EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Tecnológico Nacional de México

Centro Nacional de Investigación y Desarrollo Tecnológico

Tesis de Maestría

Modelo de calidad para la medición de tolerancia a fallas en sistemas Big Data

presentada por

Ing. Nancy Victoria Guadarrama Alvarez

como requisito para la obtención del grado de

Maestra en Ciencias la Computación

Director de tesis

Dr. Juan Carlos Rojas Pérez

Cuernavaca, Morelos, México. Enero de 2023.

cenidet
Centro Nacional de Investigación y Desarrollo Tecnológico

Centro Nacional de Investigación y Desarrollo Tecnológico
Departamento de Ciencias Computacionales

Cuernavaca, Morelos 12/diciembre/2022

OFICIO No. DCC/000/2022
Asunto: Aceptación de documento de tesis
CENIDET-AC-004-M14-OFICIO

DR. CARLOS MANUEL ASTORGA ZARAGOZA
SUBDIRECTOR ACADÉMICO
PRESENTE

Por este conducto, los integrantes de Comité Tutorial de la C. NANCY VICTORIA GUADARRAMA ALVAREZ, con número de control M21CE014, de la Maestría en Ciencias de la Computación, le informamos que hemos revisado el trabajo de tesis de grado titulado **"MODELO DE CALIDAD PARA LA MEDICIÓN DE TOLERANCIA A FALLAS EN SISTEMAS BIG DATA"**

y hemos encontrado que se han atendido todas las observaciones que se le indicaron, por lo que hemos acordado aceptar el documento de tesis y le solicitamos la autorización de impresión definitiva.

EDUCACIÓN | TECNOLÓGICO NACIONAL DE MÉXICO
14 DIC 2022
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
SERVICIOS ESCOLARES
RECIBIDO
EBD

DR. JUAN CARLOS ROJAS PÉREZ
Director de tesis

DRA. OLIVIA GRACIELA FRAGOSO DÍAZ
Revisor 1

M.C. HUMBERTO HERNÁNDEZ GARCÍA
Revisor 2

SEP TecNM CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
RECIBIDO
14 DIC 2022
LMA
SUBDIRECCIÓN ACADÉMICA

Cuernavaca, Mor., 14/diciembre/2022
No. De Oficio: SAC/183/2022
Asunto: Autorización de impresión de tesis

NANCY VICTORIA GUADARRAMA ÁLVAREZ
CANDIDATA AL GRADO DE MAESTRA EN CIENCIAS DE LA COMPUTACIÓN
P R E S E N T E

Por este conducto, tengo el agrado de comunicarle que el Comité Tutorial asignado a su trabajo de tesis titulado **"Modelo de calidad para la medición de tolerancia a fallas en sistemas Big Data",** ha informado a esta Subdirección Académica, que están de acuerdo con el trabajo presentado. Por lo anterior, se le autoriza a que proceda con la impresión definitiva de su trabajo de tesis.

Esperando que el logro del mismo sea acorde con sus aspiraciones profesionales, reciba un cordial saludo.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
"Educación Tecnológica al Servicio de México"

EDUCACIÓN | TECNOLÓGICO NACIONAL DE MÉXICO
15 DIC 2022
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
SERVICIOS ESCOLARES
RECIBIDO
EBN

DR. CARLOS MANUEL ASTORGA ZARAGOZA
SUBDIRECTOR ACADÉMICO

C. c. p. Departamento de Ciencias Computacionales
Departamento de Servicios Escolares

CMAZ/RMA

DEDICATORIAS

A mis queridos padres por el apoyo brindado a lo largo de mi vida, porque gracias a su trabajo y sacrificio he tenido la oportunidad de crecer en mi formación profesional.

¡Gracias! en especial a mi mama Herlinda por su paciencia, consejos, dedicación, amor, esfuerzo y por la confianza otorgada.
A todos mis hermanos: Viviana, Sinaí, Uriel y Esteban que son mi motivación.
A Derick Axel, por impulsarme a continuar mis estudios, por su apoyo y compañía durante toda la maestría.
Agradezco a todas las personas que forman parte de mi vida y han hecho posible este logro.

A todos mis profesores quienes han aportado conocimientos y me han guiado durante el proceso académico.

A la generación 2021-2023 del departamento de Ciencias de la Computación.

AGRADECIMIENTOS

Agradezco al Consejo Nacional de Ciencia y Tecnología (CONACyT) por su apoyo económico, sin este no hubiera sido posible el desarrollo de esta tesis.

Agradezco al Tecnológico Nacional de México campus Centro Nacional de Investigación y Desarrollo Tecnológico (TecNM/CENIDET) y a todo su personal por darme la oportunidad de realizar mis estudios de maestría en Ciencias de la Computación.

Agradezco a mis formadores por su orientación indispensable para lograr con éxito este proyecto de investigación, en primer lugar, a mi Director de Tesis, ¡Dr. Juan Carlos Rojas Pérez!, Gracias! por su constante dedicación, por el tiempo empleado para guiarme, por su apoyo e indicaciones durante mi estancia en el TecNM/CENIDET.

A mi comité revisor, conformado por la Dra. Olivia Fragoso Diaz y el Mtro. Humberto Hernández García, quienes me brindaron de su valioso tiempo en revisiones de avance, correcciones, consejos y sugerencias que contribuyeron en la mejora para esta investigación.

Resumen

Big data representa un nuevo paradigma tecnológico para los datos que se generan a alta velocidad, gran volumen, y gran variedad. Desarrollar capacidades (desarrollos tecnológicos en infraestructura, análisis y servicios, etc.) que aprovechen Big data permite a las empresas transformase en organizaciones basadas en datos para seguir siendo competitivas [1].

El Big Data es complejo y requiere de aplicaciones informáticas no tradicionales de procesamiento de datos para tratarlos adecuadamente, dichas aplicaciones requieren de otros mecanismos para evaluar atributos de calidad. Por ejemplo, la tolerancia a fallas, limita el impacto de éstas y permite que el sistema continúe funcionando correctamente aun con su presencia, la capacidad de tolerancia a fallas puede lograrse utilizando enfoques dirigidos a hardware y/o software.
El problema es que actualmente, no existen modelos de calidad para la tolerancia a fallas específicos a soluciones de Big data; por lo tanto, medir el nivel de tolerancia a fallas en un sistema de Big Data desde la perspectiva de software, se desarrolló la presente investigación.

Se realizó una búsqueda del estado del arte, que incluyó la revisión, análisis y recolección de atributos de calidad, métricas y herramientas de código fuente (definición, fórmula y umbrales) para la construcción y generación del modelo. El trabajo de tesis incluye el desarrollo de un modelo de calidad que reúne un compendio de atributos de calidad y métricas de código fuente, que de acuerdo con la literatura están relacionadas a tolerancia a fallas. Debido a que el defecto de código corresponde a una de las 6 causas de fallas en los sistemas de Big Data, Cao *et.al* [2]. Además, el trabajo incluye un catálogo de técnicas de tolerancia a fallas con el propósito de adicionar elementos preventivos que permitan asegurar el funcionamiento normal del sistema.

De modo que, se espera que el modelo propuesto y probado en 4 casos de estudio, sea una guía y contribuya en mejorar el proceso de desarrollo de sistemas Big Data. Sin embargo, es importante resaltar que el trabajo propuesto solo atiende/aborda una parte del problema, debido a que solo está enfocado en resolver el defecto de código del sistema y no significa que el problema sea eliminado de raíz. Debido a la extensibilidad y complejidad que representa un modelo de calidad para la tolerancia a fallas, es necesario plantear nuevas soluciones en el ámbito.

Abstract

Big Data represents a new technological paradigm for data generated at high speed, high volume, and great variety. To develop capabilities (technological developments in infrastructure, analysis, and services) which take advantage of big data, allows companies to transform into data-based organizations in order to remain competitive [1].

Big Data is complex and requires non-traditional data processing applications to be properly processed/used, such applications require other mechanisms to evaluate quality attributes. For example, fault tolerance limiting the impact of failures and allowing the system to continue functioning properly even with its presence. When a failure occurs, fault tolerance can be achieved implementing quality approaches targeting hardware and/or software.

The problem is that currently there are no quality models for fault tolerance specific to Big Data solutions; therefore, to measure the level of fault tolerance in a Big Data system from a software perspective was the, the present investigation objective.

A state-of-the-art search was performed, which included the review, analysis and collection of quality attributes, metrics and source code tools (definition, formula, and thresholds). For the construction and generation of the model. This thesis includes the development of a quality model that brings together a compendium of quality attributes and source code metrics, which according to the literature are related to fault tolerance. Because the code defect corresponds to one of the 6 causes of failures in Big Data systems, Cao *et.al* [2]. In addition, the presented work includes a catalog of fault tolerance techniques with the purpose to adding preventive elements that ensure the normal operation of the system.

Finally, the proposed model was tested in the 4 case studies. It is expected to be a guide and contribute to improve the development process of Big Data systems. However, it is important to note that the proposed work only addresses/attacks a part of the problem because it is only focused on solving the system code defect and does not mean that the problem is eliminated from the root. Due to the extensibility and complexity that represents a quality model for fault tolerance, it is necessary to propose new solutions in the field.

Contenido

Lista de ilustraciones

Lista de tablas

Capítulo 1. Introducción

En esta sección se presenta una pequeña introducción al tema tratado, así como trabajos antecedentes. Se describe el problema que originó el presente trabajo de investigación, así como los objetivos establecidos.

Además, se incluye el alcance, limitaciones, justificación del proyecto y organización de este documento.

Big Data se refiere a volúmenes masivos y complejos de información estructurada y no estructurada [3] como audios, videos, páginas web y textos [4]. Se caracteriza por tamaños más allá de la capacidad de las herramientas de software de uso común para capturar, curar, administrar y procesar datos dentro de un tiempo transcurrido tolerable [5]. Big Data está ganando más popularidad como herramienta para analizar cantidades significativas de datos bajo demanda [6], los cuales son generados a partir de diversas fuentes como redes sociales, Internet de las cosas y aplicaciones multimedia [7], al igual que otras áreas de investigación presenta algunos desafíos debido a que en la práctica existen problemas como la alta escalabilidad, la tolerancia a fallas, la flexibilidad, la confiabilidad y la comprobabilidad, que pueden ser considerados en términos de la ingeniería de software.

En la actualidad los sistemas de software deben de reunir diversas características o atributos de calidad, donde la tolerancia a fallas es una característica relevante, dado que de esta depende la confiabilidad del funcionamiento del sistema. La tolerancia a fallas consiste en la aplicación de técnicas que permiten a un sistema tolerar fallas de software que se pueden presentar en el sistema después de su desarrollo [8].

Una forma de determinar el estatus de un sistema de software que adolece de fallas, es midiendo, lo que permitirá evaluar el sistema, cualitativa o cuantitativamente, y de acuerdo con esta evaluación la organización podrá proponer e implementar estrategias que permitan la mejora del proceso de construcción de software, Callejas *et.al* [9].

Para abordar el problema de la tolerancia a fallas se han propuesto diversas soluciones desde diferentes perspectivas, hasta la presente investigación se ha identificado que no hay un consenso que especifique puntualmente que atributos se requieren para medir la tolerancia a fallas y como medir estos atributos. En este trabajo se presenta una taxonomía para la tolerancia a fallas, conformada por métricas cualitativas (técnicas de tolerancia a fallas) y cuantitativas (atributos de calidad: disponibilidad, rendimiento, mantenibilidad, testeabilidad, y sus respectivas métricas), las cuales servirán como medio para medir la capacidad de tolerancia a fallas que posee un sistema Big Data.

1.1 Antecedentes

En CENIDET (Centro de Investigación y Desarrollo Tecnológico) no se ha desarrollado algún proyecto que anteceda al problema planteado en este trabajo de investigación. Sin embargo, en el ámbito de Big Data se han desarrollado las tesis de maestría “Selección de un Método Ágil para el desarrollo de Sistemas Big Data”, Capistran [10] quien trabajo en seleccionar de entre un conjunto de metodologías ágiles, una metodología o combinación de metodologías que se pueden adaptar a las características cambiantes de los sistemas Big Data y Mapear la metodología ágil seleccionada al desarrollo de un sistema Big Data; y “Estudio de mapeo sistemático en el problema de la variedad en sistemas Big Data”, Gervacio [11], que trata de una investigación para evidenciar mediante un estudio de mapeo sistemático el problema de la variedad de tipos de datos en sistemas Big Data y conocer estatus actual enfocando el estudio a herramientas, IDE's, frameworks y metodologías utilizados para tratar el problema.

1.2 Planteamiento del problema

Hoy en día los sistemas de software deben de reunir diversas características o atributos de calidad, de entre estos atributos la tolerancia a fallas es una característica relevante, dado que de esta depende la confiabilidad del funcionamiento del sistema. Una forma de determinar la tolerancia a fallas en un sistema de software es medirla, el problema consiste en que actualmente, en el desarrollo de sistemas Big Data, no se cuenta con mecanismos explícitos que indiquen o guíen como medir este atributo de calidad. Un sistema desarrollado sin esta característica es un sistema que en su operación será poco fiable, puede conducir a la pérdida de información y provocar falta de disponibilidad (propiedad de que un sistema se encuentre listo para ser utilizado de inmediato) a sus usuarios.

