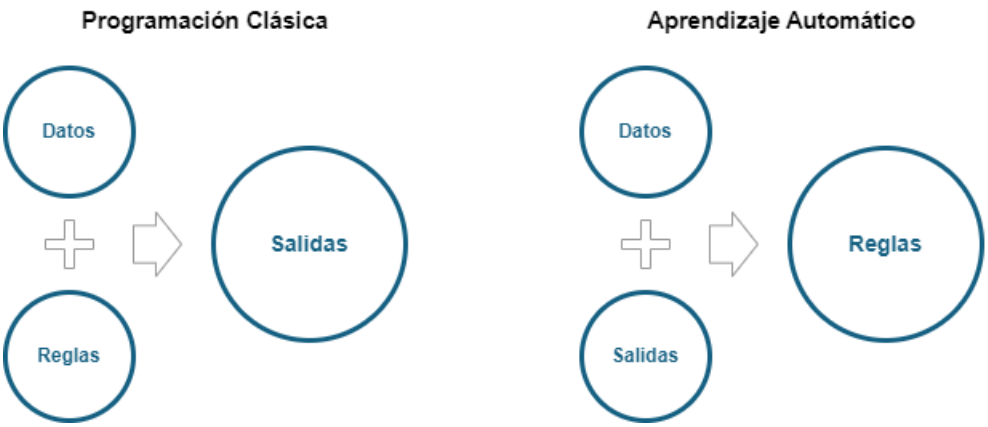
De igual manera, a lo largo de la historia han surgido varios conceptos entres los más conocidos el dado por Arthur Samuel en 1959, quien lo define como "*Aprendizaje automático es el campo de estudio que da al computador la habilidad de aprender sin haber sido explícitamente programado para ello*" y aunque podemos encontrar una infinidad de conceptos similares la definición más en la que la mayoría están de acuerdo y que engloba de gran manera su función es que el aprendizaje automático es una rama de la inteligencia artificial que

busca que un programa de computador aprenda de un conjunto de datos con los cuales se entrena (Pineda Pertuz, 2021). Una diferencia bastante significativa respecto a la programación tradicional donde el sistema recibe datos de entrada y cumpliendo una serie de reglas e instrucciones, se espera una salida con antelación ya establecida. Mientras que en el aprendizaje automático o Machine Learning (ML) los datos iniciales corresponden a los datos de entrada y los datos de salida y mediante un proceso de entrenamiento el sistema concluye cuales son las reglas de negocio o los pasos del proceso a seguir. Dicho proceso de conclusión realizado por el sistema recibe el nombre de modelo, siendo este el resultado de detectar patrones o comportamientos comunes y usar los mismos para predecir comportamientos y tendencias de la información a futuro. El éxito de la correcta predictibilidad del sistema yace en el ajuste de los distintos logaritmos usados y la calidad en los datos de entrenamiento que recibe. Dado que cada algoritmo se centra en un tipo puntual de aprendizaje para perfeccionar su entendimiento y por ende afinar su predictibilidad en datos de entrada y de salida y en donde en muchos casos para lograrlo se acude a un esquema de prueba y de error, para conseguir el mejor desempeño posible. Dado este nivel de especialización de cada algoritmo es que es muy complicado en una misma solución reutilizar un algoritmo en varios escenarios, lo que genera varios algoritmos especializados en situaciones.

La siguiente grafica muestra el esquema de la programación tradicional vs el aprendizaje automático

Figura 1

Programación Clásica vs Aprendizaje Automático



Fuente. Adaptado de Aprendizaje automático y profundo en Python: Un enfoque teórico (Pineda Pertuz, 2021)

Mientras que el aprendizaje automático o machine Learning (ML) presentan un proceso más complejo y una serie de pasos y etapas a seguir, en donde cada una describe una serie de condiciones y tareas a realizar.

Figura 2

Etapas Aprendizaje Automático

Fuente. Adaptado de Aprendizaje automático y profundo en Python: Un enfoque teórico (Pineda Pertuz, 2021)

Procesamiento de Datos: la primera etapa del aprendizaje automático consta de la limpieza y transformación de datos para que estos sean legibles y de fácil comprensión para el algoritmo. Esta etapa es una parte crucial del proceso dado que los datos que se le suministren al algoritmo deben ser numéricos y en la mayoría de los casos deben estar en una misma escala. De igual manera, se debe prestar atención tanto a los datos vacíos como nulos, ya que el algoritmo los considera y podría generar que se malinterpreten o que este trabaje de forma deficiente o que su análisis no sea el óptimo o el esperado (Pineda Pertuz, 2021).

Separación En Conjunto De Entrenamiento Y Pruebas: El conjunto de datos (Dataset) con los que se procederá a almacenar el algoritmo generalmente suelen separarse en dos grandes subconjuntos llamados entrenamiento y pruebas. El primero se usa para entrenar y estimar los parámetros de entrada y salida del sistema, mientras que el segundo se usa para que el sistema genere y mejore su modelo de predictibilidad y que logre inferir posibles comportamientos en los datos y resultados esperados. La división en el conjunto de datos usualmente se suele hacer 70%-30% o 80%-20% para entrenamiento y pruebas respectivamente (Pineda Pertuz, 2021).

Configuración del algoritmo: Se crea una instancia u objeto del algoritmo a utilizar y se definen y se afina el llamado a los hiperparámetros. Los hiperparámetros se diferencian de los parámetros en que estos últimos son generados por el sistema mientras está siendo entrenado, sin tener ningún tipo de intervención externa o humana. Algunos hiperparámetros empleados con regularidad suelen ser Numero de Épocas y la tasa de aprendizaje (Pineda Pertuz, 2021).

Entrenamiento del Modelo: es esta etapa, donde los grupos de datos preseleccionados y separados previamente proceden a alimentar al algoritmo para que este pueda aprender, logrando así generar y pulir su modelo de predicción (Pineda Pertuz, 2021).

Predicción: una vez generado el modelo, se suministrará una muestra de pruebas para que este genere predicciones, con el fin de inferir un posible resultado o valor esperado (Pineda Pertuz, 2021).

Evaluación: ya como última etapa se busca determinar de manera numérica y cuantificable, la efectividad del aprendizaje del sistema se evaluará con la base que en entre la salida generada del sistema se aleje más del resultado esperado peor será la evaluación y caso contrario mientras la salida y el resultado sea esperado sean similares la evaluación será más positiva (Pineda Pertuz, 2021).

Conceptos de Aprendizaje Automático

En la siguiente sección, se expondrán algunos conceptos a nivel general que suelen ser utilizados con regularidad en la implementación y estudio del aprendizaje automático o Machine Learning (ML):

Conjunto de Datos (Dataset): Los conjuntos de Datos corresponden a conjuntos de datos almacenados de forma específica y que le servirán al algoritmo para ser entrenado y al mismo tiempo para probarlo y realizar mediciones. Estos pueden ser tanto de tipo estructurado

como no estructurado, en donde los primeros se caracterizan por manejar una estructura predefinida, tal como una tabla o base de datos y estos suelen generalmente manejarse mediante archivos de formato csv (comma separated values). Por otro lado, los conjuntos de datos no estructurados no tienen un formato especifico o un tipo de archivo en particular (Pineda Pertuz, 2021).

Variables Descriptivas: Describen las instancias almacenadas en el conjunto de datos y suelen tener un tipo de dato asociado. Estas variables o atributos tienden a guardar una relación estrecha con el tipo de información proceda. Por ejemplo, si el conjunto de datos trata de bienes raíces o casas, algunas de las variables descriptivas usadas serán: num habitaciones, estrato, etc. (Pineda Pertuz, 2021).

Variables destino o Etiquetas: es un conjunto o característica usado para el etiquetado de las instancias para mejorar su modelo de entrenamiento. De igual manera, también son usados para representar aquellos resultados o atributos que se esperan como resultado o que se deseen predecir (Pineda Pertuz, 2021).

Inducción: Los algoritmos de aprendizaje asimilan el conocimiento a través de un proceso llamado “Inducción del Aprendizaje”, el cual es un proceso de razonamiento donde se genera un modelo de la información (Datos de Entrenamiento) (Cuevas, Avalos, Diaz, Valdivia, & Pérez, 2021).

Generalización: hace referencia a encontrar o inferir el patrón o modelo de entrenamiento para predicción de resultados o toma de decisiones (Cuevas, Avalos, Diaz, Valdivia, & Pérez, 2021).

Sobreentrenamiento: Cuando un modelo de aprendizaje aprenda de una manera tan precisa y exacta de los datos de entrenamiento que pierde la capacidad de generalizar y con ello su rendimiento frente a grupos de datos que están fuera del campo en el que se especializaron seria deficiente o de una baja calidad (Cuevas, Avalos, Diaz, Valdivia, & Pérez, 2021).

Subentrenamiento: este es el caso contrario del anterior punto, donde el modelo no aprendió lo suficiente de la muestra de datos en la etapa de entrenamiento, bien sea porque el proceso fue deficiente o no tuvo la duración necesaria. Obteniendo un resultado más guiado a la generalidad de datos, pero con un rendimiento bajo y que no es el esperado (Cuevas, Avalos, Diaz, Valdivia, & Pérez, 2021).

Tipos de Datos: Cada variable o atributo debe asumir un tipo de dato puntual el cual definirá los valores que puede tomar y el comportamiento que asumirán durante el entrenamiento. Estos tipos de datos son los siguientes:

- Numéricos: Representan Cantidades Numéricas y debe ser un número real. Ej. Largo: 2,5.

- Categóricos: Representan Valores de Tipo Cadena de caracteres de un rango limitado o hace referencia a cualidades o categorías ej. Color: Amarillo, Azul y Rojo.

- Ordinales: Son Cadenas con la diferencia que se puede establecer un orden entre ellos. Ej. Talla: L, M y S y se podrá establecer que L>M>S.

A continuación, se ejemplificará el uso de los tipos de datos en donde se genera una estructura de datos para la aprobación de un crédito bancario. En donde se usan seis atributos y cuatro muestras y en donde el salario y la edad son de tipo numérico, la historia crediticia es categórico y la ocupación es ordinal.

Tabla 1

Ejemplificación de Tipos de Datos

	Atributos Descriptivos				***Etiqueta***
ID	**Salario**	**Historia Crediticia**	**Ocupación**	**Edad**	**Préstamo**
1	$ 1.000.000,00	Bien	Profesional	20	SI
2	$ 500.000,00	-	Técnico	18	NO
3	$ 2.000.000,00	Bien	Profesional	52	SI
4	$ 3.000.000,00	-	Bachiller	18	NO
5	$ 1.500.000,00	Mala	Técnico	25	NO
6	$ 7.000.000,00	Excelente	Profesional	36	SI
7	$ 3.500.000,00	Bien	Tecnólogo	40	SI
8	$ 5.000.000,00	Excelente	Profesional	50	SI
9	$ 700.000,00	Mala	Técnico	39	NO
10	$ 5.000.000,00	Mala	Tecnólogo	35	NO

Nota. Ejemplificación estructura de datos para la aprobación de un crédito bancario

Para obtener resultados óptimos en el entrenamiento, los datos de atributos deben convertirse en valores numéricos para facilitar que el modelo se entienda y tener resultados satisfactorios en términos de predictibilidad.

Modelo: resultado del aprendizaje automático y generado por los algoritmos a partir de la muestra de datos de entrenamientos suministrados. El Modelo es el conocimiento base (Patrón o Tendencia) usado para hacer predicciones y encontrar comportamientos en nuevos datos de entrada suministrados.

Tipos de Aprendizaje Automático

Existen varios tipos de aprendizaje automatizado dependiendo el modelo o algoritmo que está usando, pero la mayoría tiende a generalizarlos o agruparlos en aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo. a continuación, presenta una visión general a cada uno de estos y de los subtipos que puedan llegar a tener cada uno:

Aprendizaje Supervisado: en este tipo de aprendizaje todas las muestras de datos suministradas al algoritmo están seleccionadas y etiquetadas con un resultado (valor predefinido) o categoría usando una variable llamada destino u objetivo, para especificar al modelo las posibles entradas y el resultado esperado y mejorar sus patrones y modelos de predictibilidad para nuevos datos de entrada según resultados previos encontrados en la muestra de información con la que se alimentó (Pineda Pertuz, 2021).

Aprendizaje No Supervisado: En este tipo de aprendizaje no contamos con etiquetas en la muestra de datos de entrenamiento y en donde el modelo analiza directamente las variables de entrada tratando de encontrar relaciones o patrones existentes entre ellas.

Haciendo poco uso de la predicción ya que él modelo se guía más por las características en común y no en los resultados etiquetados esperados.

Aprendizaje por Refuerzo: Este tipo de modelo tiene como base el aprendizaje a partir de ensayo y de error. Donde el algoritmo analizando en base a la observación del entorno y los datos muestra y seleccionado y ejecutando acciones obteniendo de esta manera feedback en forma de recompensas si los resultados son correctos o en caso contrario recibirá penalizaciones cuando los resultados no sean óptimos. Aprendiendo la mejor estrategia (Política) para obtener un resultado optimo y mayores recompensas positivas. Estos algoritmos de aprendizaje llamados agente, buscan que el algoritmo infiera y cree políticas con el fin de reducir las penalidades e ir aprendiendo paulatinamente cuales son las mejores decisiones potenciando su capacidad predictiva (Cuevas, Avalos, Diaz, Valdivia, & Pérez, 2021).

Algoritmos del Aprendizaje Automático

Regresión: La regresión permite predecir un valor continuo, con base en la muestra de variables de entrada suministradas, para encontrar una relación o dependencia entre cierto número de características y una variable objetivo-continua. Estableciendo una tendencia o patrón entre grupos de datos. Por ejemplo, se puede establecer el precio de una casa en base al histórico de precios en la misma ciudad o país o predecir la temperatura de un mes basado en los datos de temperatura histórico de años pasados.

Agrupación *o* Clustering: Principalmente son utilizados en el aprendizaje automático de tipo no supervisado ya que permite organizar y categorizar datos no etiquetados basados en similitudes. La categorización de cada punto de datos se realiza de manera iterativa y se asocian a un grupo o muestra de datos específicos tomando como base las características que se establecieron como predeterminadas. Ejemplos de este algoritmo son K-MEANS y DBSCAN, cuyo objetivo es dividir un conjunto de datos en grupo s según la similitud de cada dato o agrupar según la densidad local.

Algoritmos de Árbol de Decisión: Es una herramienta estructural muy útil para la toma de decisiones y elección de opciones en base a criterios preestablecidos. Similar a un diagrama de flujo, dividen repetidamente el conjunto de datos en subconjuntos más pequeños en función de características específicas, Dentro del árbol se generan nodos que representan variables específicas y en las ramas se puede observar el resultado de las pruebas ejecutadas. Se pueden evaluar utilizando métricas de agrupamiento o regresión, dependiendo de la tarea.

Redes Neuronales Artificiales (ANN): Las redes neuronales son modelos flexibles que pueden aprender representaciones complejas de los datos. Se pueden evaluar utilizando métricas específicas de la tarea, como precisión, recall, o MSE, dependiendo de si se trata de una tarea de clasificación o regresión.

Aprendizaje Profundo (Deep Learning): Este campo utiliza redes neuronales con múltiples capas (también conocidas como redes neuronales profundas) para aprender representaciones jerárquicas de los datos. Se pueden evaluar de manera similar a las redes neuronales artificiales, pero a menudo requieren más recursos computacionales y datos de

entrenamiento. Estás redes son conjuntos interconectados de grupos o muestras de datos que trabajan en conjunto para darle solución a problemas puntuales.

Problemas Típicos de Aprendizaje Automático

Existen varios problemas comunes cuando nos adentramos en el campo del aprendizaje automático, los expuestos a continuación son arquetipos de la mayoría de los problemas presentados al momento de implementar, entrenar o desplegar un modelo de aprendizaje automático:

Calidad de los Datos: uno de los principales inconvenientes presentados en el entrenamiento de un modelo, es garantizar la calidad de los datos y garantizar que estos no presenten valores faltantes, duplicados o incorrectos los cuales pueden generar falsos positivos o modelos impreciso o con un bajo nivel de predictibilidad.

La Maldición De La Dimensionalidad (Curse Of Dimensionality (Bellman, 1961)**):** los modelos suelen trabajar bien con un numero especifico de características o atributos, pero al aumentar estas sin un control adecuado puede aumentar exponencialmente los problemas presentados en entornos más pequeños y controlados. Entre más variables, mayor seria la capacidad de predecir de nuestro modelo, pero sin la supervisión adecuada podríamos obtener un modelo complejo con muchas variables que no aporten valor y que el rendimiento final no sea el esperado o su exactitud para predecir no sea la mejor.

Subajuste (Underfitting): este se suele presentar cuando el entrenamiento del algoritmo no fue el adecuado y no se le aportaron la suficiente cantidad de datos. Obteniendo así, un modelo simple que no puede representar o inferir la relación existente entre las variables de entrada y salida.

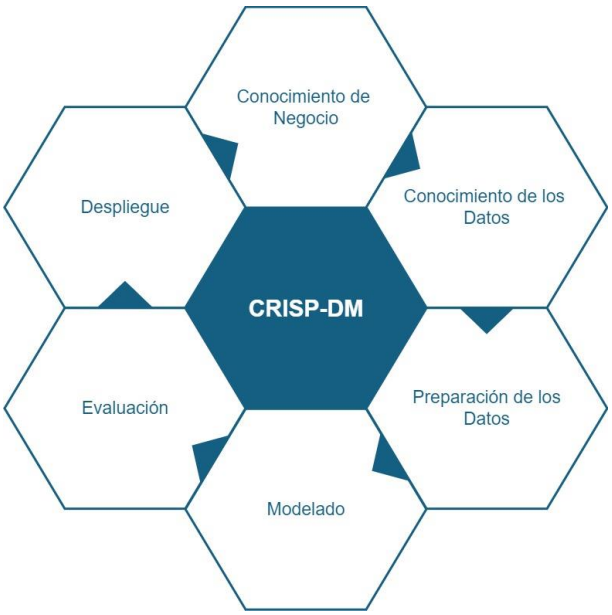
Sobreajuste (Overfitting): Sobreajuste es lo opuesto al subajuste, este ocurre cuando el entrenamiento del algoritmo fue tan robusto y complejo que género que nuestro modelo, aunque se puede ajustar a una gran cantidad de datos, no se va a poder ajustar bien a los datos nuevos y por ende no tendrá un buen nivel de generalización y de predecir patrones o comportamientos sobre nuevos datos

Metodología CRISP-DM

CRISP-DM (Cross-Industry Standard Process for Data Mining) es el estándar abierto más usado para los proyectos de aprendizaje automático. Es un modelo genérico, flexible y cíclico diseñado para adaptado en procesos de minería de datos, en donde se busca extraer patrones, tendencias y relaciones en los datos analizados (Pineda Pertuz, 2021). Esta cuenta con seis fases de desarrollo e iterativas:

Figura 3

Fases de la Metodología CRISP-DM



Fuente. Adaptado de Aprendizaje automático y profundo en Python: Un enfoque teórico (Pineda Pertuz, 2021)

Conocimiento De Negocio: la primera fase consiste en entender los objetivos de proyecto o negocio a la cual se le suplirá una necesidad por medio de la implantación de una solución con aprendizaje automático.

Conocimiento De Los Datos: en esta fase se deben adaptar y conocer la cantidad, tipo, estructura y calidad de los datos con los cuales se cuentan.

Preparación De Los Datos: Consiste en convertir los datos con los que se cuenta en un formato o estructura determinada. Para que puedan usarse por un algoritmo de aprendizaje

automático, que generará un modelo que permita identificar patrones y resultados sobre datos nuevos.

Modelado: Se analizarán varios algoritmos y modelos con el fin de evaluar y definir cual se acopla más a la necesidad y resultados esperados.

Evaluación: el modelo escogido en la fase anterior será a sometido a varias pruebas de evaluación y desempeño. Para comprobar que el nivel de predictibilidad es el esperado y que los resultados esperados sean satisfactorios.

Despliegue: la fase final busca integrar el modelo de aprendizaje automático dentro de un sistema de información y que los usuarios finales pueden hacer uso de este e interactuar con el mismo.

Librerías y Frameworks de Machine Learning

TensorFlow: Es una de las más importantes y populares bibliotecas de código abierto usadas para el desarrollo de modelos de Aprendizaje Automático (AA) y Deep Learning. Debido a su gran escalabilidad, flexibilidad, Soporte para redes neuronales y dado que cubre todas las etapas del ciclo de vida del desarrollo del Aprendizaje Automático, es una de las herramientas más usadas por desarrolladores para la construcción y entrenamiento de modelos de aprendizaje automático. Al desarrollarlo Google, tiene un amplio respaldo tanto del gigante de la industria como de la gran comunidad de usuarios con la que cuenta, que constantemente

están publicando paquetes de aplicaciones, al igual que casos de éxitos, experiencias y retroalimentación respecto a implementaciones en distintos ámbitos y sectores (Google Developer Program, 2021).

