UNIVERSIDAD MAYOR DE SAN ANDRÉS

FACULTAD DE CIENCIAS PURAS Y NATURALES

CARRERA DE INFORMÁTICA

PERFIL DE TESIS DE GRADO

MÉTRICA PARA LA EVALUACIÓN DE LA CALIDAD DEL CÓDIGO JAVASCRIPT EN TÉRMINOS DE RENDIMIENTO Y MANTENIBILIDAD

Tesis de Grado para obtener el Título de Licenciatura en Informática

Mención Ingeniería de Sistemas Informáticos

POR: AARON JOEL LIMACHI QUISPE

TUTOR: M.SC. ALDO VALDEZ ALVARADO

LA PAZ – BOLIVIA

Mayo, 2023

INDICE

1. INTRODUCCIÓN

El código *JavaScript* desempeña un papel crucial en el desarrollo de aplicaciones web modernas, y su calidad tiene un impacto significativo en el rendimiento y la mantenibilidad de estas aplicaciones según Flanagan (2020). Un código *JavaScript* de baja calidad puede afectar negativamente el rendimiento de la aplicación, provocar problemas de seguridad y dificultar la capacidad de mantenimiento y evolución del software. Por lo tanto, es fundamental contar con herramientas y enfoques que permitan evaluar de manera efectiva la calidad del código *JavaScript* en términos de rendimiento y mantenibilidad.

El objetivo principal de esta investigación es desarrollar una unidad de medida que permita evaluar y cuantificar la calidad del código *JavaScript* en los términos de rendimiento y mantenibilidad. Esta unidad de medida proporcionará una base objetiva para la evaluación y comparación de diferentes fragmentos de código *JavaScript*, lo que facilitará la identificación de áreas de mejora y permitirá una toma de decisiones informada durante el desarrollo de aplicaciones web.

La relevancia del presente trabajo radica en su potencial impacto en la industria del desarrollo de software. Una evaluación más precisa de la calidad del código *JavaScript* en términos de rendimiento y mantenibilidad permitirá a los desarrolladores crear aplicaciones de fácil escalabilidad y mantenimiento. Esto, a su vez, beneficiará las empresas al reducir costos y mejorar la satisfacción del usuario.

En la presente investigación, se utilizará un enfoque basado en la revisión de la literatura existente sobre las mejores prácticas de desarrollo de código *JavaScript*. Además, se desarrollará una unidad de medida específica que combinará y cuantificará las métricas relevantes para evaluar la calidad del código *JavaScript* en términos de rendimiento y mantenibilidad. Posteriormente, se llevará a cabo un análisis exhaustivo de casos de estudio y experimentos para validar la unidad de medida propuesta y demostrar su correcto funcionamiento en la evaluación de la calidad del código *JavaScript*.

2. ANTECEDENTES

Desde el nacimiento de *JSLint* en el 2002 que fue una de las primeras herramientas de evaluación de código para JavaScript se ha buscado el control de calidad del código para entregar mejores resultados. Esta herramienta analizaba el código en busca de errores y violaciones de estilo. Pasando por SonarQube en el 2008, plataforma enfocada en el análisis estático de código. JSHint en 2011, que fue una bifurcación de *JSLint* con mayor flexibilidad, para conocer a ESLint años más tarde en el 2013 cuya popularidad está presente hasta la actualidad. Paralelamente el surgimiento de TSLint, cuyo enfoque fue para el código *TypeScript* un superconjunto de JavaScript desarrollado por Microsoft. O alternativas a herramientas de evaluación de código como *Prettier* lanzada en 2017 pero que cumplían similares propósitos.

En los últimos años, el lenguaje de programación *JavaScript* ha experimentado un crecimiento significativo y se ha convertido en uno de los principales lenguajes utilizados en el desarrollo de aplicaciones web según el último reporte de la página web "*State Of JavaScript"* (2023). Sin embargo, a medida que la complejidad de las aplicaciones web aumenta, también lo hace la importancia de garantizar la calidad del código *JavaScript* en términos de rendimiento y mantenibilidad. En este sentido, investigaciones recientes han abordado este problema y han propuesto enfoques y métricas para evaluar y mejorar la calidad del código *JavaScript*.

Según Zozas et al. (2022) "La flexibilidad que proporciona el lenguaje puede acelerar el desarrollo de aplicaciones, pero también plantear amenazas a la calidad del producto de software final, por ejemplo, introduciendo Deuda Técnica (*TD*). La *TD* refleja el coste adicional de las actividades de mantenimiento del *software* para implementar nuevas funciones, que se producen debido a soluciones mal desarrolladas."