1.3 Objetivos

1.3.1 Objetivo general

Definir un Modelo de calidad (establecer un conjunto de atributos de calidad) para la medición de tolerancia a fallas en un sistema Big Data que apoye en etapas del desarrollo de software y aporte calidad al producto final.

1.3.2 Objetivos particulares

- Investigar, revisar y analizar métricas, modelos y atributos de calidad relacionados con tolerancia a fallas en sistemas Big Data.
- A partir del análisis, recopilar las métricas relacionadas con la tolerancia de fallas en sistemas Big Data.
- Evaluar los atributos de calidad y métricas e identificar como se pueden aplicar para poder medir la tolerancia a fallas en sistemas Big Data.
- Aplicar el modelo propuesto a un caso de estudio de un proyecto de Big Data proveniente de un repositorio.

1.4 Justificación

La tolerancia a fallas es un atributo de calidad a considerar, que tiene un impacto en la experiencia del usuario ya que afecta directamente la fiabilidad del sistema que es la garantía básica para asegurar el buen funcionamiento del mismo. El hecho de no conocer o tener la información concreta de cómo medir la tolerancia a fallas aumenta la presencia de fallas en el sistema, por lo que se requiere de un modelo de calidad que especifique al desarrollador las métricas a considerar en las etapas de diseño y desarrollo de sistemas Big Data.

1.5 Alcances y limitaciones

A continuación, se presentan los alcances y limitaciones del proyecto de tesis propuesto.

1.5.1 Alcances del proyecto

- El modelo de calidad estará enfocado a las etapas de desarrollo del ciclo de vida del software según la norma ISO/IEC 12207 [12].
- Se considerará solo un caso de estudio de sistema Big Data ya desarrollado y documentado.

1.5.2 Limitaciones

- No se analizará el contenedor donde se encuentre alojado el sistema Big Data.
- El modelo de calidad propuesto no contendrá la totalidad del conjunto de métricas y atributos encontrados en la revisión exhaustiva, sino un subconjunto de ellos.
- El modelo de calidad se enfocará específicamente al sub-atributo de tolerancia a fallas que pertenece al atributo de calidad fiabilidad.

1.6 Estructura del documento

El resto del documento se encuentra organizado en los siguientes capítulos, en el **Capítulo 2** se presenta el marco teórico donde se contextualiza al lector sobre los temas que se tratan el documento, el **Capítulo 3** presenta una recopilación del estado del arte, trabajos relacionados y su análisis, en el **Capítulo 4** se presentan las etapas que conforman el proceso para el desarrollo del modelo de calidad propuesto, en **Capítulo 5** se muestran los casos de estudio evaluados, además se presentan las pruebas y resultados obtenidos. Las conclusiones, hallazgos, trabajos futuros, del trabajo de investigación se encuentran en el **Capítulo 6**, en el **Capítulo 7** se presentan las referencias y finalmente en el **Capítulo 8** corresponde a los anexos que incluyen información complementaria para consultar detalles relacionados a casos de estudio y métricas, así como las herramientas utilizadas para la realización de las pruebas.

Capítulo 2. Marco teórico

Este capítulo presenta los conceptos (fundamentos) que fueron utilizados en el trabajo de investigación, y resultan importantes para la comprensión del tema desarrollado.

A continuación, se presenta la teoría fundamental en la cual se basa y sustentan el contenido de este documento.

2.1 Big Data

De acuerdo con Arcila *et.al* [3], Salma *et.al* [5] Big Data se refiere a volúmenes masivos y complejos de información estructurada y no estructurada, con tamaños más allá de la capacidad de las herramientas de software de uso común para capturar, seleccionar información relevante, administrar y procesar datos dentro de un tiempo transcurrido tolerable, que requiere de métodos computacionales para extraer conocimiento.

Las propiedades que definen al Big data (ver *Ilustración 1*) son:

Ilustración 1. 6 V's de Big Data[6]

El *volumen* que se refiere a la generación y recopilación de una gran cantidad de datos donde la escala de datos se vuelve cada vez más alta; la *velocidad* se refiere a la puntualidad con la que se producen, recopilan y analizan los macrodatos; *variedad* indica los diferentes tipos de datos que se pueden producir de forma estructurada y no estructurada como audios, videos, páginas web y textos [4]; *valor* se refiere al propósito o resultado comercial de los datos aportados, para facilitar el proceso de toma de decisiones [6]; *veracidad* se refiere a la calidad de los datos que se procesan. La veracidad de la fuente de datos también depende del análisis de la precisión de los datos [6]; y *variabilidad* se refiere a los datos que no son estables, que no pueden tratarse fácilmente

y son difíciles de gestionar. Explicar datos variables representa un problema importante para los investigadores [6].

2.2 Calidad

En la norma ISO 8402, la calidad se define como la capacidad de satisfacer las necesidades establecidas e implícitas Berander *et.al* [13].

2.3 Calidad del software

Grado en el que un producto de software satisface las necesidades declaradas e implícitas cuando se usa en condiciones específicas ISO/IEC [14].

2.4 Modelo de calidad (QM)

Conjunto definido de características y de relaciones entre ellas, que proporciona un marco para especificar requisitos de calidad y evaluar la calidad ISO 25010 [15]. Un QM provee una taxonomía de factores de calidad y medidas asociadas a los mismos, apropiadas para evaluar la calidad de un sistema de software y puede ser utilizado para establecer requisitos del sistema, Bermeo *et.al* [16].

2.5 Taxonomía

Una taxonomía es una colección de términos de un vocabulario controlado, organizados en una estructura jerárquica, cada término en una taxonomía está en una o más relaciones padre-hijo de otros términos de la taxonomía, las buenas prácticas limitan a que todas las relaciones padre-hijo tengan un único padre que sea del mismo tipo. Una taxonomía de atributos de calidad (AC) se denomina modelo de calidad (MC). Un MC sirve como marco de trabajo para la especificación, evaluación y testeo de la calidad de un Sistema , Blas *et.al* [17].

2.6 Atributo de calidad

Un atributo de calidad (AC) es una caracterización o propiedad específica de un sistema de software que puede tomar un valor cuantitativo o cualitativo, el cual es medible u observable, Blas *et.al* [17].

2.7 Confiabilidad

Grado en el que un sistema, producto o componente realiza funciones específicas en condiciones específicas durante un período de tiempo específico. Las limitaciones en la fiabilidad se deben a fallas en los requisitos, diseño e implementación, o debido a cambios contextuales, Pan *et.al* [19].

2.8 Tolerancia a fallas

La tolerancia a fallas es la capacidad de un sistema o componente de continuar con su funcionamiento normal y brindar un servicio correcto a pesar de la presencia de fallas activas de hardware o software, Al-Kuwaiti *et.al* [18], Pan *et.al* [19].

2.9 Falla

Una falla puede modelarse como una transición de estado no deseada (desviación del sistema del comportamiento especificado, Poola *et.al* [20]), pero sin embargo posible, en un proceso, Gärtner *et.al* [21].

2.10 Medir

Proceso por el cual se asignan números o símbolos a atributos de entidades del mundo real de tal forma que los describa de acuerdo con reglas claramente definidas [22], los resultados de la medición se pueden utilizar para análisis o conclusiones relacionadas con la entidad, Samosir *et.al* [23].

2.11 Métrica

Una métrica, es una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo determinado. "Un identificador o conjetura sobre un atributo dado" ISO 25010 [15].

Capítulo 3. Análisis de trabajos relacionados

En esta sección se presentan los trabajos que fueron encontrados en el estado del arte y relacionados al tema de investigación. Dichas investigaciones fueron analizadas y se presenta una clasificación por grupos de trabajo: 1) relacionados con atributos de calidad, 2) relacionados con métricas de código fuente y programación orientada a objetos y 3) técnicas de tolerancia a fallas. Finalmente se presenta una comparativa de los trabajos revisados.

3.1 Análisis de trabajos relacionados

En esta sección se presenta un breve resumen del estado actual de las investigaciones que dieron origen al modelo de tolerancia a fallas propuesto. A continuación, los trabajos descritos estas organizados por: trabajos que relacionan con las fallas presentes en sistemas de Big Data, atributos de calidad, métricas de código fuente y técnicas de tolerancia a fallas. El trabajo de Cao *et.al* [2] investiga a detalle los problemas existentes en el sistema de Big Data y se establece el modelo de árbol de fallas, donde la falla de operación tiene la mayor probabilidad de ocurrencia, misma que está relacionada con el error humano. Además, esta propuesta es considerada debido a que brinda un panorama de la problemática existente y que se podría abordar mediante la tolerancia a fallas en dichos sistemas. Del catálogo de fallas presentadas, las ocasionadas por defecto de código del sistema son el enfoque de esta investigación.

Zhou *et.al* [24] presenta un estudio empírico sobre 210 problemas de calidad del servicio de Big Data Microsoft ProductA causados por fallas de hardware 21,0%, defectos del lado de sistema 36,2% y fallas del lado del cliente 37,2%. Donde para el aspecto de estudio sobre las fallas del lado del sistema se clasifican en: defecto de código, limitación de diseño y fallo de funcionamiento, los defectos del código en el producto A representan el 21.0%.

3.1.1 Trabajos relacionados con atributos de calidad

De acuerdo con ISO/IEC 9126 [25] la tolerancia a fallas es una sub-característica de la confiabilidad, el trabajo de Berander *et.al* [13] proporciona una visión general de los diferentes modelos de calidad y de los diversos conceptos definidos se encuentra el de confiabilidad y tolerancia a fallas, por lo que se consideran trabajos que permiten conocer de donde parte “Tolerancia a Fallas”.

El uso de la tolerancia a fallas tiene el propósito de garantizar la calidad de las aplicaciones de Big Data, y aunque varios autores discuten diferentes sub-atributos de tolerancia a fallas, algunos de los más comunes según el trabajo de Zhang *et.al* [26] son: confiabilidad, disponibilidad y capacidad de prueba (testeabilidad) de la aplicación. En el trabajo de , Al-Kuwaiti *et.al* [18], además son considerados los atributos de rendimiento (capacidad de

ejecución), y la capacidad de mantenimiento, el trabajo presenta una taxonomía para el concepto de tolerancia a fallas, así como para la confiabilidad, la seguridad. Lo anterior permitió identificar en términos generales como medir la tolerancia a fallas.

Los trabajos de Zhang *et.al* [26], Al-Kuwaiti *et.al*. [18], ISO/IEC/IEEE 24765 [27] son utilizados para definir los conceptos relacionados a tolerancia a fallas: disponibilidad, rendimiento, mantenibilidad y testeabilidad.

3.1.2 Trabajos relacionados a métricas de código fuente para tolerancia a fallas

En la *Tabla 1* se presentan algunos trabajos relevantes en donde se consideran las métricas de POO para abordar la tolerancia a fallas y Big Data, Falih *et.al* [28] proponen 6 métricas, sin embargo el enfoque de la investigación es Web GIS (Geographical Information System) y no Big Data. Así mismo, Samosir *et.al* [23], consideran adecuado para Big Data utilizar el enfoque Orientado a Objetos para desarrollar un sistema de análisis Big Data y propone las métricas: WMC, LCOM, CBO, RFC, DIT, NOC, donde a pesar de no estar directamente relacionado a tolerancia a fallas, se entiende que es posible abordar la problemática desde esta perspectiva.

Tabla 1. Trabajos relacionados

Métricas	**Propuesta** [28]**-2016**	**Propuesta** [23]**-2017**
LOC- Líneas de código	*	
NM- Número de métodos	*	
CC- Complejidad Ciclomática	*	
WMC- Métodos de ponderación por clase	*****	*****
DIT- Profundidad del árbol de herencia	*****	*****
CBO- Acoplamiento entre objetos	*	*
LCOM- Falta de cohesión en los métodos		*
RFC- Respuesta para una clase		*
NOC Número de bloques de captura		*

Por otro lado, Nuñez *et.al* [29] presenta un mapeo sistemático, los autores realizaron una recopilación de estudios relacionados (un total de 226 artículos revisados) a atributos de calidad y sus respectivas métricas de código fuente en el periodo de 2010-2015. Finalmente, Singhani *et.al* [30] proponen un modelo de evaluación de la capacidad de prueba (testeabilidad) del software orientado a objetos, por otro lado, Nabi *et.al* [31], ISO/IEC/IEEE 24765: 2010 [27] abordan el atributo de disponibilidad.

3.1.3 Trabajos relacionados con técnicas de tolerancia a fallas

De acuerdo con Neves et.al [32] Big Data y computación en la nube son conceptos compatibles, debido a que se relacionan con las propiedades de: volumen, variedad, y variabilidad de los datos Zanoon *et.al* [33], propiedades presentes en un sistema Big Data, además puntualiza que las técnicas de tolerancia a fallas son necesarias en los sistemas Big Data para lograr sistemas disponibles, tolerantes a fallas, escalables y flexibles.

Mukwevho et.al [34] realizan una revisión de los métodos de tolerancia a fallas en los sistemas en la nube y proponen una taxonomía, Shahid *et.al* [35], Hasan et.al [36] los diferentes tipos de fallas, y sus causas.

