UNIVERSIDAD
NACIONAL
DE COLOMBIA

Una métrica para medir deuda técnica basada en el análisis de las más usadas. Caso de estudio del repositorio Square.

Didier Diaz Mena.

Universidad Nacional de Colombia

Departamento de Ciencias de la Computación y de la Decisión

Medellín, Colombia

2024

Una métrica para medir deuda técnica basada en el análisis de las más usadas. Caso de estudio del repositorio Square.

Didier Diaz Mena

Tesis presentada como requisito parcial para optar al título de:

Magister en Ingeniería Sistemas.

Director (a):

Albeiro Espinosa Bedoya, Ph.D.

Línea de Investigación:

Profundización

Universidad Nacional de Colombia

Departamento de Ciencias de la Computación y de la Decisión

Medellín, Colombia

2024

(Dedicatoria)

En primer lugar, quiero darle las gracias a Dios por darme la vida haciendo posible lograr mis metas y por dejarme recorrer un camino iluminado de enseñanza y alegría. En segundo lugar, darle las gracias a mi querida Madre (Matilde Mena), familia en general por todo ese apoyo mutuo, motivación desde el primer y último día, este logro es de ustedes.

Didier Diaz Mena.

Agradecimientos

Quisiera expresar mi más profundo agradecimiento a mi querida familia por todo el acompañamiento durante esta etapa de enseña y aprendizaje ya que sin su ayuda motivación no hubiese tenido la fuerza de poder cumplir este objetivo.

Luego, agradecerle al Profesor Albeiro Espinosa Bedoya por su dedicación, paciencia y acompañamiento en la dirección de esta tesis de profundización solo me queda decirle que muchas gracias de corazón ya que, sin su ayuda, constancia no hubiese podido yo culminar este proceso, ante mano un caluroso abrazo para su familia.

Después, darle las gracias a mi amigo el ingeniero Julio Poveda Escobar ya que ha estado presente durante todo mi proceso de formación académica apoyándome en el aprendizaje de la programación con tecnología Microsoft en la cual me ha servido para poder desempeñarme en el ámbito laboral, actualmente como Developer .Net.

En síntesis, quiero expresar el orgullo y honor en poder ser egresado de mi amada Universidad Nacional de Colombia - Sede Medellín ya que para mí es un sueño, fue un objetivo poder cumplir mi proceso de postgrado en una de las mejores universidades de Colombia, no me queda nada más que agradecerle a todos los docente de dicha Facultad de Minas en especial a los del Departamento de Ciencias de la Computación y de la Decisión por su colaboración, formación permitiéndome ser una persona integrada hacia la sociedad.

Finalmente, a todas esas personas que desde el minuto cero estuvieron apoyándome solo me queda decirle gracias.

Que Dios y la virgen los bendiga siempre. Un día lo soñé, hoy es una realidad.

Resumen

Una métrica para medir deuda técnica basada en el análisis de las más usadas. Caso de estudio del repositorio Square.

La deuda técnica es un componente principal del costo de propiedad de la aplicación, se ha convertido en una de las metáforas más importantes para expresar los atajos de desarrollo, que causan la degradación de la calidad interna del software. Es necesario recalcar que, existen métricas de código abierto las cuales proporcionan datos numéricos en algunos productos de software y que a su vez permiten medir el índice de mantenibilidad, Complejidad ciclomática, Profundidad de herencia, Acoplamiento de clases, Líneas de código fuente, Líneas de código ejecutable donde los desarrolladores pueden identificar riesgos y hacer seguimiento continuo. No obstante, la literatura informa de varias métricas de software para líneas de productos y proceso, otras orientadas a objetos. En efecto, se realiza una búsqueda centralizada de varios repositorios de código abierto en la plataforma GitHub con el objetivo de dar cumplimiento a la necesidad expuesta de acuerdo a los criterios definido donde fueron preseleccionado nueve repositorios específicamente con el lenguaje Python obteniendo A si una mayor puntuación el repositorio de código abierto (square/square-python-sdk) para la aplicación de la métrica ya que es un proyecto medible desde todos sus ámbitos. En síntesis, se efectúa el análisis de los datos mediante el método estadístico ACP, también la normalización y cálculo de los pesos sobre las referencias de las métricas más usadas en la literatura. Finalmente, se valida la métrica propuesta aplicada al repositorio Square como caso de estudio donde se realizó el desarrollo del consumo de una API tanto a nivel Backend y FrontEnd.

Palabras clave: (Deuda Técnica, Métricas, Repositorio, lenguajes de programación).

Abstract

A metric to measure technical debt based on the analysis of the most used ones. Square repository case study.

Technical debt is a major component of application cost of ownership, It has become one of the most important metaphors to express development shortcuts, which cause degradation of the internal quality of the software. It is necessary to empha size that there are open-source metrics which provide numerical data in some software products and which in turn allow measuring the maintainability index, cyclomatic complexity, depth of inheritance, class coupling, lines of source code, lines of executable code where developers can identify risks and continuously monitor. However, the literature reports several software metrics for product and process lines, others object-oriented. In effect, a centralized search of several open-source repositories is carried out on the GitHub platform with the aim of fulfilling the stated need according to the defined criteria where nine repositories were preselected specifically with the Python language, thus obtaining a higher score. open-source repository (square/square-python-sdk) for the application of the metric since it is a measurable project from all its areas. In summary, the data analysis is carried out using the ACP statistical method, as well as the normalization and calculation of the weights on the references of the most used metrics in the literature. Finally, the proposed metric applied to the square repository is validated as a case study where the development of the consumption of an API was carried out at both the Backend and FrontEnd levels.

Keywords: (Technical Debt, Metrics, Repository, programming language).

Contenido

Pág.

Lista de figuras **9**

Lista de tablas **10**

Introducción **11**

1. Capítulo: Planteamiento del Problema. **12**
1.1 Justificación. 12
1.2 Objetivos. 14
1.3 Antecedentes. 14
1.4 Contribuciones 17
1.5 Estructura de la Tesis 18

2. Capitulo: Marco Teórico **19**
2.1 Visión general de la deuda técnica 19
2.2 Medición automatizada de la Deuda Técnica. 21
2.3 Deuda técnica en la Práctica. 21
2.4 Gestión de la Deuda Técnica. 22
2.5 Deuda Técnica en proyecto de código Abierto. 22
2.6 Evolución de la Deuda técnica. 23
2.7 Medición contextual de la deuda técnica (CTDM) 25

3. Capitulo: Criterio de Selección. **26**
3.1 Métricas en general. 26
3.2 Necesidad de Métricas 27
3.3 Cuadro comparativo de Métricas 28

4. Capitulo: Métrica propuesta **36**
4.1 Metodología utilizada. 39
4.2 Análisis de Datos Vs Métrica más usadas. 40
4.2.1 Análisis factorial - Extracción de componentes principales 48
4.2.2 Correlation Matrix 48
4.2.3 Inversa de la matriz de correlación 49
4.2.4 Matriz de correlaciones parciales 50
4.2.5 KMO 50
4.2.6 Valores y vectores propios. 51
4.2.7 Full Load Matrix 51
4.2.8 Scree Plot 52
4.2.9 Factor Matrix (unrotated) vs Factor Matrix (rotated Varimax) 53
4.2.10 Reproduced Correlation Matrix 53
4.2.11 Error Matrix. 54
4.2.12 Factor Scores Matrix - Regression Method. 54
4.3 Técnica de datos sobre la métrica aplicada. 56
4.4 Simplificación de métrica propuesta 57
4.5 Formulación sobre la métrica aplicada 57

4.6 Cálculo de pesos sobre la métrica aplicada. 59
4.7 Ilustración de cálculo métrica más usadas. 60
4.8 Ilustración cálculo métrica propuesta Vs Mas usadas. 61

5. Capítulo: Validación de la métrica propuesta. 63
5.1 Ruta de validación sobre la métrica aplicada. 63
5.2 Código fuente en Backend de normalización de la métrica aplicada. 64
5.3 Tecnologias & herramientas utilizadas sobre la métrica aplicada. 65
5.4 Esquema a nivel de Base datos sobre la métrica aplicada. 66
5.5 Esquema a nivel de Backend sobre la métrica aplicada. 68
5.6 Esquema a nivel de Frontend sobre la métrica aplicada. 72
5.7 ¿Por qué la métrica propuesta es una buena métrica? 77
5.8 ¿Qué se lograría en un proyecto con la implementación de esta métrica? 78
5.9 ¿En qué etapas del desarrollo del proyecto se debe implementar? 78
5.10 ¿Qué condiciones debe cumplir un proyecto para que se pueda aplicar la métrica? 78

6. Conclusiones y recomendaciones 79
6.1 Conclusiones 79
6.2 Recomendaciones 80

Bibliografía 81

Lista de figuras

Figura 1 Metáfora de la deuda técnica. Tomado de (OMG, 2018). 13
Figura 2 Scree Plot. 52
Figura 3 Normalización Depth of Inheritance. 58
Figura 4 Normalización Class Coupling. 58
Figura 5 Normalización Lines of Source code. 58
Figura 6 Resultado cálculo métrica más usadas. 61
Figura 7 Resultado cálculo métrica propuesta Vs Mas usadas. 62
Figura 8 ERD Bases de datos – MetricsTogglesDebt 68
Figura 9 Vista de MetricsController.cs 69
Figura 10 Vista de ReportsController.cs 70
Figura 11 Capa MetricsTogglesDebt.API 71
Figura 12 Capa MetricsTogglesDebt.Data 71
Figura 13 Capa MetricsTogglesDebt.Service. 72
Figura 14 Frontend de cálculo sobre métrica implementada. 76
Figura 15 Frontend lista de datos métrica Depth of Inheritance - Class Coupling. 76
Figura 16 Frontend lista de datos métrica Lines of Source code. 77

Lista de tablas

Pág.

Tabla 1 Herramientas sobre Deuda Técnica. Tomado de (Efimova, 2021). 23
Tabla 2 Comparación de herramientas sobre Deuda Técnica. Tomado de (Efimova, 2021).24
Tabla 3 Lista de repositorio open source preseleccionado. 37
Tabla 4 Totalidad de los criterios seleccionado.. 38
Tabla 5 Puntuación de los criterios seleccionado. 38
Tabla 6 Criterio de selección cuantitativo 1.. 38
Tabla 7 Criterio de selección cuantitativo 2.. 39
Tabla 8 Análisis de datos extraído métricas seleccionadas en el repositorio Square. 41
Tabla 9 Análisis factorial - Extracción de componentes principales 48
Tabla 10 Correlation Matrix... 49
Tabla 11 Inversa de la matriz de correlación 49
Tabla 12 Matriz de correlaciones parciales.. 50
Tabla 13 Kaiser-Meyer-Olkin (KMO).. 51
Tabla 14 Valores y vectores propios.. 51
Tabla 15 Full Load Matrix. 51
Tabla 16 Scree Plot. 52
Tabla 17 Factor Matrix (unrotated).. 53
Tabla 18 Factor Matrix (rotated Varimax).. 53
Tabla 19 Reproduced Correlation Matrix. 54
Tabla 20 Error Matrix... 54
Tabla 21 Factor Scores Matrix - Regression Method. 55
Tabla 22 Matriz de puntuaciones factoriales - Método de Bartlett. 55
Tabla 23 Factor Scores Matrix - Anderson-Rubin's Method. 55
Tabla 24 Normalización de la métrica... 59
Tabla 25 Cálculo de pesos vs métrica sobre referencia. 59
Tabla 26 Resultado de Calculo métricas más usadas.. 60
Tabla 27 Resultado de Calculo Métrica Propuesta & Mas Usadas. 61

Introducción

La deuda técnica (TD) se ha convertido en una de las metáforas más importantes para expresar los atajos de desarrollo, tomados por conveniencia, pero que causan una serie de errores en la calidad interna del software. Debido a su importancia, se han lanzado varias herramientas que ofrecen medir la TD mediante el análisis estático del código. A su vez, se trata tanto de herramientas comerciales como de prototipos de investigación. Sin embargo, cada herramienta utiliza diferentes métricas, índices, modelos de calidad, reglas de análisis estático, corrección de la deuda técnica, (Avgeriou, Taibi, Ampatzoglou, Fontana, & Besker, 2020). En lo que sigue, la deuda técnica suma los esfuerzos de remediación de todos los elementos de deuda técnica detectados que se definen como ocurrencias de patrones a su vez, representan debilidades enumeradas en el código fuente automatizado.

Actualmente, existen un sin número de métricas en cuestión de tecnología que ya realizan la medición de la deuda técnica. Por tanto, se hace necesario saber definir qué tipo de métrica quiero aplicar para poder medir dicha deuda técnica esto con el objetivo de obtener éxito en el mantenimiento del producto de software. Llegados a este punto, es importante resaltar que la aplicación de la deuda técnica en las organizaciones genera un impacto significativo que no se puede ocultar (Maven Solutions, 2024). En concreto, existen un sin número de investigaciones en la literatura sobre métricas para medir la deuda técnica que pueden ser llegada hacer aplicada en diferentes proyectos de desarrollo de software, por ejemplo: en el monitoreo, evaluación, automatización, estimación, priorización de la deuda técnica. En cambio, también se identifica varios tipos de métricas como lo son de proceso, producto, orientadas a objetos y algunas métricas para estructurar las alternancias de características, otras para repositorio de código abierto. En resumen, se evidencia varias métricas (Mohan & Kumar, 2013) referenciada por diferentes autores que a su vez representan su definición y característica. Lo dicho hasta aquí supone que, la priorización de TD es esencial para A signar mejor los recursos a fin de determinar qué artículos de TD se reembolsarán primero y qué artículos se retrasarán hasta lanzamientos posteriores (Alfayez, Alwehaibi, Winn, Venson, & Boehm, 2020). Aunque, las causas relacionadas con la planificación y la gestión son protagonistas entre los responsables de generar deuda técnica. El propósito de esta tesis de profundización es formular una métrica de deuda técnica basada en el análisis de las métricas más usadas en la literatura, aplicada al repositorio Square como caso de estudio. donde se identifique las principales métricas en la literatura usadas para medir deuda técnica, con el fin de, proponer una métrica para evaluar la deuda técnica basada en el análisis de las principales métricas identificadas en la literatura. Luego, validar la métrica propuesta aplicada al repositorio Square como caso de estudio.

1. Capítulo: Planteamiento del Problema.

Las investigaciones identificadas sobre Deuda Técnica muestran un panorama de crecimiento en la parte de medición específicamente en métricas de software. Pero, su implementación en la pequeñas, mediana y grandes organizaciones ha generado un sobre costo impresionante a la hora de poner en marcha el tipo de métrica adecuada, necesaria para su producto y proceso de desarrollo. En efecto, con (OMG, 2018) no existe una definición universalmente acordada de Deuda Técnica, también la industria y la comunidad investigadora expresan objetivos diferentes a la hora de definir, medir la Deuda Técnica. Por lo que se refiere a, formular una métrica para medir la deuda técnica en el análisis de las métricas existente más usada de la literatura. Es necesario recalcar que, existen métricas de código las cuales proporcionan datos numéricos en algunos productos de software y que a su vez permiten medir el índice de mantenibilidad, Complejidad ciclomática, Profundidad de herencia, Acoplamiento de clases, Líneas de código fuente, Líneas de código ejecutable donde los desarrolladores pueden identificar los posibles riesgos y hacer seguimiento continuo durante las diferentes fases del desarrollo de software (Microsoft, 2023). Por lo tanto, el problema de investigación de esta tesis de maestría es identificar las principales métricas en la literatura usadas para medir deuda técnica.

1.1 Justificación.

La industria de tecnología requiere una medida que prediga los costos futuros del mantenimiento correctivo y otros resultados relacionados con la calidad del software. Dicho de otra manera, la deuda técnica está diseñada para predecir esos costos de mantenimiento correctivo y los factores relacionados para guiar las decisiones de TI esto es, que los componentes de la deuda técnica proporcionan una base para la economía de la calidad del software (OMG, 2018). Es necesario recalcar que, la Deuda Técnica se calcula como una estimación del esfuerzo para corregir las buenas prácticas de arquitectura y codificación que deben remediarse debido a su riesgo y costo para la empresa. Es decir, una de las bases para especificar esta medida de característica de calidad CISQ (Consortium for IT Software Quality) es aprobada como estándar OMG debido al crecimiento de la deuda técnica en los problemas de calidad y costo en las organizaciones. Sin embargo, en el marco expresado sobre la definición de la deuda técnica donde se concluye lo siguiente: Primero, no existe una definición universalmente acordada de deuda técnica. Segundo, la industria y la comunidad investigadora tienen objetivos diferentes a la hora de definir y medir la deuda técnica. Con esto quiero decir, que la deuda técnica es un componente principal del costo de propiedad de una aplicación. Así mismo: El Principal costó de solucionar los problemas que deben corregirse es en el código de producción. El Interés: Costo continuos, principalmente en TI, atribuibles a problemas que deben solucionarse mientras permanezcan en el código de producción. El Riesgo empresarial: Costó potenciales para la empresa si los problemas que deben solucionarse en el código de producción provocan incidencias operativas perjudiciales, como

interrupciones, corrupción de datos, degradación del rendimiento y fallos de seguridad. Responsabilidad: Costó para la empresa derivados de problemas operativos causados por fallos en el código de producción. Por último, el costo de oportunidad: Beneficios como los ingresos derivados de nuevas funciones, que podrían haberse obtenido si los recursos se hubieran dedicado a desarrollar nuevas capacidades en lugar de A signarlos a eliminar la Deuda Técnica (OMG, 2018). Si no también las relaciones entre los componentes de la metáfora sobre Deuda Técnica tal como se muestra en la siguiente figura 1.

Figura 1 Metáfora de la deuda técnica. Tomado de (OMG, 2018).

En la Figura 1 El costo de solucionar los problemas estructurales constituye la principal deuda, mientras que las ineficiencias causan un mayor esfuerzo de mantenimiento o un exceso de recursos informáticos que representan los costos de los interéses de la deuda. También, los problemas estructurales subyacentes a la deuda técnica crean riesgos empresariales como interrupciones y fallos de seguridad también en acontecimientos negativos que pueden causar la pérdida de ingresos por ventas en línea (OMG, 2018).Habría que decir también, que incorporar la perspectiva empresarial en la priorización de la deuda técnica es fundamental para contribuir a la toma de decisiones en la industria. Aunque, las causas relacionadas con la planificación y la gestión son protagonistas entre los responsables de generar deuda técnica. Por ejemplo, los calendarios ajustados, la competitividad, los cambios en la priorización empresarial y la dinámica empresarial son responsables de crear un entorno turbulento que conduce a la deuda técnica (de Almeida, Ribeiro, Treude, & Kulesza, 2021). En efecto, unos de los principales objetivos para medir la deuda técnica es poder cuantificar mediante métricas de código que permitan el rastreo simultaneo, además la utilización de herramientas que ayudan a comprender mejor los puntos fuertes del equipo de trabajo y críticos sobre el proyecto de desarrollo. Dicho lo anterior, es fundamental saber cómo medir la deuda técnica y cómo elegir las métricas más útiles en función de las necesidades y objetivos individuales de su negocio (Maven Solutions, 2024). Por tanto, se busca que en este trabajo sea el foco de enlace para muchos estudios y trabajos de investigaciones relacionado con el tema de métrica para medir deuda técnica en cualquier tipo de proyecto de software o repositorio de código abierto. Con la intención de, que la mayoría de los proyectos de desarrollo no fracase por no medir adecuadamente la deuda técnica mediante métricas e idónea.

1.2 Objetivos.

El propósito de esta tesis de profundización es dar a entender el objetivo general y objetivos específicos donde se visualiza el desenlace, desarrollo de la métrica propuesta.

Para simplificar, el objetivo general es proponer una métrica de deuda técnica basada en el análisis de las métricas más usadas en la literatura, aplicada al repositorio Square como caso de estudio.

Mientras tanto, para el cumplimiento del objetivo general se propone los siguientes objetivos específicos.

- Identificar las principales métricas en la literatura usadas para medir deuda técnica.
- Proponer una métrica para evaluar la deuda técnica basada en el análisis de las principales métricas identificadas en la literatura.
- Validar la métrica propuesta aplicada al repositorio Square como caso de estudio.

1.3 Antecedentes.

En cuanto a, la deuda técnica podemos observar que suma los esfuerzos de remediación hacia todos los elementos que se definen como ocurrencias de patrones que representan debilidades enumeradas en el código fuente automatizado. Por otra parte, la deuda técnica afecta la mantenibilidad del software ya que se vuelve más desafiante dado que existen estudios exploratorios donde se investiga si una métrica introducida ayuda a comprender mejor la evolución de la TD en proyecto de desarrollo, A sí mismo en los últimos años, la metáfora de la deuda técnica ha recibido mucha atención tanto del mundo académico como de la industria debido a sus ventajas para comunicar & cuantificar el impacto de los artefactos de software más aún, cuando actualmente existen investigaciones que trabajan continuamente para encontrar nuevas métricas y técnicas que permitan calcular la acumulación y el reembolso de la deuda técnica, (Al Mamun, Martini, Staron, & Berger, 2019). A sí mismo, se descubre que las herramientas contemporáneas se centran principalmente en las cualidades internas del software, es decir en la calidad del código fuente que, a su vez, definen y miden la deuda técnica o dicha mantenibilidad de manera diferente se debe agregar que, para calcular la complejidad de las unidades o módulos de código, se puede utilizar la complejidad ciclomática, el índice de mantenibilidad o la complejidad cognitiva es decir, detectar y eliminar la duplicación de código, se pueden utilizar PMD, SonarQube o Code Climate, JaCoCo, que pueden medir e informar la cobertura del código. Para identificar y refactorizar olores de código, se puede utilizar Checkstyle, ESLint, o RuboCop. (Pfeiffer & Lungu, 2022). Después, está el estado no ideal del software que se captura mediante la metáfora de la deuda técnica, en este caso (Amanatidis, Mittas, Moschou, Chatzigeorgiou, & Ampatzoglou , 2020) evalúan las principales herramientas para medir la deuda técnica luego, proponen un marco con el fin de capturar la diversidad sobre las herramientas de (TD)

examinadas dicha propuesta se ilustra mediante un estudio de caso sobre cincuenta (50) proyectos de código abierto y dos lenguajes de programación (Java y JavaScript) que emplean tres herramientas líderes de TD. En segundo lugar, podemos observar el seguimiento de la deuda técnica en un entorno industrial donde (Arvanitou, Ampatzoglou, Bibi, Chatzigeorgiou, & Stamelos, 2019) proporcionan varias vistas sobre el monitoreo de la calidad a través de un tablero en la cual se muestra algunas métricas relacionadas con deuda técnica. Por lo que se refiere a, los factores que afectan la deuda técnica (Bedi & Kaur , 2020) como lo son la frecuencia de commits, las líneas de código en realidad los autores para revelar el efecto de estas variables, utilizaron la herramienta SonarQube para encontrar la deuda técnica y las métricas relacionadas con el fin de, demostrar que a medida que aumenta la frecuencia de confirmación de commits, las líneas de código, la cobertura y los olores del código, también se produce un cambio considerable en la deuda técnica. Deseo subrayar que, la deuda técnica (TD) es un problema de la calidad del software interno, es decir, a menudo se contratan debido a plazos ajustados de los proyectos, por ejemplo, soluciones rápidas y soluciones alternativas, y pueden hacer que los cambios futuros sean más costosos o imposibles. La prevención de TD debería ser más importante que el reembolso de TD, porque la refactorización suele ser más costosa que construir la solución correcta desde el principio. (Wiese, M, 2021) Mas aun, cuando su concepto como artefacto de desarrollo de software y modelo conceptual proporcionan un enfoque coherente para la interpretación, el análisis del fenómeno de la deuda técnica (Stochel, M G; Chołda, P; Wawrowski, M R, 2020). Al mismo tiempo, (Freire, Rios, Pérez, & Mendonça, 2021) expresa que conocer los efectos de la deuda técnica (TD) puede ayudar a los equipos de desarrollo de software a priorizar su elemento. Como se ha dicho, se sabe poco sobre las relaciones entre los efectos de la TD y las prácticas de pago de la TD. Tener este conocimiento puede proporcionar información valiosa para la toma de decisiones sobre qué práctica de pago se puede aplicar dada la presencia de efectos específicos de la TD. Por otra parte, (Besker, T; Martini, A; Bosch, J, 2020) es de vital importancia mantener bajo el nivel de deuda técnica, ya que está bien establecido a partir de varios estudios así, por ejemplo, reducir la productividad del desarrollo y comprometer la calidad general del software. Por otro lado, (Taibi & Lenarduzzi, 2021) han propuesto diferentes enfoques para la priorización de la deuda técnica, todos con objetivos diferentes y proponiendo optimización en función de diferentes criterios. Dicho lo anterior, (Lenarduzzi, Besker, Taibi, Martini, & Fontana, 2020) las empresas de software necesitan gestionar y refactorizar los problemas de la deuda técnica, por lo tanto, es necesario comprender si se debe priorizar la refactorización de la deuda técnica con respecto al desarrollo de funciones o la corrección de errores. Así mismo, (Avgeriou, Taibi, Ampatzoglou, Fontana, & Besker, 2020) existen investigaciones donde se pretende estudiar la evolución y características de la deuda técnica en software de código abierto. Sin embargo, diferentes herramientas adoptan términos, métricas y formas de identificar y medir la deuda técnica ya que estas herramientas ofrecen diversas funciones y su popularidad varía significativamente. Aunque, (Molnar & Motogna, 2020) se intenta aclarar las características y la popularidad de las herramientas técnicas de medición de la deuda existente sobre su validez. Lo dicho hasta aquí supone que, las herramientas de software existentes permiten caracterizar y medir el monto de la deuda