El rendimiento es uno de los aspectos críticos en el desarrollo de aplicaciones web. En este sentido, diversos estudios han explorado técnicas para evaluar y mejorar el rendimiento del código *JavaScript*. Por ejemplo, el estudio de Cairo (2018) presentó una Revisión de la Literatura Sistemática (*SLR)* para identificar *code smells* que afectan el

calidad del código dificultando su comprensión y mantenimiento. Asimismo, el trabajo de Ardito et al. (2020) realizó una revisión sistemática de la literatura (*SLR)* bajo una perspectiva basada en herramientas sobre métricas de mantenibilidad del código de software.

La revisión de Ardito et al. (2020) encontró que "Los resultados también ponen de relieve la falta de cobertura de herramientas para algunas métricas en lenguajes de programación de uso común y la cobertura mínima de métricas para lenguajes de programación más nuevos o menos populares".

Por otro lado, existen también estudios sobre la refactorización del código en *JavaScript.* Como el estudio realizado por Linda (2021) en el que mediante el uso del análisis estático del código se planteó mostrar el efecto que tiene la refactorización de código en las aplicaciones web programadas con *JavaScript.* Y el impacto que tuvo en esta la mantenibilidad y legibilidad del código. Sus resultados determinaron que no se pueden generalizar los resultados de la refactorización del código, pero también pusieron en expuesto un código mantenible permite agregar mejoras sin incrementar el nivel de complejidad.

Tomando en cuenta también los intentos de *ECMAScript* por normar el lenguaje *JavaScript*. También conocido como ES, *ECMAScript* es un estándar de lenguaje de programación que se utiliza ampliamente en el desarrollo de aplicaciones web. Fue creado con el objetivo de estandarizar y mejorar JavaScript, el lenguaje de programación utilizado en el lado del cliente en navegadores web.

La historia de *ECMAScript* se remonta a principios de la década de 1990 cuando Brendan Eich creó *JavaScript* en *Netscape Communications Corporation*. *JavaScript* se convirtió rápidamente en un lenguaje popular para el desarrollo web según TC39 (2023), pero la falta de un estándar formal resultó en inconsistencias entre los navegadores y limitó la portabilidad de las aplicaciones.

En 1996, *Netscape Communications Corporation* y *Sun Microsystems* colaboraron para estandarizar *JavaScript* con el objetivo de unificar su implementación en diferentes navegadores. Esta colaboración dio como resultado la primera edición de *ECMAScript*, conocida como *ECMAScript 1* (*ES1*), publicada en junio de 1997. *ES1* estableció la base del lenguaje y definió características fundamentales como variables, tipos de datos, estructuras de control y funciones.

En los años siguientes según la TC39 (2023), se lanzaron nuevas versiones de *ECMAScript*, cada una introduciendo nuevas características y mejoras al lenguaje. *ECMAScript 2* (*ES2*) fue publicado en junio de 1998 y trajo consigo mejoras en la sintaxis y la adición de objetos regulares. Luego, en diciembre de 1999, se lanzó *ECMAScript 3* (*ES3*), que fue una revisión importante que incluyó nuevas características como manejo de excepciones y mejor soporte para estructuras de datos.

El siguiente hito importante en la historia de *ECMAScript* fue *ECMAScript 5* (*ES5*), lanzado en diciembre de 2009. Esta versión introdujo características esenciales como *JSON* (*JavaScript Object Notation*), métodos de matriz mejorados, acceso a objetos estrictos y la función *bind*[1]. *ES5* fue ampliamente adoptado y sentó las bases para muchas aplicaciones web.

La siguiente gran versión fue *ECMAScript 6* (*ES6*), también conocido como *ECMAScript* 2015, publicado en junio de 2015. ES6 fue una actualización significativa que introdujo una amplia gama de nuevas características, incluyendo variables *let* y *const*, funciones de flecha, clases, módulos, desestructuración y promesas. Estas adiciones mejoraron enormemente la forma en que se podía escribir y organizar el código *JavaScript*.

A partir de *ES6*, el Comité Técnico de *ECMAScript* comenzó a adoptar un enfoque de lanzamiento anual, lo que significa que se publican nuevas versiones de *ECMAScript* cada año con un conjunto de características actualizadas. Esto permite una evolución más rápida del lenguaje y la incorporación de nuevas funcionalidades más rápidamente.

[1] Función del lenguaje JavaScript para anexar elementos

Algunas versiones notables posteriores a *ES6* incluyen *ECMAScript 7* (*ES7*) en 2016, *ECMAScript 8* (*ES8*) en 2017, *ECMAScript 9* (*ES9*) en 2018 y *ECMAScript 10* (*ES10*) en 2019. Cada una de estas versiones ha agregado nuevas características y mejoras para facilitar la escritura de código más limpio y eficiente.