3.2 Tabla comparativa de trabajos encontrados

A continuación, se en *Tabla 2 se* presenta una comparativa que muestra las técnicas y métodos utilizados en los trabajos relacionados para obtener información relacionada a la tolerancia a fallas en sistemas de Big Data.

Tabla 2. Comparativa de trabajos relacionados

Año	Trabajo	Fallas en sistemas Big Data	Atributos de calidad	Técnicas de tolerancia a fallas	Uso de Métricas	Modelo de calidad	Taxonomía	Estándar	Big Data	Tolerancia a Fallas
2004	Measuring Software Product Quality: A Survey of ISO/IEC 9126 [25]		✓							✓
2005	Software quality attributes and trade-offs [13]		✓			✓		✓		✓
2009	A comparative analysis of network dependability, fault-tolerance, reliability, security, and survivability [18]		✓				✓			✓
2010	Systems and software engineering—Vocabulary ISO/IEC/IEEE 24765: 2010 [27]		✓		✓			✓		✓
2015	An Empirical Study on Quality Issues of Production Big Data Platform [24]	✓							✓	

Continuación, …Tabla 2. Comparativa de trabajos relacionados

Año	Trabajo	Fallas en sistemas Big Data	Atributos de calidad	Técnicas de tolerancia a fallas	Uso de Métricas	Modelo de calidad	Taxonomía	Estándar	Big Data	Tolerancia a Fallas
2015	Testability assessment model for object oriented software based on internal and external quality factors [30]		✓		✓	✓				
2016	Availability in the cloud: State of the art [31]		✓		✓			✓		
2016	Big Data in Cloud Computing: Features and Issues [32]			✓			✓		✓	✓
2016	Quality measurement for Web GIS using object-oriented development [28].				✓					✓
2017	Cloud computing and big data is there a relation between the two: a study [33]								✓	
2017	A survey on quality assurance techniques for big data applications [26]		✓							✓
2017	Measurement Metric Proposed For Big Data Analytics System [23]				✓				✓	
2017	Source code metrics: A systematic mapping study [29]		✓		✓					
2018	Research on reliability evaluation of big data system [2]	✓					✓		✓	
2018	Fault tolerance in cloud computing environment: A systematic survey [36]	✓		✓			✓			✓
2021	Toward a Smart Cloud: A Review of Fault-Tolerance Methods in Cloud Systems [34]			✓			✓			✓
2021	Towards Resilient Method: An exhaustive survey of fault tolerance methods in the cloud computing environment [35]			✓			✓			✓
En este trabajo		✓	✓	✓	✓	✓	✓	✓	✓	✓

3.3 Trabajos adicionales para la investigación

Durante el proceso de investigación, se consultaron trabajos (ISO/IEC ISO [14], [37], Al-Kilidar *et.al* [38], Callejas *et.al* [9], Qian *et.al* [39], Al-Kuwaiti *et.al* [19], Storey *et.al* [40], Merino *et.al* [41], Garises *et.al* [42], Reddivari *et.al* [43], Khan *et.al* [44], Davoudian *et.al* [45], Martinez et.al [46], Kasu *et.al* [47], Rosa *et.al* [48], Dyavanur *et.al* [49], Fang *et.al* [50], Wu *et.al* [51], Cowsalya *et.al* [52], Saadoon *et.al* [53], Mukwevho *et.al* [34], Boranbayev *et.al* [54], Alshehri *et.al* [55], Aziz *et.al* [56], [57], Poola *et.al* [20], Ullah *et.al* [58], Akhtar *et.al* [59], Zhang *et.al* [26], Lackovic *et.al* [60]) dado que están relacionados con:

- ¿Qué es Big Data?, sus propiedades y tipos de datos que se procesan estos sistemas.
- ¿Qué es calidad del software, modelo de calidad y atributo de calidad?
- ¿Qué es un defecto, error y falla?
- ¿Qué tipo fallas que se presentan en un sistema Big Data y cuáles son los atributos que las describen?
- ¿Cuáles son las herramientas utilizadas y Frameworks para la construcción del software de Big Data?
- ¿Qué métricas, algoritmos y principios son utilizados para bordar la tolerancia a fallas?

Los trabajos consultados aportaron una noción general relacionada con el tema de investigación y permitieron la familiarización de nuevos conceptos, su relación e importancia.

Capítulo 4. Desarrollo del modelo de calidad

Este capítulo describe el proceso de implementación de la metodología de solución para lograr este trabajo de investigación, se presenta el catálogo de atributos involucrados con la tolerancia a fallas, así como los indicadores para la evaluación de las métricas de código fuente (cualitativas) y técnicas de tolerancia a fallas (cuantitativas) que conforman la taxonomía resultante para abordar la tolerancia a fallas en sistemas Big Data.

El presente capitulo describe el proceso empleado en la realización del Modelo de Calidad para la Medición de Tolerancia a Fallas en Sistemas de Big Data, plasmando el conjunto de métricas involucradas para abordar la problemática de tolerancia a fallas desde la perspectiva "defecto de código del sistema", así como información destacada para un mejor entendimiento del tema de investigación (fallas en los sistemas de Big Data, atributos de calidad, métricas de código fuente, tolerancia a fallas y técnicas de tolerancia a fallas, etc.).

4.1 Fallas presentes en un sistema Big Data

En el trabajo de Cao *et.al* [2] se refieren 6 fallas principales que ocurren en un sistema Big Data: 1) Falla de hardware, 2) Defecto de código, 3) Limitación de diseño del sistema, 4) Falla de operación, 5) Mala configuración y 6) Falla del software de aplicación. En la *Ilustración 2*, se muestran los tipos de fallas y su clasificación.

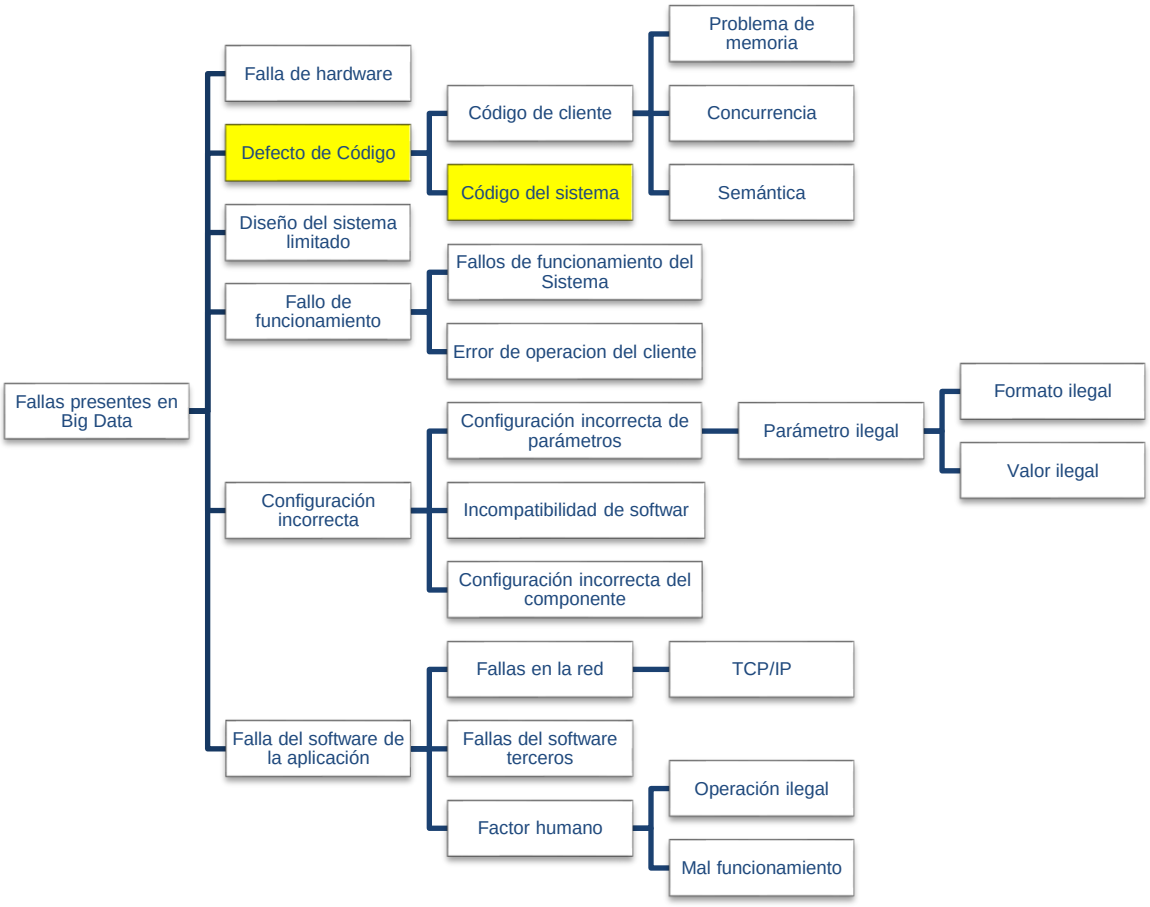


Ilustración 2. Fallas presentes en un sistema Big data [2]

De las fallas presentes en Big Data mencionadas anteriormente, el modelo propuesto en este trabajo se centra en abordar la problemática relacionada a defecto de código del sistema, resaltado en la *ilustración 1*, el cual está asociado al código del sistema debido al impacto que tiene en la degradación y defecto de los componentes que conforman el sistema, entre otras cosas, Cao *et.al* [2].

Lo deseable es que un sistema realice cada una de sus funciones correctamente, sin embargo, durante la vida del software no se puede garantizar que el funcionamiento esté libre de fallas, para aportar confiabilidad a pesar de la presencia de fallas, es necesario adoptar medidas para la **tolerancia a fallas.**

4.2 Elaboración del modelo para tolerancia a fallas en sistemas Big Data.

Un Modelo de Calidad (MC), sirve como marco de trabajo para la especificación, evaluación y testeo de la calidad de un Sistema, Blas *et.al* [17]. En la *Ilustración 3* *Ilustración 3* se presentan y describen las tres etapas secuenciales que conformaron el proceso de investigación y desarrollo.

Ilustración 3. Proceso de elaboración del Modelo de calidad

4.2.1 Identificación de los atributos de calidad relacionados a la tolerancia a fallas

La tolerancia a fallas, es un sub-atributo de *confiabilidad*, Al-Kuwaiti *et.al* [18], se define como la capacidad de un sistema o componente para realizar las funciones requeridas en condiciones establecidas durante un período de tiempo específico ISO/IEC 12207-200 [37]. La confiabilidad es un AC que otorga un producto digital de confianza ISO/IEC 25010 y se

conforma por los sub -atributos disponibilidad, madurez, tolerancia a fallas y recuperabilidad [15] (*Ilustración 4*).

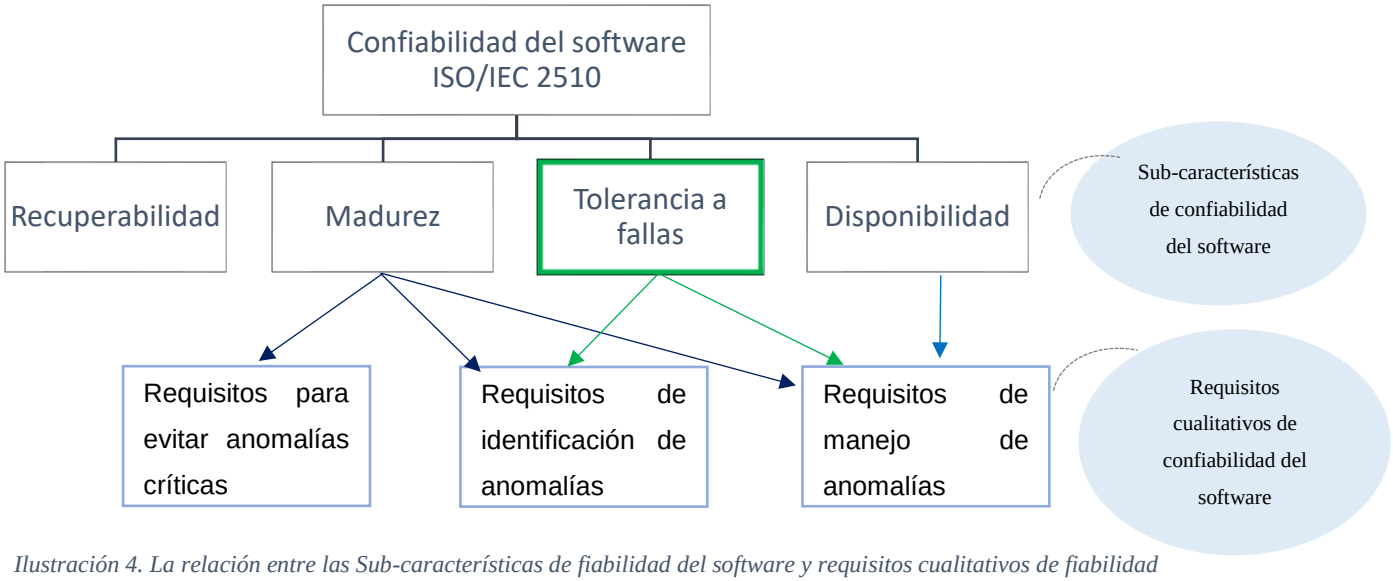


Ilustración 4. La relación entre las Sub-características de fiabilidad del software y requisitos cualitativos de fiabilidad del software, tomada de [19].