PyTorch: Es una biblioteca de código abierto desarrollado por Meta y utilizada para el desarrollo de aplicaciones de machine learning. Dentro de sus principales características se destaca por su facilidad de uso, soporte dinámico de grafos computacionales, cálculos de tensor, aceleración de GPU e integración con Python y CUDA (Compute Unified Device Architecture), lo que la destaca como una de las opciones más atractivas tanto para el desarrollo de aplicaciones como para la realización de investigaciones académicas (The PyTorch Foundation, 2023).

Keras: es una de las principales API (Application Programming Interfaz) de alto nivel usada para el desarrollo de modelos de machine learning y enfocada principalmente en experimentos con redes neuronales, destaca por su simplicidad, facilidad de uso y soporte para experimentación y generación de prototipos de forma rápida, lo que la hace ideal tanto para principiantes como para que aquellos que presentan un alto de nivel de pericia. Es compatible y se puede utilizar con backend como TensorFlow, Theano y CNTK (Google Developer Program, 2022).

Plataformas y Entornos de Desarrollo (IDEs)

Jupyter Notebooks: Es un entorno web que permite la creación, visualización y documentación de código en vivo, ecuaciones y texto narrativo. Se usa especialmente en la investigación académica, ciencia de datos y desarrollo de software. Al contar con soporte para múltiples lenguajes de programación y al poderse integrar con una gran variedad de herramientas y repositorios de datos como lo son GitHub y Google Colab, la hace una poderosa herramienta para ser usada por desarrolladores y científicos de datos (Cortés , 2023).

Microsoft Azure Machine Learning Studio: Una plataforma de desarrollo integral en la nube desarrollada por Microsoft, que proporciona una amplia gama de las herramientas necesarias para la construcción, entrenamiento y despliegue de modelos de machine learning en cada una de sus etapas de desarrollo. Cuenta con integración con otros servicios de Azure al igual que permite realizar escritura de código en lenguajes como .Net, C#, Python y R (Stott, 2019).

Amazon SageMaker: Un servicio de Amazon Web Services (AWS) enfocado en facilitar el ciclo de desarrollo de modelos de Machine Learning a gran escala de manera rápida y sencilla. Al ser desarrollado por Amazon cuenta con integración con otros servicios de AWS al igual que con herramientas propias tales como AutoML, Debugger, y MLOps. Cuenta con su propio IDE (Integrated Development Environment) llamado SageMaker Studio, el cual brinda un entorno que cuenta herramientas para la codificación, depuración, visualización de datos y gestión de distintos modelos de Machine Learning (Amazon Web Services, 2024).

Pruebas de Calidad de Software

Las pruebas de software son una etapa trascendental para garantizar la calidad y el funcionamiento correcto y eficiente bien sea del aplicativo, modulo, funcionalidad o interfaz. Estas pruebas se rigen y sostienen sus bases sobre ciertos principios, procesos y niveles. Los cuales se explorarán brevemente a continuación, con la finalidad de comprender el paso a paso y fases implicadas en el proceso de pruebas de calidad de software y de esta manera determinar cuáles de estas son susceptibles de automatización y que herramientas es la óptima para usar en cada fase o tipo de pruebas, buscando con ello aumentar significativamente la eficacia, eficiencia y precisión de las pruebas. Al igual, que reducir costos y errores provenientes de errores humanos (Sánchez Peño, 2015).

Principios Básicos de las Pruebas de Software

El proceso de pruebas o testing de calidad de software es una de las fases más importantes y uno de los pilares fundamentales dentro del ciclo de vida del desarrollo de software. ya que es en esta fase, donde se garantiza el correcto funcionamiento del producto al igual que su calidad, seguridad, rendimiento y confiabilidad. Para cumplir con estos objetivos la mayoría de los esquemas de pruebas suelen basarse en una serie de principios básicos, los cuales son usados como brújula y guía para que el proceso de pruebas se desarrolle satisfactoriamente y trate de abarcar todas las aristas que se puedan llegar a presentar durante el proceso (Realpe Mancipe, 2019).

A continuación, daremos un vistazo global a algunos de estos principios básicos, con el fin de entender las bases sobre las cuales se debe desarrollar un proceso de testeo de calidad de forma exitosa:

Las pruebas demuestran la existencia de defectos: el objetivo fundamental de las pruebas es comprobar que el software o aplicativo cuenta con errores, pero no pueden demostrar del todo que el software está libre de estos. Aunque se disminuya exponencialmente la probabilidad de errores no descubiertos en el software, no se puede corroborar que el software se desarrolló de manera perfecta y que no presentara inconvenientes al usarse (Baquero, 2023).

Las Pruebas Exhaustivas son Imposibles: abarcar todas las posibles entradas, escenarios, variables, estados y rutas de ejecución es una tarea titánica a menos que el aplicativo o software sea extremadamente pequeño o sencillo. Lo que nos obliga a realizar analizar y priorizar tanto de funcionalidades como de casos de pruebas, para priorizar estos y centrar recursos y esfuerzos en probarlos (Mera Paz, 2016).

Pruebas Tempranas: de ser posible, las pruebas deben comenzar desde etapas tempranas en el ciclo de desarrollo y deben tener objetivos claros dependiendo de la fase donde se encuentre este, ayudando a detectar errores tempranos y disminuyendo la carga y los costos, respecto a realizar todo el proceso en etapas posteriores o finales (Baquero, 2023).

Agrupamiento de Defectos: la funcionalidad o sección de código donde se encuentra un pequeño número de errores tiene un gran porcentaje probabilístico que albergue muchos más errores. Este principio se conoce como el "principio de Pareto" o "principio del 80/20". Donde el 80% de los errores pueden encontrarse en el 20% del código fuente o del total de funcionalidades. De allí, que se recomienda realizar una estrategia y análisis para concentrar las pruebas y esfuerzos en estos puntos críticos.

Paradoja del Pesticida: realizar el mismo conjunto de pruebas una y otra vez, generara un desgaste en las mismas, haciéndolas cada vez menos efectivas e ineficaces. Conllevando que cada vez se encuentren menos defectos o errores. Por eso, hay que revisar y actualizar periódicamente el grupo de pruebas ejecutado y adicionando a estos casos que puedan cubrir más casuísticas (Mera Paz, 2016).

Pruebas Dependientes del Contexto: El enfoque y las técnicas de pruebas estar reinventándose y variado respecto al contexto del proyecto y el tipo de aplicación que se esté probando (Mera Paz, 2016).

La Falacia de la Ausencia de Errores: La ausencia de errores o defectos no garantiza que cumpla con todas las necesidades, requerimientos funcionales o que satisfaga las exigencias de los usuarios finales. De allí, que no el proceso de pruebas no solo se tiene que enfocar en la detección de errores, sino en garantizar los requisitos y resultados esperados.

Proceso Fundamental de Pruebas

El proceso de pruebas de calidad de software se puede dividir o desglosar en 5 etapas claves, cada una de las cuales juega un papel importante para garantizar que el proceso de pruebas sea exitoso y se permita garantizar la calidad del aplicativo o software, que se está probando (Spillner, Linz, & Schaefer, 2014). A continuación, se describe de manera global cada una de estas fases:

Planificación Y Control De Las Pruebas: La primera etapa busca definir la planificación y alcance que tendrá el proceso de pruebas. Se determinará si se probara la totalidad del software o funcionalidades puntuales de este. Al igual, que las historias de usuario y criterio de aceptación que se cubrirán, el tipo de pruebas a realizar y Los riegos latentes que pudieran presentar durante el proceso de pruebas. De igual manera, Tanto el plan como alcance definido se deberá estar verificando, actualizando y ajustando regularmente. Con el fin, de ajustarlo a la realidad que está viviendo el proyecto.

Análisis y Diseño de Pruebas: los casos de prueba se diseñarán en base a los objetivos planteados inicialmente, realizando un análisis de riegos y de los requisitos funcionales planteados para el software. Se validará que funcionalidades y características deberán ser probadas, las condiciones iniciales de los ambientes o escenarios donde estos serán probados y los datos e información necesarios para ejecutar los casos de pruebas.

Implementación Y Ejecución De Pruebas: Los casos de pruebas deberán ser ejecutados acorde al plan detallado en fases previas, se deberá contar con un ambiente de pruebas configurado previamente y el cual contará con todos los prerrequisitos necesarios para poder ejecutar todos los casos sin inconveniente alguno. El principal objetivo de esta fase busca ejecutar todos los casos de prueba planificados y documentar comportamiento anómalo, error o defecto detectado durante la ejecución de estos.

Evaluación Y Reporte De Pruebas: Se analizan los errores o defectos encontrados y se realiza un plan de acción para la corrección de estos. En esta fase se podrían a llegar a involucrar varias áreas para diagnosticar y solucionar los inconvenientes presentados. Al igual que determinar si es necesario ampliar el plan de pruebas o centrarse en alguna funcionalidad, nivel o sección en puntual.

Actividades de Cierre de Pruebas: Se recopila toda la información obtenida y se evalúa si el plan de pruebas cumplió con los objetivos planteados en la fase de planificación. Se garantizará que el plan de pruebas se ejecutó satisfactoriamente y si todos los errores se solucionaron exitosamente. Por último, se deberá realizar un reporte con las actividades que se llevaron a cabo, los resultados finales, los defectos encontrados y las lecciones aprendidas durante el proceso.

Siguiendo estas etapas básicas en el proceso de pruebas de software, se aumentará considerablemente la detección de fallos o mal funcionamientos del aplicativo o software que este probando. Logrando a la organización a elevar los indicadores de calidad de los productos desarrolladores y mejorando la satisfacción y experiencia del usuario final.

Niveles de Pruebas de Calidad de Software

Las pruebas de calidad de software se dividen en varios niveles, donde cada uno busca enfocarse en un objetivo puntual, para abarcar todas las aristas que se pueden presentar durante el testeo de calidad. A continuación, se presentarán los niveles de pruebas más importantes y significativos:

Pruebas Unitarias: Se tienden a realizar durante la fase de codificación del aplicativo o software y sueles ser desarrolladas por el grupo de desarrollado, en muchas ocasiones se realiza de manera cruzada, probando los desarrollos ente miembros del mismo equipo de desarrollo. Estas buscan descubrir errores tempranos de codificación y basado en la experiencia de miembros más experimentados o en lecciones aprendidas obtenidas con anterioridad se busca analizar las secciones de código donde más suelen presentarse errores o inconvenientes.

Pruebas De Componentes: Busca validar que un componente o modulo funcione correctamente de forma individual. pero al mismo tiempo, si este tiene que interactuar con otro componente su funcionamiento sea satisfactorio y no presente fallos o errores (Mera Paz, 2016).

Pruebas De Integración: Guarda gran relación con la prueba de componente, ya que en este nivel se valida que las interfaces, protocolos de comunicación y flujo de datos entre dichos componentes o módulos funcionen de forma correcta, fluida y sin errores.

Pruebas De Sistema: Se evalúa el sistema o aplicativo como un todo, al igual que se debe contemplar si este tendrá algún tipo de interacción o interfaz con un componente externo y probar la funcionalidad completa en un entorno lo más parecido posible a producción. En este nivel se deberán validar aspectos claves tales como la concurrencia, el rendimiento y la seguridad.

Pruebas De Aceptación: Se valida con el cliente o usuario final que el software cumpla con sus expectativas y necesidades del usuario, al igual que con los criterios de aceptación establecidos en las historias de usuario funcionales.

Pruebas de Regresión: Valida que los cambios recientes o nuevos, introducidos a raíz del despliegue de nueva funcionalidad, mejora o corrección. no hayan generado defectos en áreas del software que no han sido modificadas y que funcionaban correctamente (Naik & Tripathy, 2008).

Pruebas De Backup/Restauración: Valida los planes de contingencia y recuperación de información en casos donde se presente un desastre o ataque cibernético. Al igual, que la consistencia y correcta distribución de la información.

Automatización de Pruebas

La automatización de pruebas de software tiene como objetivo principal ejecutar pruebas funcionales de la manera más ágil y eficaz posible, haciendo uso de herramientas y scripts para la automatización de la ejecución de los casos de pruebas. Ayudando de gran manera a los tester y analistas funcionales en tareas repetitivas o que aportan poco valor al proceso y permitiéndoles centrarse en tareas más relevantes, tales como plantear más escenarios y casos de pruebas para abarcar una mayor cantidad de casuísticas y funcionalidades que monitorear. Es por estos beneficios, que nos ofrece la automatización de pruebas que cada vez más organizaciones y empresas del sector desena incorporarla a sus procesos y líneas de trabajo. Aunque las ventajas que nos ofrecen son gigantes, también hay que considerar que su implementación conlleva retos importantes, como una interiorización integral de la organización, personal con la pericia y conocimiento del negocio para generar automatizaciones acordes a los requerimientos y necesidades del usuario. De igual manera, esta implementación también acarrea gastos tantos físicos como monetario. Ya que como hacer referencia Kaner, el costo estimado en crear una prueba automatizada es equivalente a diez veces el costo de realizar la prueba manualmente (Kaner, Bach, & Pettichord, 2002)

Herramientas de Automatización

Dado el gran auge y el naciente entusiasmo por la implantación de automatización de pruebas de software, dentro de la industria se han desarrollado varias herramientas y frameworks que facilitan la creación, ejecución y administración de pruebas automatizadas. Además, que le proporciona al especialista funcional o tester *“una base común con herramientas, bibliotecas y directrices que ayudan a estructurar y organizar las actividades de*

prueba de manera eficiente y consistente" (Desikan & Ramesh, 2009). de allí la importancia, no solo de escoger una adecuada estrategia de automatización sino de escoger las herramientas y framework adecuadas. Por lo que a continuación, se presentan algunas de las herramientas de automatización de software más comunes y populares utilizadas en el mercado:

Selenium: Es uno de los frameworks más populares para la automatización de casos de pruebas. Permitiendo ser compatible con infinidad de aplicativos y componentes dado que es multiplataforma y pudiendo funcionar sobre cualquier sistema operativo. Además de esto nos proporciona la opción de crear scripts de automatización en varios lenguajes de programación como Java, C#, Python, Ruby, JavaScript. Entre sus principales características nos permite crear, modificar y depurar casos de prueba, que después podrán ejecutarse automática e iterativamente (Selenium Developers Group, 2024).

Junit: Es un framework de código abierto enfocado en pruebas unitarias de aplicativos desarrollados en Java y el cual es generalmente usado para escribir y ejecutar pruebas de unidad de código. Este contiene varias clases que ayudan con la generación y codificación de casos de pruebas, Al igual que permite la ejecución y validación de los resultados generados por este. Una de las principales razones de su éxito consiste en su incorporación y accesibilidad desde dos de los más grandes IDE de desarrollos de Java, como lo son Eclipse y NetBeans.

Testium: Plataforma avanzada de automatización de pruebas basada en inteligencia artificial (IA) y machine learning (ML). Está diseñada para simplificar y acelerar la creación,

ejecución y mantenimiento de pruebas automatizadas, abordando algunos de los desafíos más comunes en el testeo de software, como la fragilidad de los tests y el tiempo de mantenimiento.

Appium: Herramienta para automatizar pruebas de aplicaciones móviles. Permite a los testers y desarrolladores escribir pruebas para aplicaciones nativas, híbridas y móviles web en Android y iOS. Una de las características más destacadas de Appium es su capacidad para utilizar el mismo código de prueba para diferentes plataformas móviles. Appium puede integrarse con frameworks de machine learning para mejorar las pruebas automatizadas, por ejemplo: Generación de casos de pruebas, optimización de pruebas y pruebas visuales

SonarQube: Plataforma de código abierto para la inspección continua de la calidad del código, proporcionando análisis estático del código para identificar bugs, vulnerabilidades de seguridad, y problemas de mantenimiento del código. Se usa mucho en la industria del software para garantizar que el código sea de alta calidad y ayudar a los desarrolladores a mejorar continuamente sus prácticas de codificación.

SoapUI: Es una herramienta de pruebas para aplicaciones con arquitectura orientada a servicios APIs SOAP y REST. Gracias a su interfaz gráfica fácil e intuitiva permite escribir, ejecutar y modificar pruebas de API y servicios web de manera automatizada.

Revisión del Estado del Arte

El aprendizaje automático o machine learning (ML) es una rama fundamental de uno de los campos que en la actualidad se encuentran en auge, tal como es la inteligencia artificial. La cual revoluciona cómo las aplicaciones se usan en varios ámbitos de la vida cotidiana y también en la forma de desarrollar y testear. al igual, que los retos y oportunidades nacientes que el uso de nuevas tecnologías conlleva para el desarrollo de software, donde su uso se ha incrementado exponencialmente y dado que el mercado y usuarios exigen un ritmo más acelerado para desarrollar aplicaciones y actualizarlas. Esto nos hace plantear un nuevo paradigma, ya que no solo nos hace preguntar cómo acoplar la inteligencia artificial para mejorar los tiempos y calidad de software, sino que nos hace implementar esquemas para que el proceso de pruebas sea cada vez más eficaz, rápido, riguroso y confiable, para ir a la par de las necesidades y velocidad en la implementación de software actuales (Maisueche Cuadrado, 2019). Tal como plantea Ígor Kirilenko aunque muchas de estas tareas ya son implementadas por herramientas de automatización, que no necesariamente hace uso de inteligencia artificial o aprendizaje automático, estas cuentan con ciertas limitaciones y es allí donde el uso de la inteligencia artificial busca superar y eliminar estas, para brindar un mayor valor tanto a los equipos de desarrollos como a los especialistas funcionales, reduciendo su participación en tareas repetitivas y análisis de grandes volúmenes de datos dentro del proceso de ejecución de ciclos de pruebas (Kirilenko, 2024)

Los primeros intentos de aplicar técnicas de aprendizaje automático a las pruebas software se dieron en la década de los años 90, se centraron en la generación automática de casos de prueba. Se exploraron métodos como la generación de casos de prueba basada en gramáticas y la cobertura de código utilizando algoritmos genéticos. Durante este tiempo, si

bien se exploraron algunas técnicas prometedoras, el uso práctico del aprendizaje automático en las pruebas de software estaba en una etapa inicial y no alcanzaba la amplitud y sofisticación de los enfoques actuales. Sin embargo, estos primeros trabajos sentaron las bases para el desarrollo posterior de técnicas más avanzadas de aprendizaje automático en el campo de las pruebas de software (Clark & Jacob, 1994).

En la Década de los 2000 se produce un mayor interés en la aplicación de técnicas de aprendizaje automático para la predicción de defectos y la evaluación de la calidad del software. Se utilizan algoritmos de clasificación y regresión tanto para identificar áreas de código propensas a errores como vulnerabilidades de seguridad y con ello lograr realizar una medición y cuantificar la calidad del software (Menzies, Greenwald, & Frank, 2006).

Sobre el año 2010 Se producen avances significativos en el uso de técnicas de aprendizaje automático para la detección de anomalías en el comportamiento del software y la generación automática de casos de prueba basados en técnicas de aprendizaje profundo (Xie, y otros, 2020).

A partir de la década de 2020 Se han desarrollado herramientas y plataformas que integran capacidades de aprendizaje automático para mejorar la eficiencia y la efectividad de las pruebas de software. Estas herramientas permiten a expertos de datos y los ingenieros de pruebas aprovechar algoritmos de aprendizaje automático, sin necesidad de conocimientos especializados en el campo. En esta década, el aprendizaje automático ha pasado de ser una técnica experimental a una herramienta práctica y muy utilizada en las pruebas de software. Los avances continuos en este campo prometen ser un salto significativo en la mejora de la

calidad y eficiencia de las pruebas de software en el futuro (Amalfitano, Fasolino, Tramontana, & De Carmine, 2020). A continuación, se mencionan los avances más destacados durante esta década:

- En el 2020, OpenAI creó el algoritmo de inteligencia artificial GPT-3, el cual permitía la generación de textos, lo que inmediatamente fue aprovechado por la industria del desarrollo de software y fue integrado a los procesos de prueba para la generación de casos y guiones de prueba. Al igual que permitió ampliar y tener en cuenta nuevos escenarios a los que se podría enfrentar un desarrollo u aplicativo nuevo (Jacinto, 2022).

- En el mismo año, Zapier ve un aumento exponencial en el uso de herramientas de desarrollo de aprendizaje automático tal como son AutoML de Google, SageMaker de Amazon y Azure ML de Microsoft. Permitiendo a usuarios con poca o nula experiencia, generar y entrenar modelos de aprendizaje automático para el desarrollo y testeo de aplicaciones, optimizando y automatizando los procesos de pruebas debido a los tiempos cada más cortos para el despliegue de aplicaciones y generar costos mínimos o nulos para las empresas (Zapier Editorial Team, 2022).