En la actualidad según cita TC39 (2023), la versión más reciente de *ECMAScript* es *ECMAScript 2022* (*ES2022*), lanzado en junio de 2022. *ES2022* incluye características como el operador de tubería (|>), el patrón de correspondencia (*match*), las instrucciones de registro y las funciones privadas, entre otras.

ECMAScript ha evolucionado a lo largo de los años para convertirse en uno de los lenguajes de programación más utilizados en el desarrollo web. Su continua mejora y adopción de nuevas características refleja el deseo de mantenerse al día con las demandas cambiantes de los desarrolladores y las tecnologías web emergentes. En el presente trabajo se hará un empleo indistinto del término *JavaScript* y *ECMAScript* para referirse al estándar del lenguaje de programación.

3. PLANTEAMIENTO DEL PROBLEMA

3.1. PROBLEMA CENTRAL

¿Cómo determinar la calidad del software escrito en el lenguaje de programación *JavaScript* en términos de rendimiento y mantenibilidad?

3.2. PROBLEMAS SECUNDARIOS

- La flexibilidad de desarrollo en el lenguaje *JavaScript,* causa el surgimiento de amenazas a la calidad del producto del *software* final.
- Los *code smells*[2] en los productos de software desarrollados con *JavaScript,* causan un difícil mantenimiento y comprensión del código.

[2] Término empleado para señalar secciones de código con posibles problemas futuros.

- Las herramientas de análisis de código en JavaScript no contemplan un porcentaje considerable de métricas, lo que causa una falta de visón del panorama de los productos de *software* desarrollados con *JavaScript.*
- La evaluación de la mantenibilidad en el código no puede llevarse a cabo mediante métodos cuantitativos, lo cual da lugar a una ambigüedad en cuanto a los criterios aceptables o corregibles en el código.

4. DEFINICIÓN DE OBJETIVOS

4.1. OBJETIVO GENERAL

Desarrollar una métrica para la evaluación la calidad del código estático en JavaScript

4.2. OBJETIVOS ESPECÍFICOS

- Analizar los casos en que la flexibilidad del código JavaScript ha ocasionado percances en el rendimiento del código.
- Identificar los múltiples tipos de *code smells* registrados en el código *JavaScript*.
- Identificar los criterios de calidad relevantes para la evaluación del código en JavaScript en la literatura existente.
- Plantear una unidad de medida basada en los criterios identificados para evaluar el código en JavaScript.
- Validar la unidad de medida desarrollada mediante la comparación con otras herramientas de evaluación de código en JavaScript en términos de rendimiento y mantenibilidad.
- Evaluar la aplicabilidad de la unidad de medida en proyectos de software reales.

5. HIPÓTESIS

El desarrollo de una métrica específica, basada en criterios objetivo, para evaluar la calidad del código JavaScript en términos de rendimiento y mantenibilidad permitirá identificar áreas de mejora, así como también proporcionará recomendaciones concretas para optimizar el rendimiento y mejorar la mantenibilidad del código.

5.1. OPERACIONALIZACIÓN DE VARIABLES

Variable independiente: Código *JavaScript*

Variable dependiente: Rendimiento del código JavaScript

- Indicadores operativos: Tiempo de carga de la página, tiempo de respuesta del sistema, velocidad de ejecución del código, utilización de recursos del sistema.
- Medición: Se mide el tiempo de carga de la página utilizando herramientas de análisis de rendimiento, como *Google PageSpeed Insights*[3] o *WebPageTest*. El tiempo de respuesta del sistema se puede medir utilizando técnicas de *profiling* o herramientas de monitoreo de rendimiento. La velocidad de ejecución del código se puede medir utilizando herramientas de análisis estático y dinámico, como *linters* o analizadores de código. La utilización de recursos del sistema se puede medir mediante herramientas de monitoreo de recursos, como el Administrador de tareas del sistema operativo.

Variable dependiente: Mantenibilidad del código JavaScript

- Indicadores operativos: Cohesión del código, acoplamiento del código, legibilidad del código, mantenibilidad del código, modularidad del código.
- Medición: La cohesión del código se puede medir utilizando métricas de cohesión, como el índice de cohesión de LCOM4. El acoplamiento del código se puede medir utilizando métricas de acoplamiento, como el índice de acoplamiento de *Efferent*. La legibilidad del código se puede medir utilizando métricas de complejidad, como el índice de mantenibilidad de código. La mantenibilidad del código se puede medir mediante encuestas o cuestionarios a los desarrolladores que evalúen la facilidad de mantener y modificar el código. La modularidad del código se puede medir utilizando métricas de modularidad, como la cantidad de módulos independientes y la coherencia entre ellos.