Este modelo está enfocado en proponer una forma de medir la ***tolerancia a fallas*** en los sistemas Big data, por lo que a continuación se presentan la definición y la descripción de los sub-atributos relacionados a este concepto.

4.2.1.1 Tolerancia a fallas y atributos relacionados

Aunque varios autores analizan los diferentes atributos relacionados a tolerancia a fallas, no existe un consenso que especifique puntualmente, que atributos de calidad a considerar. Sin embargo, de acuerdo a la literatura y como resultado de esta investigación se consideran los sub-atributos: 1) Disponibilidad [18], [26], 2) Rendimiento (capacidad de ejecución) [26], [18], 3) Mantenibilidad [18] y 4) Testeabilidad (capacidad de prueba) [18], sub- característica de mantenibilidad, según ISO/IEC 9126), Berander *et.al*, [13], Jung *et.al* [25].

De acuerdo a lo anterior se obtuvo la taxonomía de los sub-atributos relacionados al concepto de "Tolerancia a Fallas" (ver *Ilustración 5*), basado en los trabajos de Zhang *et.al* [26], Al-Kuwaiti *et.al* [18].

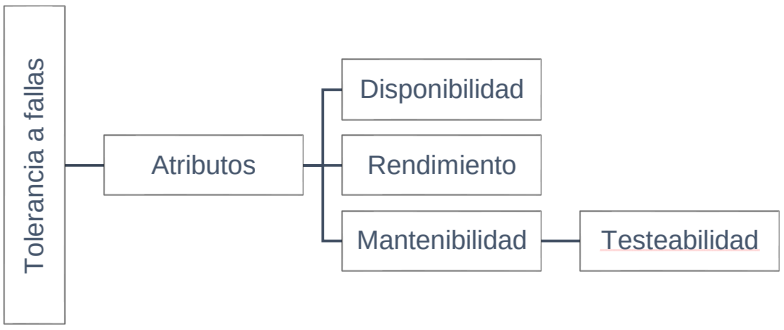


Ilustración 5. Atributos para medir la tolerancia a fallas [18].

Disponibilidad

Capacidad de un componente o servicio para realizar su función requerida en un instante determinado o durante un período de tiempo determinado, ISO/IEC/IEEE 24765 [27]**.**

$$Disponibilidad = \frac{MTTF}{MTTF+MTTR} \qquad (1)$$

Donde *MTTF* corresponde a Tiempo medio hasta el fallo y *MTTR* a Tiempo medio de reparación (para más detalle revisar la sección 4.2. 2).

Rendimiento

El grado en que un sistema o componente cumple sus funciones designadas dentro de determinadas restricciones, como la velocidad, la precisión o el uso de la memoria, Al-Kuwaiti *et.al* [18], ISO/IEC/IEEE 24765 [27]. Este factor indica el rendimiento de las grandes aplicaciones de datos, como la disponibilidad, tiempo de respuesta, etc., Zhang *et.al* [26].

Mantenibilidad

Es la rapidez [62] y facilidad con la que un sistema o componente puede modificarse para corregir fallas, mejorar el rendimiento u otros atributos, o adaptarse a un entorno modificado, Al-Kuwaiti *et.al* [18], Lackovic *et.al* [60] , ISO/IEC/IEEE 24765 [27].

Testeabilidad (Capacidad de prueba)

Grado en que se puede diseñar una prueba objetiva y factible para determinar si se cumple un requisito, ISO/IEC/IEEE 24765 [27].

4.2.2 Recopilación de métricas correspondientes a la clasificación de sub-atributos

De acuerdo con el ISO 9126:1[13], [63] hay tres enfoques para la calidad del software: calidad interna (calidad de código), calidad externa (calidad de ejecución) y calidad en uso (hasta qué punto se satisfacen las necesidades del usuario en el entorno de trabajo del usuario). Las métricas del código fuente son componentes esenciales en el proceso de medición del software, se extraen del código fuente del software, y sus valores permiten obtener conclusiones sobre la atributos de calidad medidos , Nuñez *et.al* [29]. Por lo que, esta sección tiene como objetivo presentar métricas de Programación Orientada a Objetos (recopiladas de la literatura) que permitan evaluar la calidad del código, y son consideradas aquellas métricas de código fuente que están involucradas con los sub-atributos de calidad relacionados a tolerancia a fallas.

En la Tabla 3, se listan las métricas utilizadas para medir los atributos de calidad relacionados a tolerancia a fallas (disponibilidad, rendimiento, mantenibilidad y testeabilidad).

4.2.2.1 Clasificación de métricas relacionadas a los atributos que miden la tolerancia a fallas

Tabla 3. Clasificación de las métricas relacionadas a los atributos que miden la tolerancia a fallas [18][26].

No.	Etiqueta	Métrica	ATRIBUTOS DE CALIDAD RELACIONADOS A MEDIR A TOLERANCIA A FALLAS [18][26]					
				Rendimiento [29]				
			Mantenibilidad [29]	Predecir los resultados de la compilación de software	Rendimiento del software	Eficiencia del código	Testeabilidad (Capacidad de prueba) [30] [29]	Disponibilidad [27] [31]
1	**A**	Abstracción	●	●				
2	**AHF**	Factor de ocultación de atributos				●		
3	**Ca**	Acoplamientos aferentes	●	●				
4	**CBO**	Acoplamiento entre objetos	●			●	●	
5	**Ce**	Acoplamientos eferentes	●	●				
6	**DAC**	Acoplamiento de abstracción de datos	●					
7	**DIT**	Profundidad del árbol de herencia	●	●	●	●	●	
8	**Dn/D**	Distancia normalizada desde la secuencia principal	●	●				
9	**I**	Inestabilidad	●	●				
10	**LCOM**	Falta de cohesión en los métodos	●	●			●	
11	**LOC**	Número de líneas de código	●	●		●		
12	**MHF**	Factor de ocultación del método				●		
13	**MPC**	Acoplamiento de paso de mensajes	●					
14	**NMO/NORM**	Número de métodos anulados	●				●	
15	**NEST**	Nivel de anidamiento		●				
16	**NOA/NA/NOF**	Número de atributos		●	●			
17	**NOC**	Número de bloques de captura	●		●	●		
18	**NOM/NM**	Número de métodos	●	●	●		●	
19	**NOPA**	Número de atributos públicos	●					
20	**NOPM/NPM**	Número de métodos públicos	●		●			
21	**NPAR**	Número de parámetros		●				
22	**RFC**	Respuesta para una clase	●			●		
23	**V(G)**	Complejidad Ciclomática McCabe	●	●		●		
24	**WMC**	Métodos ponderados por clase	●	●		●	●	
25	**ME**	Manejo de excepciones						●
26	**MTTF**	Tiempo medio hasta el fallo						●
27	**MTTR**	Tiempo estimado o promedio para reparar						●
28	**MTBF**	Tiempo medio entre fallos						●

Nuñez *et.al* [29], destacan que las 10 métricas más utilizadas y/o estudiadas a lo largo del tiempo (en el periodo de 2010-2015 de un total de 226 artículos revisados) son : CBO, DIT, LCOM, LOC, NA, NOC, NOM, RFC, V(G) y WMC, las cuales se resaltaron (sombreado azul) en la *Tabla 3.*

Es importante notar que algunas de las **métricas** presentadas en la taxonomía propuesta, **se aplican en 2 o más atributos**, dicha métrica **mide lo mismo para todos los casos**, sin embargo, esta puede **interpretarse** según sea el atributo al que se le relacione.

Ejemplo 1:

WMC (**Métodos ponderados por clase**), es utilizada en los atributos de rendimiento, mantenibilidad y testeabilidad.

Donde:

- **Rendimiento**: Algunos estudios experimentales indican la existencia de distintas relaciones entre la métrica de McCabe y el número de errores existentes en el código fuente por lo que los valores altos de WMC podrían conducir a más fallas de software Capiluppi *et.al* [64], Chidamber *et.al* [65].
- **Mantenibilidad**: Las clases con una gran cantidad de métodos son difíciles de reutilizar y mantener, cuanto mayor sea el número de métodos en una clase, esta métrica es un predictor de cuánto tiempo y esfuerzo se requiere para desarrollar y mantener la clase, mayor será el impacto potencial Capiluppi *et.al* [64].

- **Testeabilidad:** Define el número de caminos independientes dentro de un fragmento de código y determina el número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez. Así como el tiempo requerido para encontrar y corregir esos errores.

Ejemplo 2:

CBO (Acoplamiento entre clases de objetos), es utilizada en los atributos de rendimiento, mantenibilidad y testeabilidad**.**

- **Rendimiento**: El acoplamiento excesivo entre clases de objetos es perjudicial, revela la dependencia de una clase sobre otras clases y afecta el rendimiento, Dubey *et.al* [66], Goyal *et.al* [67].
- **Mantenibilidad**: Disminuye, la modularidad, afectando la capacidad de mantenimiento (cuando se tiene un CBO alto, mayor será la sensibilidad a los cambios en otras partes y por lo tanto el mantenimiento es más difícil modificar o corregir fallas) [66].
- **Testeabilidad**: Un valor más alto de CBO complica las pruebas de la clase y, como resultado, disminuye la comprensión y la capacidad de prueba (testeabilidad), es decir, se realiza más esfuerzo para probar y garantizar que el software realiza las

funciones deseadas. Cuanto mayor sea el acoplamiento entre clases, más rigurosa debe ser la prueba Capiluppi *et.al* [64].

4.2.3 Revisión de las técnicas de tolerancia a fallas más utilizadas

Aunque Big Data resuelve gran parte de nuestros problemas en la actualidad, todavía presenta algunas lagunas y problemas que generan preocupación y necesitan mejoras. De acuerdo con Neves *et.al* [32], Big Data y computación en la nube son conceptos compatibles, debido a que se relacionan con las propiedades de: volumen, variedad, y variabilidad de los datos[6], propiedades presentes en un sistema Big Data, para esta etapa como resultado de la investigación, además de considerar técnicas de tolerancia a fallas específicas de Big Data se incluyen técnicas aplicadas en sistemas en la nube, con el propósito de ampliar la capacidad de disponibilidad Shahid *et.al* [35], Hasan *et.al* [36].

Para poder cumplir con los requisitos actuales de los sistemas Big Data, es decir: 1) Ser disponibles, 2) Ser tolerantes a fallas, 3) Ser escalables y 4) Ser flexibles, es necesario utilizar **técnicas de tolerancia a fallas,** Zanoon *et.al* [33].

4.2.3.1 Enfoques de las técnicas de tolerancia a fallas

A continuación, se presentan los enfoques principales de técnicas de tolerancia a fallas, así como una breve descripción.

Enfoques proactivos

Se define como la capacidad del sistema para estar en un estado preparado o controlado para manejar las posibles interrupciones antes de que ocurran, Hasan *et.al* [36]. Estos métodos funcionan monitoreando constantemente el sistema y realizando predicciones de fallas para prevenir los efectos de las fallas mucho antes de que ocurran. Las técnicas clave utilizadas son el seguimiento, la predicción y la reasignación de los recursos de la nube, Mukwevho *et.al* [34]. Los sistemas tolerantes a fallas proactivos permanecen ininterrumpidos hasta que las expectativas coincidan con la experiencia, Hasan *et.al* [36].

Enfoques reactivos

Los enfoques reactivos de tolerancia a fallas manejan las fallas después de su ocurrencia, Hasan *et.al* [36], el énfasis está principalmente en la recuperación del sistema, Shahid *et.al* [35]. El estado del sistema se guarda y utiliza constantemente durante el proceso de recuperación. El funcionamiento de los enfoques reactivos se basa en la respuesta más que en la anticipación, Hasan *et.al* [36]. Las técnicas clave utilizadas son la replicación, la creación de puntos de control y el reinicio, Shahid *et.al* [35].

Enfoques resilientes

Funcionan prediciendo fallas e implementando métodos para evitar o mitigar el impacto de dichas fallas en el sistema (comparten una serie de características comunes con los métodos proactivos). Además del monitoreo y la predicción, los métodos resilientes también adaptan (afinan) la capacidad de tolerancia a fallas del sistema mediante la incorporación de aprendizaje inteligente a través de la interacción con el entorno de alojamiento. Aquí es donde los métodos resilientes difieren significativamente de los métodos proactivos, Mukwevho *et.al* [34].

4.2.3.2 Taxonomía de técnicas de tolerancia a fallas

El conjunto de técnicas presentes en la taxonomía (ver Ilustración 6), serán consideradas como métricas cualitativas para medir la tolerancia a fallas en sistemas Big Data, sin embargo, la aplicación de estas depende del alcance de información disponible en relación al proyecto, es decir se necesita una forma de comprobar que dicha técnica de tolerancia es empleada en el proyecto, por ello la necesidad de disponer de información detallada y tangible del sistema.

De lo contrario, si no es posible evaluar si el caso de estudio posee esta característica, se exponen con el objetivo de que sean contempladas y puedan considerarse en la realización de proyectos Big Data.

A continuación, se presenta la taxonomía de técnicas de tolerancia a fallas (*Ilustración 6*) aplicables a sistemas Big Data.