técnica en niveles de granularidad al proporcionar un modelo computacional, que les permiten a las partes interésadas medir y, en última instancia, controlar este fenómeno. En relación con, (Tsoukalas, Kehagias, Siavvas, & Chatzigeorgiou, 2020)se evalúa empíricamente la capacidad de los métodos de aprendizaje automático (ML) para modelar y predecir la evolución de la TD, todo esto parece confirmar, que la deuda técnica (TD) se refiere a ineficiencias durante todas las fases del ciclo de vida del desarrollo de software que conducen a un esfuerzo de mantenimiento adicional. Debido a que, en los últimos años, TD ha atraído la atención tanto del mundo académico como de la industria. Como resultado, ha habido un aumento considerable en el número y la funcionalidad proporcionada de métodos y herramientas que apoyan la gestión de TD. A su vez, (Feitosa, Ampatzoglou, Gkortzis, & Bibi, 2020) resaltan la relación y el principal interés de mantener software con mejores niveles de TD, más aún, cuando la comunidad TD se está esforzando por ofrecer métodos y herramientas para reducir la cantidad de TD. Habría que decir también, (Soliman & Avgeriou, 2021) durante el desarrollo de software, algunas decisiones de diseño arquitectónico incurren en deudas técnicas, ya sea de forma deliberada o inadvertida. Estos tienen un impacto serio en la calidad de un sistema de software y pueden costar mucho tiempo y esfuerzo cambiarlos. Dado que, (Pérez, y otros, 2021) las listas de prácticas de prevención y pago de TD pueden guiar a los equipos de software al tener un catálogo de prácticas para mantener la deuda controlada o reducida, igualmente es necesario entender cómo perciben la TD los arquitectos de software, en particular, las prácticas que apoyan la eliminación de elementos de deuda de los proyectos, y las prácticas utilizadas para reducir las posibilidades de ocurrencia de TD. Mientras tanto, (de Toledo , Martini, & Sjøberg, 2021) el uso de una arquitectura de microservicios es una estrategia popular para que las organizaciones de software ofrezcan valor a sus clientes de forma rápida y continua, a su vez, el conocimiento científico sobre cómo gestionar la deuda arquitectónica en microservicios es escaso. En efecto, (de Almeida, Ribeiro, Treude, & Kulesza, 2021) incorporar la perspectiva empresarial en la priorización de la deuda técnica es fundamental para contribuir a la toma de decisiones en la industria, aunque los autores evalúan un enfoque orientado a la empresa para la priorización de la deuda técnica, como resultado el enfoque contribuyó a alinear los criterios empresariales entre las partes interésadas y técnicas. En pocas palabras, (León-Sigg, Vázquez-Reyes, & Rodríguez-Ávila, 2020) el concepto de deuda técnica ha estado en uso desde la década de los noventa, pero, sobre todo uno de los principales problemas relacionados con la gestión reside en la complejidad de hacer que la deuda técnica sea visible para las organizaciones. Lo dicho hasta aquí supone que, introducir una gestión sistemática de la deuda técnica (Malakuti & Heuschkel, 2021) en una empresa a gran escala es un gran desafío. Es necesario recalcar que, (Mahdavi-Hezaveh, Ajmeri, & Williams, 2022) expresan la descripción y agrupación de métricas de complejidad, compresibilidad, mantenibilidad que están basadas en un estudio empírico de repositorios de código abierto. Es así que, (Microsoft, 2023) resalta las “métricas de código (Índice de mantenibilidad, Complejidad ciclomática, Profundidad de herencia, Acoplamiento de clases, Líneas de código fuente, Líneas de código ejecutable) como un conjunto de medidas de software que brindan a los desarrolladores una mejor visión del código que están efectuando”.

1.4 Contribuciones

En efecto, el propósito es poder contextualizar y mostrar un análisis de literatura sobre los temas de gran importancia en la industria del desarrollo de software como lo es la deuda técnica ya que muestra un panorama de crecimiento en la parte de medición específicamente en métricas de software.

Es preciso decir, que el objetivo de estudio es proponer una métrica de deuda técnica basada en el análisis de las métricas más usadas en la literatura, aplicada al repositorio Square como caso de estudio.

Donde se realizó el estado de arte de las métricas más usadas en la literatura obteniendo un cuadro comparativo para su análisis.

Después, tenemos el análisis metodológico implementado para la selección del repositorio la cual puede ser útil como objeto de estudio para futuras investigaciones.

Seguido, la implementación a nivel de código fuente sobre algunas métricas de literatura específicamente el repositorio seleccionado donde se podrá detallar los métodos, clases, líneas de código fuente, historia de commits, tags, caracteres especiales, remotes pull request, Releases y versionamiento sobre el desarrollo en la obtención y extracción de datos para poder analizar, formular nuestra métrica propuesta.

Por otro lado, se puede obtener una transferencia de conocimiento frente al análisis de los datos ya que se utilizó el método estadístico llamado componente de análisis principal debido a su capacidad de medir grandes cantidades de datos mediante varios métodos y factores de medición generando A si un mayor soporte a la hora de tomar decisión de las métricas más usada.

Por consiguiente, se dispone de manera publica el desarrollo del consumo de la API implementada tanto a nivel Backend, Frontend para validar la métrica, es decir, el código fuente se encuentra alojado en la plataforma de código abierto GitHub. Por lo cual, podrá ser muy útil e interés para la academia y la industria de tecnología. En definitiva, se busca que esta tesis de profundización sea el foco de enlace para muchos estudios y trabajos de investigaciones relacionado con el tema de métrica para medir deuda técnica en cualquier tipo de proyecto de software o repositorio de código abierto.

1.5 Estructura de la Tesis

Dicho brevemente, esta tesis de profundización se encuentra estructurada en siete capítulos los cuales podrán comprender la importancia de poder proponer métrica para medir deuda técnica, generando A si una transferencia de conocimiento para el lector. Es decir, se presenta los siguientes capitulo a continuación:

Capítulo 1 – Planteamiento del Problema: Contextualización de la importancia de poder realizar un análisis de literatura en cuanto a el estudio de métricas en la medición de deuda técnica por otra parte, los antecedentes, justificación del problema.

Capítulo 2 – Marco Teórico: En este apartado podemos encontrar todo el estado el estado del arte de nuestra investigación sobre deuda técnica, además, la importancia de medición de deuda técnica en métricas software.

Capítulo 3 – Criterio de Selección: Se puede detallar el proceso de búsqueda efectuado para poder seleccionar el repositorio adecuado de código abierto sobre los cumplimientos de los objetivos de la tesis.

Capítulo 4 – Principales métricas identificada en la literatura: Aquí se puede estudiar el estado del arte sobre las métricas más usadas.

Capítulo 5 – Métrica propuesta: A su vez, en este apartado se presenta el análisis de componente principal efectuado sobre los datos extraído en la implementación de las métricas más usadas por medio del repositorio seleccionado.

Capítulo 6 – Validación de la métrica propuesta: Al mismo tiempo, en este capítulo podemos visualizar la implementación de la formula y métrica propuesta de acuerdo al objetivo general, además, el código fuente a nivel de Backend sobre el cálculo de la normalización de las métricas usadas. Por otro lado, el Frontend desarrollo para el consumo de dicho cálculo, mediante la medición de la métrica propuesta con su respectiva lista de datos.

Capítulo 7 – Conclusiones y recomendaciones: Con miras a, que esta tesis de investigación sea referente para futuros estudios tanto a nivel de educación como para el sector TI y público interesado en la medición de métrica con deuda técnica.

2. Capitulo: Marco Teórico

Por lo que se refiere a, la deuda técnica es una metáfora que enmarca una serie de problemas, se utiliza para justificar el descuido de la calidad interna. El argumento es que se necesita tiempo y esfuerzo para evitar que se acumule. Si hay nuevas características que se necesitan con urgencia, entonces tal vez sea mejor asumir la deuda, aceptando que esta deuda tendrá que gestionarse en el futuro (Martin Fowler, 2019).

Como se ha dicho, la deuda técnica es una metáfora de acuerdo con Ward Cunningham en su informe del 1992. A sí mismo, cuando una persona adopta una decisión técnica no óptima o subóptima, introduce deuda técnica. Si no se corrige lo suficientemente pronto, la decisión afectará otras decisiones técnicas relacionadas y la deuda comenzará a aumentar (Samarthyam, Suryanarayana, & Sharma, 2015).

2.1 Visión general de la deuda técnica

Todavía cabe señalar, que es muy importante saber la importancia de la deuda técnica a nivel organizacional y profesional esto con el fin de no padecerla o incurrir en ella. Siendo A si se hace necesario entender los componentes de la deuda técnica (Samarthyam, Suryanarayana, & Sharma, 2015).

- La deuda técnica es principal y el interés acumulado.
- El interés es de naturaleza compuesta.
- Mientras que el interés hace que la deuda técnica sea un problema importante.

Es necesario recalcar que, la deuda técnica puede llegar afectar la productividad del equipo de desarrollo. Simultáneamente, (Cholda & Stochel, 2020) se evidencia un enfoque de valoración continua para la priorización de la deuda técnica que, a su vez, es utilizado durante todo el ciclo de vida del desarrollo de software. Por otra parte, la densidad de la deuda técnica se reduce en algunos casos (Digkas, Chatzigeorgiou, Ampatzoglou, & Avgeriou, 2021). Llegados a este punto, la necesidad de un cambio constante se ha convertido en un factor clave en el desarrollo de software (Ramirez, Tuovinen, & Mikkonen, 2021). Sírvase de ejemplo, (Feitelson, Frachtenberg, & Beck, 2013) las correcciones de errores, las nuevas funciones y las actualizaciones de tecnología, arquitectura que se realizan con frecuencia, incluso varias veces al día para solucionar un problema urgente en la base de código. Sintetizando, (Rosser & Norton, 2021) existen varios tipos de deuda técnica que pueden estar presente en la línea base de un sistema.

La deuda de requisitos: Incluye requisitos mal formados que superaron las revisiones iniciales además está relacionada con el control de la configuración y la trazabilidad de los requisitos, por otra parte, aumenta la densidad de defectos en la arquitectura y el diseño y contribuye al trabajo oculto en la implementación, integración y prueba (Rosser & Norton, 2021).

La deuda de arquitectura: Tiene un efecto perjudicial sobre la capacidad de mantener o evolucionar el sistema de software (Rosser & Norton, 2021).

La deuda de base de datos: Se puede acumular a través de problemas con los requisitos de datos, la arquitectura de datos, el diseño de la base de datos, la implementación, la documentación y la dependencia típica de un paquete de base de datos (Rosser & Norton, 2021).

Deuda de modelado y simulación: A medida que la ingeniería basada en modelos se convierte en una práctica estándar, línea de base técnica que está sujeta y que puede manifestarse como una menor productividad en el desarrollo y la integración (Rosser & Norton, 2021).

Deuda de automatización (generación): Es tanto un enfoque de origen como de resolución de la deuda técnica (Rosser & Norton, 2021).

Deuda de implementación. Fue una de las primeras fuentes reconocidas de deuda técnica (Rosser & Norton, 2021).

Deuda de documentación. Se relaciona con la utilidad y claridad de los materiales asociados con un sistema (Rosser & Norton, 2021).

La deuda de calidad: Se describe típicamente como la medida del esfuerzo para corregir los defectos que existen dentro de un sistema en un momento determinado (Rosser & Norton, 2021).

La deuda de integración: Se refiere al esfuerzo involucrado en conectar sistemas o componentes para lograr la funcionalidad deseada. Además, se minimiza mediante el uso de interfaces bien definidas y controladas entre componentes acoplados débilmente (Rosser & Norton, 2021).

La deuda de pruebas: El impacto de probar la deuda es la liberación tardía y la alta densidad de defectos de los sistemas. La deuda de pruebas a menudo se ve agravada por la falta de entornos de prueba adecuados y la falta de pruebas automatizadas (Rosser & Norton, 2021).

Deuda por depreciación: Se produce con el tiempo, como su nombre lo indica. Los sistemas de software que tienen longevidad incurren en una deuda de depreciación porque la tasa de cambio de tecnología a menudo supera incluso a un sistema de software bien escrito (Rosser & Norton, 2021).

2.2 Medición automatizada de la Deuda Técnica

La deuda técnica automatizada suma los esfuerzos de remediación de todos los elementos de deuda técnica detectados que se definen como ocurrencias de patrones que representan debilidades enumeradas en el código fuente automatizado. En cuanto algunas organizaciones pueden querer personalizar el cálculo de la Medida de Deuda Técnica Automatizada (ATDM) para reflejar las condiciones o prácticas locales. Dichas personalizaciones pueden excluir algunos patrones de código fuente del cálculo o ajustar los valores predeterminados para el esfuerzo de corrección. Estos ajustes pueden realizarse para toda la organización o para aplicaciones individuales. Sin embargo, la Medida de la Deuda Técnica Automatizada (ATDM) se calcula mediante el siguiente proceso (OMG, 2018):

1. Recopilar el código fuente de una o dos revisiones del software.
2. Generar el modelo de aplicación para la(s) revisión(es) disponible(s), ocupándose de las relaciones evolveTo/evolveFrom entre elementos de código cuando hay dos revisiones.
3. Detectar incidencias de los patrones de código fuente enumerados.
4. Calcular el esfuerzo corrector no ajustado para cada suceso.

En efecto, la presencia de TD es inevitable e incluso deseable en algunas circunstancias por una serie de razones, que a menudo pueden estar relacionadas con fuerzas comerciales o ambientales impredecibles internas o externas a la organización. A sí mismo, toda deuda técnica lleva aparejado un interés, o bien un costo adicional o impacto negativo que se genera por la presencia de una solución subóptima. En lo que sigue, las mejores prácticas actuales empleadas por las empresas de software incluyen mantener a raya la deuda técnica. Dicho de otra manera, las empresas no pueden permitir evitar o amortizar toda la deuda técnica que se genera continuamente. Es así que, los principales objetivos empresariales de las empresas son ofrecer continuamente valor a sus clientes y mantener sus productos (Lenarduzzi, T, Besker , Martini, & Arcelli, 2020).

2.3 Deuda técnica en la Práctica.

Simultáneamente, la definición convencional de la deuda técnica cubre en gran medida varios factores en cuanto al tema de requisitos, desarrollo, implementación, pruebas, arquitectura, documentación. Por esto, (Ernst, Kazman, & Delange, 2021) resaltan la importancia de como medir, monitorear el déficit técnico de dicha deuda técnica ya que con el tiempo puede salirse de control donde se va a requerir mayor esfuerzo en solucionar errores, generando a si un sobre costo para la organización. Llegados a este punto, la deuda técnica en el ciclo de vida del software se presenta cuando el equipo de desarrollo toma algunas decisiones técnicas no oportuna durante las etapas del proyecto

2.4 Gestión de la Deuda Técnica.

De la misma forma, la gestión de sobre la deuda técnica es de vital importancia para poder obtener éxito en cualquier tipo de proyecto de desarrollo de software. En efecto, hacer el proceso de resolver, verificar & validar, ayuda en si a mejorar la calidad del producto. Todas estas observaciones se, en marcar en que se recomienda tener un plan de acción con el objetivo de mitigar riesgos para luego no padecer inconveniente. Mientras tanto, solucionar la deuda técnica es una de las acciones más crítica, pero priorizarla genera un gran impacto en el software por su viabilidad y transparencia. A fin de, que esos equipos de desarrollo sean más eficientes (Holmegaard, 2023).

2.5 Deuda Técnica en proyecto de código Abierto.

A sí mismo, la deuda técnica puede acumularse en el código duplicado de una funcionalidad común entre dos componentes, afectando su futura mantenibilidad. Ya que la acumulación de deuda técnica es proporcional al tiempo y conlleva mayores costos de mantenimiento o evolución del software. Cosa parecida sucede también con, los problemas invisibles de la deuda técnica se deben a decisiones que afectan al estado del sistema con el paso del tiempo (Hoyos, 2021). Luego, se puede decir que el valor futuro de la deuda técnica podría facilitar las tareas de toma de decisiones relacionadas con el mantenimiento del software y ayudar a los desarrolladores, gerentes de proyectos a tomar acciones proactivas con respecto a la deuda técnica. Por el contrario, no existen contribuciones destacables en el campo de la predicción sobre deuda técnica, lo que indica que es un campo escasamente investigado (Tsoukalas, Kehagias, Siavvas, & Chatzigeorgiou, 2020). Ahora puedo decir, que la tendencia a la reutilización de deuda técnica en equipos de desarrollo en repositorios está generando un gran impacto por la forma en que se seleccionan e integran los componentes en los sistemas existentes. No obstante, reutilizar el software de manera oportunista puede provocar una pérdida de calidad e inducir en deuda técnica, especialmente cuando se realizan cambios en la arquitectura y en los procesos (Capilla, y otros, 2021). Podemos condensar lo dicho hasta aquí, que actualmente existen algunos estudios que analizan diversos aspectos de la evolución de la deuda técnica, pero no existe ningún estudio a gran escala que se centre en la remediación de la deuda técnica a lo largo del tiempo en proyectos Python, es decir, uno de los lenguajes de programación más completo en la industria de tecnología. Otro rasgo de, gestionar la deuda técnica tanto en repositorios de código abierto como en rastreadores de problemas pueden llevar a una supervisión más integral de esta actividad (Tan, Feitosa, & Avgeriou, 2023).

2.6 Evolución de la Deuda técnica.

Para ilustrar mejor, (Capilla, y otros, 2021) la evolución de las técnicas de TD en los últimos años ha dado lugar a una serie de herramientas comerciales y de investigación. Donde uno de los factores que puede influir en su estimación es la reutilización de componentes de terceros y su integración en un sistema existente. En pocas palabras (Efimova, 2021), la deuda técnica ocurre cuando un equipo de desarrollo acelera la entrega de un proyecto o funcionalidad que requerirá una refactorización más adelante. Es preciso decir que la deuda técnica provoca retrasos en el lanzamiento de nuevas funciones, obstaculiza la innovación y reduce la satisfacción laboral del equipo de ingeniería.

A continuación, en la tabla 1 se muestra una comparación sobe los tipos de deuda técnica, métodos y herramientas.

Tipo de deuda tecnológica	Método de seguimiento	Herramientas
Pequeños fragmentos de deuda: el tipo de deuda técnica que puede abordarse	Herramientas de análisis estático y estimación de la calidad del código	SonarQube SonarGraph Klockwork Codeclimate Teamscale VsCode Extension
	Herramientas de cobertura de pruebas.	Codecov PyCharm
Deudas medias: el tipo de deuda que puede abordarse en un sprint.	Gestión de proyectos	Atlassian jira Hansoft Squore
	Herramientas de análisis estático y estimación de la calidad del código	SonarQube SonarGraph Klockwork Codeclimate Teamscale VsCode Extension
	Gestión de la deuda técnica a 360	Tamaño del paso
Grandes piezas de deuda: la deuda tecnológica que no puede abordarse inmediatamente o incluso en un sprint.	Gestión de la deuda técnica a 360	Tamaño del paso

Tabla 1 Herramientas sobre Deuda Técnica. Tomado de (Efimova, 2021).

Dicho brevemente (Efimova, 2021), gestionar con éxito las deudas medianas y grandes se trata de adoptar los procesos correctos. El simple hecho de usar las herramientas sin adoptar el proceso correcto no es garantía de éxito. De manera semejante, en la tabla 2 se muestra una comparación con otras herramientas.

	JIRA	**Code Climate**	**Team Scale**	**SonarQube**	**Stepsize**
Identificar la deuda tecnológica en el editor.	X		X		
Integración con el flujo de trabajo de ingeniería.		X	X	X	
Impacto de la deuda en la hoja de ruta.		X	X	X	
Cuadros de mando para priorizar la deuda tecnológica.	X	X		X	
Representación gráfica de la deuda técnica.	X	X		X	
Cuantificar el impacto de la deuda técnica.	X	X	X	X	
Crear un caso de negocio para proyectos técnicos.	X	X	X	X	
Análisis automático del código.	X		X		X
Migración desde herramientas de documentación.	X	X	X	X	

Tabla 2 Comparación de herramientas sobre Deuda Técnica. Tomado de (Efimova, 2021).

Si bien la deuda técnica crece en números absolutos a medida que los sistemas de software evolucionan con el tiempo, la densidad se reduce en algunos casos. Es así que (Rosser & Norton, 2021), la identificación y evaluación proactiva de la deuda técnica del sistema puede mejorar nuestros procesos de estimación, planificación y toma de decisiones técnicas. En conclusión, tomar decisiones de diseño subóptimas durante el desarrollo de software conduce a la acumulación de Deuda Técnica en los proyectos de software (Perera, Tempero, Tu, & Blincoe , 2023). Existe una determinada clase de tipos de deuda técnica aplicada en varias interdisciplinas de la ingeniería lo cual hace pensar que su importancia en la actualidad.

2.7 Medición contextual de la deuda técnica (CTDM)

De manera semejante, algunas organizaciones pueden querer personalizar el cálculo de la Medida de Deuda Técnica Automatizada (ATDM) para reflejar las condiciones o prácticas locales. Dichas personalizaciones pueden excluir algunos patrones de código fuente del cálculo o ajustar los valores por defecto para el esfuerzo de corrección. Estos ajustes pueden realizarse para toda la organización o para aplicaciones individuales.

Los cálculos personalizados se designarán como medidas contextuales de la deuda técnica (CTDM) para distinguirlos del cálculo estándar (ATDM), que puede utilizarse para la evaluación comparativa con otras organizaciones o conjuntos de datos (OMG, 2018).

“En conclusión, estas preguntas ilustran las formas en que la medida de la Deuda Técnica Automatizada (ATDM) y la medida de la Deuda Técnica Contextual (CTDM) pueden utilizarse para ayudar a comunicar la Deuda Técnica con audiencias no técnicas. Con miras a (CTDM) pueda utilizarse para ayudar a comunicar la Deuda Técnica a un público no técnico. Para empezar, esto equivale a una estrategia de tolerancia cero a los defectos que puede ser demasiado estricta para todas las aplicaciones, así como costosa debido al gran número de casos que hay que eliminar. En segunda instancia, existe un debate conceptual sobre el contenido de la deuda técnica. Algunos dicen que la deuda técnica sólo debe contabilizar que las organizaciones tienen la intención de eliminar en algún momento. En último lugar, algunas organizaciones gestionan objetivos de calidad, como acuerdos de nivel de servicio internos o externos” Fuente propia.

Finalmente, estas medidas se incluyen en la especificación de la Deuda Técnica para proporcionar medidas estándar para su uso en la interpretación de la información. Aunque, las organizaciones pueden desarrollar sus propias medidas interpretativas, el uso de estas medidas interpretativas evita que una organización tenga que desarrollar sus propias fórmulas de ajuste sobre la Deuda Técnica (OMG, 2018).

3. Capitulo: Criterio de Selección.

No obstante, la literatura informa de varias métricas de software para líneas de productos desarrolladas mediante programación orientada a objetos. A su vez, comparten un conjunto común y gestionado de características que satisfacen las necesidades específicas de un segmento de mercado en particular (Abilio, Vale, Figueiredo, & Costa, 2016).