[3] Aplicación web de Google para el análisis de páginas web

6. JUSTIFICACIÓN

6.1. JUSTIFICACIÓN ECONÓMICA

En el contexto económico actual, las aplicaciones web desempeñan un papel crucial en diversos sectores y organizaciones. Una mala calidad del código JavaScript puede ocasionar problemas de rendimiento, como tiempos de carga prolongados o respuestas lentas, lo que puede afectar la experiencia del usuario y reducir la satisfacción del cliente. Además, un código JavaScript de baja calidad puede requerir un mayor esfuerzo y tiempo de desarrollo, así como aumentar los costos de mantenimiento. La disponibilidad de una métrica que permita evaluar la calidad del código JavaScript puede contribuir a optimizar los recursos y reducir los costos asociados con el desarrollo y mantenimiento de aplicaciones web.

6.2. JUSTIFICACIÓN SOCIAL

Las aplicaciones web desempeñan un papel cada vez más importante en la vida diaria de las personas, abarcando áreas como el comercio electrónico, la educación en línea, el entretenimiento y la comunicación. La calidad del código JavaScript utilizado en estas aplicaciones tiene un impacto directo en la experiencia del usuario. Un código JavaScript de alta calidad, que se traduzca en un rendimiento óptimo y una fácil mantenibilidad, puede mejorar la usabilidad de las aplicaciones, brindando a los usuarios una experiencia fluida, eficiente y satisfactoria. Además, una métrica que permita evaluar y mejorar la calidad del código JavaScript puede contribuir a la seguridad y confiabilidad de las aplicaciones web, protegiendo la información personal y garantizando un entorno en línea seguro. De un modo paralelo permitirá un flujo de trabajo más cómodo e informado a los desarrolladores de aplicaciones web.

6.3. JUSTIFICACIÓN CIENTÍFICA

Desde el punto de vista científico, la evaluación de la calidad del código JavaScript y el desarrollo de métricas específicas es un área de investigación en constante evolución. Existe una necesidad de enfoques más precisos y orientados a las características

particulares del lenguaje JavaScript, con el fin de proporcionar herramientas y técnicas que permitan mejorar la calidad del código en términos de rendimiento y mantenibilidad. Al desarrollar una métrica específica y realizar investigaciones en esta área, se puede contribuir al avance del conocimiento en ingeniería de software y ofrecer a los profesionales del desarrollo web herramientas más efectivas para evaluar y mejorar la calidad del código JavaScript.

7. ALCANCES Y LÍMITES

7.1. ALCANCES

A continuación, se presentan los alcances de la presente tesis.

- Para la medición de rendimiento en el código JavaScript se tomarán en cuenta los siguientes indicadores operativos: tiempo de carga de la página, tiempo de respuesta del sistema, velocidad de ejecución del código, utilización de recursos del sistema.
- Para la toma de magnitud de los indicadores en términos de rendimiento se medirá el tiempo de carga de la página utilizando herramientas de análisis de rendimiento, como *Google PageSpeed Insights* y *WebPageTest*.
- El tiempo de respuesta del sistema se medirá utilizando técnicas de *profiling* y herramientas de monitoreo de rendimiento.
- La eficiencia del código se medirá mediante la técnica *Big O Notation.*
- La velocidad de ejecución del código se medirá utilizando herramientas de análisis estático y dinámico, como *linters* y analizadores de código.
- Para la medición de la mantenibilidad se tomarán en cuenta la cohesión del código, acoplamiento del código, la legibilidad del código, la mantenibilidad del código, y la modularidad del código.
- Para la recolección de indicadores en términos de mantenibilidad se tomará en cuenta la cohesión del código que se mide utilizando métricas de cohesión, como el índice de cohesión de LCOM4.

- El acoplamiento del código se medirá empleando métricas de acoplamiento.
- La legibilidad del código se medirá con el uso de métricas de complejidad, como el índice de mantenibilidad de código.
- La mantenibilidad del código se evaluará mediante encuestas y cuestionarios a los desarrolladores que evalúen la facilidad de mantener, así como también modificar el código.
- Se desarrollará las evaluaciones bajo la perspectiva de la programación orientada a objetos.
- La modularidad del código se medirá utilizando métricas de modularidad, como la cantidad de módulos independientes y la coherencia entre ellos.
- Se contempla un método para la evolución progresiva de la métrica conforme a los cambios en los estándares de la industria.

7.2. LÍMITES

Los límites de la tesis postulada son los siguientes

- No se consideran pruebas de conexión a redes.
- No se realizarán pruebas de interfaz de usuario.
- No se consideran pruebas de velocidad en acceso a disco.
- No se mide ninguna forma de conexión a la computación en la nube.
- No se consideran técnicas de evaluación de empleadas en otros lenguajes de programación.
- No se evaluarán de ningún modo conexiones a bases de datos.
- No se consideran pruebas de vulnerabilidades de seguridad informática futuras.