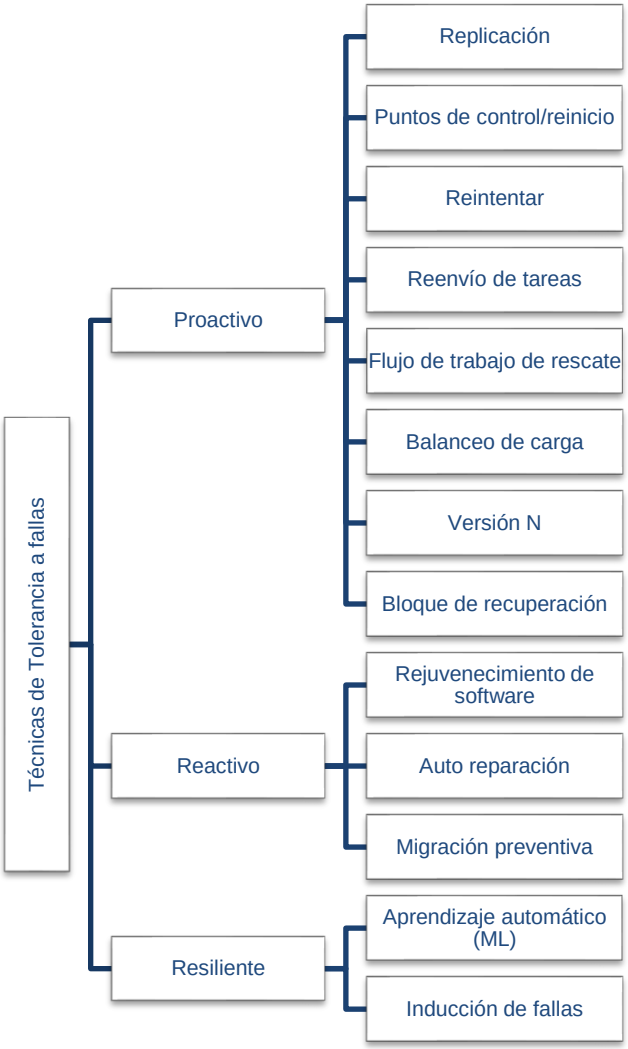


Ilustración 6. Técnicas de tolerancia a fallas, [35], [36], [34].

En la *Tabla 4* se describen las técnicas señaladas anteriormente en la taxonomía.

4.2.3.3 Descripción de las técnicas de tolerancia a fallas

Tabla 4. Descripción de las técnicas de tolerancia a fallas en sistemas Big data

Tipo	Técnica	Descripción
Proactivo [35] [36]	Punto de control [34][49] [52]	Consiste en mantener el estado de la aplicación después de cualquier finalización efectiva o almacenar el estado preliminar sin fallas (guardar periódicamente el estado más reciente, sin fallas [34]), para que en caso de que el sistema sufra una avería este pueda revertir la función y la tarea es reiniciada desde un estado funcional [35].
	Replicación [34] [49] [52] [51]	Algunos componentes del sistema se replican y se implementan simultáneamente en diferentes recursos y continuar, el objetivo es robustecer el sistema para garantizar la ejecución de los trabajos [34]. De modo que si un componente falla otro puede tomar el relevo.
	Reenvió de tareas [35] [34]	El sistema vuelve a enviar la tarea fallida al mismo o diferente recurso para su ejecución [34].
	Reintentar [68] [34] [35] [36]	El sistema reintenta solicitudes fallidas en el mismo recurso varias veces [34], hasta lograr éxito [68]. El número de intentos puede ser definido de acuerdo a la situación, después de eso se puede migrar a algún otro recurso [36].
	Flujo de trabajo de rescate [35][34]	El sistema permite que el flujo de trabajo continúe incluso si la tarea falla hasta que se vuelve imposible continuar sin atender a la tarea fallida.

Continuación, … Tabla 4. Descripción de las técnicas de tolerancia a fallas en sistemas Big data.

Tipo	Técnica	Descripción
Proactivo [35] [36]	Balanceo de carga [35][34][68]	El sistema incorpora equilibrio de carga para distribuir la carga de trabajo por igual entre los nodos que forman un sistema. La distribución justa o la carga de trabajo garantiza que los nodos no se abrumen y, por lo tanto, eviten un estado defectuoso del sistema. Esta técnica se aplica comúnmente en aplicaciones de múltiples clústeres [34].
	N-versión [35] [34] [52]	El sistema se desarrolla utilizando el modelo de programación de múltiples versiones (programas que funcionalmente equivalentes desarrollados con el mismo conjunto de especificaciones). Los programas desarrollados de forma independiente reducen en gran medida la probabilidad de fallas similares en dos o más versiones [34].
	Bloque de recuperación [36]	Es una variación de la técnica: N-versión, de todas las réplicas para una aplicación una se asigna como principal y otras se asignan como en espera, ante una falla, en lugar de reiniciar la aplicación en su totalidad, se puede verificar y así mitigar el tiempo de espera [34] [36].
Reactivo [35] [36]	Rejuvenecimiento de software [35] [34]	El sistema está diseñado para reinicios periódicos. El sistema se reinicia con un estado limpio cada vez [68]. El rejuvenecimiento también se puede categorizar como completo (todos los componentes del sistema se rejuvenecen a la vez) y parcial (solo algunos componentes del sistema se rejuvenecen)[35].

Continuación, … Tabla 4. Descripción de las técnicas de tolerancia a fallas en sistemas Big data.

Tipo	Técnica	Descripción
Reactivo [35] [36]	Auto reparación [35] [34]	El término autor reparación se utiliza para referirse a la recuperación de fallas en tiempo de ejecución. Un sistema debe ser capaz de detectar un error o lesión y luego actuar en consecuencia. La acción correctiva puede implicar que un producto altere su propio estado o efectúe cambios en otros componentes del medio ambiente. La parte restante debe intentar encontrar una manera de continuar trabajando sin la parte defectuosa [69].
	Migración Preventiva [35] [34]	Evita que los componentes del sistema que están a punto de fallar afecten al rendimiento del sistema. Esto se logra mediante la supervisión y alejando los componentes de los nodos que están a punto de fallar al ejecutarse en nodos más estables [34]. Los sistemas tolerantes a fallas siguen un enfoque probabilístico , por lo general, se enfocan en las tasas de falla de los nodos informáticos individuales [36].
Resiliente[36]	Aprendizaje automático [35] [34]	Los sistemas están capacitados para aprender automáticamente y usar la experiencia para mejorar sin ninguna programación explícita. Su motivo principal es desarrollar programas de computadora, que pueden acceder a los datos, y luego usar esos datos para aprender [68].
	Inducción a fallas [35] [34]	Se introducen errores específicos, para simular una situación catastrófica, en un sistema y se hacen observaciones sobre cómo responde el sistema. Si el sistema falla, se implementan mecanismos para manejar dichos errores y, como resultado, la resistencia del sistema mejora significativamente, se puede verificar y así mitigar el tiempo de espera [34] [36].

4.3 Modelo de calidad para medir la tolerancia a fallas en sistemas Big Data

4.3.1 Taxonomía de tolerancia a fallas en sistemas Big Data

En este trabajo el primer acercamiento del modelo se presentó mediante una taxonomía, que presenta los sub-atributos de calidad: 1) Disponibilidad, 2) Rendimiento, 3) Mantenibilidad y 4) Testeabilidad que determinará el grado total de cumplimiento presente en el atributo de tolerancia a fallas.

Es importante destacar que las métricas MTTF (Tiempo medio hasta el fallo), MTTR (Tiempo estimado o promedio para reparar) Y MTBF (Tiempo medio entre fallos) del atributo disponibilidad, para los casos de prueba realizados han sido descartadas, en vista de que no es posible evaluarlas mediante código estático y se requiere de un ambiente Big Data en ejecución. Por otro lado, para el caso de las técnicas de tolerancia a fallas (agrupadas por enfoques: reactivo, proactivo y resiliente) se considerarán para la evaluación del sistema Big Data siempre y cuando existan los elementos suficientes que permitan valorar si la técnica de tolerancia a fallas es puesta en práctica o no, y si cumple con el objetivo. Por ello, se considera necesario tener al alcance documentación relacionada al proyecto, el ambiente del proyecto y un sistema Big Data funcional.

4.3.1.1 Especificación formal del modelo de tolerancia a fallas

La especificación formal del modelo de Tolerancia a fallas (ver *Ilustración 7*), se presenta en términos de las métricas señaladas a continuación:

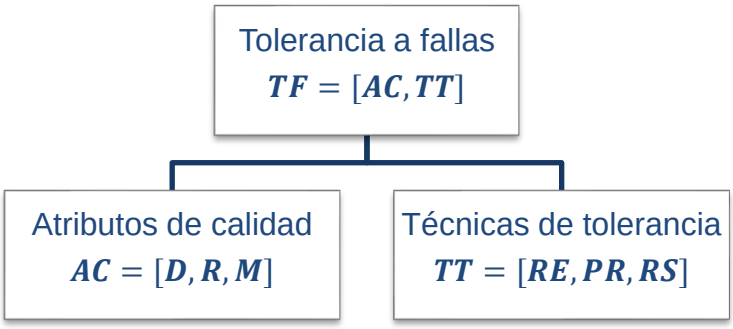


Ilustración 7. Especificación formal de tolerancia a fallas

A continuación (ver *Tabla 5*), se describen los elementos que conforman los conjuntos señalados anteriormente.

Tabla 5. Elementos del modelo de tolerancia a fallas

ELEMENTOS PARA TOLERANCIA A FALLAS	
METRICAS CUANTITATIVAS	**METRICAS CUALITATIVAS**
D= [me, mttf, mttr, mtbf]	**RE**= [rejuvenecimiento de software, auto reparación, migración preventiva]
R = [a, ahf, ca, cbo, ce, dit, dn, i, lcom, loc, nest, na, noc, nom, nopm, npar, mhf, rfc, v(g), wmc]	**PR**= [punto de control, replicación, reenvío de tareas, reintentar, flujo de trabajo de rescate, balanceo de carga, n-versión, bloque de recuperación]
M = [a, ca, ce, dac, dn, i, loc, mpc, noc, nopa, nopm, refc, v(g), {t}	
T= [cbo, dit, lcom, nmo, nom, wmc]	**RS**= [aprendizaje automático, inducción a fallas]

*Nota: Disponibilidad (**D**), Rendimiento (**R**), Mantenibilidad (**M**), Testeabilidad (**T**), Técnicas Reactivas (**TR**), Técnicas Proactivas (**TP**), Técnicas Resilientes (**TRS**).*

La *Tabla 6* presenta la descripción de las métricas mencionadas anteriormente.

Tabla 6. Acrónimos de métricas.

MÉTRICAS			
TF	Tolerancia a fallas	**NMO/NORM**	Número de métodos anulados
AC	Atributos de calidad	**NEST**	Nivel de anidamiento
TT	Técnicas de tolerancia	**NOA/NA/NOF**	Número de atributos
A	Abstracción	**NOC**	Número de bloques de captura
AHF	Factor de ocultación de atributos	**NOM/NM**	Número de métodos
Ca	Acoplamientos aferentes	**NOPA**	Número de atributos públicos
CBO	Acoplamiento entre objetos	**NOPM/NPM**	Número de métodos públicos
Ce	Acoplamientos eferentes	**NPAR**	Número de parámetros
DAC	Acoplamiento de abstracción de datos	**RFC**	Respuesta para una clase
DIT	Profundidad del árbol de herencia	**V(G)**	Complejidad Ciclomática McCabe
Dn/D	Distancia normalizada desde la secuencia principal	**WMC**	Métodos ponderados por clase
I	Inestabilidad	**ME**	Manejo de excepciones
LCOM	Falta de cohesión en los métodos	**MTTF**	Tiempo medio hasta el fallo
LOC	Número de líneas de código	**MTTR**	Tiempo estimado o promedio para reparar
MHF	Factor de ocultación del método	**MTBF**	Tiempo medio entre fallos
MPC	Acoplamiento de paso de mensajes		

Para consultar información que describe su funcionamiento, una fórmula para calcular su valor y finalmente los umbrales (bueno/común, normal/casual y malo/poco común), rangos que serán los utilizados para conocer si la métrica está bien implementada, consultar el Anexo 4, *pág. 150*. Finalmente, resultado de las etapas anteriores, se presenta la taxonomía que puede ser utilizada para medir la tolerancia a fallas en sistemas Big Data (ver *Ilustración 8*), esta se compone de atributos de calidad y técnicas de tolerancia a fallas.