Conforme, a los repositorios de código abierto GitHub es probablemente el lugar ideal para encontrar proyectos interésantes que se pueden usar de forma gratuita y en los que se pueden contribuir. En esta sección se presentará las principales métricas de la literatura sobre la medición de deuda técnica.

3.1 Métricas en general.

En síntesis (McCabe, 1976), expresa que la teoría de grafos ilustra como se puede utilizar para administrar y controlar la complejidad ciclomática de los programas desde su tamaño hasta la estructura de decisión. Mas aun, cuando la complejidad es el grado en que un sistema tiene el diseño o una implementación que es difícil de entender y verificar (Mahdavi-Hezaveh, Ajmeri, & Williams, 2022). Igualmente, la comprensión del vínculo entre las actividades de los desarrolladores y la complejidad del software dentro de una tarea de refactorización, son las adecuadas para clasificar con precisión las diferentes prácticas de refactorización (Caldeira, Cardoso, & Abreu, 2020). Por otra parte (Mohan & Kumar, 2013), existen diferentes tipos de métricas utilizadas en entornos orientados a objetos. Que tienen implicaciones significativas para el diseño de productos de software de alta calidad utilizando el enfoque orientado a objetos. Es decir, los resultados pueden ser de gran ayuda para los ingenieros de calidad a la hora de seleccionar el conjunto adecuado de métricas para sus proyectos de software (Abilio, Vale, Figueiredo, & Costa, 2016), (Chidamber, Darcy, & Kemerer, 1998), (Tang, Kao, & Chen, 2002). En relación con, (Liebig, Apel, & Lengauer, 2010) presentan varias métricas que miden la variabilidad, la complejidad, la granularidad y los tipos de extensión aplicados (Caldeira, Cardoso, & Abreu, 2020). Al mismo tiempo, las métricas de diseño juegan un papel importante para ayudar a los desarrolladores a comprender los aspectos de diseño del software y, por lo tanto, mejorar la calidad del software, la productividad del desarrollador (Subramanyam & Krishnan, 2003), (Churcher, Shepperd, Chidamber, & Kemerer, 1998), (Mohan & Kumar, 2013), (Tang, Kao, & Chen, 2002). Para concluir, (Verma & Kumar, 2017) el software de código abierto se refiere al software con acceso sin restricciones para su uso o modificación.

Dado que, muchas organizaciones de desarrollo de software están utilizando esta metodología de código abierto en su proceso de desarrollo, es necesario recalcar que los desarrolladores de software pueden trabajar en paralelo con el proyecto de código abierto utilizando la web como un recurso compartido. En otras palabras, se requiere predecir la densidad de defectos de dichos proyectos con el fin de garantizar los estándares de calidad ya que las métricas estáticas para la predicción de densidad de defectos requieren la extracción de información abstracta del código.

Habría que decir también, que las métricas de repositorio, por otro lado, son fáciles de extraer de los conjuntos de datos del repositorio. Sin embargo, (Diamantopoulos, Papamichail, Karanikiotis, Chatzidimitriou, & Symeonidis, 2020) presentan una plataforma que analiza datos de GitHub para calcular una serie de métricas que cuantifican las contribuciones de los colaboradores del proyecto, tanto desde una perspectiva de desarrollo como de operaciones.

3.2 Necesidad de Métricas

Simultáneamente, (Caldeira, Cardoso, & Abreu, 2020) expresa que las actividades de comprensión y mantenimiento del software, como la refactorización, se ven afectadas negativamente por la complejidad del software. Es por esto que, el uso de proceso sobre métricas ayuda a predecir, identificar con porcentaje la falencias y errores de código en los proyectos de software ya que la gran mayoría carecen de pruebas de software eficiente, tecnología obsoleta, documentación, comentarios y métodos, clases mal definido, también arquitectura no centralizada donde los ambientes no se encuentran de manera configurada eficientemente generando A si una deuda técnica demasiada alta para las organizaciones. Como se afirmó arriba, la comprensión del vínculo entre las actividades de los desarrolladores y la complejidad del software dentro de una tarea de refactorización, se pueden evaluar teniendo presente las métricas de proceso recopiladas del IDE, donde la utilización de métodos y herramientas de minería de procesos, son adecuadas para clasificar con precisión las diferentes prácticas de refactorización y el resultado complejidad del software (Caldeira, Cardoso, & Abreu, 2020). Todas estas observaciones se relacionan también con los resultados de esta investigación donde los autores expresan que las métricas impulsadas por procesos, permiten predecir el tipo de método de refactorización (automático o manual) para la complejidad ciclomática. Con respecto a, las métricas orientadas a objetos resaltada por (Mohan & Kumar, 2013) donde informa una revisión sistemática de literatura muy amplia y detallada en cuanto a la clasificación de métricas según sus autores. Nos permite identificar que existe un sin números de métricas definida e inconclusa para su aplicación de medición en repositorio de código abierto ya que en su momento proporcionaban el análisis, recopilación de los datos para predecir la calidad del diseño. Por lo cual, muestran su interés en un futuro en crear nuevas métricas que puedan encontrar el impacto de la reusabilidad del software en la calidad del sistema utilizando las métricas CK y otras métricas disponibles en la literatura. De manera semejante, se hace necesario fortalecer el tema de medición de deuda técnica específicamente en métricas.

En particular, (Caldeira, Cardoso, & Abreu, 2020) estos autores resaltan un grupo de métricas de producto que son importante analizar ya que permiten tener su identificación, descripción y escala de medición. Es decir, (Caldeira, Cardoso, & Abreu, 2020) éste es el primer estudio en el que, utilizaron métodos de minería de procesos. Además, se recopilaron métricas de procesos y se combinaron con métricas de producto para comprender mejor la relación entre las dimensiones de producto y proceso, especialmente en la Complejidad ciclomática. Con respecto a, las métricas de proceso estas permiten modelar las tareas de desarrollo de software. Que, a su vez estas evalúan el uso de dichas métricas, pero con IDE (Integrated Development Environment),

como forma de mejorar los modelos existentes o, eventualmente, constituido. En este caso (Caldeira, Cardoso, & Abreu, 2020) ilustra algunas métricas del proceso descripción.

En lo que toca a, las métricas en repositorio de código abierto expreso un cuadro representativo con el nombre de la métrica y su fórmula, además las referencias a nivel de investigaciones identificada en la revisión de literatura.

3.3 Cuadro comparativo de Métricas

Métrica	Formula	Referencias	Número total
Complejidad Ciclomática	$v(G) = e - n + p.$	(McCabe, 1976), (Mahdavi-Hezaveh, Ajmeri, & Williams, 2022), (Caldeira, Cardoso, & Abreu, 2020)	3
Orientado a objetos	$\text{Comp}(f) = \{\text{Con}(f) \cup \text{Ref}(f)\}$ Similarly, the components of a system *s* can be expressed by: $\text{Comp}(s) = \{\text{Con}(s) \cup \text{Ref}(s)\}$	(Mohan & Kumar, 2013), (Abilio, Vale, Figueiredo, & Costa, 2016), (Chidamber, Darcy, & Kemerer, 1998), (Tang, Kao, & Chen, 2002)	4
Número de constantes (NOCt).	Esta métrica es el número de constantes (clases, interfaces) utilizadas para realizar una característica.	(Abilio, Vale, Figueiredo, & Costa, 2016)	1
Número de refinamientos (NOR)	Esta métrica es el número de refinamientos utilizados para realizar una característica.	(Abilio, Vale, Figueiredo, & Costa, 2016)	1
Número de componentes (NOC).	Esta métrica cuenta el número de componentes (constantes/refinamientos) fueron necesarios para implementar una característica.	(Abilio, Vale, Figueiredo, & Costa, 2016), (Churcher, Shepperd, Chidamber, & Kemerer, 1998)	2
Número de refinamientos constantes (NCR).	$\text{NCR} = \lvert \{ r \mid r \in \mathit{Ref(c)} \wedge c \in \mathit{Con(s)} \} \rvert$ Esta métrica es número de refinamientos que tiene una constante. NCR se inspiró en la métrica del número de hijos.	(Abilio, Vale, Figueiredo, & Costa, 2016)	1

Número de constantes refinadas (NRC).	Esta métrica es el número de constantes que tienen refinamientos. El valor de la métrica indica cuántas número de constantes refinadas en el sistema. $NRC = \| \{c \mid \exists r \in RefCon(c) \wedge c \in Con(s)\} \|$	(Abilio, Vale, Figueiredo, & Costa, 2016)	1
Número de refinamientos del método (NMR)	Esta métrica es número de refinamientos que tiene un método. Es decir, dado un método cuántos refinamientos tiene. $NMR = \| \{mr \mid mr \in MetRef(m) \wedge m \in Met(s)\} \|$	(Abilio, Vale, Figueiredo, & Costa, 2016)	1
Número de métodos refinados (NRM).	Esta métrica es el número de métodos que fueron refinados. El valor de la métrica indica la complejidad del mantenimiento debido a la interacción entre características. $NRM = \| \{m \mid \exists r \in RefMet(m) \wedge m \in Met(s)\} \|$	(Abilio, Vale, Figueiredo, & Costa, 2016)	1
Número de características con código (NFC).	Esta métrica se basa en la métrica NOF, que indica el número de características de un sistema. $NFC = \| \{f \mid \exists com \in Comp(f) \wedge f \in Fea(s)\} \|$	(Abilio, Vale, Figueiredo, & Costa, 2016)	1
Líneas de código (LOC)	La métrica LOC representa el tamaño de un sistema de software. Se mide contando el número de líneas nuevas de cada línea de código fuente.	(Liebig, Apel, & Lengauer, 2010), (Caldeira, Cardoso, & Abreu, 2020) (Meinicke, Hoyos, Vasilescu, & Kästner, 2020) (Hoyos, 2021) (Mahdavi-Hezaveh, Ajmeri, & Williams, 2022) (Abilio, Vale, Figueiredo, & Costa, 2016) (Diamantopoulos, Papamichail, Karanikiotis, Chatzidimitriou, & Symeonidis, 2020) (Chidamber, Darcy, & Kemerer, 1998)	11

		(Chidamber, Darcy, & Kemerer, 1998) (Subramanyam & Krishnan, 2003) (Verma & Kumar, 2017)	
Número de Constantes de Características (NOFC).	La métrica NOFC refleja directamente la dimensión de configuración de un SPL y, para ello, proporciona información sobre la variabilidad y la complejidad del SPL.	(Liebig, Apel, & Lengauer, 2010)	1
Líneas de código de características (LOF)	La métrica LOF es el número de líneas de código de características que están vinculadas a expresiones de características. Nos indica si una fracción pequeña o grande del código base es variable. Esta métrica se extrae contando el número de líneas entre dos #ifdef en el código fuente y las sumamos por proyecto.	(Liebig, Apel, & Lengauer, 2010)	1
Grado de dispersión (SD) y grado de enredo (TD).	La métrica SD es el número de apariciones de constantes de características en diferentes expresiones de características. La métrica TD es el número de constantes de características diferentes que aparecen en una expresión de rasgo.	(Liebig, Apel, & Lengauer, 2010)	1
Profundidad media de anidamiento de #ifdefs (AND).	La métrica AND refleja la profundidad media donde se calcula la media y la desviación estándar de todos los #ifdef en un proyecto.	(Liebig, Apel, & Lengauer, 2010)	1
Tipo (TYPE).	El programador etiqueta varias partes en el código fuente como código de características, ya sea con extensiones distintas (heterogéneas) o con la misma extensión utilizando duplicados de código (homogéneas).	(Liebig, Apel, & Lengauer, 2010)	1
Métodos ponderados por clase (WMC)	Es la suma ponderada de todos los métodos definidos en una clase.	(Subramanyam & Krishnan, 2003), (Churcher, Shepperd, Chidamber, & Kemerer, 1998),	4
Acoplamiento entre clases de objetos (CBO)	Es un recuento del número de otras clases a las que está acoplada una determinada clase y, por tanto, denota la dependencia de una clase respecto a otras clases del diseño.	(Mohan & Kumar, 2013), (Mahdavi-Hezaveh, Ajmeri, & Williams, 2022), (Abilio, Vale, Figueiredo, & Costa, 2016), (Churcher, Shepperd,	5

		Chidamber, & Kemerer, 1998) (Subramanyam & Krishnan, 2003)	
Profundidad del árbol de herencia (DIT)	Es la longitud del camino más largo desde una clase determinada hasta la clase raíz en la jerarquía de herencia.	(Subramanyam & Krishnan, 2003), (Mohan & Kumar, 2013), (Churcher, Shepperd, Chidamber, & Kemerer, 1998), (Mahdavi-Hezaveh, Ajmeri, & Williams, 2022), (Tang, Kao, & Chen, 2002), (Caldeira, Cardoso, & Abreu, 2020)	6
Número de hijos (NOC)	Es un recuento del número de clases hijas inmediatas que han heredado de una clase determinada.	(Subramanyam & Krishnan, 2003), (Mohan & Kumar, 2013), (Tang, Kao, & Chen, 2002)	3
RFC (Respuesta para una clase)	El RFC es el recuento del conjunto de todos los métodos que pueden ser invocados en respuesta a un mensaje a un objeto de la clase o por algún método de la clase.	(Mohan & Kumar, 2013), (Tang, Kao, & Chen, 2002)	2
LCOM (Falta de Cohesión de Métodos)	La cohesión de una clase se caracteriza por el grado de relación de los métodos locales con los atributos locales.	(Mohan & Kumar, 2013)	1
(MPC) Acoplamiento de paso de mensajes	se define como el número de sentencias de envío definidas en la clase C.	(Mohan & Kumar, 2013)	1
DAC (Datos Abstracción de datos Acoplamiento)	DAC(C) se define como el número de ADTs definidos en una clase C.	(Mohan & Kumar, 2013)	1
NOM (Número de Métodos)	NOM (C) se define como el número de métodos locales en una clase C.	(Mohan & Kumar, 2013)	1
SIZE1 (Tamaño de	SIZEI (C) se calcula contando el número de puntos y comas en una clase C.	(Mohan & Kumar, 2013)	1

procedimientos o funciones)			
SIZE2 (Tamaño de propiedades definidas en una clase)	SIZE2 (C) se calcula sumando el número de atributos y el número de métodos en una clase C.	(Mohan & Kumar, 2013)	1
Número de Antepasado Clases (NAC)	Esta métrica mide el número total de clases antecesoras de las que hereda una clase en la jerarquía de la herencia de clases.	(Mohan & Kumar, 2013)	1
Número de Descendiente Clases (NDC)	Esta métrica mide el número de clases que pueden estar potencialmente influenciadas por la clase debido a las relaciones de herencia.	(Mohan & Kumar, 2013)	1
Número de métodos locales Métodos (NLM)	Esta métrica cuenta el número de métodos locales definidos en una clase que son accesibles fuera de la clase	(Mohan & Kumar, 2013)	1
Clase Métodos Complejidad (CMC)	Esta métrica es la suma de la complejidad estructural interna de todos los métodos locales, independientemente de si son visibles o no fuera de la clase.	(Mohan & Kumar, 2013)	1
Acoplamiento mediante tipos de datos abstractos (CTA)	Esta métrica cuenta el número total de clases que se utilizan como tipos de datos abstractos en la declaración de atributos de datos de una clase.	(Mohan & Kumar, 2013)	1
Acoplamiento mediante paso de mensajes (CTM)	Esta métrica mide el número de mensajes diferentes enviados desde una clase a otras clases.	(Mohan & Kumar, 2013)	1
WAC (Atributo ponderado por clase)	WAC(C) se define como el número de atributos, en una clase C, ponderados por su tamaño.	(Mohan & Kumar, 2013)	1
(NO) Número de vagabundos	NOT(C) se define como el número total de parámetros extraños en las firmas de los métodos de una clase C.	(Mohan & Kumar, 2013)	1
(VOD) Violación de la ley de Demeter	VOD(C) se define como el número de violaciones de la ley de Deméter para una clase C.	(Mohan & Kumar, 2013)	1
Complejidad estática SC (P)	La complejidad estática se refiere a la pregunta "¿Cómo de compleja es la estructura de las clases definidas"? Entonces la complejidad estática se calcula utilizando las complejidades del método y de la clase.	(Mohan & Kumar, 2013)	1
Complejidad dinámica DC(P)	La complejidad dinámica está asociada con el flujo de control y la estructura del sistema. Se mide calculando el grado de reutilización que se invoca al llamar a un método.	(Mohan & Kumar, 2013)	1
Complejidad TC(P)	$TC(P)=SC(P) + DC(P)$	(Mohan & Kumar, 2013)	1

Coeficientes (Bi's):	Los coeficientes estimados de la regresión logística se calculan a partir de los datos mediante la maximización de la función de probabilidad logarítmica, y miden la contribución de la variable independiente respectiva (métrica) en la variable dependiente.	(Tang, Kao, & Chen, 2002)	1
La significación estadística (valor p)	Representa el grado de precisión de la estimación de los coeficientes. Más concretamente, el valor p representa la probabilidad de error que supone aceptar como válidos los resultados observados. $R^2 = \frac{LL_0 - LL}{LL_0}$	(Tang, Kao, & Chen, 2002)	1
La bondad del ajuste (R2):	Es un indicador donde el modelo se ajusta a los datos.	(Tang, Kao, & Chen, 2002)	1
Acoplamiento de la herencia: (IC)	El IC proporciona el número de clases padre a las que está acoplada una determinada clase.	(Tang, Kao, & Chen, 2002)	1
Acoplamiento entre métodos (CBM)	El CBM proporciona el número total de métodos nuevos/redefinidos a los que todos los métodos heredados están acoplados.	(Tang, Kao, & Chen, 2002)	1
Número de A signación de objetos/memoria	Mide el número total de sentencias que A signan nuevos objetos o memorias en una clase.	(Tang, Kao, & Chen, 2002)	1
Complejidad media del método: (AMC)	El AMC proporciona el tamaño medio del método para cada clase. Los métodos virtuales puros y métodos heredados no se cuentan.	(Tang, Kao, & Chen, 2002)	1
GINI	se puede observar que la propensión a los defectos aumenta para el proyecto a medida que aumenta el número de committers.	(Caglayan, Bener, & Koch, 2009)	1
Métricas estáticas	Las métricas estáticas se han utilizado en la mayoría de las investigaciones con software de código abierto para la predicción de densidad de defectos.	(Verma & Kumar, 2017)	1
Tamaño del software	Las medidas de tamaño del software tienen una aplicación directa en la planificación, el seguimiento y la estimación de los proyectos de software. El tamaño del software puede medirse en líneas de código (LOC) o puede medirse en número de puntos de función (FP)	(Verma & Kumar, 2017)	1
Defectos	los defectos son imperfecciones o deficiencias en un producto de trabajo que no cumplen con sus requisitos o especificaciones y deben ser reparados o reemplazados.	(Verma & Kumar, 2017)	1
Número de desarrolladores	Los desarrolladores del software de código abierto se definen como el número total número de personas que descargan el proyecto y contribuyen con alguna modificación/actualización en el proyecto.	(Verma & Kumar, 2017)	1

Número de descargas	Es el número total de descargas de los proyectos concretos indicados en las estadísticas del proyecto en el momento de la recogida de datos.	(Verma & Kumar, 2017)	1
Commits	En el software de código abierto, los commits registran básicamente los cambios realizados por los desarrolladores en el código fuente del proyecto.	(Verma & Kumar, 2017) (Alali, Kagdi, & Maletic, 2008)	2
Densidad de defectos.	La densidad de defectos del software puede definirse como la relación entre el número total de defectos y el tamaño total del software.	(Verma & Kumar, 2017)	1
Número de rutas añadidas en el código (Numérico)	Se calcula utilizando la Complejidad Ciclomática McCabe's.	(McCabe, 1976), (Mahdavi-Hezaveh, Ajmeri, & Williams, 2022)	2
El número de métodos de comprobación del valor de la característica (Numérico)	Se mide basándose en el concepto de métodos ponderados por clase (WMC). WMC es una de las métricas orientadas a objetos de CK	(Churcher, Shepperd, Chidamber, & Kemerer, 1998), (Mahdavi-Hezaveh, Ajmeri, & Williams, 2022)	2
Nombres reveladores de intenciones (binarios)	Para variables y métodos es una práctica conocida en codificación.	(Mahdavi-Hezaveh, Ajmeri, & Williams, 2022)	1
El uso de comentarios (binarios)	Ayuda a los desarrolladores a entender el propósito y el comportamiento de los conmutadores de características	(Mahdavi-Hezaveh, Ajmeri, & Williams, 2022)	1
El uso de una descripción (binaria).	Uso de una descripción (binaria) para cada interruptor de función.	(Mahdavi-Hezaveh, Ajmeri, & Williams, 2022)	1
Número de archivos (Numérico)	Contienen una función de conmutación, incluyendo configuración, código y archivos de prueba. Cuanto mayor sea el número de archivos que hay que cambiar para soportar la conmutación de funciones, mayor es la probabilidad de cometer un error.	(Mahdavi-Hezaveh, Ajmeri, & Williams, 2022)	1
Número de lugares (numérico)	A modo de ejemplo, considere un conmutador utilizado en dos archivos.	(Mahdavi-Hezaveh, Ajmeri, & Williams, 2022)	1
Líneas de código de función de conmutación (numérico)	Se asocian directamente con una función de conmutación se añade o se elimina de un repositorio.	(Mahdavi-Hezaveh, Ajmeri, & Williams, 2022)	1
Presencia de código duplicado (binario)	Es código smell, es decir, el código duplicado es un problema de repetición del mismo bloque de código.	(Mahdavi-Hezaveh, Ajmeri, & Williams, 2022)	1

Presencia de código muerto (binario)	El código muerto es una parte del código que no se utiliza en ninguna ruta de ejecución	(Mahdavi-Hezaveh, Ajmeri, & Williams, 2022)	1
Presencia de casos de prueba (binarios)	Para las conmutaciones de características es una métrica para medir si los cambios de características se prueban. Las funciones de conmutación deberían probarse de forma similar al código de implementación.	(Mahdavi-Hezaveh, Ajmeri, & Williams, 2022)	1

4. Capitulo: Métrica propuesta.

Por lo que se refiere a, la métrica se efectuó una búsqueda centralizada y detallada tal como se informó en los capitulo anteriores. Donde se simplifica las principales métricas en la literatura usadas para medir deuda técnica. En realidad, existen un sin números de investigaciones referentes a las métricas de producto & proceso (Caldeira, Cardoso, & Abreu, 2020) en segunda instancia, las métricas orientada a objetos (Mohan & Kumar, 2013) después, en Heurística y métrica estructurada (Mahdavi-Hezaveh, Ajmeri, & Williams, 2022) Enseguida, Predicción de la densidad de defectos del software de código abierto mediante métricas de repositorio (Verma & Kumar, 2017) Finalmente, Complejidad Ciclomática (McCabe, 1976) que son la base de métricas para ilustrar grafos donde se pueden utilizar en la administración de complejidad del programa. Deseo subrayar que, en el análisis e identificación de la métrica para medir la deuda técnica construida a partir del análisis de código fuente en proyectos de repositorios de código abierto se evidencias estas causas y necesidades que están relacionada específicamente con la Deuda técnica en código abierto. En cuanto a la definición de los criterios de selección se expresan de la siguiente manera.

En primer lugar, las “lines of code” es una métrica generalmente utilizada para evaluar un programa de software o base de código según su tamaño (Rouse, 2016). En segundo lugar “Modified code” Las líneas de código modificadas son inmediatamente visibles ya que se pueden detectar fácilmente (Windev, 2023). Luego, “Unit test” es un tipo de prueba de software donde se validan unidades o componentes individuales de un software. Después, “Comment on new code implementation” Si bien es fácil medir la cantidad de comentarios en un programa, es difícil medir la calidad y los dos no están necesariamente correlacionados. Un mal comentario es peor que ningún comentario (Spertus, 2021). A sí mismo, “Obsolete source code identification” es aplicable a todos los elementos del programa excepto ensamblados, módulos, parámetros y valores devueltos. Es decir, marcar un elemento como obsoleto permite informar a los usuarios que el elemento puede eliminarse en una versión futura del producto (Microsoft, 2023). Se debe agregar que, “Refinements or identification of classes and methods” la construcción de nuevas clases a partir de otras existentes mediante herencia o subclase es un rasgo característico del desarrollo orientado a objetos. La imposición de restricciones semánticas a las subclases permite garantizar que el comportamiento de las superclases se mantenga o mejore en sus subclases (Mikhajlova & Sekerinski, 1997). Hay que mencionar, “Project without documentation” un software sin documentación es un mal software, ya que probablemente se puede decir está mal escrito. Es por esto que, tenemos un problema de calidad en la industria del software y uno de los problemas es tener poca documentación o no tener documentación (Quora, 2018). Conforme a, “Version or tag periodicity” el proceso de publicación (releasing) de una nueva versión de software es uno de los pasos más importantes en un ciclo de vida de desarrollo ya que la periodicidad es uno de los factores más importantes (Jenik, 2021).