8. APORTES

8.1. TEÓRICO

Se realizará una exhaustiva revisión de la literatura científica y técnica relacionada con la evaluación de la calidad del código JavaScript, métricas de rendimiento y mantenibilidad,

así como enfoques y técnicas utilizadas en proyectos similares. Esto permitirá comprender el estado actual del campo, identificar posibles métricas y métodos existentes, así como establecer una base sólida de conocimiento para desarrollar una nueva métrica.

Se definirá los requerimientos y objetivos específicos para la métrica propuesta, basándose en las necesidades y desafíos identificados en la revisión bibliográfica y en las metas establecidas en la tesis. Esto incluirá determinar qué aspectos de calidad se abordarán, qué tipo de información se capturará y cómo se utilizará la métrica para evaluar el código JavaScript en términos de rendimiento y mantenibilidad.

Utilizando los conocimientos adquiridos en la revisión bibliográfica y los requerimientos establecidos, se procederá al diseño y desarrollo de la métrica propuesta. Esto implicaría definir los criterios de evaluación, establecer las fórmulas y algoritmos necesarios, para luego desarrollar software específico para la implementación y cálculo de la métrica.

Se realizará pruebas para validar la efectividad y utilidad de la métrica propuesta. Esto podría implicar la evaluación de un conjunto diverso de aplicaciones web utilizando la métrica, comparando los resultados con evaluaciones manuales o con otras métricas existentes. También se considerará la realización de estudios de caso y el análisis de correlación con otras métricas establecidas para establecer la validez y fiabilidad de la métrica propuesta.

A medida que se obtengan los resultados de la experimentación y validación, se realizarán mejoras continuas en la métrica y en su implementación. Se llevarán a cabo ajustes en los criterios de evaluación, refinamientos en los algoritmos de cálculo para garantizar su efectividad en la evaluación de la calidad del código JavaScript.

8.2. PRÁCTICO

La métrica presentará una perspectiva de mayor amplitud e información respecto de la calidad de código al integrar la evaluación del rendimiento y la mantenibilidad del código. Está será aplicable al código JavaScript dando como resultado una métrica comprendida entre el cero y la unidad. La misma difiere de ser un *checklist* de características debido su

naturaleza en su desarrollo. Mediante algoritmos de aprendizaje no supervisado y aprendizaje profundo se propone obtener la medida de los fragmentos de código analizados individualmente. Mientras que la evaluación de mantenibilidad será llevada a cabo por una mesa de expertos, misma que se registrará de forma cuantitativa para su integración en los algoritmos de evaluación final.

9. METODOLOGÍA

En términos de investigación la metodología empleada es la siguiente:

En esta tesis se empleará un enfoque de investigación cuantitativa, que se basa en la recopilación y análisis de datos numéricos para obtener resultados objetivos y medibles en relación con la calidad del código JavaScript.

El tipo de investigación elegido es la investigación aplicada, ya que se busca desarrollar una métrica específica para evaluar la calidad del código JavaScript en términos de rendimiento y mantenibilidad, con el propósito de obtener resultados prácticos y aplicables en el campo de la ingeniería de software.

El método a emplear es el método experimental, que implica la realización de experimentos controlados y la recolección de datos para evaluar la eficacia y utilidad de la métrica propuesta en la evaluación de la calidad del código JavaScript.

Los instrumentos que se utilizarán incluyen herramientas de análisis estático y dinámico, herramientas de medición de rendimiento, herramientas de análisis de código fuente.

En complemento, la presente tesis utiliza la siguiente metodología en términos de ingeniería

Se empleará un enfoque de desarrollo de software basado en el ciclo de vida del software, que incluye actividades como el análisis de requerimientos, el diseño de la métrica, la implementación del software necesario para su cálculo, la realización de pruebas y su validación.

Se utilizarán diversas técnicas de ingeniería de software, como el análisis estático y dinámico del código JavaScript, la aplicación de buenas prácticas de programación, la identificación y resolución de problemas de rendimiento y la optimización del código.

Para el desarrollo de la métrica y la evaluación de la calidad del código JavaScript, se utilizarán herramientas de análisis estático y dinámico, herramientas de *profiling* y monitoreo de rendimiento, así como entornos de desarrollo integrados (*IDEs*) y herramientas de automatización del proceso de desarrollo y pruebas.

10. MARCO TEÓRICO

El presente estudio se enmarca en el campo de la evaluación de la calidad del código JavaScript en términos de rendimiento y mantenibilidad, centrándose en el desarrollo de una métrica que permita cuantificar y evaluar de manera objetiva dicha calidad. Para fundamentar este trabajo, es imprescindible realizar un exhaustivo análisis teórico que brinde las bases conceptuales y metodológicas necesarias para comprender y abordar adecuadamente la problemática planteada.