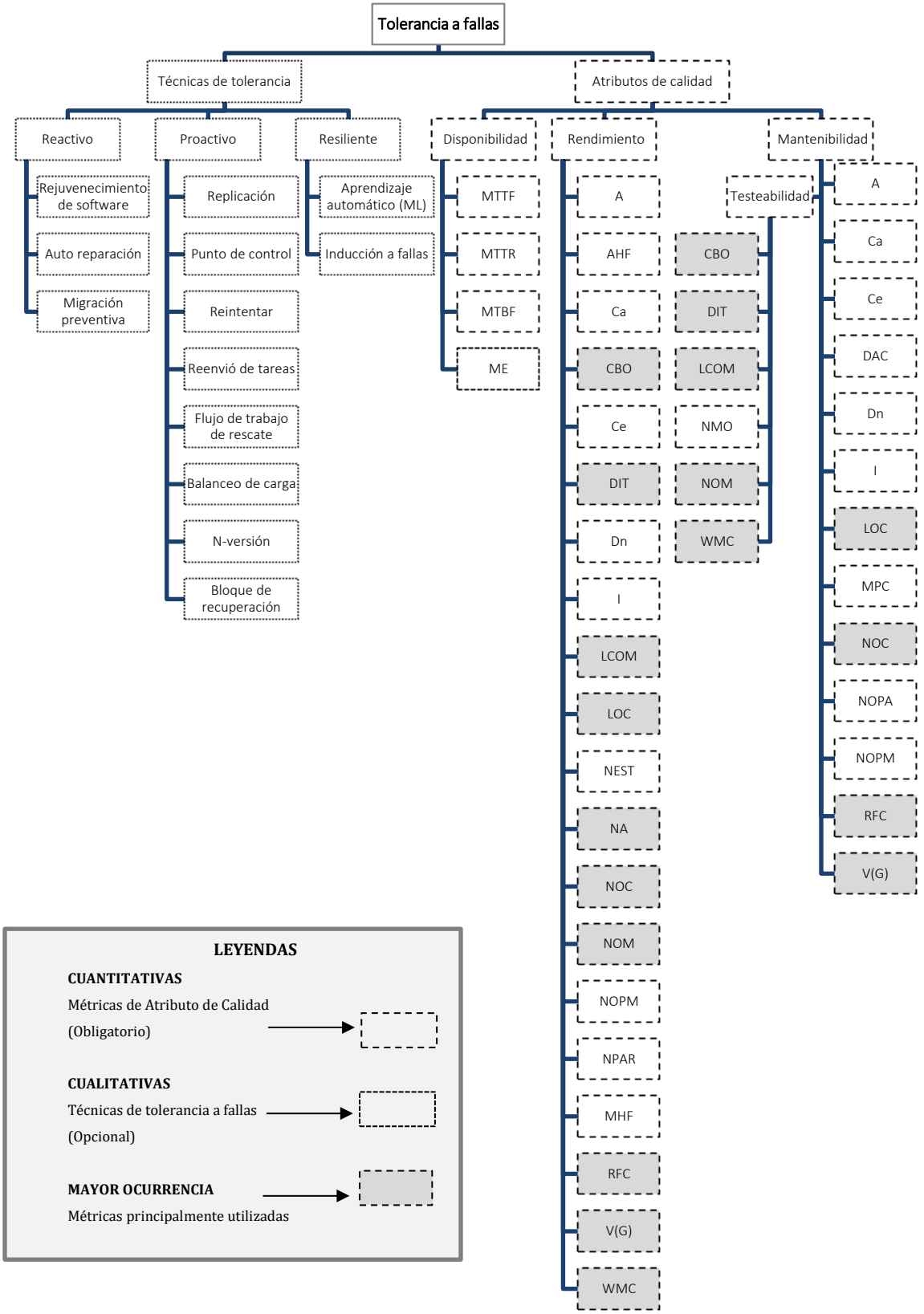


Ilustración 8. Taxonomía de tolerancia a fallas en sistemas de Big Data

4.3.2 Métricas del modelo de tolerancia a fallas en sistemas Big Data

En este trabajo, el atributo de calidad "tolerancia a fallas" se considera atributo compuesto por los sub-atributos: disponibilidad, rendimiento, mantenibilidad y testeabilidad que son evaluados a partir de métricas de código fuente. Sin embargo, también es necesario tomar medidas en la práctica, es decir, la implementación de software Big Data en su ambiente de trabajo (ver ilustración 2) no solo incluye considerar los defectos en el código, sino que también afectan factores como la configuración y el uso de componentes de hardware. Por lo que se consideran técnicas de tolerancia a fallas, que ya están establecidas en otros trabajos para complementar la tolerancia a fallas. En este contexto, deben tomarse en cuenta las métricas cuantitativas y cualitativas como se observa en la *Ilustración 9*.

Ilustración 9. Enfoques de la tolerancia a fallas en sistemas Big Data

Idealmente, aplicar ambos aspectos de calidad cubrirá en mayor medida el atributo de la tolerancia a fallas. Sin embargo, el alcance del trabajo propuesto se concentra en la parte cuantitativa dejando el aspecto cualitativo como un complemento al nivel de calidad.

En esta propuesta de modelo de calidad, las métricas expuestas pueden ser utilizadas según los requerimientos del caso de estudio, sin forzar al evaluador a utilizar todas las métricas del modelo.

A continuación, se describe y presentan las respectivas métricas que conforman los enfoques anteriormente presentados.

4.3.2.1 Métricas cuantitativas

A continuación, se presentan los indicadores para evaluar el cumplimiento de la cada una de las métricas involucradas en la medición de los atributos: disponibilidad, rendimiento, mantenibilidad y testeabilidad. (ver *Tabla 7*).

Tabla 7. Métricas del modelo de calidad para la tolerancia a fallas en los sistemas Big Data

Métrica	CL	CN	Formula	UMBRALES PARA CADA METRICA		
				Bueno/común	Normal/Casual	Malo/poco común
A [70] [71]		•	$A = \frac{N_a}{N_c}$ [70]	A =1 [70]	N/A	A =0[70]
AHF [72] [73] [74] [75] [76] [77]		•	$AHF = \frac{\sum_{i=1}^{TC}\left[\sum_{i=1}^{Ad(ei)}(1-V(A_{mi})\right]}{\sum_{i=1}^{TC} A_d(E_i)}$ *(9)* [77]	MHF= 1 [76]	0.79< MHF ≤0.99	0.79 ≤ MHF ≤1
Ca [78]		•	$CA = \sum_{i=1}^{OP}\sum_{j=1}^{TC_i} C_j(C)$	CA ≤ 7 [78]	7 < CA ≤39 [78]	CA > 39 [78]
CBO [67] [79] [80][81] [82]		•	$CBO(c) = \mid \{d \in C - \{c\} \mid usa\ (c,d) \vee usa\ (d,c)\}$ [79]	CBO ≤ 3 [82]	3 < CBO ≤9 [82]	CBO > 9 [82]
Ce [78]		•	$CE = \sum_{i=1}^{OP}\sum_{j=1}^{TC_i} C(C_j)$	CE ≤ 6 [78]	6 < CE ≤16 [78]	CE > 16 [78]
DAC [83] [79] [84]		•	$DAC(c) = \mid\{a \mid a \in A_I(c) \wedge T(a) \in C\}\mid$ [79]	DAC ≤ 2	3 ≤ DAC ≤ 4	DAC >4 [83]
DIT [67] [85] [86] [82] [87] [78] [88]		•	DIT(c) = | Ancestros (c)| [86]	DIT ≤ 2 [82]	2 < DIT ≤4 [82]	DIT > 4 [82]
Dn/D [89]		•	$D = \mid\mid A + I \mid\mid - 1$ [89]	DN =0	N/A	DN =1
I [90]		•	$I = \frac{EC}{EC+AC}$ [90]	I =0	N/A	I =1

Continuación, … Tabla 7. Métricas del modelo de calidad para la tolerancia a fallas en los sistemas Big Data.

Métrica	CL	CN	Formula	UMBRALES PARA CADA METRICA		
				Bueno/común	Normal/Casual	Malo/poco común
LCOM [91] [82] [87]		•	$LCOM = 1 - \sum MF\|(M * F)$ [82]	LCOM ≤0, 167 [78]	0,167 < LCOM ≤ 0, 725 [78]	LCOM > 0, 725 [78]
LOC [92], [89], [93], [94], [28]		•	$LOC = \sum lineasContables$ Donde: LíneasContables = líneas de código (exceptuando comentarios y saltos de línea). Se maneja umbral para clase y método.	LOC ≤ 24 [28] LOC ≤ 10 [93]	24 <LOC ≤500 [28] 10 <LOC ≤40 [93]	LOC > 500 [28] LOC > 40 [93]
MHF [74] [75] [76] [77] [72]		•	$MHF = \frac{\sum_{i=1}^{TC}\left[\sum_{i=1}^{Nd(ei)}(1-V(N_{mi})\right]}{\sum_{i=1}^{TC} N_d(E_i)}$ *(9)* [77]	MHF= 0,8 [76]	0.6< MHF ≤0.79	0< MHF ≤0.59, 0.9< MHF ≤1
MPC [91] [79] [95] [96]		•	$MPC(c) = \sum_{m \in M_I(c)} \sum_{m' \in SIM(m) - M_I(c)} NSI(m, m')$	MPC ≤ 17 [96]	N/A	MPC > 17
NMO/NORM [97]		•	$\sum_{i=1}^{n} metodosAnulados_i$ Donde: metodosAnulados, son métodos derivados que pueden anularse por extensión, restricción, optimización o conveniencia.	NMO ≤ 2 [78]	2 <NMO ≤4 [78]	NMO> 4 [78]
NEST/NBD [78] [98][99]		•	$nest(M) = Max(nivelDeBloque) + 1$ Donde: M=método Max=Función número máximo	NEST ≤ 1 [78]	1< NEST ≤3 [78]	NEST > 3 [78]
NA/NOA/NOF [84] [100] [101] [102] [78] [103]		•	$A_d(C_i) = A_v(C_i) + A_h(C_i)$ [103]	NA ≤ 3[78]	3 < NA ≤8 [78]	NA > 8[78]

Continuación, … Tabla 7. Métricas del modelo de calidad para la tolerancia a fallas en los sistemas Big Data

Métrica	CL	CN	Formula	UMBRALES PARA CADA METRICA		
				Bueno/común	Normal/Casual	Malo/poco común
NOC [67] [91] [104] [78]		•	$\sum_{i=1}^{n} \text{subclaseDirecta}_i$ Donde: subclaseDirecta = es una clase que hereda de una clase padre principal i= número de subclaseDirecta n= número total de subclasesDirectas de una clase	NOC ≤ 1 [82]	1 <NOC ≤3 [82]	NOC > 3 [82]
NOM/NM [105]		•	$\sum_{i=1}^{n} \text{metodoLocal}_i$ Donde: metodoLocal = un método definido localmente por una clase (sin contar métodos heredados ni los constructores).	NOM ≤ 6 [78]	6 < NOM ≤ 14 [78]	NOM> 14 [78]
NOPA [106] [107]		•	$\sum_{i=1}^{n} \text{atributosPublicos}_i$ Donde: atributosPublicos, cuenta todos los atributos de una clase que se declaran como públicos.	NOPA ≤ 3 [106]	5 < NOPA ≤ 10 [106] [107]	NOPA> 10 [107]
NOPM/NPM [108][109]		•	$\sum_{i=1}^{n} \text{metodosPublicos}_i$ Donde: metodosPublicos, cuenta todos los métodos de una clase que se declaran como públicos.	NOPM ≤ 1 [108]	1 < NOPM ≤ 10 [109]	NOPM > 10 [109]
NPAR/PAR [78] [89] [110] [111]		•	$\sum_{i=1}^{n} \text{parametros}_i$ Donde: parámetros, cuenta todos los valores que recibe un método	NPAR ≤ 2 [78]	2 < NPAR ≤ 4 [78]	NPAR > 4 [78]

Continuación, … Tabla 7. Métricas del modelo de calidad para la tolerancia a fallas en los sistemas Big Data

Métrica	CL	CN	Formula	UMBRALES PARA CADA METRICA		
				Bueno/común	Normal/Casual	Malo/poco común
RFC [112] [88]		•	$RFC = \mid RS \mid$ [112]	RFC ≤ 6 [78]	6 < RFC ≤ 36 [78]	RFC > 36 [78]
V(G) [78] [89] [76] [113] [114] [115] [93] [116]		•	$V(G) = numero\ de\ declaraciones\ de\ decision + 1$ [93]	V(G) ≤ 2 [78]	2 < V(G) ≤ 10 [78], [116], [89]	V(G)> 10 [116], [89]
WMC [67] [88]		•	$WMC = \sum_{i=1}^{n} c_i$ [112]	WMC ≤ 11 [78]	11 < WMC ≤ 34 [78]	WMC > 34 [78]
MTTF [18][27]		•	$MTTF = \frac{\sum_{i}^{n} t_i}{n}$ [27]	0	n/a	n/a
MTTR [18][27]		•	$MTTR = \frac{\sum_{i}^{n} R_i}{n}$ [27]	0	n/a	n/a
MTBF [27][18]		•	$MTBF = MTTR + MTTF$ [31] Donde: MTTF es el Tiempo medio hasta el fallo) y MTTR el Tiempo estimado o promedio para reparar.	0	n/a	n/a

4.3.2.2 Métricas cualitativas

El conjunto de técnicas de tolerancia a fallas presentes en la taxonomía pertenece al grupo de métricas cualitativas, mismas que representan un complemento al conjunto de métricas cuantitativas propuestas en el modelo de calidad, debido a que son considerados mecanismos necesarios e importantes para un sistema Big Data. Sin embargo, para este tipo de métricas es fundamental comprobar y/o validar su implementación, para así calificar si cumplen o no su propósito, de lo contrario, deberán ser descartadas por el momento y atender este aspecto para una futura valoración.