En resumen, “Pull request number” las solicitudes de incorporación de cambios son un mecanismo para que los desarrolladores notifiquen a los miembros de su equipo que han terminado una función. Una vez la rama de función está lista, el desarrollador realiza la solicitud de incorporación de cambios (Atlassian, 2023). En último lugar, “Number of special characters” es cualquier carácter no alfanumérico. Es decir, en Java, como en muchos otros lenguajes de programación existen un sin número de caracteres que efectúan una tarea o acción determinada en el código fuente del programa. En consecuencia, “Obsolete Technology” es un concepto que abarca software, hardware, lenguajes de programación, servicios o prácticas que ya no se utilizan (Pavel, 2023). En efecto, se realiza una búsqueda centralizada de varios repositorios tipo código abierto en la plataforma GitHub sobre organizaciones reconocida a nivel mundial con el objetivo de dar cumplimiento a la necesidad expuesta de la investigación de acuerdo a los criterios de definido anteriormente. Es por esto que, en la siguiente tabla 3 se listan los nueves repositorio de código abierto específicamente con el lenguaje Python donde se puede detallar su información correspondiente.

Tabla 3 Lista de repositorio open source preseleccionado.

Nombre	Repositorio - Github	Url	Sector económico	Año
Ryanair	Ryanair	https://github.com/Ryanair	Aeronautical	2023
Vodafone	Vodafone/chaostoolkit-elasticsearch	https://github.com/Vodafone/chaostoolkit-elasticsearch	Telecommunications	2023
Atlassian	atlassian/dc-app-performance-toolkit	https://github.com/atlassian/dc-app-performance-toolkit	Technology	2023
NBC News	nbcnews/DE-Coding-Challenge	https://github.com/nbcnews/DE-Coding-Challenge	Television	2022
HashiCorp	hashicorp/nomad-openapi	https://github.com/hashicorp/nomad-openapi	Technology	2023
Intuit	intuit/Trapheus	https://github.com/intuit/Trapheus	Technology	2023
UKG	UKG	https://github.com/orgs/UltimateSoftware/repositories	Technology	2023
SAP	SAP/SapMachine-infrastructure	https://github.com/SAP/SapMachine-infrastructure	Technology	2023
Square	square/square-python-sdk	https://github.com/square/square-python-sdk.git	Technology	2023

Por otra parte, se ordenan los repositorios desde la cantidad de commits, Realese, Pull request, branches, tags y por último # número de Project en lenguaje Python ya que con este criterio y objetivo nos permite tener un panorama de usabilidad e importancia al momento de seleccionar el repositorio sobre la librería. Donde se tendrá un puntaje según la característica de cada objetivo. En primer lugar, se expresan el siguiente criterio sobre unos objetivos de pregunta de cada ítem el cual tendrá un valor numérico que al final representará un peso.

- **A.** El repositorio está definido con lenguaje de programación Python.
- **B.** Cantidad de Commits hasta la fecha.
- **C.** Cantidad de Pull request hasta la fecha.
- **D.** Cantidad de Branches hasta la fecha.
- **E.** Cantidad de Releases hasta la fecha.
- **F.** Cantidad de Contributors hasta la fecha.
- **G.** Cantidad de Tags hasta la fecha.

En efecto, se ilustra en la siguiente tabla 4 el resultado sobre la totalidad de los criterios anteriormente referente al repositorio seleccionado.

Tabla 4 Totalidad de los criterios seleccionado.

RepositoryGithub	Languages	Commits	Pull request	Branches	Tags	Releases	Contributors	Row totals
Ryanair	1	1	1	1	1	1	1	7
Vodafone/chaostoolkit-elasticsearch	10	2	1	2	1	1	2	19
atlassian/dc-app-performance-toolkit	10	9	2	6	7	7	7	48
nbcnews/DE-Coding-Challenge	10	3	1	2	1	1	2	20
hashicorp/nomad-openapi	10	9	3	5	1	1	3	32
intuit/Trapheus	10	9	1	2	2	2	3	29
UKG	1	1	1	1	1	1	1	7
SAP/SapMachine-infrastructure	10	9	2	2	9	9	4	45
square/square-python-sdk	10	9	2	9	9	9	4	52
Row totals	72	52	14	30	32	32	27	

Como resultado, la sumatoria sobre cada uno de los criterios de la A-G de acuerdo a la puntuación de la escala 1-10 nos arroja la siguiente conclusión.

Tabla 5 Puntuación de los criterios seleccionado.

Objetivos	Puntuación	Descripción
A.	72	Repositorio con lenguaje de programación Python.
B.	52	Cantidad de Commits hasta la fecha.
C.	14	Cantidad de Pull request hasta la fecha.
D.	30	Cantidad de Branches hasta la fecha.
E.	32	Cantidad de Releases hasta la fecha.
F.	32	Cantidad de Contributors hasta la fecha.
G.	27	Cantidad de Tags hasta la fecha.

De igual modo, en la tabla 4 se evidencia que el repositorio “square/square-python-sdk” de la empresa Square obtuvo el primer lugar con un calificación alta sobre 52% de segundo lugar quedo el repositorio “atlassian/dc-app-performance-toolkit” empresa Atlassian con una calificación de 48% y tercero se encuentra el repositorio “SAP/SapMachine-infrastructure” con una calificación 45% por ultimo quedaron los repositorio de la empresas “Ryanair” & “UKG” los cuales fueron clasificado en su momento por su importancia como repositorio de código abierto.

Por otro lado, se ilustra la siguiente lista sobre los criterios base que nos permite hacer un análisis más cuantitativo de la selección definitiva sobre el repositorio código abierto en aplicación de la métrica a partir de la medición de deuda técnica. Para este escenario se utilizó una escala de 5 puntos (0-4) que suele tener una representación renumerada, donde (0-inadecuado,1-debil, 2-satisfactorio, 3-bueno,4-excelente) permitiendo A signar un factor peso a cada uno de los criterios sobre los repositorios de código abierto preseleccionado.

Tabla 6 Criterio de selección cuantitativo 1.

Objetivo	Criterio	Factor de peso	Significado
A.	Líneas de código	3	Bien
B.	Código modificado	4	Excelente
C.	Prueba unitaria	4	Excelente
D.	Comentario de nuevo código	3	Bueno
E.	Identificación de código fuente obsoleto	2	Satisfactorio

F.	Perfeccionamiento de clases y métodos	4	Excelente
G.	Clases y métodos	4	Excelente
H.	Periodicidad de la versión o etiqueta	4	Excelente
I.	Número de pull request	4	Excelente
J.	Número de caracteres especiales	2	Satisfactorio
K.	Tecnología obsoleta	1	débil

Tabla 7 Criterio de selección cuantitativo 2.

Repositorio Github	A	B	C	D	E	F	G	H	I	J	K	Sum
Ryanair	0	0	0	0	0	0	0	0	0	0	0	0
Vodafone/chaostoolkit-elasticsearch	3	3	2	3	1	3	2	1	1	2	1	22
atlassian/dc-app-performance-toolkit	3	4	3	3	2	3	4	4	3	2	1	32
nbcnews/DE-Coding-Challenge	3	2	2	3	2	3	2	2	2	2	1	24
hashicorp/nomad-openapi	3	3	3	3	2	3	2	3	2	2	1	27
intuit/Trapheus	3	3	3	3	2	3	2	3	3	2	1	28
UKG	0	0	0	0	0	0	0	0	0	0	0	0
SAP/SapMachine-infrastructure	3	3	3	3	2	3	3	3	2	2	1	28
square/square-python-sdk	3	4	4	3	2	4	4	4	4	2	1	35

En conclusión, según lo expresado anteriormente se realiza la selección del repositorio de código abierto (square/square-python-sdk) para la aplicación de la métrica, debido a que “Square” obtuvo una mayor puntación. Por lo cual, sirve para integrar los pagos de Square en aplicaciones y hace crecer cualquier tipo de negocio. Es preciso decir, que con las API de Square se incluyen una alta gama de servicios como Catálogo, Clientes, Empleados, Inventario, Mano de obra, Ubicaciones y Pedidos. En consecuencia , repositorio de empresa como SAP & Atlassian, Intuit se destacaron en este proceso de selecciòn en algunos de los ítem de “Unit test”, “Comment on new code implementation”, “Refinements or identification of class and methods” mientras que otros repositorio como lo son Vodafone & NBC y HashiCorp estuvieron un puntaje medio o bajo frente al cumplimiento de los criterios definidos para esta investigación es por eso que se nota una variación baja en algunos ítem en este caso el de Obsolete Technology. En consecuencia, el repositorio square-python-sdk nos permite cumplir con todos los criterios expuesto y objetivos en esta investigación ya que es un proyecto medible desde todos sus ámbitos.

4.1 Metodología utilizada.

Esta tesis de profundización busca proponer una métrica basada en el análisis de las métricas más usadas en la literatura para medir deuda técnica en repositorio de código abierto. La presente investigación es un caso de estudio con enfoque teórico & práctico, en la cual se identifica las principales métricas en la literatura usadas para medir deuda técnica. Con la finalidad de, evaluar la deuda técnica basada en el análisis de las principales métricas identificadas en la literatura. De modo que, se pueda validar la métrica propuesta aplicada al repositorio seleccionado.

En este caso la propuesta metodológica está en marca en los siguientes puntos:

1. Búsqueda centralizada acerca de métricas de la literatura, deuda técnica.
2. Identificación de las principales métricas en la literatura para medir deuda técnica.
3. Análisis de las investigaciones identificadas vs la seleccionada.
4. Definir los criterios de selección sobre el repositorio de código abierto.
5. Búsqueda y selección del repositorio de código abierto.
6. Análisis de componente principal sobre las métricas más usadas.
7. Proponer la métrica para medir deuda técnica.
8. Aplicar la métrica propuesta sobre consumo de la API del repositorio seleccionado.
9. Validar la métrica propuesta aplicada a la API del repositorio seleccionado.
10. Garantizar la viabilidad del estudio propuesto para futuras investigaciones sobre el tema.

4.2 Análisis de Datos Vs Métrica más usadas.

En cuanto, análisis de componentes principales (ACP) es uno de los métodos estadísticos de minería de datos más populares ya que permite extraer fácilmente información de grandes conjuntos de datos. Es por esto que, es necesario resaltar su importancia debido a su finalidad de identificar un conjunto de características, es decir, los PCA y sus métodos relacionando proporciona los medios para cualquier tipo de investigación en el resumir, extraer los datos haciendo que estos métodos sean particularmente útiles en la era del Big Data y la medicina personalizada (Kherif & Latypova, 2020). Además, Los componentes principales son algunas combinaciones lineales de las variables originales que explican al máximo la varianza de todas las variables. En el proceso, el método proporciona una aproximación de la tabla de datos original utilizando sólo estos componentes principales (Greenacre, Groenen, & Hastie, 2022) Al mismo tiempo, Technical Debt (TD) es una metáfora de los problemas técnicos que no son visibles para los usuarios y clientes pero que obstaculizan el trabajo de los desarrolladores, dificultando los cambios futuros. A menudo se incurre en TD debido a plazos ajustados de los proyectos y puede hacer que los cambios futuros sean más costosos o imposibles. La gestión de proyectos normalmente se centra en los beneficios para el cliente y presta menos atención a la calidad interna de sus sistemas de TI (Wiese, Rachow, Riebisch, & Schwarze, 2022). Mas aun, cuando varios estudios en la literatura investigan la identificación de la deuda técnica y sus consecuencias para que los desarrolladores de software permitan una planificación adecuada de las actividades de mantenimiento y mejora del software, como la refactorización y la prevención de la degradación del sistema (Ardimento, Aversano, Bernardi, Cimitile, & Iammarino, 2022). Por tanto, el ACP puede considerarse un método de minería de datos, ya que permite extraer fácilmente información de grandes conjuntos de datos (Lumivero, 2023). Además, se expresa que tiene varios usos, entre ellos:

- El estudio y visualización de las correlaciones entre variables para poder limitar el número de variables a medir posteriormente.
- La obtención de factores no correlacionados que son combinaciones lineales de las variables iniciales para poder utilizar estos factores en métodos de modelización como la regresión lineal, la regresión logística o el análisis discriminante.

- Visualizar las observaciones en un espacio bidimensional o tridimensional para identificar grupos uniformes o atípicos de observaciones.

Con esto quiero decir, el análisis de datos que se efectuó en esta de tesis de profundización están subjetos a las métricas de deuda técnica más usadas. Por otro lado, su aplicación al repositorio de código abierto de la empresa Square el cual contiene su propia API de forma interactiva, sin escribir ninguna línea de código eso la hace más usable a la hora de poder adaptarse a cualquier lenguaje de programación o entorno de desarrollo mediante su SDK, para todo tipo de sector o negocio donde se puede crear, verificar & validar, monitorear sus aplicaciones que se interconecta con los servicios y producto ofrecido por Square. Igualmente, se expresa la importancia de este tipo de repositorio el cual fue seleccionado para esta tesis cumpliendo con unas series de criterios los cuales fueron detallado anteriormente. En relación con, el desarrollo & análisis de una aplicación para consumir la información alojada en el repositorio GitHub sobre la librería de API square en Python, donde se utilizaron varios tipos de métricas de deuda técnica basada en el análisis de las más usadas como lo son Cyclomatic Complexity, Class coupling, Lines of code, Depth of inheritance permitiéndonos tener un análisis más completo a la hora de proponer nuestra métrica para evaluar la deuda técnica. En efecto, se expresa los principales factores de extracción sobre las siguientes métricas de acuerdo a los datos del repositorio square.

Tabla 8 Análisis de datos extraído métricas seleccionadas en el repositorio Square.

Es decir, se muestra las 6 métricas aplicada de deuda técnica en el análisis de la librería de API square.

Mantenibilidad	Complejidad ciclomática	Profundidad de herencia	Acoplamiento de clases	Líneas de código fuente	Líneas de código ejecutable
86	22	1	18	158	38
72	15	1	18	139	38
72	15	1	18	136	38
100	0	0	1	1	0
96	1	0	1	1	1
78	1	0	12	14	3
72	2	0	4	17	5
63	3	0	9	24	9
72	2	0	4	17	5
72	2	0	4	17	5
72	2	0	4	17	5
72	2	0	4	17	5
100	7	0	6	19	0
100	7	0	6	16	0
100	1	0	2	1	0
100	1	0	1	1	0
100	1	0	1	1	0
100	1	0	1	1	0
100	1	0	1	1	0

100	1	0	1	1	0
100	1	0	1	1	0
83	72	2	59	396	127
47	4	1	31	35	32
56	2	0	31	35	16
55	35	2	27	277	91
55	35	2	27	274	91
100	0	0	1	1	0
96	1	0	2	1	1
52	12	0	15	48	15
49	9	0	18	58	21
54	4	0	14	46	15
57	3	0	13	36	13
57	3	0	13	36	13
57	3	0	13	36	13
96	33	1	5	84	4
100	1	1	0	4	0
93	8	1	3	20	4
100	2	0	1	1	0
100	1	0	1	1	0
100	1	0	1	1	0
100	2	0	1	1	0
100	1	0	1	1	0
100	1	0	1	1	0
100	2	0	0	1	0
100	1	0	0	1	0
100	1	0	0	1	0
85	1	0	3	5	2
85	1	0	3	5	2
96	8	1	2	14	0
100	2	0	1	2	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	1	2	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	1	2	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	2	2	0
100	1	0	1	1	0
100	1	0	1	1	0

95	10	1	2	17	0
100	2	0	1	2	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	1	2	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	2	2	0
100	1	0	1	1	0
100	1	0	1	1	0
100	2	0	2	2	0
100	1	0	1	1	0
100	1	0	1	1	0
100	2	0	2	2	0
100	1	0	1	1	0
100	1	0	1	1	0
96	6	1	2	11	0
100	2	0	1	2	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	2	2	0
100	1	0	1	1	0
100	1	0	1	1	0
100	2	0	1	2	0
100	1	0	0	1	0
100	1	0	0	1	0
79	108	2	46	1334	291
100	12	2	10	18	1
100	12	2	10	15	1
100	1	0	3	1	0
100	2	0	2	1	0
100	1	0	2	1	0
100	1	0	2	1	0
100	2	0	2	1	0
100	1	0	2	1	0
100	1	0	2	1	0
100	2	0	2	1	0
100	1	0	2	1	0
100	1	0	2	1	0
100	2	0	2	1	0
100	1	0	2	1	0

100	1	0	2	1	0
100	2	0	2	1	0
100	1	0	2	1	0
100	1	0	2	1	0
100	1	0	2	4	1
97	80	1	10	139	0
98	4	1	3	8	0
100	2	0	2	2	0
100	1	0	1	1	0
100	1	0	1	1	0
100	2	0	2	2	0
100	1	0	1	1	0
100	1	0	1	1	0
98	10	1	3	12	0
100	2	0	1	2	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	0	1	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	0	1	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	0	1	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	2	2	0
100	1	0	1	1	0
100	1	0	1	1	0
90	20	1	8	39	0
100	2	0	1	2	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	1	2	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	3	3	0
100	1	0	1	1	0
100	1	0	1	1	0
100	2	0	1	2	0
100	1	0	0	1	0
100	1	0	0	1	0

100	2	0	3	3	0
100	1	0	1	1	0
100	1	0	1	1	0
100	2	0	3	3	0
100	1	0	1	1	0
100	1	0	1	1	0
100	2	0	1	2	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	3	3	0
100	1	0	1	1	0
100	1	0	1	1	0
100	2	0	3	3	0
100	1	0	1	1	0
100	1	0	1	1	0
100	2	0	3	3	0
100	1	0	1	1	0
100	1	0	1	1	0
98	4	1	3	8	0
100	2	0	2	2	0
100	1	0	1	1	0
100	1	0	1	1	0
100	2	0	2	2	0
100	1	0	1	1	0
100	1	0	1	1	0
98	4	1	1	8	0
100	2	0	1	2	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	1	2	0
100	1	0	0	1	0
100	1	0	0	1	0
99	14	1	3	14	0
100	2	0	1	2	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	0	1	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	0	1	0
100	1	0	0	1	0

100	1	0	0	1	0
100	2	0	0	1	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	0	1	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	0	1	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	2	2	0
100	1	0	1	1	0
100	1	0	1	1	0
100	10	1	1	13	0
100	2	0	1	2	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	0	1	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	0	1	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	0	1	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	0	1	0
100	1	0	0	1	0
100	1	0	0	1	0
100	10	1	1	10	0
100	2	0	1	2	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	0	1	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	0	1	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	0	1	0
100	1	0	0	1	0
100	1	0	0	1	0

100	2	0	0	1	0
100	1	0	0	1	0
100	1	0	0	1	0
98	4	1	1	8	0
100	2	0	1	2	0
100	1	0	0	1	0
100	1	0	0	1	0
100	2	0	1	2	0
100	1	0	0	1	0
100	1	0	0	1	0
51	16	2	31	1177	290
51	3	2	28	201	59
38	1	0	13	166	54
70	1	0	11	19	4
94	1	0	3	6	1
55	3	2	28	165	41
68	1	0	11	24	4
94	1	0	3	6	1
43	1	0	13	125	36
52	3	2	21	288	54
61	1	0	5	75	9
61	1	0	5	87	9
43	1	0	13	116	36
59	3	2	25	123	32
60	1	0	11	37	8
85	1	0	3	9	2
51	1	0	10	67	22
38	1	2	16	169	54
38	1	0	13	165	54
53	3	2	28	176	50
41	1	0	13	141	45
70	1	0	11	19	4
94	1	0	3	6	1

En síntesis, el análisis de la deuda técnica está ganando popularidad a medida que hoy en día los investigadores y la industria están adoptando diversas herramientas de análisis de código estático para evaluar la calidad de su código. A pesar de esto, los estudios empíricos sobre proyectos de software son costoso debido al tiempo necesario para analizarlo. Además, los resultados son difíciles de comparar ya que los estudios suelen considerar proyectos diferentes (Lenarduzzi, Saarimäki, & Taibi, The Technical Debt Dataset, 2019). Precisamente, las empresas de software necesitan gestionar y refactorizar los problemas de deuda técnica.

Por lo tanto, es necesario comprender si se debe priorizar la refactorización de la deuda técnica y cuándo con respecto al desarrollo de funciones o la corrección de errores (Taibi & Lenarduzzi, 2021). En lo que sigue, a los datos expresado en la tabla 6, nos plasma un análisis de cada una de las métricas y variables representada donde se muestra a continuación los principales factores de extracción para este tipo de datos.

4.2.1 Análisis factorial - Extracción de componentes principales

En cuanto a, los factores de extracción implican elegir el tipo de modelo y la cantidad de factores a extraer. Es por esto, que la rotación de factores se produce después de que se extraen los factores, con el objetivo de lograr una estructura simple para mejorar la interpretabilidad. Es decir, hay dos enfoques para la extracción de factores que surgen de diferentes enfoques para la partición de la varianza: En primer lugar, análisis de componentes principales. En segundo lugar, análisis de factores comunes. En relación con, el análisis de componentes principales o PCA se supone que no existe una varianza única, la varianza total es igual a la varianza común tal como lo expresa (UCLA, 2021). De igual manera, en la siguiente tabla 9 se muestra los factores de extracción de análisis de componentes principal (PCA) que se aplicaron en cada una de las métricas clasificada en la medición de los datos extraído en el repositorio Python client library for the Square API.

Tabla 9 Análisis factorial - Extracción de componentes principales

	Mantenibilidad	Complejidad ciclomática	Profundidad de herencia	Acoplamiento de clases	Líneas de código fuente	Líneas de código ejecutable
Media	93,13846154	3,542307692	0,176923077	3,665384615	27,91923077	6,830769231
Desviación estándar	15,57830084	10,22498541	0,496557798	7,597471755	120,0197105	29,29958121
A simetría	-2,212386034	7,325674259	2,817692747	3,640871935	8,707360158	7,646814462
Kurtosis	3,542537384	61,38372726	6,909496031	16,56903025	86,23942414	68,27366695

Es preciso decir, que en la tabla 9 de Análisis factorial - Extracción de componentes principales, nos muestra unos ítems de variables: Media, Desviación estándar, A simetría, Kurtosis. Que en su media la métrica de Mantenibilidad nos arroja un resultado muy alto a comparación de la A simetría, Kurtosis. Mientras que la métrica profundidad de herencia está por debajo de las otras métricas a nivel de Mean. Pero la métrica línea de código fuente, se encuentra a un nivel mucho mayor en cada una de las variables dando a entender su importancia a la hora de medir & calcular la deuda técnica.

4.2.2 Correlation Matrix

Mientras tanto, el objetivo de un PCA es replicar la matriz de correlación utilizando un conjunto de componentes que sean menos en número y combinaciones lineales del conjunto original de elementos. En la tabla 8 se muestra las siguientes variables como lo son:

- Índice de mantenibilidad.
- Complejidad ciclomática.
- Profundidad de la herencia.
- Acoplamiento de clases.
- Líneas de código fuente.
- Líneas de código ejecutable.

Similarmente, estas variables a su vez son métricas que se aplicaron en la medición de los datos extraído en el repositorio Python client library for the Square API, arrojando unos resultados medibles tal como se detalla en esta tabla 10 de Correlation Matrix con una totalidad de 11,78231377.

Tabla 10 Correlation Matrix.

	Mantenibilidad	Complejidad ciclomática	Profundidad de herencia	Acoplamiento de clases	Líneas de código fuente	Líneas de código ejecutable
Mantenibilidad	1	-0,163675445	-0,395991347	-0,728481168	-0,431055753	-0,500957785
Complejidad ciclomática	-0,163675445	1	0,555926144	0,614119232	0,678329252	0,635362861
Profundidad de la herencia	-0,395991347	0,555926144	1	0,692245267	0,571324938	0,584841475
Acoplamiento de clases	-0,728481168	0,614119232	0,692245267	1	0,703473067	0,761216799
Líneas de código fuente	-0,431055753	0,678329252	0,571324938	0,703473067	1	0,98392685
Líneas de código ejecutable	-0,500957785	0,635362861	0,584841475	0,761216799	0,98392685	1
Total	1,151051374	1,5768025	1,613518271	2,461356123	2,435338224	2,544247279

4.2.3 Inversa de la matriz de correlación

Es importante, resaltar que la inversa de una matriz de correlación es la matriz de precisión ya que juega un papel muy importante en la estadística. Para comprender mejor, en la tabla 11 se muestra los resultados inversos sobre dicha matriz entre las variables expuesta en términos de su asociación lineal.