- Amazon DevOps Guru fue anunciado en 2020 y el cual facilito de gran manera el monitoreo y detección de anomalías y comportamiento irregulares en tiempo real de la aplicación. De igual manera, con la recopilación y análisis automática de datos es capaz de alertar a los grupos de desarrollos y testers sobre potenciales interrupciones y sugerencias de reparación.

- En 2023, con el auge de los LLM o Grandes Modelos Lingüísticos y con la masificación en el público en general de herramientas como ChatGPT y DALL-E, también se impulsó de gran manera los sistemas de gestión documental y de conocimiento, basados en aprendizaje automático y en donde apoyados en patrones de comportamiento y datos de rendimiento, se puede generar bases de conocimiento e incentivar la transferencia de conocimiento dentro de los grupos de desarrollo y testeo.

Un artículo de Vladimir Berrio (Berrio, 2024), describe como la aplicación de la inteligencia Artificial en el proceso de testing está transformando el panorama del desarrollo de software y donde grandes lideres de la industria tales como Google, Microsoft y Facebook están a la vanguardia de esta transformación, integrando la Inteligencia Artificial para mejorar la eficiencia, la precisión y la seguridad en sus procesos de testing. También indica que medida que la tecnología avanza, podemos esperar aún más innovación en este campo, lo que conducirá a una mejora continua en la calidad del software y una mayor satisfacción del usuario final.

Berrio también resalta que la implementación de prácticas de inteligencia artificial (IA) en el testing de software ya no es exclusiva de grandes empresas tecnológicas. A pesar de que estas compañías fueron pioneras en este ámbito, hoy en día existen herramientas y técnicas basadas en Inteligencia Artificial a precios razonables disponibles para una amplia gama de organizaciones, independientemente de su tamaño o presupuesto. El crecimiento del mercado de Inteligencia Artificial, que se ha triplicado en los últimos 5 años, en busca de impulsar una transformación digital dentro sus organizaciones y la amplia variedad de soluciones disponibles

tales como permiten a empresas de todos los tamaños encontrar opciones adecuadas para mejorar la calidad y eficiencia de su proceso de testing de software.

Natasha Sharygina y Doron Peled, Señalan que el testeo manual es más exhaustivo pero limitado a sistemas de estado finito, lo que puede llevar a la omisión de algunos errores e incluso estar sujeto a la generación de nuevos inconvenientes dados por errores humanos. Por lo tanto, se propone metodologías de aplicación de machine learning para mejorar la verificación del software mediante la realización de algunas comprobaciones de consistencia entre el código fuente final y los criterios de aceptación establecidos en las historias de usuario, específicamente mediante la generación de casos y guiones de pruebas a contemplar en el ciclo de vida de pruebas (Sharygina & Peled, 2001).

Dichos factores pueden llegar a ser controlados e incluso llegarse mitigar con el uso de machine learning, dado que muchas de las tareas repetitivas o inoficiosas realizadas por los tester o especialistas funcionales pueden llegar a ser automatizadas y optimizadas tales como la generación de guiones y planes de pruebas, planteamiento de escenarios de pruebas, análisis de los datos resultantes o el mantenimiento a esquemas de pruebas tradicionales. De igual manera, como plantea Eduardo Hoe, se puede llegar a lograr que el aplicativo o software logre adaptarse mejor a los factores cambiantes de su entorno basándose en patrones de comportamiento y datos históricos, reduciendo la intervención humana y el esfuerzo manual realizados por los tester o analistas funcionales permitiéndolos centrarse en las áreas con mayor riesgo de fallo (Hoe, 2023). Sin embargo, aunque esto podría llegar a plantear una reducción en la intervención de los tester o disminuir el papel protagónico que ocupan en la ejecución de los planes de pruebas. Esto podría varias, ya que el rol de ellos transmutará y se

convertirá en una relación simbiótica. dado que serán estos, los encargados de entrenar los algoritmos de machine learning y generar el grupo de datos de entrenamiento para permitir que estos sean capaces de forma autónoma de tomar dediciones, realizar predicciones o realizar una tarea repetitiva, sin la necesidad que exista una programación explicita (Kosowski, 2023). Es en este contexto, que se presentan nuevas perspectivas en la implementación del aprendizaje automático para revolucionar las metodologías y esquemas de pruebas tradicionales, mejorando su eficiencia y rendimiento. Con el fin de llevar a cabo este objetivo, la implementación de machine learning en procesos de pruebas, suelen plantear algunas soluciones generales a problemas comunes presentados con regularidad en las metodologías de pruebas tradicionales, tales como:

Generación Automática de Casos de Prueba: Mediante el adecuado entrenamiento y uso de algoritmos de aprendizaje supervisado, no supervisado o semisupervisado, el modelo de machine learning tendrá la capacidad de identificar patrones en el código y analizar los requisitos funcionales del sistema, para generar casos de prueba automáticamente. De igual manera, podrá plantear una amplia gama de posibles escenarios y comportamientos a la que se pueda ver sometido el software (Beta Arrays, 2023).

Detección de Defectos y Anomalías: dada la capacidad de los modelos de machine learning para procesar y analizar grandes volúmenes de datos de pruebas y comparar datos históricos de ejecuciones previas. Es capaz de detectar comportamientos anómalos que podrían repercutir en errores, defectos en el sistema o fallos de seguridad (Navas Ajenjo, 2020).

Optimización de Pruebas: Haciendo uso de técnicas de aprendizaje por refuerzo, el modelo será capaz de seleccionar y priorizar los casos de prueba más críticos. Para enfocar tanto tiempo como recursos en probar estos exhaustivamente y disminuir la atención prestada a casos de prueba sin peso o relevancia en el resultado final (Neora, 2022).

Predicción de Fallos: Se pueden generar modelos predictivos basados en Machine Learning para predecir posibles fallos en el software. Utilizando datos históricos de pruebas y ejecución, permitiendo al algoritmo predecir secciones de código donde se podrían presentar errores o presentar vulnerabilidades de seguridad. Para que el equipo de desarrollo le preste atención al modificarlas o prevenir errores de este tipo (Beta Arrays, 2023).

Análisis de Impacto de Cambios: Realizando un análisis del historial de modificaciones y de bugs o inconvenientes reportados. El modelo podrá analizar el comportamiento del software, sugiriendo pruebas o escenarios adicionales, asegurando que todas las casuísticas y posibles entradas se analicen mitigando los posibles fallos que se puedan presentar.

Desafíos y Retos en la Implementación de Machine Learning

Como se ha analizado previamente la implementación de machine learning (ML) o aprendizaje automático en el proceso de pruebas de software representa una oportunidad sin igual y que supone una revolución importante a las metodologías tradicionales de pruebas. Sin embargo, la integración de este a sistemas maduros o demasiados complejos suele suponer

una serie de desafíos y que en muchos casos las compañías u organizaciones no están disponibles a afrontar. Dado que las fases iniciales de la implantación suelen ser las más complejas de encarar. Principalmente, porque no se suele tener un punto de partida claro. Además, que los prerrequisitos iniciales suelen ser exigentes y no se suele contar con el personal indicado y la asesoría de expertos en el tema.

Dada la gran repercusión que en la actualidad tiene el concepto de inteligencia artificial y machine learning en el mercado, donde es constantemente mencionado en varios medios. Las empresas desean vehementemente implementarlos dentro de sus organizaciones e integrarlos dentro de sus procesos, sin antes plantearse, si cuenta con los recursos y cultura necesarios para implantar un modelo de aprendizaje automático. Lo que conlleva que cuando inicia la implementación se encuentren con barreras que a veces son desalentadoras y que en algunos casos hacen desistir a las organizaciones de integrarlo a sus procesos. Dentro de los principales retos iniciales con los que se topan las empresas podemos encontrar:

- Se deben encontrar, organizar y estandarizar los datos, al igual que estos deben tener un alto grado de calidad.
- Se debe contar con los recursos físicos y hardware para manejar las cargas de trabajo.
- Se requiere asesoramiento de expertos y Científicos de datos para crear y entrenar los algoritmos.
- Los productos más robustos y antiguos de la organización también deberán integrarse en los procesos del negocio.

- Los costos iniciales suelen ser elevados y el retorno de inversión puede tardar en reflejarse y muchas veces no se cuantificar totalmente.
- La organización se debe acoplar al cambio y debe haber una adopción cultural al modelo

Sin embargo, como hace referencia Larry Pizette, gerente senior y líder del laboratorio de soluciones de machine learning (ML) en Amazon Web Services "*Comenzar implica solo seguir tres pasos: aprender sobre Machine Learning, crear ideas basadas en los casos de uso y los enfoques para validar las ideas, e implementar una o dos pruebas de conceptos de alto valor*" (Pizette, 2018). Lo que nos lleva a analizar que la implementación de un modelo de machine learning va más allá de los desafíos tecnológicos o técnicos que se puedan llegar a presentar. Ya que la organización y sus principales líderes deben pasar por un proceso de comprensión, estudio e interiorización sobre el aprendizaje automático y la importancia que tendrá para la organización. Igual que liberarse de preconceptos previos y empezar a ser más creativos y generar ideas nuevas y revolucionarias que antes no se habían aplicado o se podían imaginar que podían aplicarse en sus organizaciones. También es necesario recalcar la realización de pruebas de concepto (PoC) y la generación de prototipos para evidenciar y mostrar avances a las organizaciones y con ello integrar a la organización, motivar y evidenciar los beneficios y oportunidades que podría traer la implementación de un modelo de aprendizaje automático.

Aplicaciones de Machine Learning

El uso de Machine Learning (ML) ha supuesto una revolución en el proceso y ciclo de pruebas de calidad de software. Al igual, que en las estrategias y técnicas para detectar, prevenir y corregir errores. Álvaro Fernández Villar indica que, a medida que el código fuente crece, el tamaño y complejidad de los sets de pruebas aumentan según se deben cubrir nuevos escenarios y casuísticas que se pueden presentar. De igual manera, se dificulta establecer criterios para definir que escenarios priorizar o tienen más peso. Para dar solución a estos problemas, es que se construyen modelos y soluciones basadas en aprendizaje automático los cuales buscan analizar el estado de las pruebas, detectar cambios recientes en el código fuente, analizar los resultados para establecer métricas y planes de ejecución a futuro (Fernández Villar, 2023). A continuación, se describirá brevemente las aplicaciones más significativas e importantes del machine learning en el proceso de calidad y pruebas de software. Acompañado de ejemplos y casos de éxito de cómo está siendo integrado por organizaciones y empresas lideres de la industria:

Detección de Errores y Defectos: la detección de anomalías es una de las principales aplicabilidades del uso del machine learning en el proceso de pruebas de calidad de software. Según una muestra de datos de entrenamiento o en el histórico de datos suministrado, el modelo de aprendizaje automático podrá detectar anomalías o comportamientos inusuales que podrían indicar errores, fallas o vulnerabilidades con mayor rapidez y en tiempo real. Un ejemplo de esta aplicación en la industria es implementado por Netflix en donde hacen uso del machine learning para analizar los logs de uso de comportamiento de sus usuarios de streaming, para detectar de manera rápida y eficaz comportamientos anómalos en la

plataforma y darle una pronta respuesta a posibles errores o problemas presentados (Ekanadham, 2018).

Predicción de Fallos y Errores: mediante el uso de aprendizaje automático es posible la realización de predicción de errores y análisis de comportamientos de las células o grupos de codificación, basándose en el historial de cambios de los repositorios es posible inferir practicas comunes en la codificación del código fuente. Al igual, que determinar cuáles son las áreas de código que son más sensibles de sufrir modificaciones, permitiendo generar recomendación tanto para prevenir futuros errores sino también consejos en lo que respecta a escalabilidad y eficiencia. Tal como describen Titus Winters y Hyrum Wright antiguos trabajadores de Google, dentro del gigante de tecnología se hace uso de redes neuronales y técnicas de procesamiento de lenguaje natural (NLP) con el fin de entender el contexto de cada uno de los cambios realizados, en busca de hallar potenciales vulnerabilidades y generar sugerencias y planes de acción frente a estos (Winters, Manshreck, & Wright, 2022).

Generación Automática de Casos de Prueba: El modelo de machine learning haciendo uso de técnicas como Árboles de Decisión o Clustering (k-means, DBSCAN) es capaz de generar casos de prueba de manera eficiente y eficaz. Abarcando una gran cantidad de escenarios y casuísticas, para garantizar que la totalidad de la funcionalidad y módulos sea probada. Con ello, se puede reducir los costos y esfuerzos humanos, mejorar la calidad del software y la cobertura del plan de pruebas. Microsoft al ser una de las empresas de la industria que más herramientas y software crea y genera actualizaciones necesita un proceso de pruebas continuo y eficaz para garantizar la calidad y correcto funcionamiento de sus aplicativos. De allí, que implemente algoritmos de machine learning para analizar el código y

los datos históricos de pruebas previas, generando casos de pruebas automáticos que se enfoquen en sectores de código o funcionalidad que presenten un gran porcentaje de fallos con la finalidad de tener una mayor cobertura y eficiencia de las pruebas (Amershi, y otros, 2019).

Optimización y Priorización de Pruebas: la adecuada selección de casos y la priorización de estos, para darle un mayor peso de pruebas a aquellos que puedan llegar a tener más impacto en el aplicativo a largo plazo, es un proceso crucial para determinar el exitoso en la ejecución un plan de pruebas. De allí que, con el uso de redes neuronales, algoritmos de regresión y Aprendizaje por Refuerzo se puedan plantear varias estrategias para priorizar y darle mas importancia a los casos que puedan llegar a tener mas relevancia en el resultado final y tanto en los niveles de calidad como en la disminución de errores o inconvenientes del aplicativo y tiempos en la ejecución de los esquemas de pruebas. dentro de estas estrategias se pueden plantear el análisis histórico de ejecuciones previas y clasificarlos en base a las similitudes de estas, cobertura del código, coste y tiempos de ejecución (Sánchez, Segura, & Ruiz-Cortés, 2013).

Optimización del Proceso de Pruebas: Haciendo uso de algoritmos de machine learning se puede lograr a ampliar la cobertura del plan de pruebas, la optimización de recursos y mejorar la eficiencia y eficacia en la ejecución de los casos de pruebas ejecutados. De igual manera, se pueden llegar a priorizar mediante el uso de modelos de regresión los casos de pruebas que tienen más importancia o definir que escenarios o secciones de código tienen más peso en la calidad final del producto, basados en los datos de entrenamiento, redes neurales y algoritmos predictivos (Random Forest). Gigantes de la industria como son Google y SAP para realizar la priorizar casos de pruebas y automatizar las pruebas de regresión. Por ejemplo,

hace uso de técnicas de machine learning para optimizar la ejecución de pruebas en su ciclo de integración continua, seleccionando los casos de prueba más relevantes. Logrando con ello la reducción de tiempos de ejecución y aumentando exponencialmente el porcentaje de detección temprana de errores y defectos (Google Developer Program, 2023).

Optimización del Código Fuente: Mediante la combinación de reglas predefinidas de codificación y el entrenamiento de un modelo de Aprendizaje Supervisado es posible ajustar, dinamizar y retroalimentación dichas reglas permitiendo generar un esquema robusto y actualizable para el análisis de código fuente y generando planes de optimización y refactorización de código con el fin de aumentar y flexibilidad. Empresas como BMW han implementado herramientas como SonarQube en sus sistemas de software para vehículos conectados y autónomos con el fin de garantizar que su software se lo más seguro y fiable posible (Wagner, 2012).

Análisis de Sentimiento de Revisión de Código (Sentiment Analysis): Hace referencia a la aplicación de técnicas de procesamiento del lenguaje natural (NLP) y machine learning (ML) para evaluar el ambiente y opiniones presentadas en la revisión de código, permitiendo evaluar el comportamiento e interacciones presentadas entre los desarrolladores y miembros del equipo, Con el fin de realizar una mejor retroalimentación y fomentar una mejor colaboración en los equipos de desarrollo. Una de las aplicaciones más conocida dentro de la industria la realiza Airbnb, quien usa técnicas de machine learning para proporcionar una mejor retroalimentación a los desarrolladores y lideres de proyecto, mejorando el ambiente laboral del proyecto y reduciendo los conflictos internos que se puedan llegar a presentar (Alharbi, 2023).

Automatización de Pruebas Funcionales y de Regresión: la automatización de pruebas funcionales son un punto crítico para mejorar la eficiencia del proceso de pruebas de calidad y regresión, reduciendo el tiempo invertido en la ejecución del plan de pruebas y garantizando el cumplimiento de los requisitos funcionales y casos de pruebas planteados. A la vez, garantizando que las actualizaciones realizadas no introducirán nuevos defectos a secciones o módulos no modificadas. Facebook al requerir probar continuamente una gran cantidad de funcionalidades tanto en sus aplicativos webs como móviles, realiza la automatización de pruebas funcionales y de regresión, integrando herramientas como Selenium y Appium con su pipeline de integración continua (CI/CD). obteniendo una mejor cobertura en su plan de pruebas y reducido el tiempo necesario para ejecutar regresiones (Riggins, 2019).

Análisis Predictivo de Logs: Haciendo uso de distintos algoritmos predictivos de machine learning tal como regresiones lineales, árboles de decisión y bosques aleatorios (*random forests*), es posible identificar patrones y distintos comportamientos en grandes volúmenes tal como lo son logs o registros de salida de aplicativos, permitiendo entrenar el algoritmo para que identifique el comportamiento normal del aplicativo o software, permitiendo detectar cuando este se sale del comportamiento esperado o presenta resultados anómalos o fuera de los esperados. de igual manera, el algoritmo puede ser entrenado para que genere alertamientos y posibles causas cuando identifique un error o un evento que este fuera de las métricas. En el mercado ya existen herramientas y plataformas como es el caso de Splunk que es usado por empresas de talla mundial como Cisco, IBM y Nasdaq para el análisis y monitoreo de grandes volúmenes de datos y logs de salida (Fernandez, 2024).

Automatización de Pruebas UI (Interfaz de Usuario): implementando Redes Neuronales Convolucionales (CNN) se puede realizar la identificación y clasificación de componentes de una interfaz tal como lo son menús, campos y botones, permitiendo definir y clasificar la interacción y comportamiento del usuario para de esta manera generar o actualizar los casos de prueba para que se ajusten y acerquen lo más posible a la interacción que pueda tener el usuario final con el aplicativo. Empresas como Tesla han empezado a implementar dichos algoritmos por medio de herramientas como Test.AI para mejorar la calidad de su interfaz mediante la automatización de pruebas en donde se prioriza las funcionalidades que tienden a ser más usadas por los usuarios (Testers.AI, 2024).

Pruebas de Seguridad Automatizadas: Dado los crecientes retos que enfrentan las organizaciones y empresas respecto a la seguridad y vulnerabilidad de sus aplicativos e información, se ha generado la necesidad de implementar nuevos esquemas y métodos que sean capaces de adaptarse y aprender de los ataques o posibles brechas a los cuales se puedan enfrentar. De allí, que el machine learning cobre una gran importancia debido a que, con el adecuado entrenamiento, el algoritmo podrá estar en la capacidad de identificar vulnerabilidades y grietas que puedan llegar aprovechadas por un externo. De allí, que aplicativos como Snyk que son usados por empresas como Spotify haya integrado herramientas basadas en aprendizaje automático que analizan, aprenden y son capaces de adaptarse en base a patrones de comportamiento y datos históricos (Piper, 2022).

Adaptación Continua de Pruebas: Mediante el uso de algoritmos de clustering y análisis predictivo, es posible detectar comportamientos en el uso de la aplicación por parte del usuario final y adaptarse a estos. Google Maps constantemente está recopilando y analizando

datos de uso sobre la aplicación, permitiendo enfocar los escenarios de pruebas en las funcionalidades que puedan llegar a ser más usadas, asegurando que las pruebas sigan siendo relevantes y efectivas a lo largo del tiempo. De igual manera, se analiza las preferencias del usuario para optimizar la calidad de resultados consultados y detectando los reseñas o lugares inconsistentes, permitiendo detectar posibles engaños o errores en la aplicación (Dinukamarlon, 2023).

Automatización de Pruebas de Rendimiento: el machine learning ha comprobado ser una herramienta sumamente poderosa para optimizar y mejorar el código fuente. La igual que las maneras en que este se testea, pero sus beneficios no llegan solo hasta ahí dado que los distintos algoritmos de aprendizaje automático también pueden ser usados para la optimización de redes y tiempos de respuesta a nivel de infraestructura del sistema, recopilando datos históricos y analizándolos para mejorar el volumen del tráfico, encontrar interrupciones o detectar cuellos de botellas en la respuesta de la red (Red Hat, 2024).