En este sentido, se abordarán diversos temas de relevancia, incluyendo la calidad del software, los aspectos clave del lenguaje JavaScript, las métricas utilizadas en la evaluación de código y los enfoques existentes para medir el rendimiento y mantenibilidad. Este marco teórico, construido a partir de fuentes confiables y actualizadas, servirá como cimiento sólido para el desarrollo y la justificación de la métrica propuesta, así como para orientar las etapas subsiguientes de investigación y desarrollo. Se describen a continuación los conceptos empleados para la investigación y desarrollo de la presente tesis.

JavaScript es un lenguaje de programación interpretado de alto nivel ampliamente utilizado para crear páginas y aplicaciones web interactivas según la documentación oficial extraída de *Mozilla Developer Network*. Es un lenguaje de programación del lado del cliente que se ejecuta en el navegador y se utiliza para añadir interactividad a las

páginas web. JavaScript también se utiliza en el lado del servidor con Node.js para crear aplicaciones del lado del servidor.

Los *Code smells* es un término utilizado para describir cualquier característica en el código fuente de un programa que pueda indicar un problema más profundo acuñado por Fowler (1999). Es un término metafórico utilizado para describir el olor desagradable que proviene de un código que ha sido mal diseñado o implementado. Los *code smells* pueden ser una indicación de problemas de diseño, problemas de rendimiento u otros problemas que pueden hacer que el código sea difícil de mantener o modificar.

El *refactoring* según Fowler (1999) es el proceso de mejorar el diseño y la calidad del código existente sin cambiar su comportamiento. Consiste en realizar cambios en el código para hacerlo más fácil de mantener, legible y eficiente. La refactorización puede ayudar a reducir los olores y mejorar la calidad general del código. La refactorización es un proceso iterativo que implica hacer pequeños cambios en el código y probarlo para asegurarse de que sigue funcionando como se esperaba. Se espera obtener un análisis profundo gracias a las técnicas del *refactoring* en la elaboración de la nueva métrica.

Según la documentación oficial de *ESLint* (2023) las herramientas de *profiling* o perfilado en código son herramientas de software que se utilizan para medir el rendimiento de un programa. Pueden utilizarse para identificar cuellos de botella en el rendimiento y optimizar el código para mejorar su rendimiento. Las herramientas de perfilado pueden utilizarse para medir el tiempo de ejecución de diferentes partes del código, el uso de memoria del programa y otras métricas de rendimiento. Las herramientas de perfilado se utilizan a menudo junto con la refactorización para identificar las áreas del código que deben optimizarse. Mismas que tendrán lugar en la aportación del cálculo para la nueva métrica.

Los *Linters* en *JavaScript* para los desarrolladores oficiales de *ESLint* (2023) y Mozilla Developer Network (2023) son herramientas que se utilizan para analizar el código JavaScript en busca de posibles errores y olores de código. Pueden utilizarse para identificar violaciones de las normas de codificación, errores de sintaxis y otros problemas

que pueden dificultar el mantenimiento del código. Los *linters* pueden utilizarse para aplicar las normas de codificación y las mejores prácticas, y para garantizar que el código sea coherente y legible. Los *linters* más conocidos para *JavaScript* son *ESLint, JSLint* y *JSHint*.

La notación *Big O* es una notación matemática utilizada para describir la complejidad de un algoritmo según afirma Martin (2009). Se utiliza para describir el peor escenario posible para la cantidad de tiempo y espacio requerido por un algoritmo para resolver un problema. La notación *Big O* se utiliza para comparar la eficiencia de diferentes algoritmos y determinar qué algoritmo es el más eficiente para un problema dado. La notación se basa en la tasa de crecimiento del algoritmo a medida que aumenta el tamaño de los datos de entrada. Las complejidades comunes de la notación Big O incluyen O(1), O(log n), O(n), O(n log n), O(n^2) y O(2^n). Esta es la medida base que sustenta la afirmación de rendimiento de los distintos fragmentos de código en *JavaScript.*

Así como también es necesario precisar conceptos sobre la calidad del código de software. La calidad del software se refiere al grado en que un producto de software cumple sus requisitos y satisface a sus usuarios como sostiene Sabouri (2017) y también Agrahari (2020). Abarca varios aspectos, como la funcionalidad, la fiabilidad, la facilidad de uso, la eficacia, la capacidad de mantenimiento y la portabilidad. En el contexto de JavaScript, la calidad del software puede verse afectada por factores como el olor del código, el rendimiento y la facilidad de mantenimiento.

En paralelo se entiende por rendimiento según Sarafim (2022) a la velocidad y eficiencia con la que se ejecuta un programa. En el contexto de JavaScript, el rendimiento puede verse afectado por factores como la optimización del código, la complejidad del algoritmo y el uso de recursos. Un rendimiento deficiente puede provocar tiempos de carga de páginas lentos, interfaces de usuario que no responden y otros problemas que pueden afectar negativamente a la experiencia del usuario.