Tabla 11 Inversa de la matriz de correlación

	Mantenibilidad	Complejidad ciclomática	Profundidad de herencia	Acoplamiento de clases	Líneas de código fuente	Líneas de código ejecutable
Mantenibilidad	3,032458958	-1,532594341	-0,335587587	3,324607368	1,375451856	-1,194938004
Complejidad ciclomática	-1,532594341	3,388571368	-0,101624068	-3,013254871	-6,991854775	6,311910375

Profundidad de la herencia	-0,335587587	-0,101624068	2,09017033	-1,602944699	-1,197431483	1,072408065
Acoplamiento de clases	3,324607368	-3,013254871	-1,602944699	8,068241895	9,884289671	-11,34963255
Líneas de código fuente	1,375451856	-6,991854775	-1,197431483	9,884289671	54,30379154	-55,12333014
Líneas de código ejecutable	-1,194938004	6,311910375	1,072408065	-11,34963255	-55,12333014	58,64069989

4.2.4 Matriz de correlaciones parciales

Luego, la matriz de correlación parcial calcula los coeficientes de las columnas de una matriz. Que, a su vez esta también es una matriz simétrica es por esto que en la tabla 12 se muestra los resultados obtenidos de acuerdo a las variables aplicada sobre los datos extraído en el repositorio Python client library for the Square API con una totalidad de 6,282372136.

Tabla 12 Matriz de correlaciones parciales

	Mantenibilidad	**Complejidad ciclomática**	**Profundidad de herencia**	**Acoplamiento de clases**	**Líneas de código fuente**	**Líneas de código ejecutable**
Mantenibilidad	1	0,478102961	0,133296143	-0,67213025	-0,107184713	0,08960832
Complejidad ciclomática	0,478102961	1	0,038185406	0,576286032	0,515429183	-0,447767876
Profundidad de la herencia	0,133296143	0,038185406	1	0,390335674	0,112394422	-0,096865619
Acoplamiento de clases	-0,67213025	0,576286032	0,390335674	1	-0,472216255	0,521786422
Líneas de código fuente	-0,107184713	0,515429183	0,112394422	-0,472216255	1	0,976834067
Líneas de código ejecutable	0,08960832	-0,447767876	-0,096865619	0,521786422	0,976834067	1
Total	0,71762759	1,028309471	0,19360338	1,431475864	1,466981297	1,444374534

4.2.5 KMO

En seguida, esta KMO el cuál es la Medida Kaiser-Meyer-Olkin de adecuación de muestreo estadístico que indica la proporción de varianza en las variables que pueden ser causadas por factores subyacentes. Donde, los valores altos generalmente indican que un análisis factorial puede ser útil con los datos. Si el valor es menor que 0,50, los resultados del análisis factorial probablemente no serán muy útiles según (IBM, 2023). En la tabla 13 se detalla el KMO de la medida sobre los datos para el análisis de cada una de las variables expresada anteriormente dando A si un resultado total 0,652229097.

Tabla 13 Kaiser-Meyer-Olkin (KMO).

Mantenibilidad	Complejidad ciclomática	Profundidad de herencia	Acoplamiento de clases	Líneas de código fuente	Líneas de código ejecutable
0,615970638	0,605272448	0,892866438	0,632279053	0,624074531	0,637876289

4.2.6 Valores y vectores propios.

Consideremos ahora, la teoría espectral que hace referencia a los valores propios y vectores propios de una matriz. Es necesario recalcar que, son vectores que no cambian su dirección cuando se les aplica una transformación lineal. Por esto, que en la tabla 14 se representan su magnitud para una transformación lineal de acuerdo a los datos extraído en el repositorio Python client library for the Square API según las variables expresada anteriormente.

Tabla 14 Valores y vectores propios.

4,062706111	0,914275957	0,556172045	0,344147272	0,113980742	0,008717873
-0,315119027	-0,77439191	-0,068262894	-0,228101612	-0,493826706	-0,021498505
0,371512461	-0,530527109	-0,165294192	0,673747917	0,302714383	0,087321562
0,381839393	-0,047864727	-0,749580058	-0,510693823	0,170211461	0,016039378
0,45283615	0,249722421	-0,155417431	0,278364676	-0,781462301	-0,142314215
0,449129718	-0,207185691	0,443675616	-0,273148041	0,132864784	-0,682821815
0,458365669	-0,106207002	0,430284922	-0,284830606	-0,084798979	0,710743647

4.2.7 Full Load Matrix.

No obstante, Full Load Matrix es un enfoque sistemático para generar nuevos programas de validación u optimizar los existentes. Ya que se basa en factores de aceleración de prueba específica sobre cada componente a la vez también se utiliza en modelos estadístico. Es preciso decir, que el uso del enfoque Load Matrix ayuda a validar programas de prueba completos y, al mismo tiempo, evita pruebas excesivas poco realistas según lo expresa (Hick & Denkmayr, 2004). Se debe agregar, que en tabla 14 expresamos dicho enfoque frente a cada una de las variables donde nos arroja que la métrica de Lines of Executable code, Lines of Source code, Class Coupling son un poco eficiente a la hora de validar y optimizar mientras que las otras como Cyclomatic Complexity, Maintainability Index tiene un nivel muy bajo sobre las tres métricas anteriores dando A si como resultado una mayor fiabilidad, usabilidad y efectividad la métrica de line of code. En la siguiente tabla 15 se puede detallar dicha afirmación.

Tabla 15 Full Load Matrix.

	1	2	3	4	5	6
Mantenibilidad	-0,635158815	-0,740456341	-0,05090838	-0,133813682	-0,166721002	-0,002007306
Complejidad ciclomática	0,7488263	-0,507278236	-0,123271356	0,395247927	0,102199506	0,008153175
Profundidad de la herencia	0,769641424	-0,045767189	-0,559013894	-0,299593765	0,057465149	0,00149759

Acoplamiento de clases	0,912743593	0,238779031	-0,115905569	0,163300039	-0,263829753	-0,013287814
Líneas de código fuente	0,90527285	-0,198106355	0,330879712	-0,160239749	0,044856525	-0,063754765
Líneas de código ejecutable	0,923888978	-0,101552776	0,320893342	-0,167093216	-0,028629012	0,066361814

4.2.8 Scree Plot

Algo semejante, ocurre con Scree Plot que hace parte de los enfoques sobre análisis de componente principal (PCA) donde ayuda a la toma de decisiones. A su vez, es un método común para determinar la cantidad que se conservara en una representación gráfica muy conocida como diagrama scree Plot. Ya que es un gráfico de segmento de línea simple que muestra los valores propios de cada PC individual. Por lo tanto, cuando los valores propios disminuyen drásticamente de tamaño, un factor adicional agregaría relativamente poco a la información ya extraída de acuerdo con (Mangale, 2020). En el caso de, la tabla 16 sobre Scree Plot nos muestra unos valores y componente de menor, mayor porcentaje según las variables anteriormente expresada. Sin embargo, la figura 2 nos representa de manera gráfica dicha escala.

Tabla 16 Scree Plot.

eValue	%	Cum %
4,062706111	67,71%	67,71%
0,914275957	15,24%	82,95%
0,556172045	9,27%	92,22%
0,344147272	5,74%	97,96%
0,113980742	1,90%	99,85%
0,008717873	0,15%	100,00%

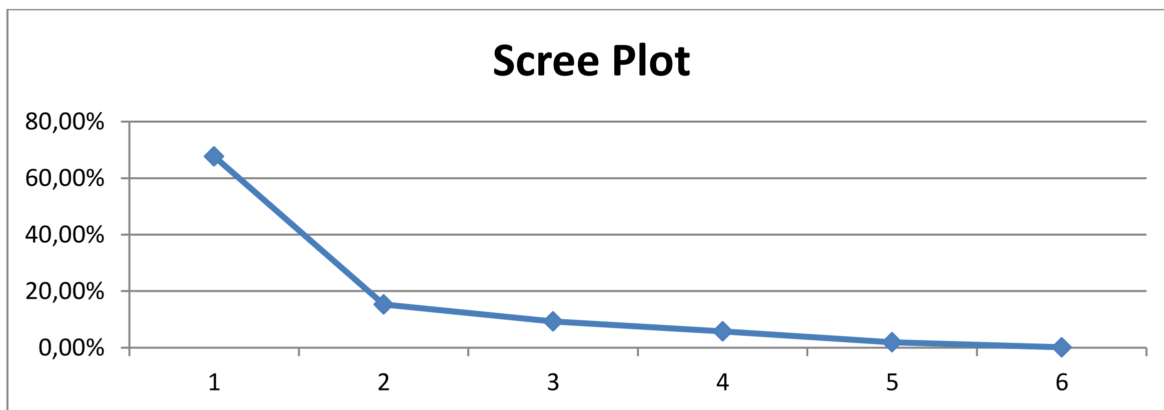


Figura 2 Scree Plot.

De modo que, esto se debe a que el primer componente suele representar gran parte de la variabilidad, los siguientes componentes explican una cantidad moderada y los últimos componentes sólo explican una pequeña fracción de la variabilidad general (Mangale, 2020).

4.2.9 Factor Matrix (unrotated) vs Factor Matrix (rotated Varimax)

A cerca de, los factores rotados o no rotados son irrelevante cuando se refiere a un solo factor, pero cuando son dos o más factores se debe rotar, ya que los no rotados son bastante difícil de interpretar. Teniendo en cuenta que, la matriz de factores rotados o no rotados las siguientes tablas 17 y 18 contiene la ponderación de dichas variables también su relación dando A si una interpretación más significativa ya que estos factores a nivel estructural son iguales como podemos detallar en las tablas 17 y 18.

Tabla 17 Factor Matrix (unrotated).

	1	COMUN	ESPECIFICO
MANTENIBILIDAD	-0,635158815	0,403426721	0,596573279
COMPLEJIDAD CICLOMÁTICA	0,7488263	0,560740827	0,439259173
PROFUNDIDAD DE LA HERENCIA	0,769641424	0,592347921	0,407652079
ACOPLAMIENTO DE CLASES	0,912743593	0,833100866	0,166899134
LÍNEAS DE CÓDIGO FUENTE	0,90527285	0,819518933	0,180481067
LÍNEAS DE CÓDIGO EJECUTABLE	0,923888978	0,853570843	0,146429157
	4,062706111	4,062706111	1,937293889

Tabla 18 Factor Matrix (rotated Varimax).

	1	COMMUN	SPECIFIC
MANTENIBILIDAD	-0,635158815	0,403426721	0,596573279
COMPLEJIDAD CICLOMÁTICA	0,7488263	0,560740827	0,439259173
PROFUNDIDAD DE LA HERENCIA	0,769641424	0,592347921	0,407652079
ACOPLAMIENTO DE CLASES	0,912743593	0,833100866	0,166899134
LÍNEAS DE CÓDIGO FUENTE	0,90527285	0,819518933	0,180481067
LÍNEAS DE CÓDIGO EJECUTABLE	0,923888978	0,853570843	0,146429157
	4,062706111	4,062706111	1,937293889

4.2.10 Reproduced Correlation Matrix.

En síntesis, la matriz de correlación reproducida es la matriz de correlación basada en los factores extraídos. Donde los valores sean lo más parecidos posible a los valores de la matriz de correlaciones original. Como se ha dicho en este caso la tabla 19 se muestra los resultados sobre la matriz correlación de acuerdo a los datos extraído en el repositorio Python client library for the Square API según las variables expresada.

Tabla 19 Reproduced Correlation Matrix.

	Mantenibilidad	Complejidad ciclomática	Profundidad de herencia	Acoplamiento de clases	Líneas de código fuente	Líneas de código ejecutable
Mantenibilidad	0,403426721	-0,475623626	-0,488844535	-0,579737139	-0,574992031	-0,586816229
Complejidad ciclomática	-0,475623626	0,560740827	0,57632774	0,683486407	0,677892119	0,691832365
Profundidad de la herencia	-0,488844535	0,57632774	0,592347921	0,702485278	0,696735485	0,711063228
Acoplamiento de clases	-0,579737139	0,683486407	0,702485278	0,833100866	0,826281993	0,843273745
Líneas de código fuente	-0,574992031	0,677892119	0,696735485	0,826281993	0,819518933	0,836371608
Líneas de código ejecutable	-0,586816229	0,691832365	0,711063228	0,843273745	0,836371608	0,853570843

4.2.11 Error Matrix.

Sobre todo, la matriz de covarianza de errores (ECM) es un conjunto de datos que especifica las correlaciones en los errores de observación entre todos los pares posibles de niveles verticales. Se presenta como una matriz bidimensional, de tamaño NxN, donde N es el número de niveles verticales en los productos de datos de sondeo. Para este caso la tabla 20 sobre Error Matrix nos muestra una correlaciones y variaciones de acuerdo a las variables o métrica Acoplamiento de clases con unos datos menor hacia otras variables de Profundidad de la herencia, Líneas de código fuente, que sería importante tener presente a la hora de tomar una decisión frente a la métrica propuesta.

Tabla 20 Error Matrix.

	Mantenibilidad	Complejidad ciclomática	Profundidad de herencia	Acoplamiento de clases	Líneas de código fuente	Líneas de código ejecutable
Mantenibilidad	0,596573279	0,311948181	0,092853188	-0,148744029	0,143936278	0,085858444
Complejidad ciclomática	0,311948181	0,439259173	-0,020401595	-0,069367175	0,000437134	-0,056469503
Profundidad de la herencia	0,092853188	-0,020401595	0,407652079	-0,010240011	-0,125410547	-0,126221753
Acoplamiento de clases	-0,148744029	-0,069367175	-0,010240011	0,166899134	-0,122808927	-0,082056946
Líneas de código fuente	0,143936278	0,000437134	-0,125410547	-0,122808927	0,180481067	0,147555242
Líneas de código ejecutable	0,085858444	-0,056469503	-0,126221753	-0,082056946	0,147555242	0,146429157

4.2.12 Factor Scores Matrix - Regression Method.

De manera semejante, los métodos alternativos para calcular las puntuaciones de los factores son la regresión, Bartlett y Anderson-Rubin.

Es decir, las puntuaciones que se producen tienen una media de 0 y una varianza igual a la correlación múltiple al cuadrado entre factores estimado y los valores reales (IBM, 2021). En resumen, la tabla 19, 20 y 21 detalla unas puntuaciones a cada una de las variables y métrica analizada sobre la información extraída en el repositorio Python client library for the Square API.

Tabla 21 Factor Scores Matrix - Regression Method.

Mantenibilidad	-0,156338853
Complejidad ciclomática	0,184317122
Profundidad de la herencia	0,189440585
Acoplamiento de clases	0,224663948
Líneas de código fuente	0,222825089
Líneas de código ejecutable	0,227407288

Sin embargo, en la tabla 21 de Factor Scores Matrix - Regression Method. se muestra los coeficientes por los que se multiplican las variables para obtener puntuaciones de los factores.

Tabla 22 Matriz de puntuaciones factoriales - Método de Bartlett.

Mantenibilidad	-0,0567299
Complejidad ciclomática	0,090835108
Profundidad de la herencia	0,100598675
Acoplamiento de clases	0,291399094
Líneas de código fuente	0,267264535
Líneas de código ejecutable	0,336190713

Finalmente, tras un análisis factorial exploratorio, las puntuaciones pueden calcularse y utilizarse en análisis posteriores. En pocas palabras las puntuaciones factoriales son variables compuestas que proporcionan información sobre la posición de un individuo en sus factores (DiStefano, Zhu, & Mîndrilã, 2019). Como podemos observar en la tabla 23 se detalla el cálculo sobre el Factor Scores Matrix - Anderson-Rubin's Method que fue determinado de análisis total sobre las variables & métrica propuesta. Para concluir, el análisis de todos estos métodos descrito anteriormente para la crear de las puntuaciones nos ayudaron a tener un mayor entendimiento a la hora de tomar una decisión más practica en la puesta en marcha de la métrica.

Tabla 23 Factor Scores Matrix - Anderson-Rubin's Method.

	357,2355704

En conclusión, tabla 15: sobre Full Load Matrix, revela que las métricas Lines of Executable code, Lines of Source code, Class Coupling tiene tienen unos valores muy alto con respecto hacia las otras métricas como los son Maintainability Index, Cyclomatic Complexity, Depth of Inheritance, que están a un valor bajo.

4.3 Técnica de datos sobre la métrica aplicada.

En cuanto a, el análisis de los datos sobre las métricas más usadas me permito expresar lo siguiente, En definitiva, la implementación de los diferentes factores & técnica estadística como lo son:

- Análisis factorial - Extracción de componentes principales.
- Correlation Matrix
- Inversa de la matriz de correlación
- Matriz de correlaciones parciales
- KMO
- Valores y vectores propios
- Full Load Matrix
- Scree Plot
- Factor Matrix (unrotated)
- Factor Matrix (rotated Varimax)
- Reproduced Correlation Matrix
- Error Matrix
- Factor Scores Matrix - Regression Method
- Factor Scores Matrix - Bartlett's Method
- Factor Scores Matrix - Anderson-Rubin's Method

“Como se ha Dicho, anteriormente en su especificación este análisis estadístico sobre los datos extraído nos ha permitido tener una mayor claridad a la hora de poder cumplir con el objetivo general de la investigación ya que se puede decir, que las métricas Depth of Inheritance, Class Coupling, Lines of Source code tiene una mayor nivel de puntuación, efectiva de medir deuda técnica en cualquier tipo de repositorio open source en este caso de estudio el repositorio que fue seleccionado cumpliendo unos criterio especifico fue: repositorio Square https://github.com/square/square-python-sdk/ el cual nos permite identificar e implementar, verificar la fórmula de una manera centralizada de acuerdo al tipo de lenguaje de programación que este caso para su puesta en marcha se utilizó C# por su efectividad e integración del consumo de API hacia cualquier tipo de archivo” Fuente Propia. Por lo que se refiere a, la formula se expresa de la siguiente manera el método Full Load Matrix nos muestra que las métricas pueden tener diferentes umbrales a nivel de componente (clase) y de método. Por otra parte, algunas métricas pueden ser aplicables sólo en un nivel, mientras que otras pueden ser aplicables tanto a nivel de clase como de método a continuación se realiza una descripción sobre dichas métricas:

Depth of Inheritance: Indica el número de clases diferentes que heredan entre sí, hasta la clase base. La profundidad de herencia es similar al acoplamiento de clases en el sentido de que un cambio en una clase base puede afectar a cualquiera de sus clases heredadas (Microsoft, 2023).

Class Coupling: mide el acoplamiento a clases únicas a través de parámetros, variables locales, tipos de retorno, llamadas a métodos, instancias genéricas o de plantilla, clases base, implementaciones de interfaz, campos definidos en tipos externos y decoración de atributos (Microsoft, 2023).

Lines of Source code: indica el número exacto de líneas de código fuente que están presentes en su archivo fuente, incluidas las líneas en blanco (Microsoft, 2023).

Lines of Executable code: indica el número aproximado de líneas de código ejecutable u operaciones. Este es un recuento del número de operaciones en el código ejecutable (Microsoft, 2023).

“En síntesis, la métrica propuesta está basada en las tres primeras métricas descrita anteriormente y relacionada a él análisis de componente principal. En primer lugar, sería la métrica Depth of Inheritance. En segundo lugar, Class Coupling y en seguida, la métrica Lines of Source code. Es preciso decir, que la métrica Lines of Executable code no se aplicó en cuenta a la hora de poner en ejecución la formula ya que se evidencia su poca aplicación en la literatura en cuanto a referencia teniendo en cuenta que sería un recuento de la métrica de Lines of source code, pero su soporte a nivel de referencia quedaría corto para nuestra investigación. Por lo tanto, se tomó la decisión de poder trabajar con dichas métricas descrita (Depth of Inheritance, Class Coupling, Lines of Source code) esto con el fin de dar cumplimiento al objetivo general de la tesis de maestría” Fuente Propia.

4.4 Simplificación de métrica propuesta.

Métrica = Peso 1*Normalización Depth of Inheritance + Peso 2* Normalización Class Coupling + Peso 3* Normalización Lines of Source code.

El proceso de normalización se realiza con el fin de obtener valores de la métrica en el rango de 0 a 1, no solamente para el repositorio usado, sino, para que pueda ser comparable con otros resultados obtenidos en varios repositorios.

4.5 Formulación sobre la métrica aplicada.

A continuación, se describe la formulación de la métrica propuesta.

1. Para empezar, se efectúa una normalización por cada una de las tres métricas.

Luego, se realiza el cálculo de normalización de cada métrica usada en el repositorio Square sobre la cantidad de total, es decir:

- **Columna 1:** Depth of Inheritance # total divido el máximo valor de la columna 1.

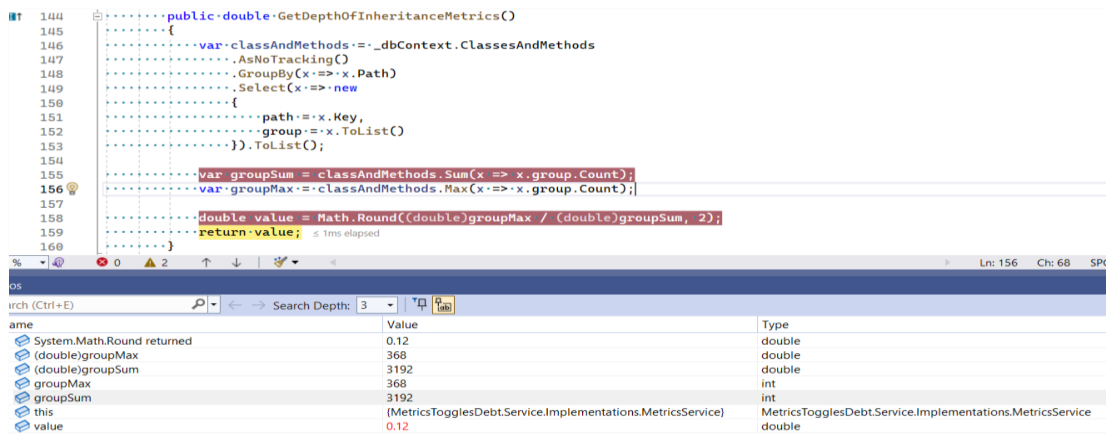


Figura 3 Normalización Depth of Inheritance.

- **Columna 2:** Class Coupling # total divido el máximo valor de la columna 2.

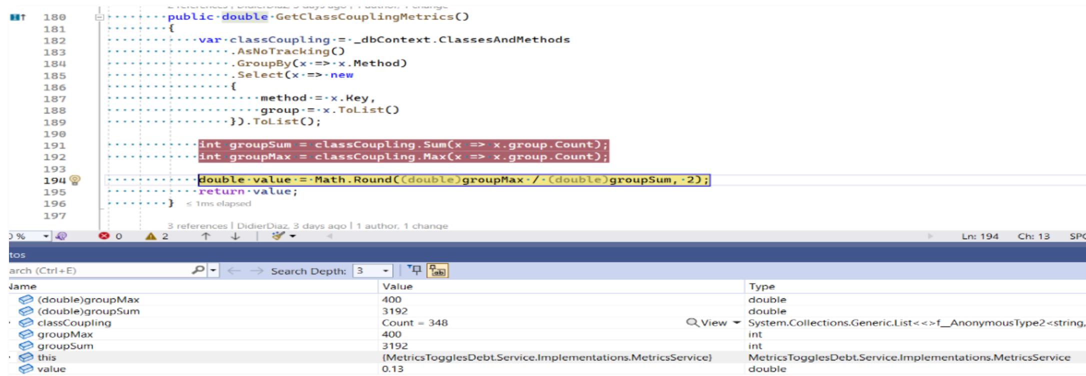


Figura 4 Normalización Class Coupling.

- **Columna 3:** Lines of Source code # total divido el máximo valor de la columna 3.