Automatización de Pruebas de Accesibilidad: haciendo uso de árboles de decisión, redes neurales y algoritmos de clasificación como K-Beans, se puede llegar a automatizar la detección de problemas de accesibilidad o de ingreso tanto en aplicativos móviles como sitios web. Herramientas como AXE implementada por Deque Systems es usada empresas como Duolingo o HubSpot, los cuales la usan no solo para detectar posibles fallos en el ingreso de sus plataformas, sino que buscan hacer sus sitios web y aplicaciones mas amigables con el usuario y mejorar de manera inclusiva con aquellos clientes que puedan llegar a tener algún tipo de discapacidad (Deque Systems, 2024).

Automatización de Pruebas de API: con la creciente necesidad de integrar diferentes aplicativos que pueden llegar a pertenecer a varias empresas o proveedores, se ha acrecentado la necesidad de tener métodos que garanticen la fiabilidad y eficiencia en la integración u orquestación entre dichos aplicativos. De allí, que herramientas como Postman han empezado a implementar técnicas de machine learning tales como algoritmos de aprendizaje supervisado y clustering. Al igual que modelos predictivos para simular escenarios reales de pruebas y con ello generar casos automáticos de pruebas, que puedan conllevar a la automatización de pruebas de regresión cuando uno de los aplicativos que hace parte de la integración sufra cambios. De igual manera, es capaz de detectar anomalías en el rendimiento y generar alertamientos o informes cuando una de estas se presente o presente tiempos excedidos en la respuesta (Postman, 2024).

Herramientas y Software para el Desarrollo de Machine Learning

Herramientas para el Preprocesamiento de Datos

Pandas: Es una biblioteca de software para el lenguaje de programación Python utilizada para el procesamiento y análisis de datos. Cuenta con varias funciones para la manipulación, transformación y análisis de datos. Haciendo uso de diferentes métodos para seleccionar, agrupar y filtrar grandes conjuntos de datos de manera eficiente (Quiroga Saldaña, 2023).

NumPy (Numerical Python): Una librería de Python para la computación científica con soporte para matrices multidimensionales y operaciones matemáticas de alto rendimiento. es

ampliamente utilizado en el manejo y preprocesamiento de grandes volúmenes de datos y en aplicaciones o software que requieren cálculos numéricos de alta complejidad, lo que la convierte en la base de muchas aplicaciones científicas y de investigación académica (Cardellino, 2021).

Herramientas de Visualización

Matplotlib: Una librería de Python utilizada para la creación de gráficos estáticos, gráficos interactivos y visualizaciones de datos en dos dimensiones. Su capacidad para generar una amplia variedad de gráficos, junto con su alto nivel de personalización. Suele usarse en distintos campos de investigación y ciencia de datos, por la gama de gráficos que puede generar, desde los más básicos hasta gráficos de gran complejidad y alta calidad. Su fácil integración con herramientas como NumPy, Pandas y Jupyter Notebooks la hacen una de las más usadas en investigaciones académicas (Trespaderne & Mazaeda Echevarría, 2020).

Seaborn: Es una biblioteca de visualización de datos basada en Matplotlib con el agregado que proporciona una interfaz más robusta, la cual facilita la generación de gráficos estadísticos. Al igual que Matplotlib, cuenta con integración con estructuras de datos de pandas, facilitando la generación de gráficos estadísticos y de análisis exploratorios complejos con una cantidad mínima de código (Carmona, 2020).

Plotly: Una biblioteca de gráficos interactivos para Python, R y otros lenguajes de programación que permite la creación de visualización de datos de alta calidad. Se usa

principalmente en aplicaciones web y análisis de datos en tiempo real, gracias a su capacidad de generar gráficos interactivos y personalizables (Zapata, 2023).

Herramientas de Implementación y Producción

TensorFlow Serving: Es una plataforma de alto rendimiento desarrollada por Google para el despliegue de modelos de machine learning en entornos de producción de manera fácil y eficiente. Destaca por su arquitectura, capaz de manejar muchas solicitudes concurrentemente. Además, que permite gestionar múltiples versiones de un modelo, lo que repercute en que se pueden desplegar con transparencia y sin ningún tipo de afectación para los usuarios finales.

MLflow: Una plataforma de código abierta dedicada a la gestión del ciclo de vida del desarrollo de modelos de machine learning. Proporciona herramientas para el seguimiento de experimentos, gestión y despliegue de modelos, empaquetado de proyectos e implementación de flujos de trabajo. Los cuales son ideales para equipos académicos y de ciencia de datos, que buscan mejorar la eficiencia y la colaboración en sus flujos de trabajo de Machine Learning.

Kubeflow: Un proyecto de código abierto diseñado para facilitar la implementación y gestión de flujos de trabajo de aprendizaje automático en Kubernetes. Su objetivo es simplificar el proceso de despliegue de modelos de machine learning en producción, brindando una

interfaz de fácil uso y componentes que permiten a los grupos de trabajo de ciencia de datos crear, entrenar y desplegar modelos de manera eficiente.

Herramientas de Anotación de Datos

Labelbox: Una plataforma para la anotación y gestión de datos de entrenamiento de inteligencia artificial (IA), La cual está diseñada para ayudar a los equipos de ciencia de datos y desarrolladores a construir modelos de machine learning de alta complejidad, simplificando el proceso de anotación, gestión y mejora de múltiples conjuntos de datos, reduciendo el esfuerzo manual requerido y acelerando el proceso de desarrollo.

SuperAnnotate: Una plataforma de anotación de datos diseñada para gestionar y optimizar el proceso de anotación de datos de entrenamiento en proyectos de inteligencia artificial (IA) y aprendizaje automático. Entre sus principales características, cuenta con herramientas de anotación avanzadas, automatización y asistencia por IA, garantizando la calidad de los datos anotados y reduciendo el tiempo y esfuerzo invertido en el proceso de anotación. También tiene la particularidad de anotar imágenes, videos y datos en 3D.

Entornos de Desarrollo Integrado (IDE)

PyCharm: Es un entorno de desarrollo integrado (IDE) para Python desarrollado por JetBrains, que brinda una amplia gama de herramientas, como lo son la depuración avanzada, autocompletado de código, integración con sistemas de control de versiones y herramientas de

base de datos. De igual manera, cuenta con un depurador visual potente y herramientas integradas para realizar el análisis, depuración y refactorización de código.

Visual Studio Code (VS Code): Un editor de código abierto desarrollado por Microsoft, altamente extensible y personalizable, el cual soporta múltiples lenguajes de programación y cuenta con una ampla biblioteca de extensiones ampliando las funcionalidad y funciones del editor. Cuenta con autocompletado de código, depuración avanzada, integración con sistemas de control de versiones y distintas herramientas como Docker, Kubernetes y Azure.

Técnicas y Herramientas de Machine Learning más Convenientes para el Testeo de Software

Las técnicas y herramientas de machine learning emergen como una alternativa prometedora para mejorar el proceso de pruebas de software. Estas tienen la capacidad de aprender patrones y comportamientos a partir de los datos de prueba, lo que permite generar casos de prueba más efectivos y optimizar el proceso de manera iterativa.

Se determinan algunas de las técnicas y herramientas de machine learning más convenientes para el testeo de software, analizando sus fortalezas, aplicaciones y beneficios en el contexto de esta importante actividad del ciclo de vida del desarrollo de software.

Herramientas más Convenientes Para el Testeo de Software

Appium

Appium es una herramienta de automatización de código abierto, la cual tiene como principal objetivo automatizar el proceso y plan de pruebas tanto de aplicaciones móviles como de escritorio. Al tener soporte multiplataforma permite a los equipos de testers y desarrollo generar guiones, casos de uso y planes de pruebas para aplicaciones nativas, híbridas y móviles tanto en Android como en iOS. De igual manera, nos permite evolucionar casos de pruebas para que sean más robustos y avanzados, permitiendo la realización de pruebas más complejas que pueden ir desde pruebas de accesibilidad hasta la validación de contenido dinámico e incluyendo detección de anomalías en la interfaz de usuario. Dada su arquitectura

basada en WebDriver y su facilidad de integrarse con diferentes frameworks de machine learning lo convierten en una solución innovadora para automatizar varias fases y tareas del proceso de pruebas de software (Orozco Díaz, 2022) como, por ejemplo:

Generación de Casos de Prueba: la herramienta al hacer uso de modelos de Machine Learning para interpretar y analizar comportamientos y cambios en la interfaz de usuario es capaz de generar o sugerir casos de prueba basados en patrones de uso reales. Al mismo tiempo, se puede hacer uso de los informes y reportes generados por la herramienta para ampliar los casos de prueba haciéndolos más complejos y permitiendo cubrir escenarios más diversos tal como lo son pruebas de experiencia de usuario y pruebas de rendimiento (Martín, 2023).

Optimización de Pruebas: Mediante el análisis de los resultados de las pruebas es posible detectar patrones de fallos y priorizar áreas de la aplicación que requieren más atención. Igualmente, usando algoritmos de identificación visual, se pueden detectar comportamientos de uso sobre la interfaz y con los que se pueden plantear casos de pruebas más complejos y generar la automatización de tareas repetitivas.

Pruebas Visuales: al usar modelos de aprendizaje automático entrenado para reconocer elementos en la interfaz de usuario y al Integrase con herramientas como Applitools que utilizan machine learning para realizar comparaciones visuales inteligentes, es posible que con un entrenamiento continuo el modelo sea cada vez más eficaz, fiable y preciso, permitiendo detectar pequeños cambios en la interfaz de usuario y ajustando los esquemas de pruebas para que se enfoquen en secciones puntuales de mayor relevancia (Martín, 2023).

En conclusión, Appium es una potente herramienta para la automatización de pruebas, la cual nos brinda un gran abanico de posibilidades, ya que al ser multiplataforma y permitir la integración con varios lenguajes y herramientas, la hace una opción robusta y a la vez flexible, que nos brinda distintas posibilidades que pueden ir desde el análisis de interfaces y el comportamiento que los usuarios tienen sobre estas hasta evaluar la experiencia de los usuarios al usar el aplicativo y con los reportes e informes de la herramienta generar planes de optimización o contingencia.

SonarQube

Plataforma de código abierto para la inspección continua de la calidad del código, proporcionando análisis estático del código para identificar bugs, vulnerabilidades de seguridad, y problemas de mantenimiento del código. Se usa mucho en la industria del software para garantizar que el código sea de alta calidad y ayudar a los desarrolladores a mejorar continuamente sus prácticas de codificación. A continuación, se presentan algunas posibles aplicaciones de SonarQube:

Detección de Vulnerabilidades y Bugs: Identifica problemas en el código que pueden ser potenciales bugs o vulnerabilidades de seguridad. SonarQube clasifica estos problemas en diferentes niveles de gravedad (bloqueadores, críticos, mayores, menores y de información) (Campbell & Papapetrou, 2013).

Ejecución de Pruebas: Ejecuta las pruebas automatizadas (como pruebas unitarias) en el proyecto. La herramienta de cobertura de pruebas recolectará datos sobre qué partes del código fueron ejecutadas durante las pruebas (ElEmam, 2022).

Puertas de Calidad (Quality Gates): Utiliza puertas de calidad para establecer umbrales que el código debe cumplir antes de ser aceptado, lo cual incluye métricas como cobertura mínima de pruebas y cantidad máxima de vulnerabilidades (Campbell & Papapetrou, 2013).

Evaluación de la Complejidad del Código: Analiza la complejidad cognitiva, proporcionando métricas que permiten identificar y abordar partes del código que pueden ser difíciles de entender o mantener (Campbell & Papapetrou, 2013).

SonarQube puede instalarse localmente en un servidor o usarse como servicio en la nube. La instalación local implica configurar una base de datos para almacenar los resultados del análisis y un servidor para ejecutar el servicio de SonarQube. Para el uso en la nube, existen versiones comerciales de SonarQube que proporcionan hosting y administración simplificados.

Testium

Plataforma avanzada de automatización de pruebas basada en inteligencia artificial (IA) y machine learning (ML). Está diseñada para simplificar y acelerar la creación, ejecución y

mantenimiento de pruebas automatizadas, abordando algunos de los desafíos más comunes en el testeo de software, como la fragilidad de los tests y el tiempo de mantenimiento. Las características principales y aplicación de Testium para la automatización de pruebas son:

Creación Inteligente de Pruebas: Utiliza algoritmos de ML para identificar y fijar elementos de la interfaz de usuario de manera más resistente a los cambios en el código, reduciendo la fragilidad de las pruebas.

Auto-Mantenimiento: Testim actualiza automáticamente los selectores y otros aspectos de las pruebas cuando detecta cambios en la aplicación, minimizando la necesidad de intervención manual.

Feedback de Pruebas: Proporciona información detallada y visual sobre el estado de las pruebas y los errores detectados.

Parallel Execution: Permite ejecutar pruebas en paralelo en múltiples navegadores y entornos, acelerando significativamente el tiempo total de prueba.

Test Analytics: Proporciona análisis detallados sobre la cobertura de las pruebas, tiempos de ejecución, tasas de fallos, y otras métricas clave.

Reporting: Genera informes visuales que ayudan a los equipos a comprender rápidamente el estado de las pruebas y a identificar áreas problemáticas.

Test Management: Facilita la gestión y organización de casos de prueba, permitiendo a los usuarios agrupar, etiquetar y priorizar pruebas según sea necesario.

Amazon DevOps Guru

Es un servicio que hace uso de machine learning que nos ayuda a mejorar el rendimiento operativo y la disponibilidad de una aplicación o sistema, realizando la detección de comportamientos anómalos o que se salen de los patrones operativos o métricas establecidas. De igual manera, nos permite identificar, diagnosticar y solucionar automáticamente los problemas de rendimiento y operación que tradicionalmente han sido difíciles de identificar, permitiendo generar planes de mejora y medidas preventivas frente a estas situaciones (Artistizábal, 2024). Entre los principales beneficios de Devops Guru se pueden destacar los listados a continuación:

Mejora De La Disponibilidad Y Rendimiento: al estar analizando continuamente las métricas y flujos de la plataforma, permite identificar fácilmente los primeros signos de errores o futuras anomalías, corrigiéndolos antes que tenga una afectación importante sobre el sistema y generando planes de mejora.

Simplifica la gestión: dada su interfaz amigable y de sencilla, permite realizar una gestión fácil y adecuada del servicio, sin requerir de conocimientos o experiencia previa en machine learning, lo que la hace accesible a todos los usuarios.

Escale Y Mantenga La Disponibilidad: Permite el despliegue de actualizaciones y la fácil adaptación de nuevas cargas de trabajo con configuración mínima.

Microsoft Azure ML

Azure ML es un servicio en la nube de Microsoft que te permite acelerar y administrar el ciclo de vida completo de tus proyectos de aprendizaje automático. Desde la preparación de datos hasta la implementación de modelos en producción, Azure ML te ofrece una plataforma integral y escalable para desarrollar soluciones de inteligencia artificial. Algunas de las funciones claves de Microsoft Azure Ml se presentan a continuación:

Preparación de datos: Permite la carga de datos desde diversas fuentes (bases de datos, archivos CSV, etc.). Adicional la limpieza y transformación de datos para eliminar valores faltantes, normalizar datos y crear nuevas características.

Entrenamiento de modelo: Realiza selección de algoritmos adecuados para tu problema (regresión, clasificación, Clustering, etc.) y configuración de hiperparámetros para ajustar el comportamiento del modelo, además de entrenamiento del modelo utilizando grandes cantidades de datos.

Amazon SageMaker

Amazon SageMaker es una plataforma completa para machine learning que ofrece una serie de herramientas y servicios para acelerar y simplificar el desarrollo, entrenamiento e implementación de modelos. Una de sus aplicaciones más destacadas se encuentra en el ámbito del testeo de software. A continuación, se detallan algunas opciones de uso de Amazon SageMaker:

Generación de datos sintéticos: Puede generar grandes cantidades de datos sintéticos altamente realistas para entrenar modelos de machine learning que identifiquen patrones y anomalías en los datos de producción.

Automatización de pruebas: SageMaker permite automatizar la creación y ejecución de pruebas, lo que reduce significativamente el tiempo y los recursos necesarios para garantizar la calidad del software.

Detección de anomalías: Los modelos de machine learning entrenados con SageMaker pueden identificar patrones anómalos en los datos de producción, lo que permite detectar errores y problemas de seguridad de manera temprana.

Keras

Keras es una biblioteca de aprendizaje profundo de alto nivel que se ha vuelto muy popular debido a su facilidad de uso y su capacidad para construir modelos de manera rápida y eficiente.

Generación de datos sintéticos: Keras puede ser utilizado para crear modelos generativos que produzcan datos sintéticos muy realistas. Estos datos pueden ser utilizados para ampliar los conjuntos de datos de prueba y mejorar la cobertura de las pruebas.

Predicción de fallas: Keras puede ser utilizado para construir modelos de predicción que anticipen fallas en el software. Al analizar los datos de logs y métricas del sistema, estos modelos pueden identificar patrones que indican una posible falla.

Técnicas más Convenientes Para el Testeo de Software

Técnicas de Clasificación

Las técnicas de clasificación se usan para predicción de defectos en software. Entre las técnicas más comunes se encuentran los procedimientos estadísticos, los métodos basados en árboles, las redes neuronales y los enfoques basados en analogías. Estas técnicas ayudan a categorizar módulos de software en propensos a fallos (FP) y no propensos a fallos (NFP)

basándose en modelos derivados de datos de proyectos de desarrollo previos (Lessmann, Baesens, Mues, & Pietsch, 2008).

Procedimientos Estadísticos: Son utilizados para modelar la calidad del software y predecir defectos. Estas técnicas incluyen la regresión logística y el análisis discriminante lineal (Lessmann, Baesens, Mues, & Pietsch, 2008).

Redes Neuronales: Aplicadas para modelar la calidad del software en sistemas complejos, especialmente en grandes sistemas de telecomunicaciones (Khoshgoftaar, Allen, Hudepohl, & Aud, 1997).

Técnicas de Agrupación

DBSCAN (Density-Based Spatial Clustering of Applications with Noise): El algoritmo DBSCAN es aplicado para identificar clústeres en grandes bases de datos. Este método es útil para encontrar estructuras en los datos sin necesidad de especificar el número de clústeres previamente. Además, DBSCAN puede identificar datos como ruido, como ejemplo, Cagatay Catal menciona en su artículo que, para el diseño experimental, DBSCAN detectó data ruidosa en un 13% y 1% para el conjunto de datos creados (Catal, Sevim, & Diri, 2009).

Algoritmo de Clustering K-Means: El K-Means es un algoritmo de agrupamiento no jerárquico. La técnica consiste en identificar grupos de casos de prueba similares, eliminando

redundancias y optimizando el conjunto de pruebas. El proceso para la implementación de este algoritmo inicia por la recopilación de datos sobre los defectos encontrados, el procesamiento de datos, ejecución del algoritmo y análisis de los agrupamientos para identificar patrones comunes y áreas del software que necesitan más atención.

Estudio de Caso: Machine Learning para el Testeo de Software en Cobis-Topaz.

Cobis-Topaz, una empresa multinacional dedicada al desarrollo de software, especializada en brindar soluciones financieras y digitales a diferentes instituciones financieras en Latinoamérica y Estados Unidos. Por la gran cantidad de proyectos y clientes a los que les provee servicios, y el gran tamaño y volumen de datos que maneja, se ha visto enfrentado a varios desafíos en cuanto a calidad y eficiencia de sus procesos de testeo. La naciente necesidad de reducir tiempos en la ejecución de casos de pruebas, detectar cuellos de botella y problemas de rendimiento. Al igual que aumentar y mejora la detección de fraudes o errores, han llevado a la empresa a explorar soluciones innovadoras. Una de estas soluciones es la integración de aplicativos con técnicas de Machine Learning en su proceso de testeo de software, específicamente utilizando herramienta tales como SonarQube, Amazon DevOps Guru y plataformas como Microsoft Azure.

Problemas En Proceso De Pruebas De Cobis-Topaz.