Sarafim (2022) también define la mantenibilidad como la facilidad con la que un programa puede modificarse, actualizarse y ampliarse a lo largo del tiempo. En el contexto de

JavaScript, la mantenibilidad puede verse afectada por factores como la organización del código, la documentación y los olores del código. Una mala mantenibilidad puede aumentar el tiempo de desarrollo, incrementar los costes y reducir la productividad.

Para las métricas de calidad del código según Fowler (1999) y Sarafim (2022) son medidas cuantitativas que se utilizan para evaluar la calidad del código fuente. Se pueden utilizar para identificar problemas potenciales como olores de código, cuellos de botella en el rendimiento y problemas de mantenibilidad. Algunas métricas comunes de calidad del código para JavaScript incluyen la complejidad ciclomática, la cobertura del código y el índice de mantenibilidad.

Otro factor a tener en cuenta para la creación de la métrica planteada es la cohesión del código. La cual se refiere al grado en que los elementos de un módulo o componente están relacionados entre sí. Una cohesión alta significa que los elementos están estrechamente relacionados y trabajan juntos para lograr un objetivo común. El acoplamiento del código se refiere al grado en que un módulo o componente depende de otro. Un acoplamiento alto significa que los cambios en un módulo o componente pueden tener un impacto significativo en los demás. En el contexto de JavaScript, la cohesión y el acoplamiento del código pueden afectar al mantenimiento, el rendimiento y la calidad del software.

Para el desarrollo de la presente tesis es también menester detallar la definición de metodologías a utilizar. Como el ciclo de vida de desarrollo de software (*SDLC*), que según Saboury (2017) es un proceso utilizado para diseñar, desarrollar y mantener software. Suele incluir fases como la planificación, la recopilación de requisitos, el diseño, la implementación, las pruebas, la implantación y el mantenimiento. El *SDLC* proporciona un marco para gestionar el proceso de desarrollo de software y garantizar que el software se entrega a tiempo, dentro del presupuesto y cumple los requisitos del usuario.

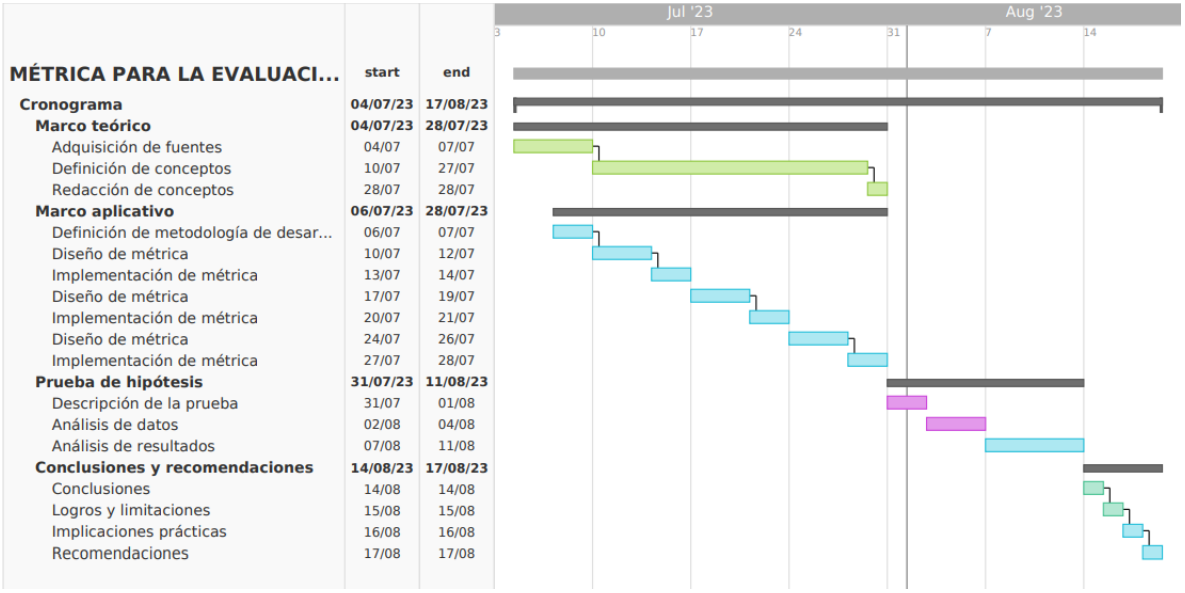
La definición de Sarafim (2022) para el análisis estático del código es una técnica utilizada para analizar el código fuente sin ejecutarlo. Puede utilizarse para identificar problemas potenciales como olores de código, vulnerabilidades de seguridad y cuellos de botella en el rendimiento. El análisis dinámico del código, por su parte, consiste en analizar el código

mientras se ejecuta. Puede utilizarse para identificar problemas como fugas de memoria, condiciones de carrera y problemas de rendimiento. En el contexto de JavaScript, tanto el análisis estático como el dinámico pueden utilizarse para mejorar la calidad y el rendimiento del código.