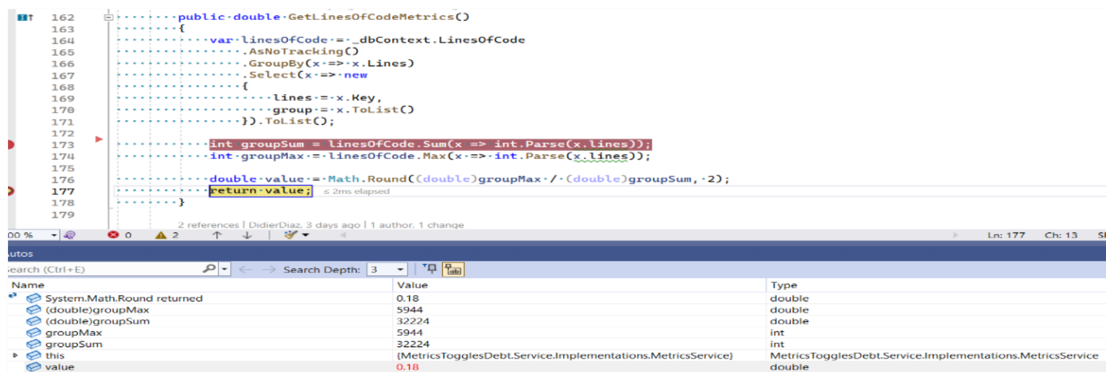


Figura 5 Normalización Lines of Source code.

Mientras tanto, la normalización de columna sobre cada métrica nos arroja el siguiente resultado de acuerdo a la tabla 24 donde se visualiza normalización de métrica usada.

Tabla 24 Normalización de la métrica.

Metrics	Cantidad	Suma	# Mayor	Calculo
Depth of Inheritance	51	3192	368	0,12
Class Coupling	384	3192	400	0,13
Lines of Source code	182	32224	5994	0,18

4.6 Cálculo de pesos sobre la métrica aplicada.

2. En segunda instancia, se calcula los pesos que son tomado de la literatura.

Donde, se identifica las referencias de literatura sobre la métrica Depth of Inheritance, Class Coupling, Lines of Source code donde se habla de cada una.

- **#** Referencia sobre la métrica Depth of Inheritance divido el # total de referencia.
- **#** Referencia sobre la métrica Class Coupling divido el # total de referencia.
- **#** Referencia sobre la métrica Lines of Source code divido el # total de referencia.

Entonces el número de referencia se muestra en la siguiente tabla 25 donde podemos visualizar los pesos de cada referencia luego el número total de referencia divido por cada peso de referencia.

Tabla 25 Cálculo de pesos vs métrica sobre referencia.

Metrics	Referencia	Peso	Tipo
Depth of Inheritance	6	0,29	Peso 1 Normalización
Class Coupling	5	0,24	Peso 2 Normalización
Lines of Source code	10	0,48	Peso 3 Normalización
Total, de Referencia	21	1,00	Calculo

Donde, la normalización del peso 1,2 y 3 se efectúa a la cantidad de referencia de literatura expresada en la tabla 25 donde se encuentra las tres métricas (Depth of Inheritance, Class Coupling, Lines of Source code) más usadas de la literatura.

En efecto, la formula se efectuaría de la siguiente manera de acuerdo a lo escrito anteriormente en la tabla de pesos y normalización de cada métrica.

Métrica = 0,29*0,12+ 0,24* 0,13 + 0,48*0,18.

Finalmente, con el propósito de poder realizar el cálculo de la normalización de cada métrica según el resultado de la cantidad divido por la suma total de la métrica tal como se pueden visualizar en la tabla 24 Normalización de métrica.

4.7 Ilustración de cálculo métrica más usadas.

Para simplificar, se expresa un analisis detallado sobre las seis métricas más usadas (Mantenibilidad, Complejidad ciclomática, Profundidad de herencia, Acoplamiento de clases, Líneas de código fuente, Líneas de código ejecutable) frente a la métrica propuesta donde de manera preliminar se representa los siguiente registro de acuerdo a los datos y la dependencia de herencia que en este caso fueron tres capas las cuales estan clasificada según la arquitectura implementada para consumir la API del repositorio Square.

- Project: MetricsTogglesDebt.Data
- Project: MetricsTogglesDebt.API
- Project: MetricsTogglesDebt.Service

Donde se calculan los siguientes datos extraído sobre las métricas más usadas en este caso la tabla 26 nos muestra dicha información donde se observa una cantidad mayor en las métricas de líneas de código ejecutable.

Tabla 26 Resultado de Calculo métricas más usadas.

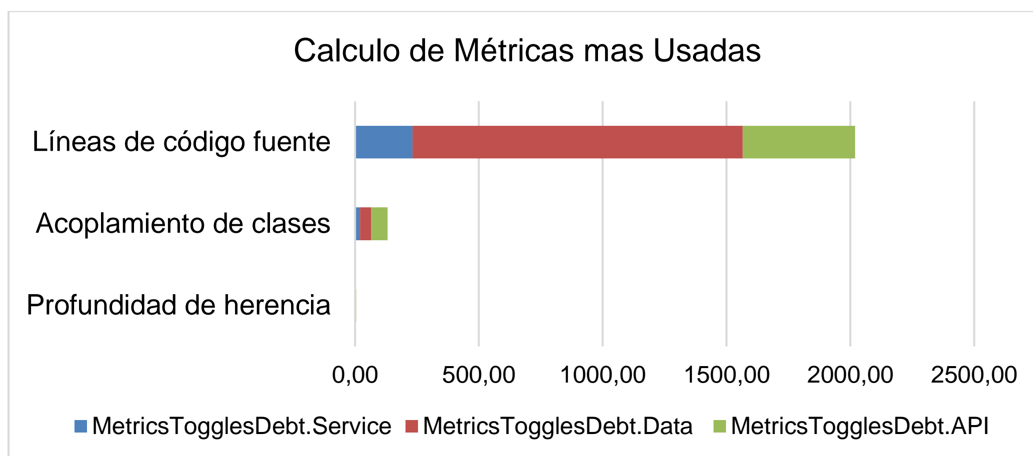
Project	Mantenibilidad	Complejidad ciclomática	Profundidad de herencia	Acoplamiento de clases	Líneas de código fuente	Líneas de código ejecutable
MetricsTogglesDebt.Service	84	32	1	20	232	69
MetricsTogglesDebt.Data	79	108	2	46	1334	291
MetricsTogglesDebt.API	83	81	2	65	453	141

Como resultado, la tabla 26 en la parte del Project sobre la capa de MetricsTogglesDebt.Service tiene la mejor mantenibilidad y la menor complejidad y acoplamiento, siendo el módulo más sencillo de mantener y entender. Luego MetricsTogglesDebt.Data presenta la mayor complejidad y tamaño, lo que podría dificultar su mantenimiento. En último lugar, MetricsTogglesDebt.API tiene una buena mantenibilidad, pero su alta complejidad ciclomática y acoplamiento pueden ser áreas de preocupación.

En pocas palabras, se debería prestar especial atención a MetricsTogglesDebt.Data, MetricsTogglesDebt.API para mejorar su mantenibilidad y reducir su complejidad y acoplamiento. Por esto, estas métricas juntas proporcionan una visión completa del código fuente de cualquier proyecto de software open source, facilitando la gestión proactiva, priorizando A si su resolución de deuda técnica, según su impacto en el negocio donde se recomienda realizar revisiones de código enfocadas en calidad. Todavía cabe señalar, en la figura 6 para medir la deuda técnica, es crucial analizar la profundidad de herencia, el acoplamiento de clases y las líneas de código fuente. En efecto, al monitorear y optimizar estas métricas, se puede reducir la deuda técnica, haciendo el código más fácil de mantener, entender y evolucionar. Dicho brevemente, la métrica de profundidad de herencia se detalla: Un bajo nivel sugiere una estructura de clases simple y fácil de mantener. Mientras que en la métrica de acoplamiento de clases: se visualizan valores bajos indican menor dependencia entre clases, favoreciendo el modularidad y facilitando cambios en el código.

Finalmente, la métrica línea de código fuente: Mantiene un tamaño moderado ayudando a evitar complejidad innecesaria en el código.

Figura 6 Resultado cálculo métrica más usadas.



Definitivamente, las métricas con más usabilidad frente al tema de la medición de la deuda tecnica en el repositorio open source Square fue la métrica de líneas de código ya que se obtiene un mayor nivel de efectividad a la hora de poder medir, calcular cualquier proyecto de software.

4.8 Ilustración cálculo métrica propuesta Vs Mas usadas.

Como resultado, en este apartado expresamos ya una comparación conjunta sobre la métrica propuesta vs las métricas más usadas teniendo presente los datos anteriormente analizados según el modelo de analisis de componente principales ya que nos permitió tener una mayor objetividad, sustento a la hora de poder proponer, medir nuestras métricas hacia las demás. Hecha esta salvedad, en la tabla podemos visualizar el cálculo de resultados.

Tabla 27 Resultado de Calculo Métrica Propuesta & Mas Usadas.

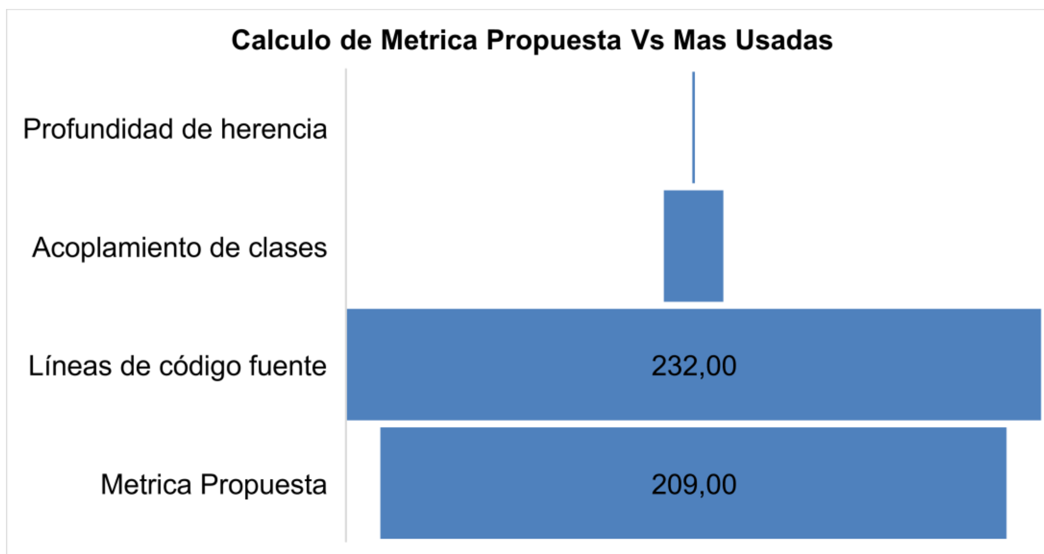
Project	Profundidad de herencia	Acoplamiento de clases	Líneas de código fuente	Métrica Propuesta
MetricsTogglesDebt.Service	1	20	232	209
MetricsTogglesDebt.Data	2	46	1334	22
MetricsTogglesDebt.API	2	65	453	60
Metrics Technical-Debt	0,12	0,13	0,18	0,152

Sintetizado la tabla 27 el análisis de las métricas de profundidad de herencia, acoplamiento de clases y líneas de código fuente revela diferentes niveles de deuda técnica entre los proyectos. Es decir, MetricsTogglesDebt.Service muestra una baja profundidad de herencia (1), acoplamiento de clases moderado (20) y un tamaño de código manejable (232 líneas), resultando en una métrica propuesta alta (209), indicando buena calidad del código.

Mas aun, cuando MetricsTogglesDebt.Data tiene una mayor profundidad de herencia (2), alto acoplamiento (46) y un tamaño de código considerable (1334 líneas), reflejando una métrica propuesta baja (22), lo que sugiere una mayor deuda técnica. MetricsTogglesDebt.API también tiene una profundidad de herencia de 2, acoplamiento de clases alto (65) y un tamaño de código moderado (453 líneas), con una métrica propuesta intermedia (60). La métrica Technical-Debt combinada muestra valores relativos de 0.12 para profundidad de herencia, 0.13 para acoplamiento de clases y 0.18 para líneas de código fuente, resultando en una métrica propuesta general de 0.152.

Estos resultados indican que MetricsTogglesDebt.Service es el más mantenible, mientras que MetricsTogglesDebt.Data presenta mayores desafíos en términos de deuda técnica.

Figura 7 Resultado cálculo métrica propuesta Vs Mas usadas.



Podemos condensar lo dicho hasta aquí, en la figura 7 se analizaron métricas de software en el repositorio "Square" y se obtuvieron los siguientes resultados: La métrica propuesta, con valores de 0,12 en profundidad de herencia, 0,13 en acoplamiento de clases, 0,18 en líneas de código fuente y 0,15 en una métrica general, proporciona una visión de la estructura y complejidad del código en términos de herencia, acoplamiento y tamaño. Este enfoque puede ser aplicado en otros repositorios. En consecuencia, La deuda técnica en repositorios open source se puede medir utilizando la métrica propuesta, que considera la profundidad de herencia, el acoplamiento de clases, las líneas de código fuente y una métrica general. Ya que estos valores permiten evaluar la complejidad y calidad del código en proyectos diversos.

5. Capítulo: Validación de la métrica propuesta.

5.1 Ruta de validación sobre la métrica aplicada.

Con respecto a, la métrica propuesta para evaluar la deuda técnica basada en el análisis de las principales métricas identificadas en la literatura. Se hace necesario expresar lo siguiente en referente a la validación:

1. Identificación de las métricas más usadas para la medición de deuda técnica.
2. Luego, la selección del repositorio de código abierto Biblioteca cliente Python para la API de Square https://github.com/square/square-python-sdk/ de acuerdo a los criterios de selección. En seguida, el análisis estático del repositorio seleccionado.
3. A si mismo, la verificación & validación del análisis de literatura sobre métrica & deuda técnica ya que nos permite tener una mayor credibilidad y confianza en la investigación propuesta. Por otra parte, el análisis & desarrollo de las métricas más usadas en el repositorio seleccionado square-python-sdk.
4. Seguido, se efectuó un análisis de los datos extraído con la técnica de estadística Análisis de componentes principales hacia las métricas más usada.
5. Desarrollo e implementación con herramientas de BD y lenguajes de programación ya que se utilizó a nivel de bases de datos “PostgreSQL” en la parte de Backend “Microsoft Visual Studió 2022 con el entrono C# Net Core 6 hacia el consumo de API. Y en cuanto a la parte FrontEnd se realizó con “Angular 17” dando A si una exclusividad en tecnologías y herramientas avanzada para ejecutar & validar la métrica propuesta.
6. Finalmente, se implementa la formula sobre la métrica propuesta teniendo en cuenta los objetivos específico y general de la investigación a nivel de código fuente tanto para Backend y Frontend alojado en la plataforma GitHub.
7. Código fuente en Backend: https://github.com/DidierDiaz/Metrics-Toggles-Debt
8. Código fuente en Frontend: https://github.com/DidierDiaz/MetricsAPIFrontend

Teniendo en cuenta que, el cálculo de normalización de dichas métricas se implementa y se efectúa directamente desde el consumo de nuestra API desarrollada sobre el repositorio descrito anteriormente tal como se muestra a continuación en el desarrollo sobre código fuente.

5.2 Código fuente en Backend de normalización de la métrica aplicada.

Normalización GetDepthOfInheritanceMetrics

```
public double GetDepthOfInheritanceMetrics()
{
    var classAndMethods = _dbContext.ClassesAndMethods
        .AsNoTracking()
        .GroupBy(x => x.Path)
        .Select(x => new
        {
            path = x.Key,
            group = x.ToList()
        }).ToList();

    var groupSum = classAndMethods.Sum(x => x.group.Count);
    var groupMax = classAndMethods.Max(x => x.group.Count);

    double value = Math.Round((double)groupMax / (double)groupSum, 2);
    return value;
```

Normalización GetClassCouplingMetrics

```
public double GetClassCouplingMetrics()
{
    var classCoupling = _dbContext.ClassesAndMethods
        .AsNoTracking()
        .GroupBy(x => x.Method)
        .Select(x => new
        {
            method = x.Key,
            group = x.ToList()
        }).ToList();

    int groupSum = classCoupling.Sum(x => x.group.Count);
    int groupMax = classCoupling.Max(x => x.group.Count);

    double value = Math.Round((double)groupMax / (double)groupSum, 2);
    return value;
}
```

Normalización GetLinesOfCodeMetrics

```
public double GetLinesOfCodeMetrics()
{
    var linesOfCode = _dbContext.LinesOfCode
        .AsNoTracking()
        .GroupBy(x => x.Lines)
        .Select(x => new
        {
            lines = x.Key,
            group = x.ToList()
        }).ToList();
```

```
        int groupSum = linesOfCode.Sum(x => int.Parse(x.lines));

        int groupMax = linesOfCode.Max(x => int.Parse(x.lines));

        double value = Math.Round((double)groupMax / (double)groupSum, 2);

        return value;
    }
```

5.3 Tecnologias & herramientas utilizadas sobre la métrica aplicada.

En cuanto, a la verificación & validación de la métrica propuesta aplicada al repositorio Square como caso de estudio. Sobre, el cumplimiento de este objetivo se realiza el desarrollo e implementación de nuestra formula mediante el cálculo de referencia de literaturas de las métricas usada en el repositorio Square ya que para esto se efectuó el desarrollo con la siguiente tecnología:

Microsoft Visual Studio 2022: IDE de Visual Studio es una plataforma de lanzamiento creativa que puede utilizar para editar, depurar y compilar código y, finalmente, publicar una aplicación. Además del editor y depurador estándar que ofrecen la mayoría de IDE, Visual Studio incluye compiladores, herramientas de completado de código, diseñadores gráficos y muchas más funciones para mejorar el proceso de desarrollo de software (Microsoft, 2024).

Visual Studio Code: Visual Studio Code es un editor de código fuente ligero pero eficaz que se ejecuta en el escritorio y está disponible para Windows, macOS y Linux. Incluye compatibilidad integrada con JavaScript, TypeScript y Node.js, y cuenta con un amplio ecosistema de extensiones para otros lenguajes y entorno de ejecución (como C++, C#, Java, Python, Go, .NET) (Microsoft, 2024).

PostgreSQL: Es un potente sistema de bases de datos objeto-relacional de código abierto con más de 35 años de desarrollo activo que le ha granjeado una sólida reputación por su fiabilidad, robustez de funciones y rendimiento (PostgreSQL, 2023).

Angular: Es un marco de diseño de aplicaciones y una plataforma de desarrollo para crear aplicaciones de una sola página eficaces y sofisticadas (Angular, 2023).

De acuerdo con, el proceso de base de datos relacionado con el consumo de la implementación de nuestra API para el desarrollo de las métricas más usada de la literatura como referencia del repositorio seleccionado square a continuación, se resalta los scripts que efectuó de la BD y las tabla (ClassesAndMethods, Commits, LinesOfCode, Remotes, Tags, __EFMigrationsHistory) para el consumo de los datos extraído en formato Json de dicho repositorio.

5.4 Esquema a nivel de Base datos sobre la métrica aplicada.

```
BEGIN;

CREATE TABLE IF NOT EXISTS public."ClassesAndMethods"

(

    "Id" integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),

    "Path" text COLLATE pg_catalog."default",

    "Method" text COLLATE pg_catalog."default",

    "Found" boolean NOT NULL,

    "FoundPaths" jsonb,

    CONSTRAINT "PK_ClassesAndMethods" PRIMARY KEY ("Id")

);

CREATE TABLE IF NOT EXISTS public."Commits"

(

    "Id" integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),

    "Sha1" text COLLATE pg_catalog."default",

    "Parents" jsonb,

    "Tree" text COLLATE pg_catalog."default",

    "Author" jsonb,

    "Committer" jsonb,

    "Message" text COLLATE pg_catalog."default",

    "GPG" jsonb,

    "Refs" jsonb,

    "Tags" jsonb,

    CONSTRAINT "PK_Commits" PRIMARY KEY ("Id")

);

CREATE TABLE IF NOT EXISTS public."LinesOfCode"
```

```
(
   "Id" integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),
   "File" text COLLATE pg_catalog."default",
   "Lines" text COLLATE pg_catalog."default",
   "Language" text COLLATE pg_catalog."default",
   "Comments" text COLLATE pg_catalog."default",
   "Blank" text COLLATE pg_catalog."default",
   "SpecialCharacters" jsonb,
   CONSTRAINT "PK_LinesOfCode" PRIMARY KEY ("Id")
);
CREATE TABLE IF NOT EXISTS public."Remotes"
(
   "Id" integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),
   "Name" text COLLATE pg_catalog."default",
   "SHA1" text COLLATE pg_catalog."default",
   "Type" text COLLATE pg_catalog."default",
   "Size" integer NOT NULL,
   CONSTRAINT "PK_Remotes" PRIMARY KEY ("Id")
);
CREATE TABLE IF NOT EXISTS public."Tags"
(
   "Id" integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),
   "Tag" text COLLATE pg_catalog."default",
   "SHA1" text COLLATE pg_catalog."default",
   "Type" text COLLATE pg_catalog."default",
```

```
    "Size" integer NOT NULL,
    CONSTRAINT "PK_Tags" PRIMARY KEY ("Id")
);
CREATE TABLE IF NOT EXISTS public."__EFMigrationsHistory"
(
    "MigrationId" character varying(150) COLLATE pg_catalog."default" NOT NULL,
    "ProductVersion" character varying(32) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "PK___EFMigrationsHistory" PRIMARY KEY ("MigrationId")
);
END;
```

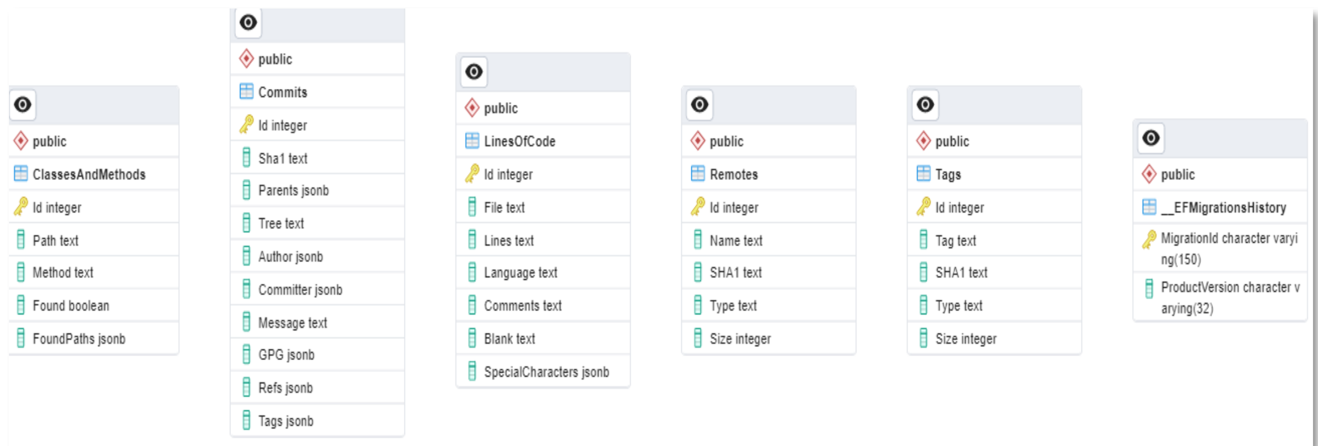


Figura 8 ERD Bases de datos – MetricsTogglesDebt.

5.5 Esquema a nivel de Backend sobre la métrica aplicada.

Todavía cabe señalar, que para la integración de Backend se realizó utilizó mediante API (Interfaz de programación de aplicaciones) debido a su funcionalidad, mantenibilidad de adaptación con otras aplicaciones por lo que nos permite hacer integración de manera paralela con otros servicios. Llegado a este punto, en la siguiente figura 8 se muestra a nivel de vista el desarrollo de algunas métricas de la literatura que se implementaron en su momento el repositorio square para la validación correspondiente y poder medir su nivel de importancia a la hora de extraer información de los datos para A si tener insumo en el cálculo de nuestra métrica propuesta.