Aunque la implementación de machine learning no es ajena en la implementación de proyectos y productos ofrecidos por Cobis-Topaz a sus principales clientes, dado que entre las soluciones brindadas por la empresa podemos encontrar distintas herramientas y aplicaciones potenciadas por aprendizaje automático tales como modelos de prevención de fraude y lavado de activos, modelos predictivos de fuga de clientes, modelos de propensión de pagos y modelos de recomendación para productos. sin embargo, la implementación de machine learning en la ejecución e implementación de planes de pruebas ha enfrentado desafíos importantes dentro de la organización, dado que esta no ha sido totalmente integrada en todos

los procesos de pruebas de los distintos proyectos, aunque bien se han realizado iniciativas interesantes con la implementación de herramientas como Amazon DevOps Guru en el monitoreo y detección de errores en la implantación y desarrollos de servicios web AWS de clientes tan importantes como lo es Walmart y su aplicación móvil Cashi. De igual manera, se está implementando el análisis inteligente de código con SonarQube en los distintos proyectos, para aumentar el porcentaje de detección de defectos y mitigar el impacto de errores en la calidad del producto final. Entre los inconvenientes y problemas que también se busca aminorar, se encuentran los expuestos a continuación:

Problemas De Calidad

Bugs e Incidentes en Producción: Defectos o errores no detectados en la fase de desarrollo pueden generar que al desplegarse en producción el software o aplicativo generen bugs o incidentes, afectando el comportamiento de las funcionalidades impactadas durante el desarrollo, al igual que funcionalidades que no se vieron impactadas, pero sobre las que no se realizaron pruebas de regresión, afectando la disponibilidad en línea o la experiencia y precepción del usuario final.

Degradación del Rendimiento: Problemas de rendimiento no identificados o la falta de pruebas de carga y estrés durante la ejecución del plan de pruebas pueden causar que la aplicación funcione lentamente, consuma más recursos de los necesarios o incluso se bloquee. Lo cual puede generar tiempos de respuesta excesivos generando time out, cuellos de botella y fallos operativos en el aplicativo.

Falta de Monitoreo y Escalabilidad: La falta de herramientas de monitoreo y evaluación del comportamiento del sistema durante el proceso de pruebas de calidad, puede llegar a imposibilitar la identificación de problemas de rendimiento nacientes. Al igual, que problemas como el uso ineficiente de recursos, bloqueos de memoria e inconvenientes de rendimiento y comportamientos anómalos.

Costos Elevados

Costo de Corrección Tardía: Detectar y corregir errores en las etapas finales del desarrollo o en producción es significativamente más costoso que hacerlo durante las etapas iniciales, requiriendo retrabajo y afectando considerablemente la eficiencia y efectividad del desarrollo y despliegue de software. Dado que requiere la inversión de esfuerzos tanto humanos como físicos, ya que se tienen que destinar recursos para reescribir, probar y validar las soluciones desarrolladas, aumentando el presupuesto y gastos destinados para el proyecto.

Retrasos en el Proyecto: La necesidad de corregir errores críticos en etapas avanzadas puede provocar retrasos significativos en el cronograma del proyecto. extendiendo la planeación inicial de las distintas fases de desarrollo, prologando la ejecución del ciclo de pruebas y retrasando significativamente la puesta en producción del aplicativo desarrollado o bien posponiendo el inicio de desarrollos que se encuentren en cola en el backlog de desarrollos planificados.

Impacto en la Reputación

Percepción Negativa del Usuario: La presencia de errores puede afectar negativamente la percepción del usuario sobre la calidad y fiabilidad del software, teniendo un impacto reputacional negativo de cara a los clientes y afectando la competitividad directa de la empresa frente a otros clientes o proveedores rivales. Desgastando e incluso llegando a perder uno de los activos intangibles más importantes de la empresa tal como es la reputación y conllevando a perder oportunidades de negocio dentro del mercado.

Pérdida de Confianza: Clientes y usuarios pueden perder la confianza en la empresa, si el software presenta fallos frecuentes y graves, que pongan en riesgo la disponibilidad de los aplicativos, generen una experiencia de usuario deficiente o incluso llegando a arriesgar la seguridad e integridad de la información y datos críticos del cliente. Conllevando a pérdidas financieras e impactando directamente en la confianza, lealtad y percepción de la empresa ante el público en general.

Mantenimiento y Soporte

Incremento en los Costes de Mantenimiento: Corregir errores en producción no solo es más costoso, sino que también incrementa el esfuerzo de mantenimiento y soporte necesario. Impactando directamente en los cronogramas y presupuestos estimados para la implementación de distintos proyectos. Ya que no solo hay que destinar recursos humanos y físicos para solucionar los inconvenientes presentados, sino que alarga los ciclos de desarrollos y pruebas, haciéndolos menos eficientes y complejos, ya que en muchos casos hay que

modificarlos y replantearlos para enfocarlos en aquellas escenarios o funcionalidades a los que no se les dio la suficiente relevancia y donde se terminaron presentados defectos o malos funcionamientos.

Soporte Técnico Sobrecargado: La necesidad de manejar y resolver problemas de usuarios relacionados con errores no detectados puede sobrecargar al equipo de soporte técnico. Debido a la gran cantidad de incidencias o bugs reportados y generando tiempos excedidos de respuesta e incumpliendo los tiempos de respuesta establecidos en los Acuerdo de Nivel de Servicio (ANS) y reduciendo la capacidad de reacción del equipo ante nuevos sucesos o defectos, al presentarse encolamientos en la respuesta de tickets o casos. Impactando negativamente en la moral y salud emocional del equipo haciéndolo menos productivo y eficaz.

Impacto en el Negocio

Pérdida de Clientes: Los clientes pueden optar por productos de la competencia sí experimentan problemas frecuentes con el software, al sentir que no se está proporcionando un servicio a la altura de las expectativas, generando un sentimiento que los productos entregados no están cumpliendo con los estándares de calidad esperados o presentar retrasos en las fechas comprometidas para el despliegue y puesta en producción de productos, actualización o funcionalidades acordadas.

Interrupciones Operacionales: Errores críticos pueden causar interrupciones en los servicios o fallos recurrentes, afectando las operaciones diarias de la empresa, disponibilidad del servicio o problemas de rendimiento. Teniendo un impacto negativo en la experiencia de los usuarios y clientes finales.

Ante estos nacientes desafíos, Cobis-Topaz ha tenido la necesidad e iniciativa de implementar Machine Learning para automatizar y optimizar el proceso de testeo dentro de varios de sus proyectos, aprovechando herramientas como SonarQube y Amazon DevOps Guru para mejorar la detección y corrección de errores en el código. Al igual, que el rendimiento, disponibilidad y posibles anomalías sobre sus servicios web AWS.

Exploración Problemas en Proceso De Pruebas De Cobis-Topaz

El proceso de pruebas en Cobis-Topaz enfrenta varios problemas críticos que hacen imperativo buscar soluciones efectivas cuanto antes. En primer lugar, la falta de automatización y el uso de métodos tradicionales para la detección de errores han generado dificultades en garantizar la calidad del software. Esto no solo aumenta el riesgo de fallos en producción, sino que también impacta directamente en la experiencia del usuario final. En un mercado donde los plazos para lanzar nuevas versiones de software son cada vez más ajustados, las pruebas manuales resultan insuficientes, provocando retrasos, sobrecostos y posibles defectos en el producto final.

Es fundamental reconocer que, a medida que las aplicaciones se vuelven más complejas y los estándares de calidad se elevan, los métodos de prueba tradicionales no son suficientes para cumplir con las expectativas. La empresa se enfrenta a la necesidad de mejorar sus procesos de prueba para poder garantizar la fiabilidad, seguridad y eficiencia del software. Además, el no abordar estos problemas de manera oportuna puede derivar en una pérdida de competitividad, ya que un mal desempeño del software puede generar descontento entre los usuarios y dañar la reputación de la empresa.

La urgencia radica en la creciente presión del mercado por ofrecer soluciones rápidas, seguras y de alta calidad. Implementar tecnologías innovadoras, como SonarQube, que utiliza técnicas de Machine Learning, podría optimizar significativamente los procesos de prueba, permitiendo una detección más eficiente de errores y ayudando a mantener los altos estándares de calidad que la empresa necesita. Resolver estos problemas no solo es necesario, sino urgente para asegurar que Cobis-Topaz continúe siendo competitivo en un entorno empresarial cada vez más exigente.

Aplicación de Machine Learning en Cobis-Topaz

SonarQube es una herramienta muy útil en tareas de testing y aseguramiento de la calidad del software. Aunque está principalmente orientada a los grupos de desarrolladores para evaluar la calidad del código en etapas tempranas de codificación y pruebas unitarias o cruzadas realizadas entre integrantes que conforman los equipos de desarrollo, los testers o analistas funcionales pueden beneficiarse de sus grandes capacidades para identificar problemas potenciales, determinar que funcionalidades presenta déficit o un nivel bajo respecto

a la calidad o bien concluir en que secciones de código se agrupa el mayor nivel de defectos o errores históricamente, permitiendo priorizar ciertos escenarios y casos de prueba para asegurar que el código cumple con los estándares de calidad. Aquí hay algunas maneras en que los testers están implementado SonarQube dentro de varios proyectos de la organización como los son Banco Agrario de Colombia, Fondo Nacional de Ahorros y depósitos a plazo fijo del Banco Davivienda:

Cobertura de Pruebas

Informe de Cobertura: Muestra qué porcentaje del código está cubierto por pruebas unitarias. Esto ayuda a los testers a identificar áreas del código que necesitan más pruebas y priorizar las mismas. De igual manera, les permite establecer planes de mejores y definir qué tipo de pruebas son las más adecuadas para ser aplicadas a secciones de código en específico.

Ejecución de Pruebas Automáticas: Dado que varios de los proyectos manejados por Cobis-Topaz al regirse sobre la metodología ágil SAFe y hacer el uso de DevOps por medio de Azure, integra pruebas automáticas y análisis de la cobertura de estas, al proceso de integración y entrega continuas (CI/CD) implantado en los distintos proyectos.

Monitoreo Continuo

Dashboard Personalizado: dado que en los distintos proyectos manejados por la empresa se manejan necesidades distintas y los estándares de codificación distintas, es posible configurar dashboards personalizados para monitorear y analizar la calidad del código fuente a lo largo del tiempo. Al igual, que estipular reglas y excepciones particulares para cada proyecto, permitiendo enfocarse bien sea en la Fiabilidad, Seguridad, Mantenimiento, cobertura, duplicación o Tamaño de acuerdo con la necesidad naciente en el momento.

Trends e Historial: Dado que se almacena un histórico de cada uno de los análisis realizados sobre el código tanto a nivel de requerimiento como de proyecto, es posible realizar mediciones y análisis de tendencias e historial de calidad para determinar cómo evoluciona el código y tomar decisiones informadas. así como determinar, en qué secciones o funcionalidades se tienden a producir más defectos y errores.

Colaboración y Feedback

Feedback Temprano: Los testers pueden proporcionar feedback y retroalimentación temprano a los desarrolladores basándose en los informes, reportes e historial de SonarQube, ayudando a corregir incidentes antes de que se conviertan en problemas mayores o bien detectando posibles vulnerabilidades y brechas de seguridad que en el futuro podría llegar a poner en riesgo al aplicativo mismo y la información vulnerable tanto de los clientes como de la organización.

Documentación de Issues: El uso constante y la ejecución de análisis de código en diferentes fases de desarrollo y pruebas haciendo uso de SonarQube ha permitido documentar y realizar un seguimiento continuo sobre los incidentes y problemas encontrados durante todas las fases del desarrollo, permitiendo registrar tanto en la documentación técnica de los requerimientos como en la plataforma de colaboración integrada (Atlassian Confluence) de la compañía, la solución a problemas o errores recurrentes que se pueden presentar durante el desarrollo y despliegue de los distintos aplicativos.

Análisis Cualitativo De Resultados De La Implementación

La implementación de SonarQube en la empresa Cobis-Topaz tuvo un impacto significativo en la mejora de la calidad del código y la eficiencia en procesos de pruebas. Los resultados mostraron una notable reducción en la cantidad de errores críticos detectados en el código, así como una disminución considerable en el mantenimiento de software, mejorando la mantenibilidad del software. Además, la cobertura de pruebas automatizadas se incrementó, lo que permitió una mayor detección de defectos y optimización del tiempo en el proceso de pruebas. A continuación, se presentan los resultados detallados:

- **Mejora en la detección de errores:** Mayor precisión en la identificación de defectos y vulnerabilidades.

- **Reducción del tiempo de pruebas:** Automatización de tareas repetitivas y análisis continuo del código.

- **Mejora en la productividad del equipo**: SonarQube al proporcionar retroalimentación continua sobre la calidad del software permite que los desarrolladores y tester pueda gestionar los errores de manera inmediata

- **Mejora la calidad del software:** Con la implementación de SonarQube se pueden identificar errores y vulnerabilidades en el código desde las primeras etapas del desarrollo.

- **Facilita el mantenimiento del código:** Al monitorear la evolución de la calidad del código a lo largo del tiempo, ayuda a la identificación de tendencias y áreas de mejora. Además de disminuir el soporte a corrección de error productivos que suelen ser costosos, pueden ocasionar multar y tienen un tiempo de solución mínimo, incrementando el esfuerzo de mantenimiento y soporte.

- **Impacto positivo al usuario:** La baja presencia de errores puede beneficiar positivamente en la percepción del usuario sobre la calidad y fiabilidad del software.

Análisis Cuantitativo De Resultados De La Implementación

El uso de técnicas de machine learning y la integración de inteligencia artificial se ha establecido como una política de optimización e innovación al interior de las organizaciones y Cobis-Topaz no es la excepción en donde se han planteado estrategias internas con el fin de utilizar tanto machine learning como herramientas de análisis de código e inteligencia artificial para garantizar la calidad y la seguridad de sus productos. De igual manera, también ha buscado ofrecer a sus clientes productos innovadores y ha buscado integrar técnicas de aprendizaje automático en el desarrollo de las soluciones tecnológicas que ofrece.

Implementación de SonarQube y Beneficios Obtenidos: Cobis-Topaz, en colaboración con el Banco Agrario de Colombia, ha integrado SonarQube como parte de su ciclo de desarrollo y ecosistema de herramientas de calidad de software. Desde el año 2020 como requisito técnico y documental de cada requerimiento de software entregado al banco y en donde en conjunto con el cliente, se han establecido métricas que exige, que la calificación del análisis de código no sea inferior a "A" en aspectos clave como Bugs, Code Smells y vulnerabilidades. De igual manera, el informe generado por la herramienta abarca puntos tan importantes como lo son problemas relacionados con fiabilidad, seguridad, mantenibilidad, cobertura, y duplicación de código. Esta implementación ha permitido establecer reglas claras y precisas para evaluar la calidad del código, especialmente en los componentes más críticos del sistema.

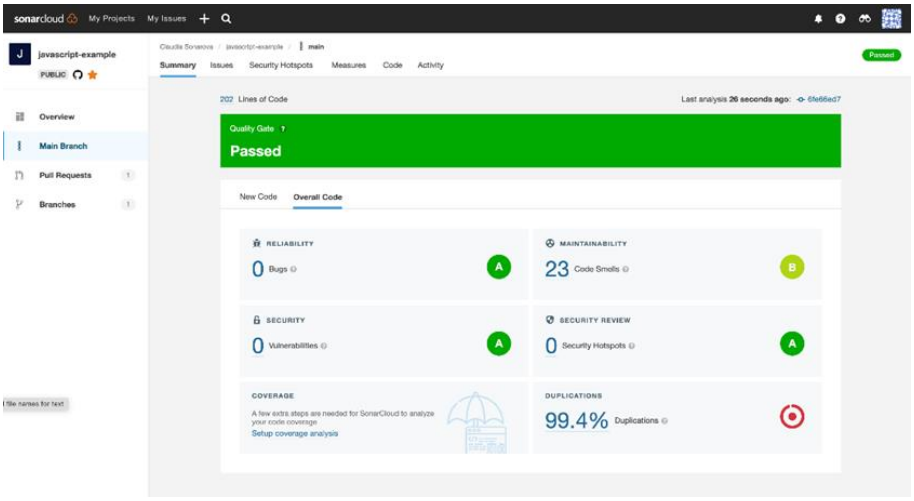
Como política de calidad del ciclo de desarrollo se ha establecido la ejecución de análisis del código fuente de backend en dos puntos críticos. El primer análisis se ejecuta previo al inicio del desarrollo y el cual tiene como finalidad detectar la deuda técnica acumulada que puedan llegar a tener las fuentes a modificarse durante del desarrollo y tratar de resolver o mitigar a un porcentaje menor a 5% respecto al total de líneas. Al mismo tiempo, el informe generado permite establecer que secciones o funcionalidades presentan un mayor peso o cantidad de deuda técnica, para de esta manera enfocar los casos de pruebas en estas. El segundo análisis, se realiza una vez se haya terminado la codificación del requerimiento y en este se busca determinar que la deuda técnica se vio solventada y que el nuevo código o modificaciones, tengan una calidad y optima y no haya generado alertas respecto a Bugs, Code Smells y vulnerabilidades. De ser así, estas tendrán que ser resueltas previo a la entrega del requerimiento al cliente para garantizar la calidad del producto final.

Definición De Quality Gate Para Proyectos: Una puerta de calidad (*QualityGate*) es un indicador que nos permite cuantificar si el código fuente cumple con el nivel mínimo de calidad requerido. Evaluándolo con un conjunto de condiciones que se aplican a los resultados de cada análisis y nos permite concluir si las fuentes analizadas cumplen o superan las condiciones de la puerta de calidad establecidas, dando una calificación de Aprobado o en caso contrario una calificación de Fallido.

Las puertas de calidad se muestran en la interfaz de SonarCloud junto con los resultados del análisis de la rama principal del proyecto indicando un estado aprobado o fallido, según el análisis dado por Sonar

Figura 4

Ejemplo Calificación QualityGate



Fuente. Autoría Propia

Dentro de todos los análisis ejecutados para un mismo proyecto, siempre se utiliza la misma definición de puerta de calidad para todos los componentes, configuraciones y ramas. Sin embargo, la forma en que se realizan los cálculos difiere un poco entre la rama principal, rama de larga duración y rama de corta duración. Para la rama principal, se aplican tanto las condiciones definidas en el código general como las condiciones definidas en el nuevo código. Para ramas de larga duración distintas de la rama principal, Se aplican tanto las condiciones definidas en el código general como las condiciones definidas en el nuevo código. por último, Para ramas de corta duración, al igual que en los otros casos, se utiliza la puerta de calidad definida a nivel de proyecto, pero solo se aplican las condiciones definidas al código nuevo.

Como principales pilares o características a analizar, SonarQube evalúa tres puntos cruciales:

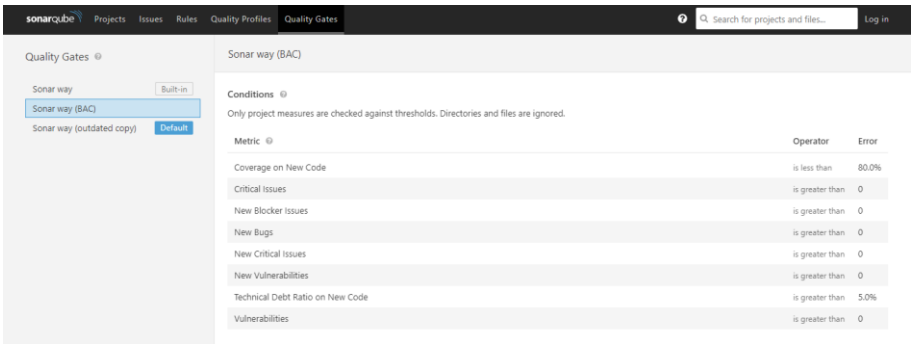
- La confiabilidad que deberá tener una calificación de al menos A
- La capacidad de mantenibilidad deberá tener una calificación de al menos B
- La cobertura de la prueba será de al menos el 80 %

Estas condiciones se aplican a los resultados del análisis para determinar si el código cumple con el nivel de calidad requerido. Si la rama principal cumple o supera las condiciones de la puerta de calidad, muestra el estado Aprobado, Si la rama principal no cumple con las condiciones de la puerta de calidad, mostrará el estado Fallido y Si la rama o el análisis no funcionará o presentará algún error, el estado será No Computado.

Como puertas de calidad para las distintos componentes y ramas del proyecto Banco Agrario de Colombia, se establecieron las siguientes métricas con el fin de mantener los máximos estándares de calidad del código en los proyectos de backend.