Por otro parte es necesario adoptar una definición apropiada para las llamadas “mejores prácticas de programación” según Dalton (2021) se refieren a un conjunto de directrices y principios que se utilizan para escribir código de alta calidad, mantenible y eficiente. Pueden incluir prácticas como la organización del código, la documentación, las pruebas y la optimización. En el contexto de JavaScript, las mejores prácticas de programación pueden ayudar a mejorar la calidad, el rendimiento y la facilidad de mantenimiento del código.

Junto a las “mejores prácticas de programación” está también la optimización del código. La cual según Saboury (2017) se refiere al proceso de mejorar el rendimiento y la eficiencia del código. Puede implicar técnicas como la optimización de algoritmos, la gestión de memoria y la refactorización del código. En el contexto de JavaScript, la optimización del código puede ayudar a mejorar la velocidad y la capacidad de respuesta de las aplicaciones web.

11. ÍNDICE TENTATIVO



MÉTRICA PARA LA EVALUACI...	start	end
Cronograma	**04/07/23**	**17/08/23**
Marco teórico	**04/07/23**	**28/07/23**
Adquisición de fuentes	04/07	07/07
Definición de conceptos	10/07	27/07
Redacción de conceptos	28/07	28/07
Marco aplicativo	**06/07/23**	**28/07/23**
Definición de metodología de desar...	06/07	07/07
Diseño de métrica	10/07	12/07
Implementación de métrica	13/07	14/07
Diseño de métrica	17/07	19/07
Implementación de métrica	20/07	21/07
Diseño de métrica	24/07	26/07
Implementación de métrica	27/07	28/07
Prueba de hipótesis	**31/07/23**	**11/08/23**
Descripción de la prueba	31/07	01/08
Análisis de datos	02/08	04/08
Análisis de resultados	07/08	11/08
Conclusiones y recomendaciones	**14/08/23**	**17/08/23**
Conclusiones	14/08	14/08
Logros y limitaciones	15/08	15/08
Implicaciones prácticas	16/08	16/08
Recomendaciones	17/08	17/08

13. REFERENCIAS

Agrahari V. (2020)*An Exploratory Study of Code Smells in Web Games*

Cairo, A. (2022) *The Impact of Code Smells on Software Bugs: A Systematic Literature Review. Programa de Pós-Graduação em Sistemas e Computação (PPGCOMP), Universidade Salvador (UNIFACS), Salvador*

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

Costa, L. (2021) *Testing Javascript aplications.* Editorial Manning. *New York, USA*

Dalton N. (2021) *Investigating Test Smells in JavaScript Test Code* Proceedings of the 6th Brazilian Symposium on Systematic and Automated Software Testing

ESLint. (n.d.). *About*. Retrieved from https://eslint.org/docs/about/

Flanagan , D. (2020) *JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language 7th Ediction* Editorial *O'Reilly Media New York, USA*

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional.*

Mozilla Developer Network. (n.d.). *JavaScript*. Retrieved from https://developer.mozilla.org/en-US/docs/Web/JavaScript

Hahn, E. (2019) *JavaScript Testing with Jasmine: JavaScript Behavior-Driven Development.* Editorial *O'Reilly Media New York, USA*

Haverbeke, M. (2018) *Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming* Sin editorial

Levanen, L. (2021) The effects of refactoring on a web application's quality of code. Master's Programme in Computer, Communication and Information Sciences, *Learning Centre*

Luca Ardito, Riccardo Coppola, Luca Barbato, Diego Verga, (2020) *A Tool-Based Perspective on Software Code Maintainability Metrics: A Systematic Literature Review*, Scientific Programming

Martin, R. C. (2009). *Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall. Oracle. (n.d.). Java Performance Tuning Guide.* Retrieved from https://docs.oracle.com/cd/E15289_01/doc.40/e15058/underst_jvm.htm

Oracle. (n.d.). *Java Performance Tuning Guide.* Retrieved from https://docs.oracle.com/cd/E15289_01/doc.40/e15058/underst_jvm.htm

Sabouri A. (2017) *An empirical study of code smells in JavaScript projects* IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)

Sarafim D. (2022) *Random Forest for Code Smell Detection in JavaScript* Anais do XIX Encontro Nacional de Inteligência Artificial e Computacional (ENIAC 2022)

Zozas, I (2023) *Forecasting the Principal of Code Technical Debt in JavaScript Applications* IEEE