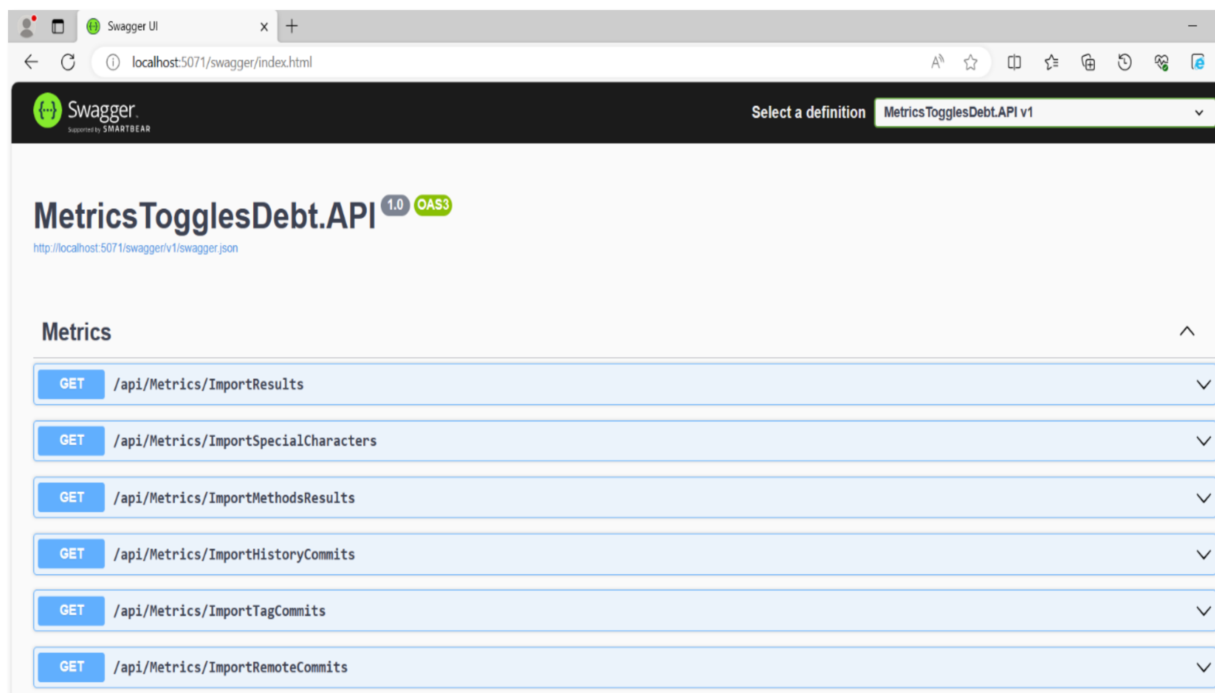


Figura 9 Vista de MetricsController.cs

A si que, las métricas de literaturas implementada que se efectuaron para el análisis, fueron generada con un tipo de operación y recurso GET donde nos permitió tener base para proponer & validar la métrica nuestra.

- GET/api/Metrics/ImportResults
- GET/api/Metrics/ImportSpecialCharacters
- GET/api/Metrics/ImportMethodsResults
- GET/api/Metrics/ImportHistoryCommits
- GET/api/Metrics/ImportTagCommits
- GET/api/Metrics/ImportRemoteCommits

Conforme a, la implementación del cumplimiento de validar la formula sobre el objetivo general de esta investigación se añade una vista para las métricas más utilizada de acuerdo al análisis de componente principal donde se aplica dicha formula en cuanto a la normalización de cada una y posteriormente calculo en este caso la figura 10 nos muestra la integración que se efectuó también bajo el tipo de recurso GET.

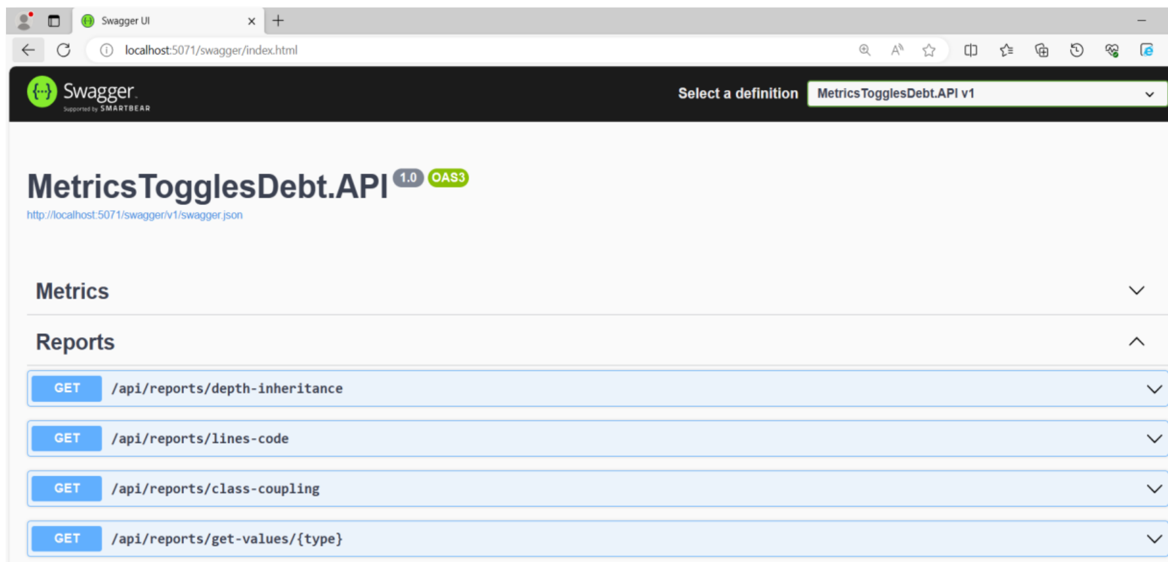


Figura 10 Vista de ReportsController.cs

- GET/api/reports/depth-inheritance
- GET/api/reports/lines-code
- GET/api/reports/class-coupling
- GET/api/reports/get-values/{type}

Es preciso decir, que en este ajuste se realiza todo el proceso de implementación de la métrica propuesta la cual se podrá detallar en el proyecto open source de GitHub "https://github.com/DidierDiaz/Metrics-Toggles-Debt" donde se visualizará el código fuente. En otras palabras, este desarrollo se encuentra enfocado en el consumo de una API desarrollada en Backend .NET Core 6 bajo el lenguaje de programación C# y en el IDE Microsoft Visual Studio 2022, la cual funciona de la siguiente manera:

Proyecto - Metrics-Toggles-Debt - Contiene 3 capas.

1. **Capa 1:** MetricsTogglesDebt.API
2. **Capa 2:** MetricsTogglesDebt.Data
3. **Capa 3:** MetricsTogglesDebt.Service

Frente a la capa 1: Se encuentra toda la estructura de consumo y configuración de la API en este caso los archivos tipo JSON que son los que contiene la información extraída del repositorio seleccionado (Python client library for the Square API) como se ha dicho esta librería sirve para integrar los pagos de Square en su aplicación.

Simultáneamente, en la figura 11 capa 1(MetricsTogglesDebt.API) se puede corroborar dicha estructura en cuanto al controlador al consumo de los archivos JSON, las clases sobre cada método y atributo declarado de acuerdo a la métrica.

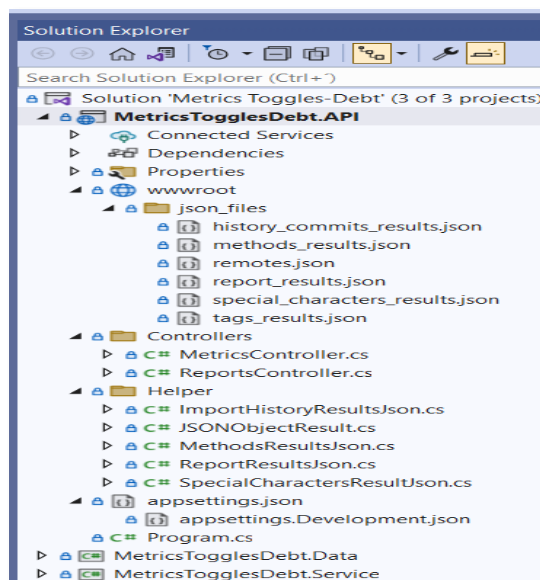


Figura 11 Capa MetricsTogglesDebt.API

Frente a la capa 2: En este apartado de capa podemos encontrar las entidades declarada sobre cada una de las clases según la métrica tal como muestra en la siguiente figura 12 capa MetricsTogglesDebt.Data.

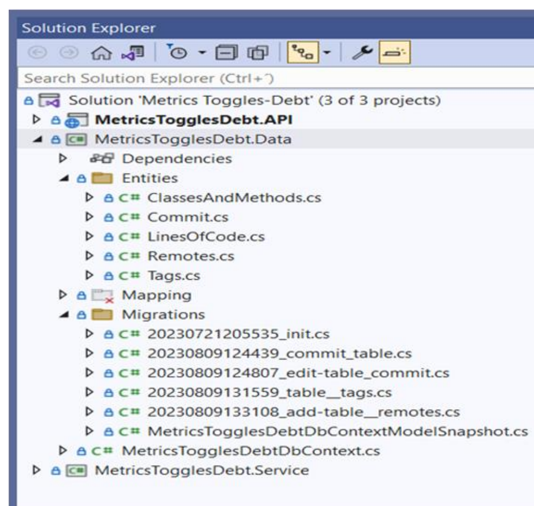


Figura 12 Capa MetricsTogglesDebt.Data

Frente a la capa 3: Aquí podemos detallar la lógica de interfaces e implementación de las métricas más usadas y la métrica propuesta de nuestra investigación dando A si cumplimiento del objetivo general, específicos en cuanto a proponer, validar la métrica aplicada al repositorio Square como caso de estudio es preciso decir que la figura 13 se visualiza dicha información.

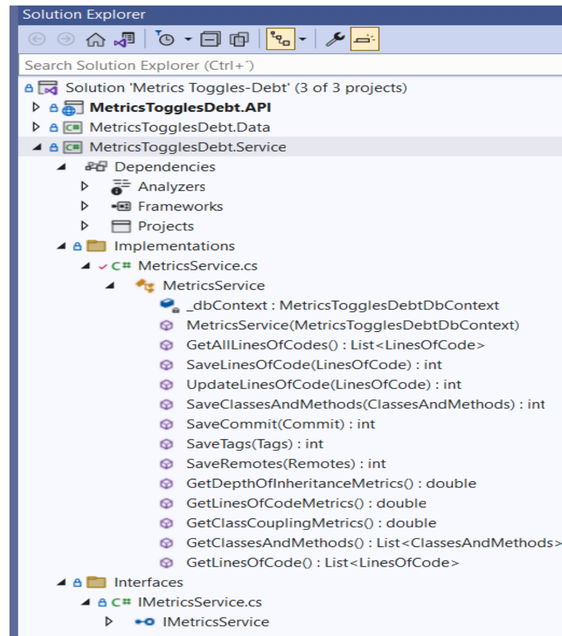


Figura 13 Capa MetricsTogglesDebt.Service.

5.6 Esquema a nivel de Frontend sobre la métrica aplicada.

Por otra parte, el desarrollo a nivel Frontend donde se encuentra implementado toda la lógica de consumo de la API en cuanto a la visualización de pagina sobre el cálculo de las métricas más usadas de la literatura según el criterio de selección y al análisis de componente de principal de dicha métrica propuesta. En relación con, dicho desarrollo el cual contiene la vista, formula y cálculo de nuestra métrica esto se efectuó bajo el marco de aplicaciones para web Angular v17 conjuntamente con el IDE o editor Visual Studio Code. En este apartado, report.component.ts se puede encontrar el consumo de la API en Backend de los reportes de las métricas más usada en cuanto al resultado de normalización efectuado tal como podemos ver en el siguiente código fuente implementado.

```
import { CommonModule } from '@angular/common';
import { HttpClient, HttpClientModule } from '@angular/common/http';
import { Component } from '@angular/core';
import { MatDialog, MatDialogModule } from '@angular/material/dialog';
import { DialogListRowsComponent } from '../dialogs/dialog-list-rows/dialog-list-rows.component';

@Component({
  selector: 'app-report',
  standalone: true,
  imports: [HttpClientModule, CommonModule, MatDialogModule],
  templateUrl: './report.component.html',
  styleUrl: './report.component.css'
```

```
})
export class ReportComponent {
  public report = {
    inheritance: 0,
    linesOfCode: 0,
    classCoupling: 0
  };
  constructor(private httpClient: HttpClient, private dialog: MatDialog) {
    this.getDepthOfInheritance();
    this.getLinesOfCode();
    this.getClassCoupling();
  }

  getDepthOfInheritance(): void {
    this.httpClient.get('http://localhost:5071/api/reports/depth-inheritance').subscribe((value: any) => {
      this.report.inheritance = value;
    });
  }

  getLinesOfCode(): void {
    this.httpClient.get('http://localhost:5071/api/reports/lines-code').subscribe((value: any) => {
      this.report.linesOfCode = value;
    });
  }
  getClassCoupling(): void {
    this.httpClient.get('http://localhost:5071/api/reports/class-coupling').subscribe((value: any) => {
      this.report.classCoupling = value;
    });
  }
```

Simultáneamente, en la vista report.component.html podemos detallar la formula, el reporte de normalización de las métricas vs el cálculo total de nuestra métrica.

```
<div class="container-fluid mt-2">
    <div class="row">
        <div class="col-12 col-sm-12 col-md-4">
            <div class="card border-top border-success shadow-sm cursor-click" (click)="listValues('inheritance')">
                <div class="card-body">
                    <h4 class="text-success">Depth of Inheritance</h4>
                    <h2>{{report.inheritance}}</h2>
                </div>
            </div>
        </div>
        <div class="col-12 col-sm-12 col-md-4">
```

```
        <div class="card border-top border-warning shadow-sm cursor-click" (click)="listValues('linesOfCode')">
          <div class="card-body">
            <h4 class="text-warning">Lines of Source Code</h4>
            <h2>{{report.linesOfCode}}</h2>
          </div>
        </div>
      </div>
      <div class="col-12 col-sm-12 col-md-4">
        <div class="card border-top border-info shadow-sm cursor-click" (click)="listValues('classCoupling')">
          <div class="card-body">
            <h4 class="text-info">Class Coupling</h4>
            <h2>{{report.classCoupling}}</h2>
          </div>
        </div>
      </div>
    </div>
    <div class="row mt-3">
      <div class="col-12 col-sm-12 col-md-4" *ngIf="report.inheritance && report.classCoupling &&
report.linesOfCode">
        <div class="card border-top border-primary shadow-sm">
          <div class="card-body">
            <h4 class="text-primary">Métrica Didier</h4>
            <h2>{{(0.29 * report.inheritance + 0.24 * report.classCoupling + 0.48 * report.linesOfCode) | number: '1.3-
3'}}</h2>
          </div>
        </div>
      </div>
    </div>
  </div>
```

Resumiendo, en dialog-list-rows.component.html podemos detallar el código fuente donde se realiza la implementación de visualización de los datos de las métricas más usadas en la literatura.

```
<h1 mat-dialog-title>
  Lista de Datos
</h1>
<div mat-dialog-content>
  <table class="table table-hover table-striped" *ngIf="data.type == 'inheritance' || data.type == 'classCoupling'">
    <thead>
      <tr>
        <th>Id</th>
        <th>Method</th>
        <th>Path</th>
      </tr>
```

```
      </thead>
      <tbody>
        <tr *ngFor="let value of values">
          <td>
            <span class="text-muted">{{value.id}}</span>
          </td>
          <td>{{value.method}}</td>
          <td>{{value.path}}</td>
        </tr>
      </tbody>
    </table>
    <table class="table table-hover table-striped" *ngIf="data.type == 'linesOfCode'">
      <thead>
        <tr>
          <th>Id</th>
          <th>Language</th>
          <th>Lines</th>
          <th>File</th>
          <th>Comments</th>
        </tr>
      </thead>
      <tbody>
        <tr *ngFor="let value of values">
          <td>
            <span class="text-muted">{{value.id}}</span>
          </td>
          <td>{{value.language}}</td>
          <td>{{value.lines}}</td>
          <td>{{value.file}}</td>
          <td>{{value.comments}}</td>
        </tr>
      </tbody>
    </table>
  </div>
  <div mat-dialog-actions>
    <button mat-button mat-dialog-close>Cerrar</button>
  </div>
```

Como resultado, del desarrollo sobre el consumo de la API a nivel del Backend & Frontend de nuestra métrica propuesta basada de las métricas más usadas en la literatura a continuación se muestra la parte final del resultado esperando como cumplimiento del objetivo general y específicos de esta investigación, en este caso podemos detallar en la figura 14 la parte grafica de dicho desarrollo a nivel del Frontend.

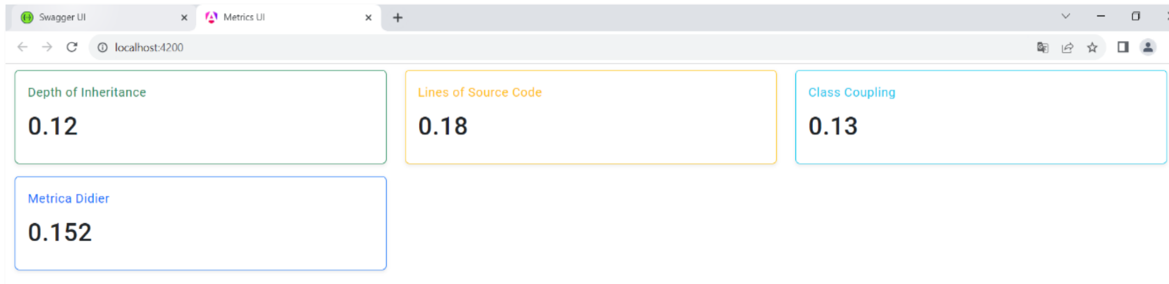


Figura 14 Frontend de cálculo sobre métrica implementada.

En la figura 15 se puede observar la lista de datos sobre dos de la métrica más usada en este caso Depth of Inheritance - Class Coupling la cual contiene los siguientes campos (Id- Method- Path). que a su vez fueron extraído del repositorio square.

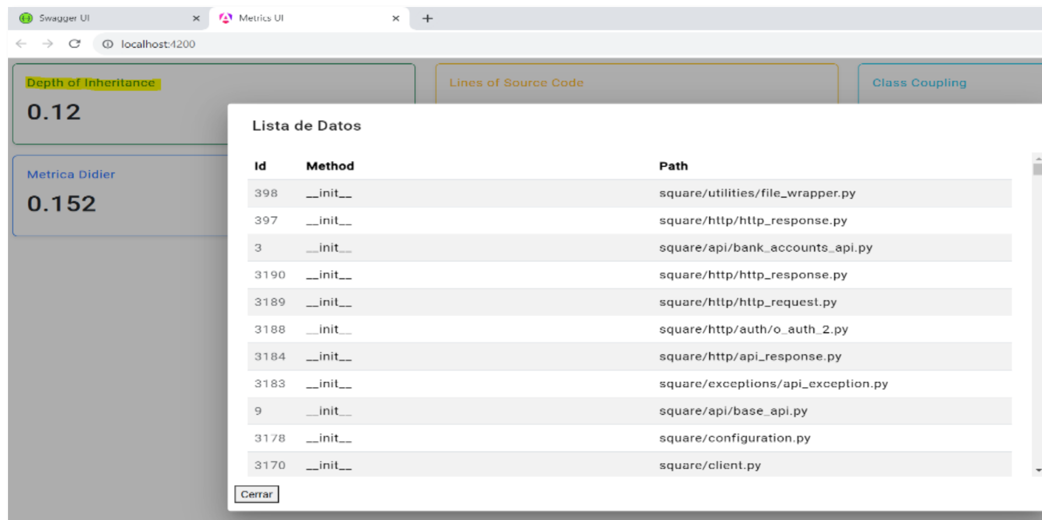


Figura 15 Frontend lista de datos métrica Depth of Inheritance - Class Coupling.

Por consiguiente en la figura 16 se muestra la lista de datos referente a la métrica mas usada Lines of Source code ya que esta fue las que nos arrojo un mayor nivel de efectividad en cuanto a las demas métrica por lo cual en este caso podemos observar los siguiente campos (Id- Language- Lines – File – Comments) que en marca esta métrica que a su vez fueron extraido del repositorio square.

Depth of Inheritance 0.12

Lines of Source Code

Class Coupling

Metrica Didier 0.152

Lista de Datos

Id	Language	Lines	File	Comments
11753	Ini	6	/square-python-sdk/tox.ini	0
8228	Ini	6	/square-python-sdk/tox.ini	0
3528	Ini	6	/square-python-sdk/tox.ini	0
3	Ini	6	/square-python-sdk/tox.ini	0
1178	Ini	6	/square-python-sdk/tox.ini	0
9403	Ini	6	/square-python-sdk/tox.ini	0
10578	Ini	6	/square-python-sdk/tox.ini	0
12928	Ini	6	/square-python-sdk/tox.ini	0
5878	Ini	6	/square-python-sdk/tox.ini	0
7053	Ini	6	/square-python-sdk/tox.ini	0
2353	Ini	6	/square-python-sdk/tox.ini	0

Cerrar

Figura 16 Frontend lista de datos métrica Lines of Source code.

Finalmente, se anexa el código fuente alojado en la plataforma de código abierto sobre la implementación de la métrica propuesta dando A si cumplimiento a los objetivos de esta tesis de investigación. En síntesis, este estudio nos generó una transferencia de conocimiento frente a diferentes temas como lo son: Deuda técnica, métricas de software, análisis de componente principal, aprendizaje en el lenguaje de programación Python también en el IDE Microsoft visual Studio 2022 hacia el consumo de API. Además, en el lenguaje de FrontEnd Angular por su versatilidad al momento de poder integrar, consumir nuestra API desarrollada. En efecto, el manejo de PostgreSQL como motor de base datos brindándonos una relación dinámica en el análisis y extracción de la información del repositorio “Square” seleccionado etc.

5.7 ¿Por qué la métrica propuesta es una buena métrica?

En síntesis, las grandes organizaciones cada día se esfuerzan en desarrollar productos de software con alto estándares de calidad haciendo posible la mitigación de la deuda tecnica ya que se debe controlar y medirse de una manera muy satisfactoria durante las etapas del desarrollo de software, esto con el objetivo de que no se aumente generando A si sobre costos en el mantenimiento de dicho producto. Es por esto, que la métrica propuesta basada en las más usadas permite tener un análisis de literatura demasiado amplio sobre unas métricas especifica que al juntarla, unirla nos daría una serie de datos medible en la incorporación e implementación en cualquier producto de software independientemente del lenguaje de programación, plataforma tecnológica. Por tanto, medir y mejorar las métricas de deuda técnica es una de las primeras y más importantes formas de abordarla, eliminando los problemas de interrupciones en las operaciones. Esta métrica que se implemento puede ser una gran alternativa de solución a esos productos de desarrollo de software para las organizaciones.

5.8 ¿Qué se lograría en un proyecto con la implementación de esta métrica?

Llegados a este punto, la deuda técnica en su literatura describe el diseño o construcción a corto plazo que crea un contexto técnico que puede hacer que los cambios futuros sean más costosos o imposibles. Igualmente, es importante resaltar que la aplicación de la deuda técnica en las organizaciones genera un impacto significativo que no se puede ocultar. En consecuencia, la implementación de las métricas más usadas en las literaturas en esta tesis como los son Mantenibilidad, Complejidad ciclomática, Profundidad de la herencia, Acoplamiento de clases, Líneas de código fuente, Líneas de código ejecutable para medir la deuda técnica nos da la garantía a la hora de proponer, identificar, validar nuestra métrica propuesta en cualquier proyecto de repositorio open source independientemente del lenguaje de programación y el sector sobre la empresa interesada. Ya que lograría un impacto significado en primero identificar dichas métricas descrita anteriormente, después poderla medir de acuerdo al código fuente de la aplicación de software. Es decir, le brindaría al equipo de desarrollo una visión amplia del código sobre el programa que se está desarrollando y edificando A si los posibles riesgos, amenazas, sobre costo del producto mediante un seguimiento continuo.

5.9 ¿En qué etapas del desarrollo del proyecto se debe implementar?

En concreto, es importante analizar la necesidad del proyecto de desarrollo de software a la hora de medir la deuda técnica mediante métricas en una organización ya que es una tarea muy difícil para cualquier equipo de desarrollo. Es por esto que, se debe tener presente una visión del mercado, sus atributos internos & externo a la hora de implementar esta métrica teniendo muy presente todos los criterios definidos durante el proceso de identificar las métricas más usadas y validándola al momento de aplicar en cualquier repositorio open source. Por consiguiente, se hace necesario iniciar con la etapa de recopilación de los datos, a continuación, analizar los resultados, después, abordar una hoja de ruta con la finalidad de corregir las acciones correctivas. Luego, el desarrollo mediante un caso de estudio para A si, obtener y analizar esos resultados que permita mejorar su aplicación de una manera interactiva, medible.