Figura 5

Quality Gate Establecidos para el Proyecto Banco Agrario de Colombia



Fuente. Autoría Propia

Una vez establecidas las puertas de calidad y la ejecución del análisis de código, la herramienta Sonar nos genera un informe completo en donde se detalla además del estado de la puerta de calidad, nos muestra varias categorías, métricas y medidas generales respecto a la calidad del código analizado. Dentro de estas podemos encontrar:

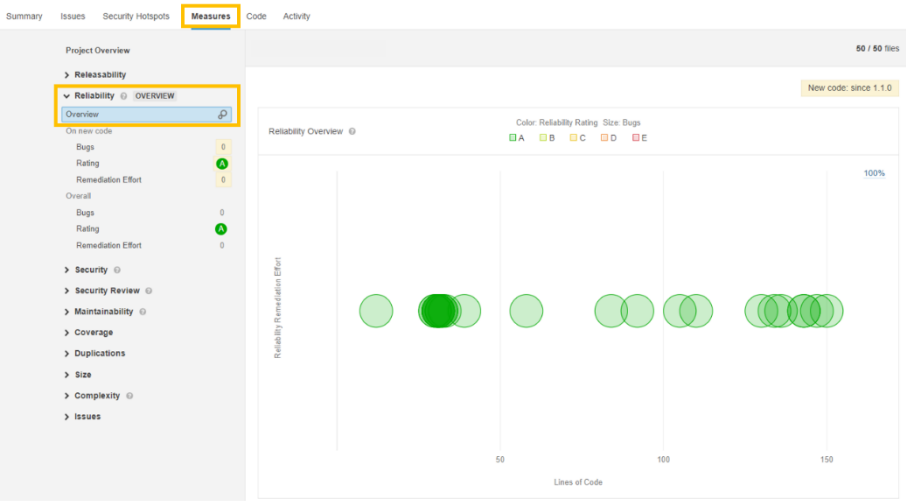
Reliability / Fiabilidad: Muestra la cantidad de bugs y su criticidad que provocan que el sistema no sea confiable. al igual que muestra, la calificación de se realiza en base al número de inconvenientes y su complejidad de la siguiente manera (Larios Calleja, 2021):

- A: Sin Bugs
- B: 1 Bug Menor

- C: 1 Bug Mayor
- D: 1 Bug Crítico
- E: 1 Bug Bloqueante

Figura 6

Categoría de Reliability / Fiabilidad - Informe SonarQube



Fuente. Autoría Propia

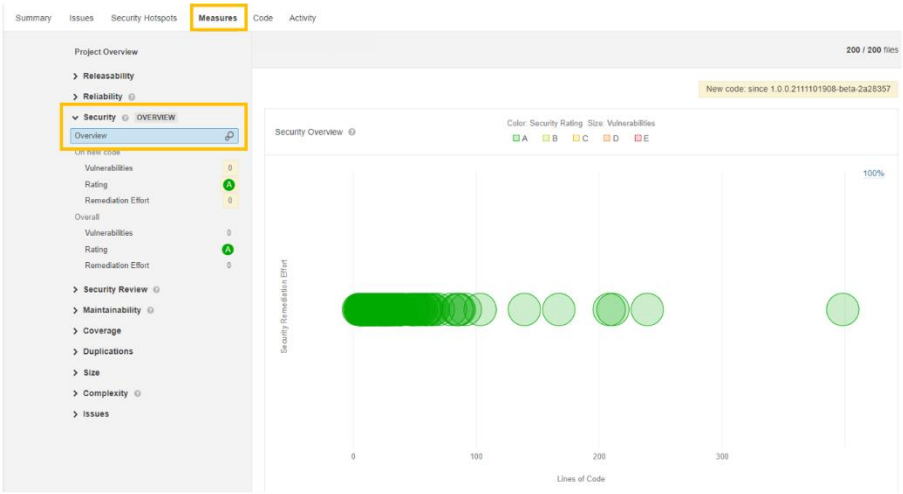
Security / Seguridad: evalúa la protección de la información y las debilidades o accesos que pueda llegar a tener un externo, bien sea, para leer o modificar la información o datos almacenados, el rating se realiza en base a las vulnerabilidades encontrados de la siguiente manera:

- A: 0 vulnerabilidades
- B: 1 Vulnerabilidad Menor
- C: 1 Vulnerabilidad Mayor
- D: 1 Vulnerabilidad Critico

- E: 1 Vulnerabilidad Bloqueante

Figura 7

Categoría Security / Seguridad - Informe SonarQube



Fuente. Autoría Propia

Coverage / Cobertura: Muestra el porcentaje de líneas de código potencialmente comprobables que en realidad están cubiertas por casos de prueba. el informe define el cálculo de la cobertura (CV) partiendo de la suma de líneas más las condiciones cubiertas sobre el total de líneas y condiciones que compone todo el proyecto (SonarSource, 2020). Se expone según la siguiente formula:

Figura 8

Cálculo de Cobertura

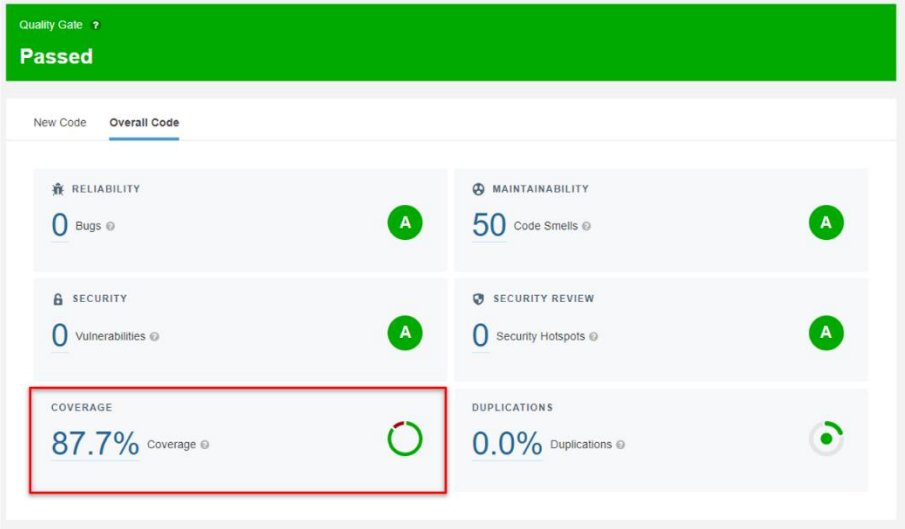
$$Cv = \frac{(Lines\_to\_Cover - Uncovered\_Lines) + (Conditions\_to\_Cover - Uncovered\_Conditions)}{(Lines\_to\_Cover + Conditions\_to\_Cover)}$$

Fuente. Sonarqube Calculation Coverage (SonarSource, 2020)

En el resumen principal de la rama tendremos el detalle sobre la cobertura:

Figura 9

Categoría Coverage / Cobertura - Informe SonarQube

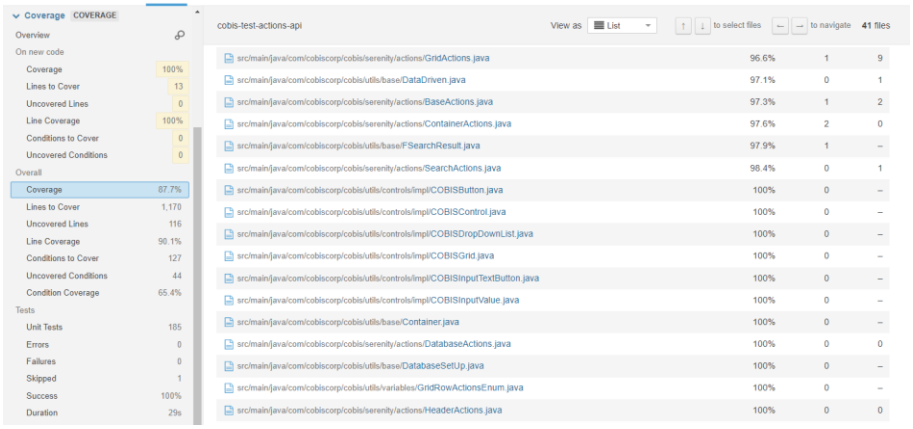


Fuente. Autoría Propia

Si pulsamos sobre la métrica nos da información detallada y los ficheros afectados:

Figura 10

Información Detallada Coverage / Cobertura

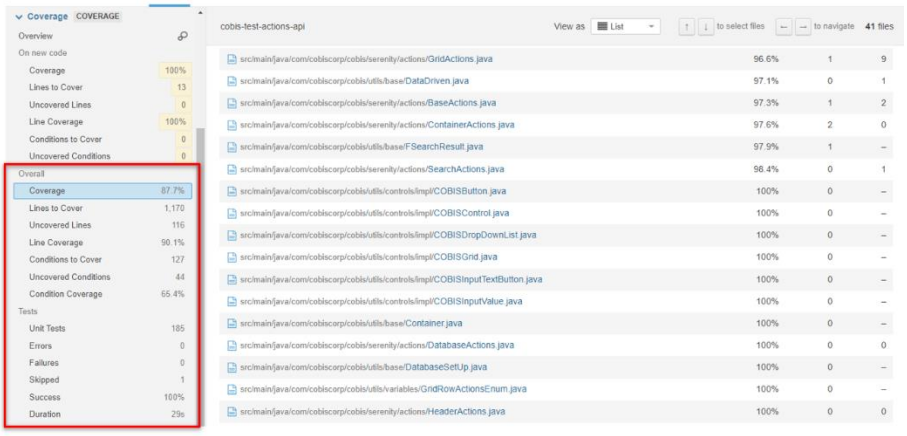


Fuente. Autoría Propia

En el menú de la izquierda, nos da información del resultado de la ejecución de las pruebas unitarias

Figura 11

Información Del Resultado De La Ejecución De Las Pruebas Unitarias

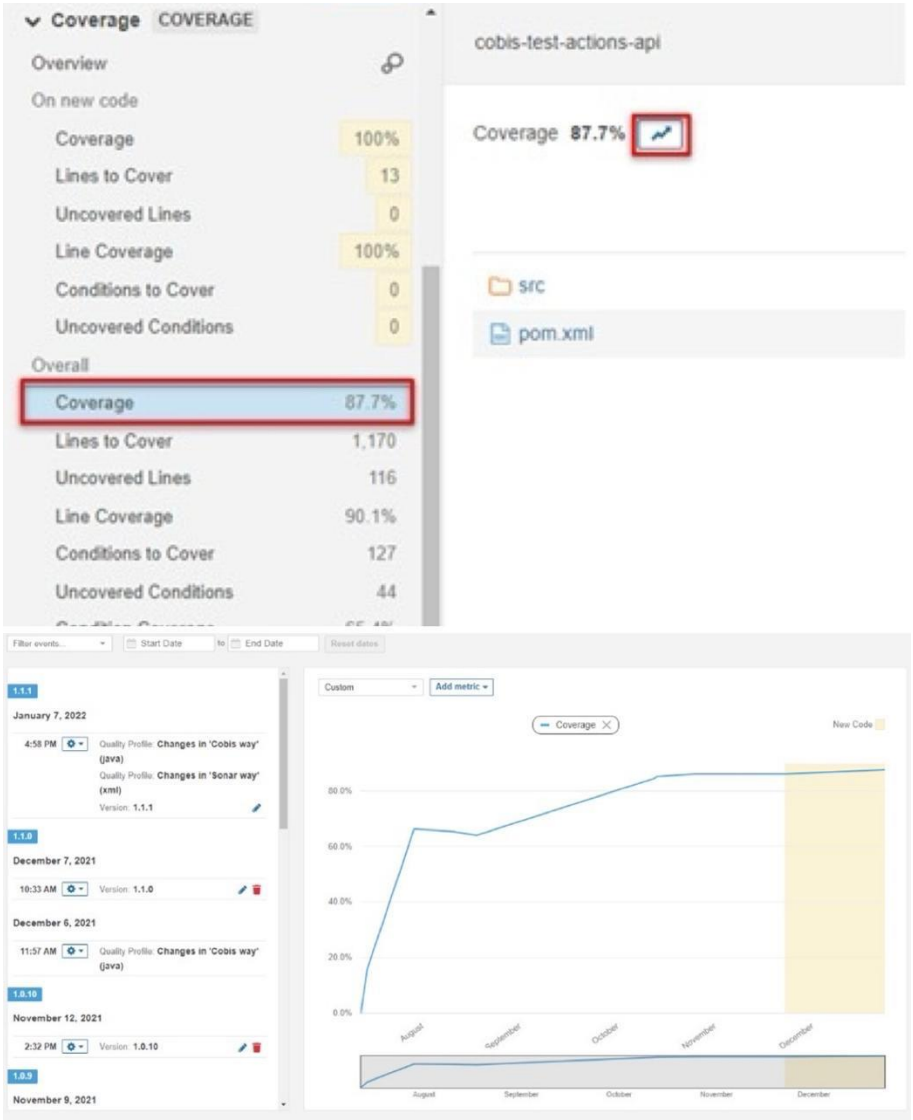


Fuente. Autoría Propia

Además, y muy importante, podemos comprobar su evolución mediante gráficas de evolución e histórico por rama de larga duración analizada en Sonar. Dando clic en la opción de grafico que se encuentra en el menú Coverage

Figura 12

Porcentaje de Cobertura

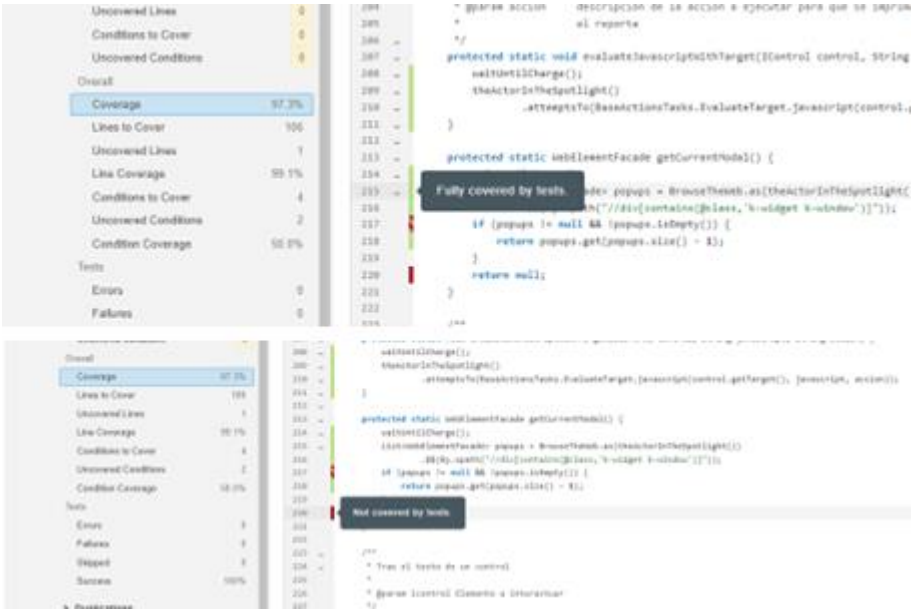


Fuente. Autoría Propia

Si accedemos a un archivo, en este caso Java, podemos ver como Sonar señala mediante una barra roja las líneas que no tienen cobertura, y en barra verde aquellas que si tienen cobertura.

Figura 13

Líneas de Código con y sin Cobertura



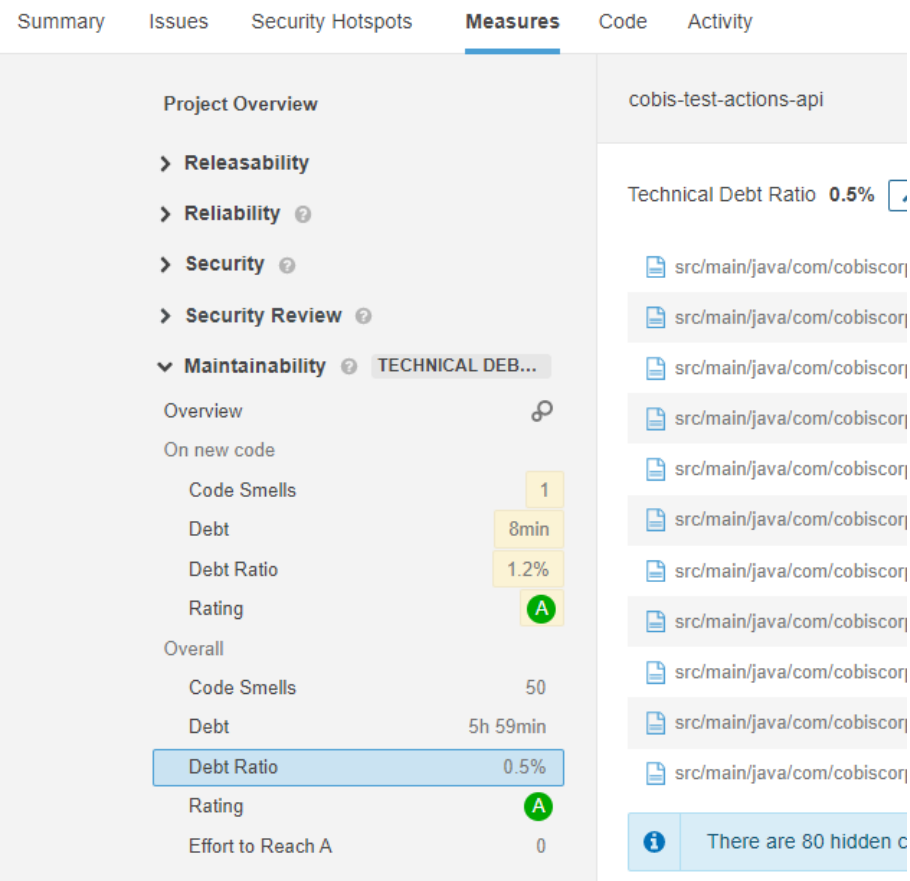
Fuente. Autoría Propia

Maintainability / Mantenibilidad: en esta sección se evalúa la capacidad del proyecto de código para ser modificado efectiva y eficientemente en el tiempo, debido a necesidades evolutivas o correctivas (Larios Calleja, 2021). La escala de calificación de mantenibilidad se puede establecer alternativamente dado el costo de remediación pendiente:

- <=5% del tiempo que ya ha pasado en la aplicación, la calificación es A
- entre 6 a 10% la calificación es una B
- entre 11 a 20% la calificación es una C
- entre 21 a 50% la calificación es una D
- cualquier Medición por encima del 50% es una E

Figura 14

Categoría Maintainability / Mantenibilidad - Informe SonarQube

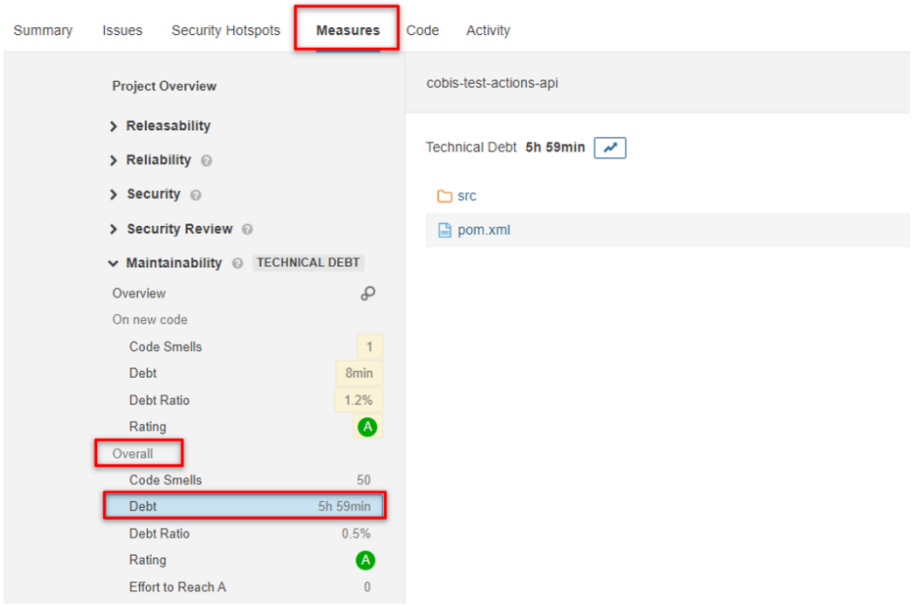


Fuente. Autoría Propia

Deuda Técnica: La deuda técnica se calcula como el esfuerzo futuro que se debe realizar para arreglar defectos, vulnerabilidades y problemas de mantenimiento que en su momento se introdujeron por alguna urgencia, plazo de entrega ajustado o falta de conocimiento (Larios Calleja, 2021). El informe de sonar calcula este esfuerzo en minutos y días (8 horas).