Derivado de la recopilación de las técnicas de tolerancia a fallas más comunes para abordar la tolerancia a fallas, a continuación, en la *Tabla 8*, se presenta una plantilla utilizada para la clasificación de técnicas de tolerancia a fallas y su correspondiente método de evolución.

Donde:

SI, corresponde a una correcta implementación de la técnica. NO, revela que es deficiente y no cumple el propósito. N/A, esta conclusión se da por dos situaciones: 1) Simplemente indica que ni siquiera es considerada como un mecanismo de tolerancia a fallas y 2) Debido a que no se cuenta con información suficiente y/o disponible que permita obtener una valoración del caso de estudio.

Tabla 8. Técnicas de tolerancia del modelo de calidad para la tolerancia a fallas en los sistemas Big Data

MÉTRICA			TIPO		DESRIPCIÓN	EVALUACIÓN		
Enfoque		Técnica(s) de tolerancia	CL	CN	Método de evaluación	SI	NO	N/A
Técnicas de tolerancia a fallas	**Reactivo** (8 Técnicas)	Replicación	●		Algunos elementos del sistema se repiten y distribuyen alrededor de varias fuentes (replicas configuradas), para garantizar el funcionamiento de cada trabajo [35].			
		Puntos de control	●		El sistema está configurado para reiniciarse desde el ultimo estado señalado verificado y activa el punto de control [68]. Esta práctica es más efectiva, que iniciar el ciclo desde el principio [35].			
		Reintentar	●		Intentar consultas fallidas en el mismo activo de diversas maneras [35]			
		Reenvío de tareas	●		En cualquier punto, cuando se detecta una tarea fallida, a enviar/reenviar al recurso idéntico y/o a un activo alternativo para su ejecución [68] [117].			
		Flujo de trabajo de rescate	●		El sistema emplea flujos de trabajo alternativos, con el propósito de lograr su función, hasta el punto de ser necesario atender y reparar la falla.			
		Balanceo de carga	●		El balanceador de carga monitoriza las condiciones de cada servidor y direcciona las peticiones al más apto.			
		N-Versión	●		La aplicación se ejecuta simultáneamente en todas las réplicas y el resultado final se decide por votación. El resultado final en activo es la primera respuesta recibida correctamente [34]			
		Bloqueo de recuperación	●		Si la aplicación principal falla, se reinicia secuencialmente en las réplicas en espera disponibles [36].			

Acrónimos: Tipo de métrica: CL→ Cualitativa, CN → Cuantitativa

Continuación, … Tabla 8. Técnicas de tolerancia del modelo de calidad para la tolerancia a fallas en los sistemas Big Data

MÉTRICA			TIPO		DESRIPCIÓN	EVALUACIÓN		
Enfoque		Técnica(s) de tolerancia	CL	CN	Método de evaluación	SI	NO	N/A
Técnicas de tolerancia a fallas	**Proactivo** (3 Técnicas)	Rejuvenecimiento de software			Se realizan copias de seguridad periódicas, el sistema se limpia y repara de cualquier tipo de errores o fallas y se restablece la copia logrando un estado de sistema actualizado [36].			
		Auto reparación	•		Un sistema de recuperación automática tiene la capacidad de modificar su propio comportamiento en respuesta a los cambios en su entorno [69].			
		Migración preventiva	•		Las tareas de los nodos probables de fallo se desplazan de forma preventiva a otros nodos.			
	Resiliente (2 Técnicas)	Aprendizaje automático (ML)	•		El sistema es capaz de predecir fallas en función de los datos previos del sistema [68].			
		Inducción a fallas	•		Si el sistema falla, se implementan mecanismos para manejar dichos errores y, como resultado, la resistencia del sistema mejora significativamente, se puede verificar y así mitigar el tiempo de espera [34] [36].			

Acrónimos: Tipo de métrica: CL→ Cualitativa, CN → Cuantitativa

Capítulo 5. Pruebas y resultados

En este capítulo se presentan los casos de estudio utilizados para aplicar el modelo de calidad propuesto, así como los resultados obtenidos y un breve análisis de lo encontrado. Estas pruebas tienen como objetivo evaluar si la taxonomía propuesta es útil para medir la tolerancia a falas en sistemas de Big Data

5.1 Casos de estudio

En esta sección, se presentan proyectos de software que han sido clasificados como casos de estudio, debido a que cumplen con las características de ser un sistema Big Data y desarrollo *open source,* se encontró un medio para consultar información del proceso de alguna de las etapas de desarrollo (en este caso repositorio GitHub[1]). El objetivo fue utilizar en la medida de lo posible esta información (código estático) como objeto de evaluación del modelo de calidad para medir la tolerancia a fallas en este tipo de sistemas.

Las condiciones para contemplar un desarrollo como caso de estudio son: Ser un software Big Data, desarrollo *open source,* contar con un repositorio para consultar información requerida y desarrollado en lenguaje JAVA.

A continuación (ver *Tabla 9*), se presentan los 4 casos de estudios en los que se probó el modelo de calidad propuesto, así como los resultados obtenidos.

Tabla 9. Descripción de los Casos de estudio

Caso de estudio	Objetivo del software	Framework y/o herramientas relacionados	Lenguaje JAVA %
Analytics on Amazon Customer Reviews, Gandhi [118] (**2020**)	Obtener información sobre las opiniones de los clientes sobre diferentes productos, analizar grandes datos y producir un resultado informativo sobre las opiniones de los clientes sobre el producto Cámara presente en Amazon	Hadoop MapReduce Apache Pig	100%
Twitter Analyser, Mehrotra [119] (**2021**)	Un algoritmo de conteo con pérdida implementado para determinar los hashtags de mayor tendencia usando la API de Twitter para obtener un flujo continuo de tweets.	Apache Storm Apache Hadoop Scala, Twitter (Tweetstream API)	100%
Modeling Adverse Drug Reactions, Koziol [120] (**2022**)	OpenFDA Big Data Pipeline permite la recopilación, el procesamiento y la presentación en tiempo real de datos sobre eventos adversos de medicamentos de la base de datos de openFDA.	Apache Kafka Mongo DB, Spring Boot.	37.2%
CRIMEGRAPH, Marciani *et.al* [121] (**2017**)	Aplicación de análisis visual en tiempo real para la predicción y detección de enlaces en redes criminales.	Apache flink Apache Kafka Neo4j	98.3%

[1] https://www.github.com

La siguiente sección corresponde a los resultados obtenidos por la aplicación del modelo de calidad propuesto, se presentan tablas para listar las métricas evaluadas, donde la columna *detalle* indica el número de elementos que conforman el proyecto y han sido revisados: número de paquetes, numero de clases y numero de métodos, la columna *métrica* presenta un sombreado identificador y el conjunto de métricas aplicadas para cada tipo de elemento, por ejemplo: los paquetes se distinguen por ser color azul aqua y fueron calculadas 6 métricas: abstracción (A), Acoplamiento Aferente (CA), Acoplamiento Eferente (CE), Distancia Normalizada (DN), Inestabilidad (I), Factor de Ocultación de Métodos (MHF).
Por otro lado, el valor obtenido del cálculo por métrica está presente en el apartado de resultados, es importante destacar que la medición de las métricas está basada en umbrales identificados en la literatura y en este trabajo se clasifican en Bueno/común (color verde), Normal/casual (amarillo) y Malo/poco común (rojo), lo que nos permitió obtener una perspectiva del nivel de tolerancia a fallas que posee el desarrollo Big Data.

A continuación, en *Tabla 10* se muestran los resultados obtenidos en el caso de estudio #1: Modeling Adverse Drug Reactions

5.2 Resultados Caso de estudio 1- Modeling Adverse Drug Reactions, Koziol [120]

Tabla 10. Resultados a detalle de caso de estudio #1

DETALLES		MÉTRICAS		
No. Paquetes: 12	PAQUETE	CLASE	MÉTODO
No. Clases: 28	A, CA, CE, DN, I	AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO	LOC, NEST, NPAR, V(G) (4)
No. Métodos: 18	MHF (6)	NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)	
Líneas de código totales: 305			

		RESULTADOS DE MÉTRICAS EVALUADAS																										
		CODIGO ESTATICO																									EJECUCIÓN		
ID	ELEMENTOS REVISADOS:	A	AHF	CA	CBO	CE	DAC	DIT	Dn/D	I	LCOM	LOC	MHF	MPC	NMO-NORM	NEST	NA/NOF	NOC	NOM	NOPA	NOPM	NPAR	RFC	V(G)	WMC	ME	MTTF	MTTR	MTBT
1	consumer	0		0		1			0.0	1.0		8	1.0												1				
2	OpenfdaConsumerApplication		N/A		1		0	1			0.0	6		0	0		0	0	0	0	1		2		1				
3	main ( )				1							2				1						1		1		0			
4	OpenfdaConsumerApplicationTests		N/A		0		0	1			0.0	2		0	0		0	0	0	0	4		0		0				
5	config	0		0		1			0.0	1.0		33	1.0												2				
6	KafkaConsumerConfig		0.5		8		1	1			1.0	33		0	0		4	0	0	0	2		10		2				
7	consumerFactory ( )				4							13				1						0		1		0			
8	drugAdverseEventKafkaListenerContainerFactory ( )				4							9				1						0		1		0	NO APLICA		
9	kafka	0		0		1			0.0	1.0		16	N/A												1				
10	DrugAdverseEventKafkaConsumer		0.0		2		3	1			0.0	16		1	0		1	0	1	0	0		4		1				
11	handleMessage ( )				2							11				1						3		1		0			
12	mongo	0.7		1		3			0.4	0.8		24	1.0												2				
13	DrugAdverseEventDocument		0.3		1		0	1			0.0	18		0	0		12	0	0	0	0		0		0				
14	DrugAdverseEventMapper		N/A		2		2	1			0.0	5		0	0		0	0	2	0	0		2		2				
15	map(Collection)				0							1				1						1		1		0			
16	map(DrugAdverseEvent)				2							1				1						1		1		0			
	Métricas más utilizadas [29]	Bueno/Común→							Normal/Casual→								Malo/Poco común→												

Continuación, … Tabla 10. Resultados a detalle de caso de estudio #1

DETALLES		MÉTRICAS																												
No. Paquetes: 12 **No. Clases: 28** **No. Métodos: 18** **Líneas de código totales: 305**		**PAQUETE** **A, CA, CE, DN, I** **MHF (6)**									**CLASE** **AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO** **NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)**											**MÉTODO** **LOC, NEST, NPAR, V(G) (4)**							
		RESULTADOS DE MÉTRICAS EVALUADAS																												
		CODIGO ESTATICO																										EJECUCIÓN		
ID	**ELEMENTOS REVISADOS:**	**A**	**AHF**	**CA**	**CBO**	**CE**	**DAC**	**DIT**	**Dn/D**	**I**	**LCOM**	**LOC**	**MHF**	**MPC**	**NMO-NORM**	**NEST**	**NA/NOF**	**NOC**	**NOM**	**NOPA**	**NOPM**	**NPAR**	**RFC**	**V(G)**	**WMC**	**ME**	**MTTF**	**MTTR**	**MTBT**
17	DrugAdverseEventRepository		N/A		0		1	1			0.0	1		0	0		0	0	0	0	0		0		0				
18	kafka	0		1		0			1.0	0.0		16	N/A												0				
19	DrugAdverseEvent		0.2		0		0	1			0.0	16		0	0		11	0	0	0	0		0		0				
20	openfda	0		0		0			0.0	1.0		2	N/A												0				
21	OpenfdaProducerApplicationTests		N/A		0		0	1			0.0	2		0	0		0	0	0	0	0		0		0				
22	producer	0		0		1			0.0	1.0		6													1				
23	OpenfdaProducerApplication		N/A		1		0	1			0.0	6	1.0	0	0		0	0	0	0	1		2		1				
24	main ( )				1							2				1						1		1		0			
25	client	0		0		4			0.0	1.0		22	0.8												8				
26	ClientRequestException		N/A		0		0	4			0.0	3		0	0		0	0	1	0	1		3		1				
27	ClientRequestException				1							2				1						1		1		1			
28	RestTemplateResponseErrorHandler		N/A		6		3	1			0.0	13		3	0		0	0	2	0	1		4		5				
29	handleError				6							8				3						1		4		1			
30	hasError				3							3				1						1		1		1			
31	ResultsNotFoundException		N/A		1		0	4			0.0	3		0	0		0	0	1	0	1		3		1				
32	ResultsNotFoundException ( )				1							1				1						1		1		1			
33	ServerProcessingException		N/A		1		0	4			0.0	3		0	0		0	0	1	0	1		3		1				
34	ServerProcessingException ( )				1							1				1						1		1		1			
35	config	0		0		2			0.0	1.0		36	1.0												3				
36	KafkaProducerConfig		N/A		2		1	1			0.0	15		0	0		1	0	2	0	2		3		2				

NO APLICA

Métricas más utilizadas [29] **Bueno/Común→** **Normal/Casual→** **Malo/Poco común→**

Continuación, … Tabla 10. Resultados a detalle de caso de estudio #1

DETALLES		MÉTRICAS		
No. Paquetes: 12		**PAQUETE**	**CLASE**	**MÉTODO**
No. Clases: 28		**A, CA, CE, DN, I**	**AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO**	**LOC, NEST, NPAR, V(G) (4)**
No. Métodos: 18		**MHF (6)**	**NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)**	
Líneas de código totales: 305		**RESULTADOS DE MÉTRICAS EVALUADAS**		
		CODIGO ESTATICO		EJECUCIÓN