5.10 ¿Qué condiciones debe cumplir un proyecto para que se pueda aplicar la métrica?

Resumiendo, se debe tener muy presente los criterios de selección sobre las métricas más usadas de literatura ya que le permitiría a cualquier empresa, organización un contexto amplio de dichas métricas de deuda tecnica en la parte de medición, métodos, generación de código fuente. Debido a que el equipo de desarrollo de software pueda analizar, recopilar datos del proyecto donde sean medible mediante la complejidad y capacidad del mantenimiento de su código administrado teniendo siempre presente implementar las mejores prácticas. Finalmente, su aplicación se adapta a cualquier lenguaje de programación y tecnología o proyecto open source de la industria.

6. Conclusiones y recomendaciones

6.1 Conclusiones

Para concluir, se da cumplimiento al objetivo general y específicos donde proponemos una métrica de deuda técnica basada en el análisis de las métricas más usadas en la literatura, aplicada al repositorio Square como caso de estudio.

En primer lugar, se realiza un estado de arte sobre los diferentes estudios de literatura en cuanto a la medición deuda técnica en las métricas existente. En segunda instancia, se analiza las diferentes métricas más usadas (Depth of Inheritance, Class Coupling, Lines of Source code) de la literatura donde se logró comprobar por medio de estadística en cuanto al análisis de componente principal frente a los datos extraído del repositorio square, su nivel importancia y efectividad en la medición, cálculo de la deuda técnica. Luego, se realizó la formulación de nuestra métrica propuesta en cuanto a los resultados generando mediante el ACP dándonos soporte en la integración de las tres métricas más usadas, A si pudiendo aplicar la normalización de cada métrica posterior, la normalización de los pesos referente a la literatura sobre las métricas y por último aplicar el cálculo correspondiente obteniendo esta fórmula

"**Métrica** = Peso 1*Normalización Depth of Inheritance + Peso 2* Normalización Class Coupling + Peso 3* Normalización Lines of Source code".

Por otra parte, medir la deuda técnica se centra estrictamente en la definición de una medida principal de una Deuda Técnica que puede calcularse a partir de las Características de Calidad. A su vez, el no uso pertinente de cálculo sobre la medición puede llegar a tener una deuda técnica elevada. Con respecto a, esto se hace necesario poder profundizar más en investigaciones sobre poder medir métrica bajo la metáfora de la deuda técnica ya que ayudaría a las organizaciones, equipos de desarrollo tener éxito en el mantenimiento de su producto. Al mismo tiempo, unos de los principales objetivos para medir la deuda técnica es poder cuantificar mediante métricas de código que permitan el rastreo simultaneo, además la utilización de herramientas que ayudan a comprender mejor los puntos fuertes del equipo de trabajo y críticos sobre el proyecto de desarrollo.

Por consiguiente, el proponer nuestra métrica en base a las métricas más usadas nos permite generar una base de conocimiento frente al tema deuda técnica hacia futuras investigaciones de literatura ya que su implementación estuvo en marca en tecnología de desarrollo muy existente en la industria de software. En suma, que la mayoría de los proyectos de desarrollo no fracase por no medir adecuadamente la deuda técnica mediante métricas e idónea.

6.2 Recomendaciones

Podemos condensar lo dicho hasta aquí, que se hace necesario poder profundizar más en el tema de nuevas métricas para medición de deuda técnica ya que permitiría a las organizaciones y equipo de desarrollo poder mitigar cualquier sobre costo a la hora de medir, calcular dicha deuda técnica.

Sin embargo, esta tesis de profundización permite abarca más hacia las métricas de literatura más usada por lo que la hace vigente para una posible mejora como trabajo futuro.

Habría que decir también, realizar una profundización en cuanto al tema de análisis de datos ya que existen otro tipo de análisis y extracción de información que ayuda a tener una mayor visibilidad a la hora de tomar decisiones.

Al mismo tiempo, se recomienda utilizar otros tipos de lenguajes de programación en la actualidad tanto a nivel Backend y Frontend generando A si una mantenibilidad con cualquier tipo de lenguaje y motor de base datos.

Finalmente, se resalta la facilidad que genera el tipo de plataforma GitHub la cual es de código abierto ya que nos permite consumir y extraer información para cualquier tipo de repositorio independientemente de su lenguaje de programación, es por esto que se recomienda explorar más y utilizar esta plataforma hacia futuro.

Bibliografía

Abilio, R., Vale, G., Figueiredo, E., & Costa, H. (16 de Mayo de 2016). Metrics for Feature-Oriented Programming. *IEEE*, 1-7. doi:10.1109/WETSoM.2016.014

Al Mamun, M., Martini, A., Staron, M., & Berger, C. (7 de Noviembre de 2019). Evolution of technical debt: An exploratory study. *Digitala Vetenskapliga Arkivet, 2476*, 1-16. Recuperado el 24 de Enero de 2024, de https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1368627&dswid=-9383

Alali, A., Kagdi, H., & Maletic, J. I. (2 de Julio de 2008). What's a Typical Commit? A Characterization of Open Source Software Repositories. *IEEE*, 1-10. doi:10.1109/ICPC.2008.24

Alfayez, R., Alwehaibi, W., Winn, R., Venson, E., & Boehm, B. W. (28 de Junio de 2020). A systematic literature review of technical debt prioritization. *ACM Digital Library*, 1-42. doi:10.1145/3387906.3388630

Amanatidis, T., Mittas, N., Moschou, A., Chatzigeorgiou, A., & Ampatzoglou , A. (26 de Agosto de 2020). Evaluating the agreement among technical debt measurement tools: building an empirical benchmark of technical debt liabilities. *Springer Nature, 25*, 4161-4204. doi:https://doi.org/10.1007/s10664-020-09869-w

Angular. (12 de Diciembre de 2023). *Angular*. Recuperado el 16 de Enero de 2024, de https://angular.io/docs

Ardimento, P., Aversano, L., Bernardi, M., Cimitile, M., & Iammarino, M. (Diciembre de 2022). Using deep temporal convolutional networks to just-in-time forecast technical debt principal. *Science direct, 194*. doi:https://www.sciencedirect.com/science/article/pii/S0164121222001649

Arvanitou, E. M., Ampatzoglou, A., Bibi, S., Chatzigeorgiou, A., & Stamelos, I. G. (2019). Monitoring Technical Debt in an Industrial Setting. *ACM Digital Library*, 123–132. doi:10.1145/3319008.3319019

Atlassian. (22 de Mayo de 2023). *Atlassian.* Recuperado el 22 de Mayo de 2023, de https://www.atlassian.com/es/git/tutorials/making-a-pull-request#:~:text=Las%20pull%20requests%20son%20una,integrarlos%20en%20el%20proyecto%20oficial.

Avgeriou, P., Taibi, D., Ampatzoglou, A., Fontana, F., & Besker, T. (18 de Septiembre de 2020). An Overview and Comparison of Technical Debt Measurement Tools. *IEEE*, 1-19. doi:10.1109/MS.2020.3024958

Bedi, J., & Kaur , K. (24 de Junio de 2020). Understanding factors affecting technical debt. *Springer Nature, 14*, 1051–1060. doi:https://doi.org/10.1007/s41870-020-00487-9

Besker, T; Martini, A; Bosch, J. (20 de Junio de 2020). Carrot and stick approaches when managing technical debt. *ACM Digital Library*, 21-30. doi:https://doi.org/10.1145/3387906.3388619

Caglayan, B., Bener, A., & Koch, S. (18 de Mayo de 2009). Merits of using repository metrics in defect prediction for open source projects. *IEEE*, 1-6. doi:10.1109/FLOSS.2009.5071357

Caldeira, J., Cardoso, J., & Abreu, B. F. (30 de Octubre de 2020). Unveiling process insights from refactoring practices. *ScienceDirect*. doi:https://doi.org/10.1016/j.csi.2021.103587

Capilla, R., Mikkonen, T., Carrillo, C., Fontana, F. A., Pigazzini, I., & Lenarduzzi, V. (25 de Junio de 2021). Impact of Opportunistic Reuse Practices to Technical Debt. *IEEE*, 1-10. doi:10.1109/TechDebt52882.2021.00011

Chidamber, S. R., Darcy, D. P., & Kemerer, C. F. (Agosto de 1998). Managerial use of metrics for object-oriented software: an exploratory analysis. *IEEE*, 1-12. doi:10.1109/32.707698

Cholda, P., & Stochel, M. (16 de Octubre de 2020). Continuous Debt Valuation Approach (CoDVA) for Technical Debt Prioritization. *IEEE*, 1-5. doi:10.1109/SEAA51224.2020.00066

Churcher, N. I., Shepperd, M. J., Chidamber, S., & Kemerer, C. F. (Marzo de 1998). Comments on "A metrics suite for object oriented design. *IEEE*, 263 - 265. doi:10.1109/32.372153

de Almeida, R. R., Ribeiro, R. N., Treude, C., & Kulesza, U. (25 de Junio de 2021). Business-Driven Technical Debt Prioritization: An Industrial Case Study. *IEEE*, 1-10. doi:10.1109/TechDebt52882.2021.00017

de Toledo , S. S., Martini, A., & Sjøberg, D. I. (9 de Abril de 2021). Identifying architectural technical debt, principal, and interest in microservices: A multiple-case study. *Elsevier, 177*, 1-21. doi:https://doi.org/10.1016/j.jss.2021.110968

Diamantopoulos, T., Papamichail, M., Karanikiotis, T., Chatzidimitriou, K. C., & Symeonidis, A. L. (Junio de 2020). Employing Contribution and Quality Metrics for Quantifying the Software Development Process. *ACM Digital Library*, 558-562. doi:https://doi.org/10.1145/3379597.3387490

Digkas, G., Chatzigeorgiou, A. N., Ampatzoglou, A., & Avgeriou, P. C. (20 de Octubre de 2021). Can Clean New Code reduce Technical Debt Density. *IEEE*, 1-18. doi:10.1109/TSE.2020.3032557

DiStefano, C., Zhu, M., & Mîndrilã, D. (Noviembre de 2019). Understanding and Using Factor Scores: Considerations for the Applied Researcher. *Scholarworks, 14*(20), 1-12. doi:https://doi.org/10.7275/da8t-4g52

Efimova, P. (26 de Enero de 2021). *Stepsize Ltd.* Recuperado el 18 de Octubre de 2021, de https://www.stepsize.com/blog/tools-to-track-and-manage-technical-debt

Ernst, N., Kazman, R., & Delange, J. (2021). *Technical Debt in Practice: How to Find It and Fix It.* doi:https://doi.org/10.7551/mitpress/12440.001.0001

Feitelson, D. G., Frachtenberg, E., & Beck, K. L. (4 de Febrero de 2013). Development and Deployment at Facebook. *IEEE*, 1-10. doi:10.1109/MIC.2013.25

Feitosa, D., Ampatzoglou, A., Gkortzis, A., & Bibi, S. (23 de Mayo de 2020). CODE reuse in practice: Benefiting or harming technical debt. *Elsevier , 167*, 1-12. doi:https://doi.org/10.1016/j.jss.2020.110618

Freire, S., Rios, N., Pérez, B., & Mendonça, M. (21 de Mayo de 2021). How do Technical Debt Payment Practices Relate to the Effects of the Presence of Debt Items in Software Projects? *IEEE*, 1-5. doi:10.1109/SANER50967.2021.00074

Greenacre, M., Groenen, P., & Hastie, T. (22 de Diciembre de 2022). Principal component analysis. *Springer Nature Limited*. doi:https://doi.org/10.1038/s43586-022-00184-w

Hamilton, T. (6 de Mayo de 2023). *Guru99.* Recuperado el 20 de Mayo de 2023, de https://www.guru99.com/unit-testing-guide.html

Hick, H., & Denkmayr, K. (2004). The Load Matrix — a method for optimising powertrain durability and reliability test programmes. *Springer Nature*. Recuperado el 6 de Noviembre de 2023, de https://link.springer.com/chapter/10.1007/978-0-85729-410-4_262

Hodgson, P. (9 de Octubre de 2017). *Martin Fowler*. Recuperado el 12 de Agosto de 2021, de https://martinfowler.com/articles/feature-toggles.html

Holmegaard, E. (20 de Abril de 2023). *Medium.* Recuperado el 25 de Enero de 2024, de https://medium.com/@emilholmegaard/managing-technical-debt-31b52e83b510

Hoyos, J. D. (2021). *Repositorio Institucional Biblioteca Digital.* Recuperado el 19 de Octubre de 2021, de https://repositorio.unal.edu.co/bitstream/handle/unal/80530/71360425.2021.pdf?sequence=4&isAllowed=y

IBM. (22 de Marzo de 2021). *IBM.* Recuperado el 13 de Noviembre de 2023, de https://www.ibm.com/docs/en/spss-statistics/25.0.0?topic=analysis-factor-scores

IBM. (3 de Marzo de 2023). Recuperado el 6 de Noviembre de 2023, de https://www.ibm.com/docs/es/spss-statistics/29.0.0?topic=detection-kmo-bartletts-test

Jenik, F. (2 de Febrero de 2021). *Sudolabs.* Recuperado el 22 de Mayo de 2023, de https://sudolabs.com/blog/automated-release-process-for-lerna-monorepo

Kherif, F., & Latypova, A. (2020). *Machine Learning Methods and Applications to Brain Disorders.* (S. V. Andrea Mechelli, Ed.) Copyright © 2020 Elsevier Inc. All rights reserved. doi:https://doi.org/10.1016/B978-0-12-815739-8.00012-2

Lenarduzzi, V., Besker, T., Taibi, D., Martini, A., & Fontana, F. A. (30 de Junio de 2020). *arXiv*, 1-42. Recuperado el 27 de Agosto de 2021, de https://arxiv.org/abs/1904.12538

Lenarduzzi, V., Saarimäki, N., & Taibi, D. (18 de Septiembre de 2019). The Technical Debt Dataset. *ACM Digital Library*, 1-10. doi:https://doi.org/10.1145/3345629.3345630

Lenarduzzi, V., T, D., Besker , T., Martini, A., & Arcelli, F. F. (14 de Octubre de 2020). A systematic literature review on Technical Debt prioritization: Strategies, processes, factors, and tools. *sciencedirect*, 1-16. doi:https://doi.org/10.1016/j.jss.2020.110827

León-Sigg, M., Vázquez-Reyes, S., & Rodríguez-Ávila. (31 de Diciembre de 2020). Towards the Use of a Framework to Make Technical Debt Visible. *IEEE*, 1-7. doi:10.1109/CONISOFT50191.2020.00022

Li , X., Moreschini, S., Zhang, Z., & Taibi, D. (Junio de 2022). Exploring factors and metrics to select open source software components for integration: An empirical study. *Sciencedirect, 188*, 1-19. doi:https://doi.org/10.1016/j.jss.2022.111255

Liebig, J., Apel, S., & Lengauer, C. (Mayo de 2010). An Analysis of the Variability in Forty Preprocessor-Based Software Product Lines. *ResearchGate*. doi:10.1145/1806799.1806819

Lumivero. (Octubre de 2023). *Xlstat*. Recuperado el 14 de Octubre de 2023, de https://www.xlstat.com/es/soluciones/funciones/analisis-de-componentes-principales-acp

Mahdavi-Hezaveh, R., Ajmeri, N., & Williams, L. (10 de Febrero de 2022). Feature toggles as code: Heuristics and metrics for structuring feature toggles. *Science direct, 145*, 1-14. doi:https://doi.org/10.1016/j.infsof.2021.106813

Malakuti, S., & Heuschkel, J. (25 de Junio de 2021). The Need for Holistic Technical Debt Management across the Value Stream: Lessons Learnt and Open Challenges. *IEEE*, 1-5. doi:10.1109/TechDebt52882.2021.00021

Mangale, S. (28 de Agosto de 2020). *Medium.* Recuperado el 6 de Noviembre de 2023, de https://sanchitamangale12.medium.com/scree-plot-733ed72c8608

Martin Fowler. (21 de Mayo de 2019). Recuperado el 25 de Enero de 2024, de https://martinfowler.com/bliki/TechnicalDebt.html

Maven Solutions. (10 de Enero de 2024). Tips and Tricks for Measuring and Improving Technical Debt Metrics. Recuperado el 23 de Enero de 2024, de https://www.mavensolutions.tech/blog/metrics-for-technical-debt/

McCabe, T. J. (1976). A Complexity Measure. *IEEE*. doi:10.1109/TSE.1976.233837

Meinicke, J., Hoyos, J., Vasilescu, B., & Kästner, C. (2020). Capture the Feature Flag: Detecting Feature Flags in Open-Source. *ACM Digital Library*, 169–173. doi:https://doi.org/10.1145/3379597.3387463

Microsoft. (1 de Marzo de 2023). Recuperado el 5 de Mayo de 2023, de https://learn.microsoft.com/en-us/dotnet/api/system.obsoleteattribute?view=net-7.0

Microsoft. (29 de Noviembre de 2023). *Microsoft.* Recuperado el 15 de Enero de 2024, de https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values?view=vs-2022

Microsoft. (10 de Enero de 2024). *Microsoft*. Recuperado el 16 de Enero de 2024, de https://visualstudio.microsoft.com/es/

Mikhajlova, A., & Sekerinski, E. (Septiembre de 1997). Class Refinement and Interface Refinement in Object-Oriented Programs. *ResearchGate*, 1-21. Recuperado el 22 de Mayo de 2023, de https://www.researchgate.net/publication/303992360_Class_Refinement_and_Interface_Refinement_in_Object-Oriented_Programs

Mohan, B., & Kumar, P. (20 de Enero de 2013). An Overview of Various Object Oriented Metrics. *ResearchGate, II*, 1-11. Recuperado el 14 de Febrero de 2022, de https://www.researchgate.net/publication/236616349_An_Overview_of_Various_Object_Oriented_Metrics

Molnar, A. J., & Motogna, S. (20 de Octubre de 2020). Long-Term Evaluation of Technical Debt in Open-Source Software. *ACM Digital Library*, 1-10. doi:10.1145/3382494.3410673

OMG. (1 de Septiembre de 2018). *Object Management Group.* Recuperado el 15 de Febrero de 2023, de https://www.omg.org/spec/ATDM/?__hstc=64371056.313d6ed65e49ea25f2bfedd7ce2e4cea.1698498226843.1698498226843.1698498226843.1&__hssc=64371056.2.1698498226843&__hsfp=1453860330

Pathak, A. (11 de Abril de 2022). *Kinsta Inc.* Recuperado el 02 de Marzo de 2023, de https://kinsta.com/es/blog/herramientas-devops/

Pavel, K. (24 de Marzo de 2023). *Modlogix.* Recuperado el 22 de Mayo de 2023, de https://modlogix.com/blog/3-main-threats-resulting-from-outdated-technology/

Perera, J., Tempero, E. D., Tu, Y., & Blincoe , K. (Junio de 2023). Understanding the relationship between Technical Debt, New Code Cost and Rework Cost in Open-Source Software Projects: An Empirical Study. *ACM Digital Library*, 247–252. doi:https://doi.org/10.1145/3593434.3593490

Pérez, B., Castellanos, C., Correal, D., Rios, N., Freire, S., & Spínola, R. (22 de Julio de 2021). Technical debt payment and prevention through the lenses of software architects. *Elsevier, 140*. doi:https://doi.org/10.1016/j.infsof.2021.106692

Pfeiffer, R. H., & Lungu, M. (27 de Febrero de 2022). Technical Debt and Maintainability: How do tools measure it? *arXiv*, 1-22. Recuperado el 24 de Enero de 2024, de https://arxiv.org/abs/2202.13464

PostgreSQL. (9 de Noviembre de 2023). *PostgreSQL*. Recuperado el 16 de Enero de 2024, de https://www.postgresql.org/

Quora. (18 de Agosto de 2018). *Quora.* Recuperado el 22 de Mayo de 2023, de https://www.quora.com/What-are-the-steps-one-should-take-to-understand-a-software-system-with-no-documentation

Ramirez, J., Tuovinen, A. P., & Mikkonen, T. (25 de Junio de 2021). Experiences on Managing Technical Debt with Code Smells and AntiPatterns. *IEEE*, 1-29. doi:10.1109/TechDebt52882.2021.00013

Rosser, L. A., & Norton, J. H. (7 de Junio de 2021). A Systems Perspective on Technical Debt. *IEEE*, 1-10. doi:10.1109/AERO50100.2021.9438359

Rouse, M. (27 de Diciembre de 2016). *Techopedia.* Recuperado el 20 de Mayo de 2023, de https://www.techopedia.com/definition/8073/lines-of-code-loc#:~:text=27%20December%2C%202016-,What%20Does%20Lines%20Of%20Code%20Mean%3F,used%20to%20write%20a%20program.

Samarthyam, G., Suryanarayana, G., & Sharma, T. (25 de Septiembre de 2015). *InfoQ.com.* Recuperado el 25 de Enero de 2024, de https://www.infoq.com/articles/pragmatic-technical-debt/

Schermann, G., Cito, J., & Leitner, P. (22 de Marzo de 2016). An empirical study on principles and practices of continuous delivery and deployment. *PeerJ*, 1-13. doi:10.7287/peerj.preprints.1889v1

Soliman, M., & Avgeriou, p. (14 de Junio de 2021). Architectural design decisions that incur technical debt — An industrial case study. *Elsevier, 139*, 1-17. doi:https://doi.org/10.1016/j.infsof.2021.106669

Spertus, E. (23 de Diciembre de 2021). *stackoverflow.* Recuperado el 20 de Mayo de 2023, de https://stackoverflow.blog/2021/12/23/best-practices-for-writing-code-comments/

Stochel, M G; Chołda, P; Wawrowski, M R. (16 de Octubre de 2020). On Coherence in Technical Debt Research : Awareness of the Risks Stemming from the Metaphorical Origin and Relevant Remediation Strategies. *IEEE*, 1-9. doi:10.1109/SEAA51224.2020.00067

Subramanyam, R., & Krishnan, M. S. (23 de Abril de 2003). Empirical analysis of CK metrics for object-oriented design complexity: implications for software defects. *IEEE*, 297 - 310. doi:10.1109/TSE.2003.1191795

Taibi, D., & Lenarduzzi, V. (2021). A systematic literature review on Technical Debt prioritization: Strategies, processes, factors, and tools. *Science Direct, 171*, 1-16. doi:https://doi.org/10.1016/j.jss.2020.110827

Tan, J., Feitosa, D., & Avgeriou, P. (Julio de 2023). The lifecycle of Technical Debt that manifests in both source code and issue trackers. *Science direct, 159*, 1-14. doi:https://doi.org/10.1016/j.infsof.2023.107216

Tang, M. H., Kao, M. H., & Chen, M. H. (6 de Agosto de 2002). An empirical study on object-oriented metrics. *IEEE*. doi:10.1109/METRIC.1999.809745

Tsoukalas, D., Kehagias, D., Siavvas, M., & Chatzigeorgiou. (8 de Agosto de 2020). Technical debt forecasting: An empirical study on open-source repositories. *Elsevier*(170), 1-31. doi:https://doi.org/10.1016/j.jss.2020.110777

UCLA. (22 de Agosto de 2021). Recuperado el 6 de Noviembre de 2023, de https://stats.oarc.ucla.edu/spss/seminars/efa-spss/

Verma, D., & Kumar, S. (Junio de 2017). Prediction of Defect Density for Open Source Software using Repository Metrics. *ACM Digital Library*, 1-18. Recuperado el 2 de Marzo de 2022, de https://dl.acm.org/doi/abs/10.5555/3177580.3177587

Wheatley, M. (13 de Octubre de 2022). *DEV Community.* Recuperado el 02 de Marzo de 2023, de https://dev.to/maximwheatley/an-open-source-solution-to-dora-devops-metrics-72l

Wiese, M. (25 de Junio de 2021). Preventing Technical Debt by Technical Debt Aware Project Management. *IEEE*, 1-10. doi:10.1109/TechDebt52882.2021.00018

Wiese, M., Rachow, P., Riebisch, M., & Schwarze, J. (2022). Preventing technical debt with the TAP framework for Technical Debt Aware Management. *Science direct, 148*, 0950-5849. doi:https://doi.org/10.1016/j.infsof.2022.106926

Windev. (22 de Marzo de 2023). *Windev.* Recuperado el 20 de Mayo de 2023, de https://help.windev.com/en-US/?9000091&name=instant_spotting_modified_codecurrent_code