Figura 15

Categoría Deuda Técnica - Informe SonarQube



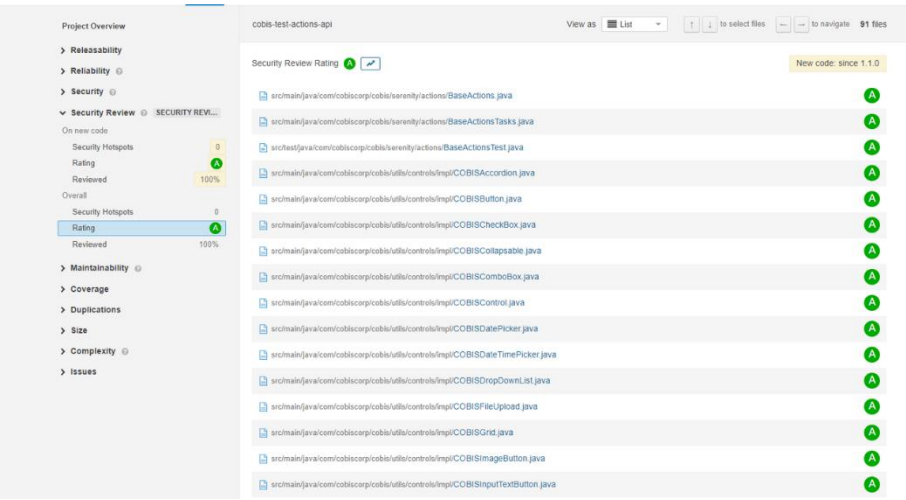
Fuente. Autoría Propia

Security Review / Revisión de Seguridad: El informe de *Sonar* realiza la revisión de seguridad, en busca de puntos de accesos o posibles vulnerabilidades de seguridad, que deben ser revisada y validar si corresponde a una grieta de seguridad. Por ello, el rating de realiza en base al número de vulnerabilidades encontradas de la siguiente manera:

- A: > 80%
- B: 70% - 80%
- C: 50% - 70%
- D: 30% - 50%
- E: < 30%

Figura 16

Categoría Deuda Técnica - Informe SonarQube

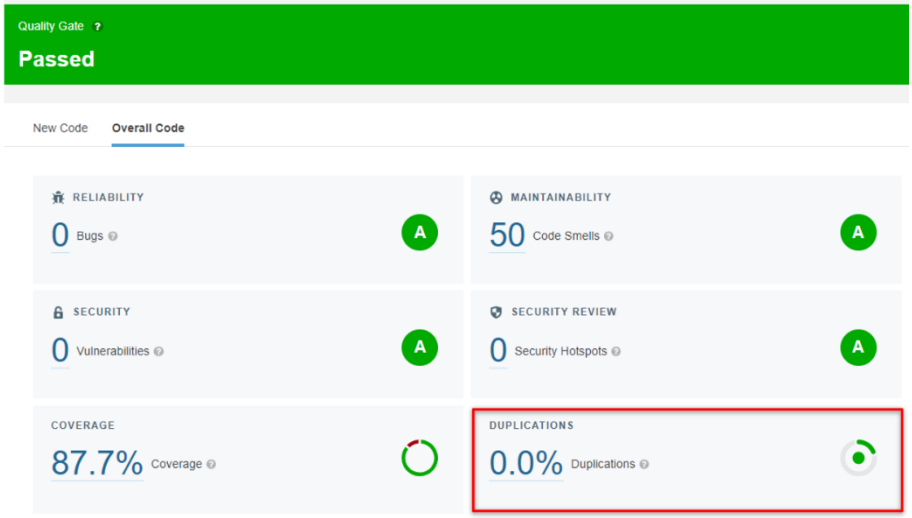


Fuente. Autoría Propia

Duplications / Duplicidad: Es una de las métricas que evidencia una serie de malas prácticas, al replicar código repetidamente o copiar código sin un patrón de reutilización. Esta sección se enfoca en encontrar líneas, bloques o ficheros duplicados en el código fuente de los proyectos.

Figura 17

Categoría Duplications / Duplicidad - Informe SonarQube



Fuente. Autoría Propia

A continuación, se detallará y analizara el impacto que esta política de calidad ha tenido sobre uno de los módulos más importantes y críticos en el proyecto de banco agrario de Colombia, el módulo de clientes o MIS – Managment Information System. En donde realizando una comparativa de los análisis de SonarQube de los 25 componentes clave del módulo de clientes, desde la implementación del análisis de código en 2020 hasta la fecha, se ha logrado identificar áreas de mejora que anteriormente no eran visibles a través de métodos tradicionales. Los resultados no solo han mejorado la seguridad y la mantenibilidad del código, sino que también han optimizado la eficiencia operativa del equipo de desarrollo.

Tabla 2

Comparativa Informe Sonar 2020 vs 2024

	Total de Líneas	Fiabilidad (Bugs)	Critical Issues (Vulnerabilidades)	Mantenibilidad (Code Smells)	Duplicidad
2020	29.645	142	32	743 (2.51%)	4.986 (16.8%)
2024	42.245	0	0	875 (2.07%)	7.980 (18.9%)
Variación	12.600 (142.5%)	-142	-32	-0.44%	2.994 (2.1%)

Nota. Se Hace una comparativa cuantitativa de los resultados del análisis entre los años de 2020 y 2024

Tal como se puede evidenciar en la gráfica comparativa se puede visualizar el gran impacto a nivel de calidad de software que ha significado la implementación de análisis con SonarQube y el refinamiento del código con ayuda de SAI Library. ya que, aunque la cantidad de líneas de código ha aumentado en un 142.5%, se ha visto una drástica disminución en ítems cruciales como lo son bugs e issues críticos, pasando de 142 y 32 a 0 y 0 respectivamente. De igual manera, la cantidad de Code Smells se ha visto afectada disminuyendo en un 0.44% respecto al total de líneas. asimismo, aunque la duplicidad de código aumento en un 2.1%, este aumento es mínimo si se tiene en cuenta el aumento exponencial en la cantidad total de líneas de código. Estos resultados evidencian un aumento exponencial en la calidad y mantenibilidad del código fuente de uno de los módulos más importantes y al cual el banco agrario de Colombia le presta más atención. dado que allí, se almacena la información sensible de sus clientes y la cual es usada por todos los demás productos.

El uso de machine learning, en particular, ha permitido a Cobis-Topaz abordar problemas complejos de forma anticipada, lo que ha reducido drásticamente la aparición de errores críticos y ha garantizado que el código cumpla con los más altos estándares de calidad

exigidos por clientes como el Banco Agrario de Colombia. Además, las recomendaciones de SAI Library continúan contribuyendo a la mejora continua, asegurando que el software evolucione de manera sostenible y eficiente.

Uso de SAI Library - Inteligencia Artificial para la Mejora Continua: Una de las innovaciones más destacadas de Cobis-Topaz ha sido la incorporación de su propia herramienta de inteligencia artificial, SAI Library. Esta plataforma utiliza los datos recopilados de todos los proyectos de la empresa y la extensa base de conocimientos del grupo Stefanini, para brindarnos opciones y consejos personalizados para la optimización del código y la mejora de la calidad del software.

SAI Library aprende continuamente a partir de los datos históricos y el Data Lake centralizado, identificando patrones que sugieren oportunidades de mejora. Gracias a esto, los equipos de desarrollo de Cobis-Topaz reciben recomendaciones y consejos no solo para mejorar la calidad del código, sino también para reducir los tiempos de entrega, minimizar los errores en producción y aumentar la productividad y la creatividad de los desarrolladores.

Implementación Code Guru Profiler en el Microservicio de Xsell (ECS): COBIS Xsell es una solución que integra y orquesta en un único portal los procesos y transacciones de negocio y operación mediante la utilización de microservicios, buscando fortalecer la gestión comercial en la atención a clientes y ventas de productos y servicios sobre todos los canales de distribución de la Institución Financiera. Con el fin de generar un análisis en el rendimiento del código y poder optimizar el mismo, se optó por realizar la integración de AWS CodeGuru Profiler en Amazon Elastic Container Service (ECS), específicamente en el microservicio de

Reglas de Xsell (gg-xsell-rules-service) con el objetivo de explorar la capacidad del CodeGuru Profiler para proporcionar informes detallados sobre el rendimiento del código en el entorno de ECS.

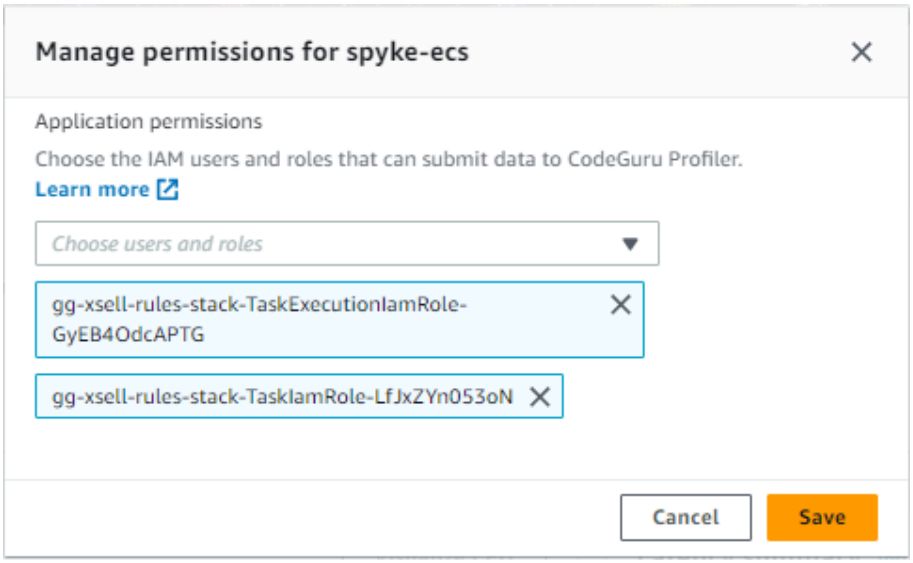
Para su implementación, se requirió la definición tanto de roles como de grupos de perfiles (profiling groups), para poder asociar a estos las respectivas políticas de acceso y seguridad que requiere CodeGuru Profiler para poder capturar, recopilar y analizar información. Mas puntualmente se requiere la autorizar de las políticas AmazonCodeGuruProfilerAgentAccess y AmazonCodeGuruProfilerFullAccess.

AmazonCodeGuruProfilerAgentAccess: Esta política otorga acceso a los recursos necesarios para instalar y ejecutar el agente del CodeGuru Profiler en las instancias de Amazon EC2 (Elastic Compute Cloud). Este agente recopila datos de perfil de rendimiento de las aplicaciones en ejecución en las instancias de EC2 y los envía al servicio de CodeGuru Profiler para su análisis (AWS Política gestionada, 2024).

AmazonCodeGuruProfilerFullAccess: Esta política proporciona acceso al servicio CodeGuru Profiler. Permite a los usuarios realizar todas las acciones disponibles en el servicio, como analizar perfiles de rendimiento, ver recomendaciones de optimización y administrar configuraciones de perfil. Esta política es más adecuada para usuarios que necesitan realizar análisis detallados y gestionar el servicio en su totalidad (AWS Política gestionada, 2024).

Figura 18

Configuración de Permisos

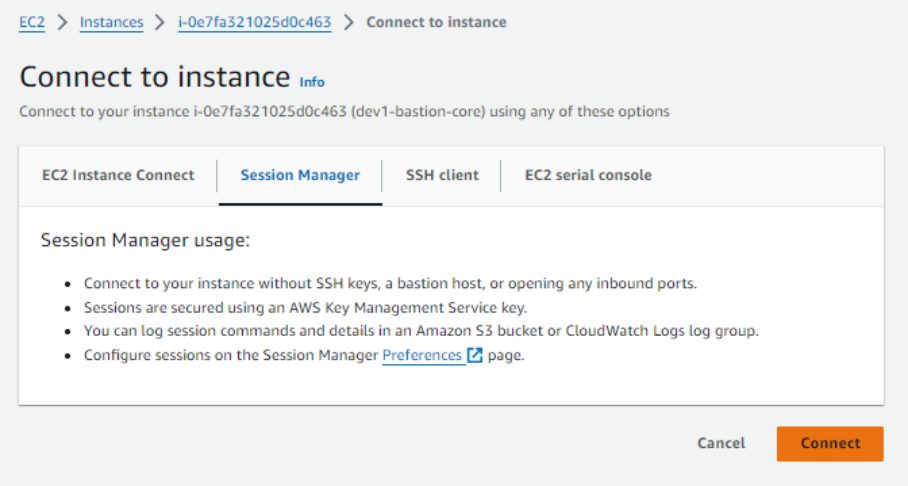


Fuente. Autoría Propia

Para la implementación del análisis de código mediante Codeguru Profiler para el servicio de Reglas de Xsell (gg-xsell-rules-service) se incluye la configuración y la inicialización del CodeGuru Profiler ubicándolo en el método de inicio main. Esto con el fin de que, al iniciar la aplicación, se garantiza que el Codeguru Profiler se inicialice y comience a recopilar datos de rendimiento tan pronto como la aplicación se ejecute. Una vez, cumplidas estas tareas se procede a configurar el host bastión del AWS para él envió de solicitudes al servicio y realizar pruebas de carga.

Figura 19

Configuración de AWS Bastión



Fuente. Autoría Propia

Para el plan piloto se optó por realizar una prueba de carga con 10.000 solicitudes sobre el servicio especificado y se monitoreo tanto la utilización de CPU como de memoria sobre el servidor

Figura 20

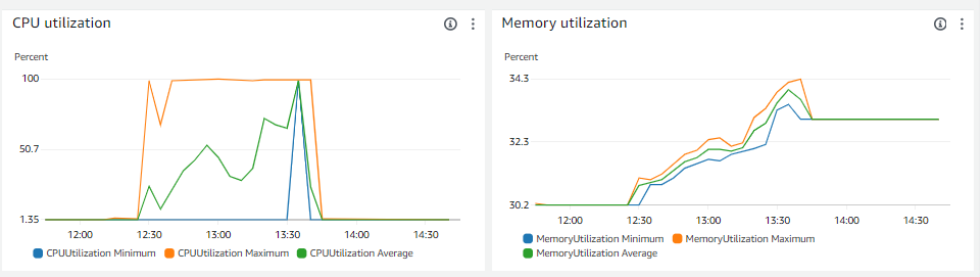
Prueba de Carga

```
Server Software:
Server Hostname:        general-alb.cob.cobiscloud.int
Server Port:            8084

Document Path:          /cobis/xsell/business-rules/actuator/health/
Document Length:        294 bytes

Concurrency Level:      10
Time taken for tests:   22.622 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      6900000 bytes
HTML transferred:       2940000 bytes
Requests per second:    442.04 [#/sec] (mean)
Time per request:       22.622 [ms] (mean)
Time per request:       2.262 [ms] (mean, across all concurrent requests)
Transfer rate:          297.86 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        1    1   0.4      1      14
Processing:     3   22  30.1      6     108
Waiting:        3   22  30.1      5     108
Total:          3   23  30.1      6     108

Percentage of the requests served within a certain time (ms)
  50%      6
  66%      9
  75%     15
  80%     71
  90%     79
  95%     81
  98%     85
  99%     88
 100%    108 (longest request)
```

Fuente. Autoría Propia

Figura 21

Prueba de Carga

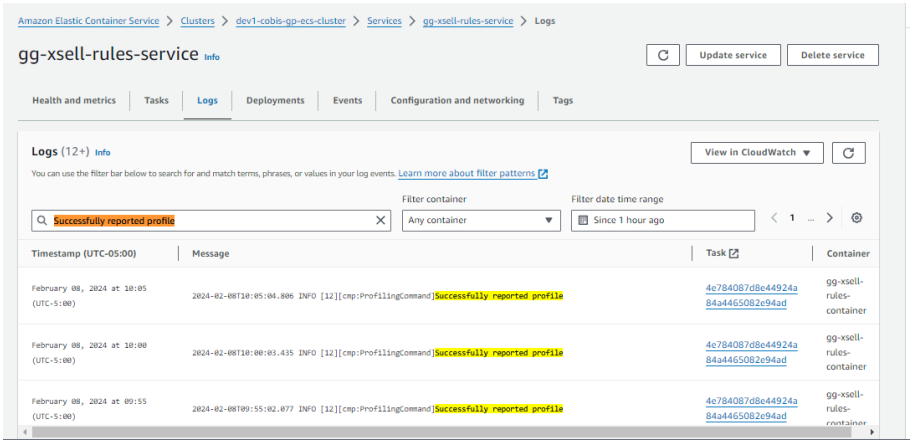


Fuente. Autoría Propia

Una vez finalizada la realización de la prueba de carga, se procede a validar los logs del servicio de reglas de Xsell, para garantizar que el AWS CodeGuru Profiler se inicializo correctamente. Al igual que el agente se encuentra operativo y que está enviando datos al Profiler se está realizando de forma correcta.

Figura 22

Monitoreo de Utilización de CPU y Memoria en Prueba de Carga

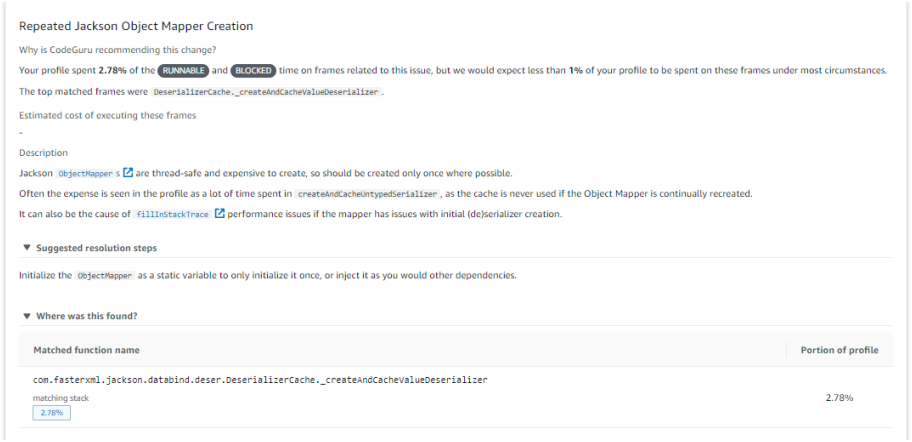


Fuente. Autoría Propia

Posteriormente, ya se pude entrar a validar y analizar los hallazgos del informe de AWS Codeguru. De igual manera, examinar sus recomendaciones y sugerencias. Como resultado del informe de AWS Codeguru se recomienda evitar la recreación innecesaria de clientes de servicios AWS SDK. Un 2.78% del tiempo de ejecución se gasta en marcos relacionados con este problema, lo cual es más alto de lo esperado. La creación repetida de clientes de servicios AWS SDK, como DynamoDB, puede ser una pérdida de tiempo de CPU. Se recomienda crear solo uno o pocos clientes de servicios AWS SDK y reutilizarlos en toda la aplicación, preferiblemente a través de la inyección de dependencias o el almacenamiento en caché.

Figura 23

Logs de Microservicio para garantizar inicialización de AWS CodeGuru

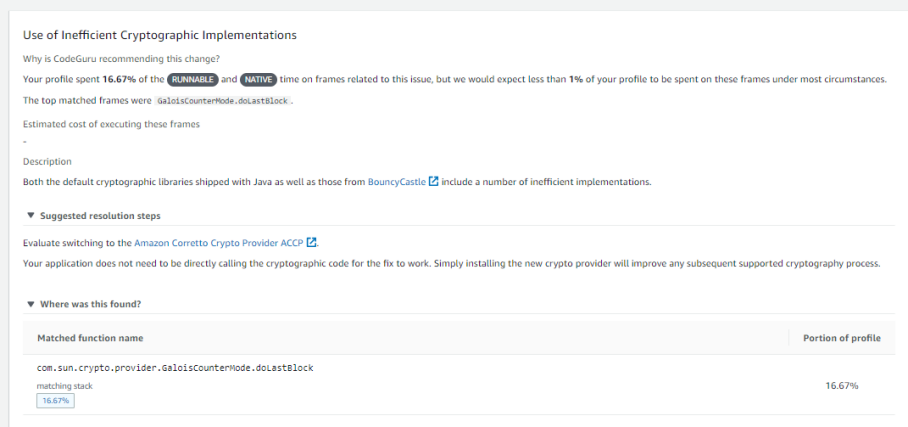


Fuente. Autoría Propia

De igual manera, recomienda evitar la creación repetida de Object Mappers de Jackson. El 2.78% del tiempo de ejecución se gasta en marcos relacionados con este problema, más de lo esperado. Los Object Mappers de Jackson son seguros para subprocesos y costosos de crear, por lo que se deben crear solo una vez cuando sea posible. La creación repetida puede causar un gasto innecesario de tiempo de CPU y problemas de rendimiento. Se sugiere inicializar el ObjectMapper como una variable estática o inyectarlo como otras dependencias para evitar este problema. Por último, en días posteriores se procedió a realizar una nueva prueba de carga en donde con el historial de ejecuciones previas el AWS CodeGuru fue capaz de evidenciar nuevos hallazgos y generar nuevas recomendaciones

Figura 24

Informe AWS CodeGuru



Fuente. Autoría Propia

Como resultado de este nuevo análisis, el informe recomienda cambiar el uso de implementaciones criptográficas ineficientes, ya que una parte significativa del tiempo de ejecución (16.67%) se gasta en marcos relacionados con este problema. La función GaloisCounterMode.doLastBlock fue identificada como la principal causa de este gasto. Se sugiere evaluar el cambio al Amazon Corretto Crypto Provider (ACCP), ya que proporciona implementaciones más eficientes. Este cambio puede mejorar el rendimiento de las operaciones criptográficas en la aplicación, incluso si no se llama directamente al código criptográfico.

Conclusiones

La conclusión principal de esta monografía es que el uso e integración de machine learning en el testeo de software, tiene un impacto significativo en varios aspectos y fases del proceso de pruebas de software, logrando potenciar y complementar la ejecución de planes y esquemas de pruebas, contribuyendo con ello una mejora considerable en la calidad del producto final. a continuación, se presentan conclusiones obtenidas.

La revisión del estado del arte mostró que Machine Learning (ML) está transformando el testeo de calidad de software a través de la automatización y mejora de la eficiencia en la detección de errores, generación de casos de prueba y optimización del código. Las principales tendencias incluyen la integración de aprendizaje supervisado para la detección automática de defectos, y técnicas de aprendizaje no supervisado para la identificación de patrones y anomalías en datos.

A pesar de los avances, la integración de Machine Learning presenta desafíos. La calidad y volumen de datos de entrenamiento sigue siendo un reto, ya que la efectividad del modelo depende en gran medida de la calidad de los datos suministrados. Además, existe una curva de aprendizaje importante para los equipos de pruebas tradicionales, que deben adaptarse a estas nuevas tecnologías. En algunos casos, los modelos presentan problemas de sobreajuste (overfitting) al intentar predecir fallos en aplicaciones complejas, lo que requiere ajustes constantes.

El estado del arte en Machine Learning aplicado al testeo de calidad de software muestra que la evolución de estas técnicas ha impactado positivamente en diversas áreas del ciclo de pruebas, con modelos que permiten reducir la intervención manual y mejorar la cobertura de pruebas. Sin embargo, los estudios también subrayan la necesidad de un mayor control sobre la fiabilidad de los datos y la necesidad de herramientas más flexibles que puedan adaptarse a entornos cambiantes y requisitos específicos del software en desarrollo.

Otro punto relevante que se puede extraer de este apartado es la importancia de la elección adecuada de técnicas y modelos de Machine Learning para diferentes fases del testeo de calidad de software. La selección correcta de estos modelos no solo impacta en la precisión de la detección de defectos, sino también en la optimización de los recursos y el tiempo dedicados a las pruebas. Diferentes técnicas de Machine Learning tienen sus propias fortalezas y debilidades en diversos contextos. Algunos algoritmos pueden ser más efectivos en la identificación de defectos mediante el análisis de patrones en datos históricos, y otros pueden ser más adecuados para la generación automática de casos de prueba y la evaluación de escenarios de prueba complejos. Este enfoque diferenciado permite una mayor flexibilidad y adaptabilidad en el testeo, maximizando la eficacia de las pruebas y mejorando la calidad del software.

El estudio identificó que las técnicas más efectivas para el testeo de calidad de software incluyen el uso de algoritmos de clasificación como los árboles de decisión para priorizar casos de prueba, y algoritmos de agrupación para detectar áreas de código problemáticas. En cuanto a las herramientas, SonarQube destaca por su capacidad de realizar análisis estático de código, detectando defectos y problemas de mantenibilidad con precisión. Otras herramientas,

como Amazon DevOps Guru y Testium, permiten integrar machine learning para optimizar la cobertura de pruebas y priorizar áreas críticas del software. Estas herramientas demuestran su aplicabilidad en distintas fases del ciclo de vida del software, particularmente en la automatización de pruebas funcionales y de regresión, reduciendo tiempos y costos.

Tras la implementación de *SonarQube* en *Cobis-Topaz*, se logró una reducción significativa en la aparición de errores críticos en producción. Los análisis de 2020 y 2024 muestran una disminución de más del 50 % en errores críticos y vulnerabilidades de seguridad. Además, se optimizó la capacidad de detección de anomalías mediante algoritmos de *Machine Learning*, lo que permitió identificar fallos críticos en tiempo real y reducir duplicaciones de código, contribuyendo así a una mayor eficiencia y calidad en el desarrollo del software.

También se ve que la implementación de SonarQube permitió reducir la deuda técnica a menos del 5% del tiempo estimado para la corrección de errores acumulados.

La cobertura de código se incrementó del 65 % en 2020 al 85 % en 2024, facilitando una detección más efectiva de defectos y una mayor seguridad y calidad del software. Este avance se logró mediante la automatización de pruebas, que permitió una ejecución constante y exhaustiva de los casos de prueba, así como la integración de Machine Learning, que optimizó la detección de vulnerabilidades y errores críticos al analizar patrones y priorizar áreas de riesgo en el código. Juntos, estos enfoques contribuyeron a la robustez del producto, elevando los estándares de calidad y seguridad en el desarrollo del software

El uso de *Machine Learning* y herramientas como *SonarQube* contribuyó a una mejora del 20% en la productividad, al facilitar tanto la reducción en el tiempo de retrabajo como la detección temprana de problemas. Esto permitió que el equipo pudiera centrarse en otras áreas clave del desarrollo, optimizando el proceso de manera integral.

En resumen, la aplicación de machine learning en el testeo de calidad de software representa un avance significativo en la industria, ofreciendo mejoras en la precisión, eficiencia y efectividad del proceso de pruebas. Estos progresos no solo benefician a los desarrolladores y testers, sino también a los usuarios finales, al asegurar productos de software más robustos y confiables.

Referencias Bibliográficas

Alharbi, Z. (2023). A Sustainable Price Prediction Model for Airbnb Listings Using Machine Learning and Sentiment Analysis. *Sustainability*, 13159. doi:10.3390/su151713159

Amalfitano, D., Fasolino, A. R., Tramontana, P., & De Carmine, S. (2020). *Test Generation as a Learning Problem: Towards Intelligent Software Testing.* Napoli: Università Federico II Napoli.

Amazon Web Services. (2024). *AWS Documentation*. Obtenido de Amazon SageMaker - Developer Guide: https://docs.aws.amazon.com/sagemaker/latest/dg/whatis.html

Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., . . . Zimmermann, T. (2019). Software Engineering for Machine Learning: A Case Study. *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP).* Montreal.

Artistizábal, M. (26 de Abril de 2024). *Pragma*. Obtenido de Devops Guru: usa machine learning para monitoreo de activos digitales: https://www.pragma.co/es/blog/devops-guru-machine-learning-para-el-monitoreo-proactivo-de-activos-digitales

AWS Política gestionada. (2024). *Amazon AWS Documentation.* Obtenido de AWS Política gestionada: Guía de referencia: https://docs.aws.amazon.com/es_es/aws-managed-policy/latest/reference/aws-managed-policy.pdf

Baquero, L. (2023). *Guía rápida de Fundamentos del Testing.* Miami: Asociación Internacional de Calidad de Software (AICS).

Bellman, R. (1961). *Adaptive Control Processes: A Guided Tour.* New Jersey: Princeton University Press.

Berrio, V. (01 de Mayo de 2024). *BetaBrains*. Obtenido de https://www.betabrains.io/

Beta Arrays. (25 de Julio de 2023). *Medium*. Obtenido de Leveraging AI and Machine Learning in Automation Testing: https://medium.com/@betaarrays/leveraging-ai-and-machine-learning-in-automation-testing-4a8a3ed38faa

Cahuasa, P. B. (27 de Febrero de 2024). *Universidad Franz Tamayo*. Obtenido de La IA Aprende Como Un Niño, En Base A La Repetición Y La Observación: https://unifranz.edu.bo/blog/la-ia-aprende-como-un-nino-en-base-a-la-repeticion-y-la-observacion/

Calvo, J., Guzmán, M., & Ramos, D. (2018). *Management Solutions*. Obtenido de Machine Learning, una pieza clave en la transformación de los modelos de negocio: https://www.managementsolutions.com/es/publicaciones-y-eventos/informes-sectoriales/white-papers/machine-learning-una-pieza-clave-en-la-transformacion-de-los-modelos-de-negocio

Campbell, G. A., & Papapetrou, P. (2013). *SonarQube in Action.* Manning Publications.

Cardellino, F. (21 de Marzo de 2021). *FreeCodeCamp*. Obtenido de La guía definitiva del paquete NumPy para computación científica en Python: https://www.freecodecamp.org/espanol/news/la-guia-definitiva-del-paquete-numpy-para-computacion-cientifica-en-python/

Carmona, P. (20 de Abril de 2020). *Medium*. Obtenido de Data Visualization con pandas y seaborn: https://medium.com/ironhack/data-visualization-con-pandas-y-seaborn-1044906af34f

Catal, C., Sevim, U., & Diri, B. (2009). Clustering and Metrics Thresholds Based Software Fault Prediction of Unlabeled Program Modules. *2009 Sixth International Conference on*

Information Technology: New Generations, (págs. 199-204). Las Vegas. doi:10.1109/ITNG.2009.12

Chacón, J. M., García, A., Hervera, M., Barrios, L., & Moreno, J. (08 de Agosto de 2022). *Juice Studio*. Obtenido de ¿Cómo cambia el machine learning las pruebas de software?: https://juice-studio.com/como-esta-cambiando-el-machine-learning-el-futuro-de-las-pruebas-de-software/

Clark, J. A., & Jacob, J. L. (1994). Testing Software Using Finite State Models. *IEEE Transactions on Software Engineering, 20*, 217–237.

Cortés , D. E. (24 de Agosto de 2023). *Medium*. Obtenido de Primeros pasos en Jupyter Notebook con Python: https://medium.com/@diego.coder/primeros-pasos-en-jupyter-notebook-con-python-bff43c73621

Cuevas, E., Avalos, O., Diaz, P., Valdivia, A., & Pérez, M. (2021). *Introducción al Machine Learning con MATLAB.* Madrid: Alpha Editorial.

Dark, S. (2019). *Aprendizaje Automático: La Guía Definitiva Para Principiantes Para Comprender el Aprendizaje Automático.*

Deque Systems. (01 de Octubre de 2024). *Deque Systems*. Obtenido de Axe DevTools and Artificial Intelligence (AI): https://www.deque.com/axe/devtools/ai/

Desikan, S., & Ramesh, G. (2009). *Software Testing: Principles and Practice.* Pearson Education Canada.

Dinukamarlon. (02 de Septiembre de 2023). *Medium*. Obtenido de Google Maps with Machine Learning: https://medium.com/@dinukamarlon/google-maps-with-machine-learning-d9b51db22b49

Durelli, V., Durelli, R., Borges, S., Endo, A., Eler, M., Dias, D., & Guimarães, M. (10 de Septiembre de 2019). Machine Learning Applied to Software Testing: A Systematic Mapping Study. *IEEE Transactions on Reliability, 68*(3), 1189 - 1212.

Ekanadham, C. (22 de Marzo de 2018). *Using Machine Learning to Improve Streaming Quality at Netflix*. Obtenido de Netflix Technology Blog: https://netflixtechblog.com/using-machine-learning-to-improve-streaming-quality-at-netflix-9651263ef09f

ElEmam, M. (29 de Marzo de 2022). *Medium*. Obtenido de What is Software Testing and What is SonarQube: https://medium.com/@Mohamed-ElEmam/what-is-software-testing-and-what-is-sonarqube-bcd668a46b93

Fernández Villar, Á. (02 de Enero de 2023). *CaixaBak*. Obtenido de Testing automatizado de software basado en IA: https://www.caixabanktech.com/es/blogs/testing-automatizado-de-software-basado-en-ia/#

Fernandez, O. (14 de Julio de 2024). *Aprender Big Data*. Obtenido de Qué es Splunk y por qué Debería Importarte: https://aprenderbigdata.com/splunk/

Google Developer Program. (01 de Septiembre de 2021). *TensorFlow*. Obtenido de Introducción a TensorFlow: https://www.tensorflow.org/learn?hl=es-419

Google Developer Program. (24 de Enero de 2022). *TensorFlow*. Obtenido de Keras: https://www.tensorflow.org/guide/keras?hl=es-419

Google Developer Program. (20 de Junio de 2023). *Google for Developers*. Obtenido de Machine Learning - Pruebas y depuración: https://developers.google.com/machine-learning/testing-debugging?hl=es-419

Hoe, E. (29 de Agosto de 2023). *LinkedIn*. Obtenido de La implementación de pruebas automatizadas con AI: https://www.linkedin.com/pulse/la-implementaci%C3%B3n-de-pruebas-automatizadas-con-ai-eduardo-henriquez/

Jacinto, A. (9 de Septiembre de 2022). *Startechup*. Obtenido de Historia del Aprendizaje Automático: La Cronología Completa: https://www.startechup.com/es/blog/machine-learning-history/

Kaner, C., Bach, J., & Pettichord, B. (2002). *Lessons Learned in Software Testing: A Context-Driven Approach.* New York: Wiley.

Khoshgoftaar, T., Allen, E., Hudepohl, J., & Aud, S. (1997). Application of neural networks to software quality modeling of a very large telecommunications system. *IEEE transactions on neural networks*, 902-909. doi:10.1109/72.595888

Kirilenko, Í. (17 de Junio de 2024). *Parasoft*. Obtenido de ¿Qué es la inteligencia artificial en las pruebas de software?: https://es.parasoft.com/blog/what-is-artificial-intelligence-in-software-testing/

Kosowski, D. (27 de Septiembre de 2023). *NetGuru*. Obtenido de What's the Role of Quality Assurance in Machine Learning Projects?: https://www.netguru.com/blog/whats-the-role-of-quality-assurance-specialists-in-machine-learning-projects

Larios Calleja, J. (15 de Abril de 2021). *En Mi Local Funciona*. Obtenido de Las tripas de SonarQube: Métricas y cómo calcula el Rating: https://www.enmilocalfunciona.io/las-tripas-de-sonarqube-metricas-y-como-calcula-el-rating-parte-2/#:~:text=Reliability%20%2F%20Fiabilidad%3A%20Capacidad%20de%20un,base%20a%20las%20Bugs%20encontrados.

Lessmann, S., Baesens, B., Mues, C., & Pietsch, S. (2008). Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. *IEEE Transactions on Software Engineering, 34*, 485 - 496. doi:10.1109/TSE.2008.35

Maisueche Cuadrado, A. (2019). *Utilización Del Machine Learning En La Industria 4.0.* Valladolid: Escuela de Ingenierías Industriales - Universidad de Valladolid.

Martín, S. (06 de Mayo de 2023). *Testealo*. Obtenido de Desarrollo de pruebas automatizadas con Appium – Guía completa: https://testealo.com/desarrollo-de-pruebas-automatizadas-en-android-con-appium-guia-completa/

Menzies, T., Greenwald, J., & Frank, A. (11 de Diciembre de 2006). Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE Transactions on Software Engineering, 33*(1), 2-13.

Mera Paz, J. A. (2016). *Análisis del proceso de pruebas.* Popayán: Universidad Cooperativa de Colombia.

Montalván Vélez, C. L., Mogrovejo Zambrano, J. N., Romero Vitte, I. J., & Pinargote Carrera, M. L. (Enero de 2024). Introducción a la Inteligencia Artificial: Conceptos Básicos y Aplicaciones Cotidianas. *Journal of Economic and Social Science Research, 4*, 173-183. doi:10.55813/gaea/jessr/v4/n1/93

Naik, K., & Tripathy, P. (2008). *Software Testing and Quality Assurance: Theory and Practice.* New Jersey: Wiley-Spektrum.

Navas Ajenjo, A. (2020). *Estudio, análisis y detección de ciertas anomalías en el contexto de seguridad informática.* Madrid: Escuela Politécnica Superior Universidad Autónoma de Madrid.

Neora. (10 de Enero de 2022). *Neora*. Obtenido de ¿Cómo utilizar datos, análisis y machine learning en nuestra automatización de pruebas de software?: https://www.neora-consulting.com/neora-como-utilizar-datos-analisis-y-machine-learning-en-nuestra-automatizacion-de-pruebas-de-software/

Noorian, M., Bagheri, E., & Du, W. (2011). *Machine Learning-based Software Testing: Towards a Classification Framework.* Fredericton: University of New Brunswick.

Orozco Díaz, J. A. (2022). *Implementación de pruebas funcionales automatizadas con Selenium, Appium y Rest-Assured en la empresa Valid Colombia.* Santa Marta: Universidad del Magdalena.

Pineda Pertuz, C. M. (2021). *Aprendizaje Automático Y Profundo En Python*. Bogota: Ediciones de la U.

Piper, B. (13 de Septiembre de 2022). *Snyk*. Obtenido de How Spotify uses Snyk to secure the SDLC: https://snyk.io/blog/how-spotify-uses-snyk-to-secure-the-sdlc/

Pizette, L. (2018). *Cómo comenzar con Machine Learning: consejos de expertos de vanguardia.*

Postman. (01 de Octubre de 2024). *Postman*. Obtenido de ML Tools for Developer Professionals Documentation: https://www.postman.com/ai-in-postman/machine-learning-tools-for-developer-professionals/overview

Quiroga Saldaña, J. D. (03 de Abrol de 2023). *LinkedIn*. Obtenido de La librería Pandas de Python: Ventajas, desventajas y consejos prácticos: https://www.linkedin.com/pulse/la-librer%C3%ADa-pandas-de-python-ventajas-desventajas-y-quiroga-salda%C3%B1a/

Realpe Mancipe, L. C. (2019). *Metodología para el proceso de pruebas de software: Un estudio de caso enfocado a una empresa de software colombiana.* Bogotá: Universidad Nacional de Colombia.

Red Hat. (09 de Mayo de 2024). *Red Hat*. Obtenido de Casos prácticos de inteligencia artificial y machine learning: https://www.redhat.com/es/topics/cloud-computing/ai-ml-use-cases

Riggins, J. (04 de Febrero de 2019). *The New Stack*. Obtenido de Facebook's Tool for Automated Testing at 2 Billion Users Scale: https://thenewstack.io/facebooks-tool-for-automated-testing-at-2-billion-users-scale/

Sánchez Peño, J. M. (2015). *Pruebas de Software. Fundamentos y Técnicas.* Madrid: Universidad Politécnica de Madrid.

Sánchez, A., Segura, S., & Ruiz-Cortés, A. (2013). Priorización de casos de prueba. Avances y retos. *Novatica – Revista de la Asociación de Técnicos de Informática*, 26-32.

Selenium Developers Group. (21 de Mayo de 2024). *Selenium*. Obtenido de Selenium Projects: https://www.selenium.dev/projects/

Sharygina, N., & Peled, D. (2001). A Combined Testing and Verification Approach for Software Reliability. *FME 2001: Formal Methods for Increasing Software Productivity, International Symposium of Formal Methods Europe.* Berlin.

SonarSource. (Marzo de 2020). *Sonar Community*. Obtenido de Sonarqube Calculation Coverage: https://community.sonarsource.com/t/sonarqube-calculation-coverage

Spillner, A., Linz, T., & Schaefer, H. (2014). *Software Testing Foundations* (4 ed.). Heidelberg, Alemania: Rocky Nook Computing.

Stott, L. (21 de Marzo de 2019). *Microsoft Community Hub*. Obtenido de Using Azure for Machine Learning: https://techcommunity.microsoft.com/t5/educator-developer-blog/using-azure-for-machine-learning/ba-p/381215

Testers.AI. (2024). *Test.AI: Intelligent Test Automation*. Obtenido de https://test.ai/

The PyTorch Foundation. (2023). *PyTorch*. Obtenido de PyTorch documentation: https://pytorch.org/docs/stable/index.html

Trespaderne, F. M., & Mazaeda Echevarría, R. (2020). *Universidad de Valladolid*. Obtenido de Introducción a Matplotlib: https://www2.eii.uva.es/fund_inf/python/notebooks/Bibliotecas/01_Introduccion_a_Matplotlib/Introduccion_a_Matplotlib.html

Wagner, E. (2012). How BMW Improved Software Quality and Agility of a Mission Critical Software System. (A. von Zitzewitz, Entrevistador) Obtenido de How BMW Improved Software Quality and Agility of a Mission Critical Software System.

Winters, T., Manshreck, T., & Wright, H. (2022). *Ingeniería de software en Google: Lecciones sobre programación aprendidas a lo largo del tiempo.* Marcombo.

Xie, X., Zhang, Z., Chen, T. Y., Liu, Y., Poon, P.-L., & Xu, B. (Diciembre de 2020). METTLE: A METamorphic Testing Approach to Assessing and Validating Unsupervised Machine Learning Systems. *IEEE Transactions on Reliability, 69*(4), 1293 - 1322. doi:10.1109/TR.2020.2972266

Zapata, J. R. (14 de Noviembre de 2023). *Jose Ricardo Zapata*. Obtenido de Visualizacion Interactiva con Plotly: https://joserzapata.github.io/courses/python-ciencia-datos/visualizacion/plotly/

Zapier Editorial Team. (11 de Marzo de 2022). *Zapier*. Obtenido de Zapier data report: The rise of no-code: https://zapier.com/blog/no-code-report/