ID	ELEMENTOS REVISADOS:	A	AHF	CA	CBO	CE	DAC	DIT	Dn/D	I	LCOM	LOC	MHF	MPC	NMO-NORM	NEST	NA/NOF	NOC	NOM	NOPA	NOPM	NPAR	RFC	V(G)	WMC	ME	MTTF	MTTR	MTBT
37	kafkaTemplate ( )				1							3				1						0		1		0			
38	producerFactory ( )				3							8				1						1		1		0			
39	KafkaTopicConfig		0.5		2		0	1			0.0	21		0	0		4	0	1	0	1		6		1				
40	topic ( )				2							10				1						0		1		0			
41	job	0		0		1			0.0	1.0		15	1.0												2				
42	FdaSynchronizationJob		N/A		1		1	1			0.0	15		0	0		1	0	1	0	1		2		2				
43	fetchEvents ( )				1							9				2						0		2		1			
44	kafka	0		0		0			0.0	1.0		15	1.0												0				
45	DrugAdverseEvent		N/A		0		0	1			0.0	15		0	0		11	0	0	0	0		0		0		NO APLICA		
46	response	0		0		1			0.0	1.0		112	1.0												2				
47	FdaDrug		0.5		1		1	1			0.0	11		0	0		3	0	0	0	0		0		0				
48	FdaEventResponse		0.3		1		2	1			0.0	9		0	0		2	0	1	0	1		2		1				
49	getResults ( )				0							2				1						0		1		0			
50	FdaEventResponsePage		0.5		2		1	1			0.0	10		0	0		2	0	1	0	1		2		1				
51	hasNextPage ( )				1							2				1						0		1		0			
52	FdaMeta		0.5		1		1	1			0.0	9		0	0		2	0	0	0	0		0		0				
53	FdaMetaResult		0.5		0		0	1			0.0	8		0	0		3	0	0	0	0		0		0				

Métricas más utilizadas [29] **Bueno/Común→** **Normal/Casual→** **Malo/Poco común→**

Continuación, … Tabla 10. Resultados a detalle de caso de estudio #1

DETALLES		MÉTRICAS		
No. Paquetes: 12	PAQUETE	CLASE	MÉTODO	
No. Clases: 28	A, CA, CE, DN, I	AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO	LOC, NEST, NPAR, V(G) (4)	
No. Métodos: 18	MHF (6)	NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)		
Líneas de código totales: 305	RESULTADOS DE MÉTRICAS EVALUADAS			

		CODIGO ESTATICO																							EJECUCIÓN		
ID	ELEMENTOS REVISADOS:	A	AHF	CA	CBO	CE	DAC	DIT	Dn/D	I	LCOM	LOC	MHF	MPC	NMO-NORM	NEST	NA/NOF	NOC	NOM	NOPA	NOPM	NPAR	RFC	V(G)	WMC	ME	MTTF	MTTR	MTBT
54	FdaPatient		0.3		0		2	1			0.0	13		0	0		4	0	0	0	0		0		0				
55	FdaPrimarySource		0.5		0		0	1			0.0	7		0	0		1	0	0	0	0		0		0				
56	FdaReaction		0.5		0		0	1			0.0	7		0	0		1	0	0	0	0		0		0			NO APLICA	
57	FdaResult		0.5		2		2	1			0.0	21		0	0		8	0	0	0	0		0		0				
58	OpenFda		0.0		0		0	1			0.0	17		0	0		6	0	0	0	0		0		0				
	Métricas más utilizadas [29]		Bueno/Común→								Normal/Casual→									Malo/Poco común→								

De los datos anteriores se realizó un análisis (ver *Tabla 11*), que consiste en presentar para cada métrica evaluada, que umbral tuvo mayor frecuencia de aparición.

Donde:

La columna *elemento*, clasifica el total de los elementos revisados (columna *tipo*) por métrica en umbrales (bueno -B, normal-N y malo-M) y finalmente se presentan las *observaciones* de acuerdo a los resultados obtenidos. Es importante destacar que el cálculo de las métricas se realizó mediante el uso de las herramientas mencionadas en el **Anexo 2,** *pág. 144* sin embargo, la metodología que siguen y/o su implementación son caja negra.

5.2.1 Análisis resultados-Caso de estudio 1

A continuación, se presentan gráficas correspondientes a la distribución de valores por umbral definidos en la tabla anterior.

Tabla 11. Resultados de Caso de Estudio 1

CASO DE ESTUDIO #1: Modeling Adverse Drug Reactions, Koziol [120]						
Tipo	**Métrica**	**Elementos Umbral**				**Observaciones**
		B	N	M	N/A	
Paquete(s) **(12)**	A	0	0	11	1	Las métricas afectadas son 4: Abstracción (A), Distancia Normalizada (DN), Inestabilidad (I) y Factor de Ocultación de Métodos (MHF).
	CA	12	0	0	0	
	CE	12	0	0	0	
	DN	10	0	1	1	
	I	1	0	10	1	
	MHF	0	1	8	1	
Clase(s) **(28)**	AHF	0	0	15	13	Las métricas afectadas son 3: Factor de Ocultamiento de Atributos (AHF), Falta de Cohesión en los Métodos (LCOM), y Numero de Atributos (NA).
	CBO	26	2	0	0	
	DAC	26	2	0	0	
	DIT	25	3	0	0	
	LCOM	27	0	1	0	
	LOC	27	1	0	0	
	MPC	28	0	0	0	
	NMO	28	0	0	0	
	NA	20	5	3	0	
	NOC	28	0	0	0	
	NOM	28	0	0	0	
	NOPA	28	0	0	0	
	NOPM	25	3	0	0	
	RFC	27	1	0	0	
	WMC	28	0	0	0	
Método(s) **(18)**	LOC	16	2	0	0	No hay valores clasificados en el umbral MALO, por lo tanto, los valores para las respectivas métricas son aceptables.
	NEST	16	2	0	0	
	NPAR	17	1	0	0	
	V(G)	17	1	0	0	

De un total de 12 paquetes evaluados, la *Ilustración 10* muestra el porcentaje de distribución de los umbrales utilizados para las métricas A, CA, CE, DN, I, MHF.

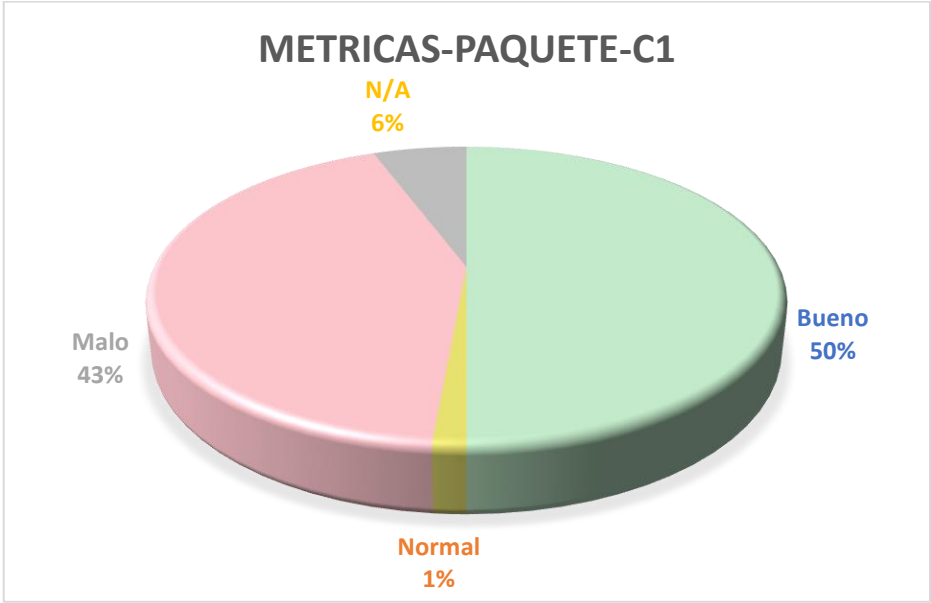


Ilustración 10. Gráfica de Resultados por paquete del C1

De acuerdo a la gráfica anterior se entiende que los umbrales más comunes son Bueno 50% y Malo con 43%.

El siguiente grafico (ver *Ilustración 11*) corresponde a la distribución de los resultados obtenidos en la evaluación de métricas: AHF,CBO, DAC, DIT, LCOM, LOC, MPC, NMO,

NA, NOC, NOM, MOPA, NOPM, RFC, WMC aplicadas a las 28 clases JAVA que el sistema Big Data posee donde prevalece el umbral Bueno con 88%.

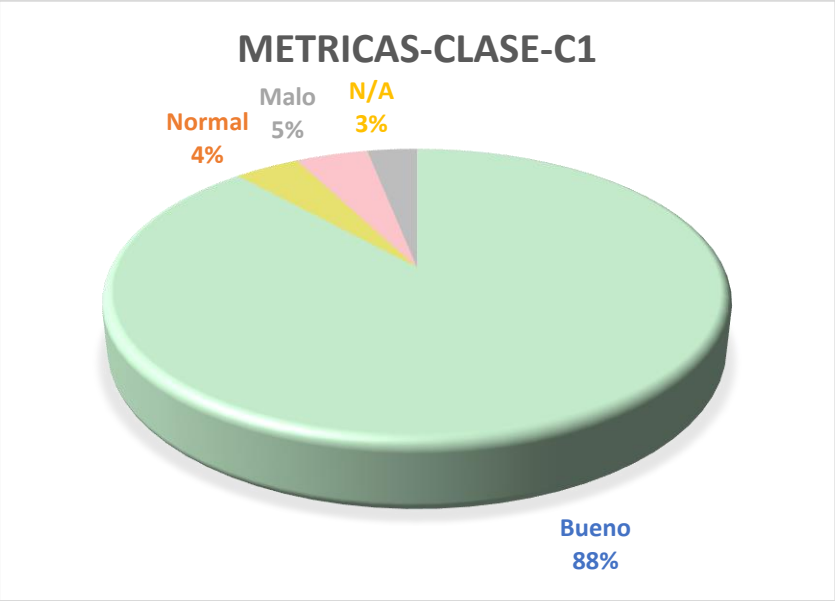


Ilustración 11. Gráfica de Resultados por Clase del C1

Finalmente, en *Ilustración 12* presenta las métricas aplicadas a 18 métodos: LOC, NEST, NPAR, V(G).

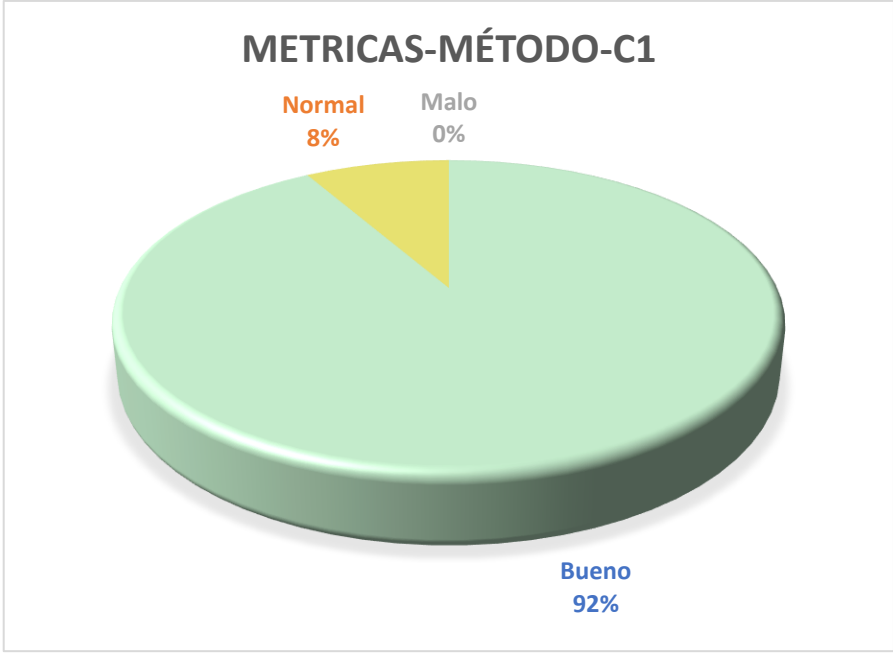


Ilustración 12. Gráfica de resultados por método del C1

De acuerdo a la gráfica anterior se identifica que 92% de los métodos implementados en las clases están dentro de los rangos permitidos o "adecuados" para las métricas señaladas anteriormente.

Por otro lado, los resultados del proyecto, agrupados por atributo de calidad se presentan a continuación.

Cada uno de los atributos relacionados a “Tolerancia a Fallas” se conforma por métricas de programación, que fueron evaluadas a partir de los umbrales donde, al clasificar los resultados de acuerdo al atributo que pertenecen, se obtiene el una representación (ver *Ilustración 13*,*Ilustración 14*,*Ilustración 15*) del porcentaje bueno, normal, malo que posee el proyecto en relación a los atributos: rendimiento, mantenibilidad y testeabilidad con el propósito de identificar que propiedad representa mayor riesgo.

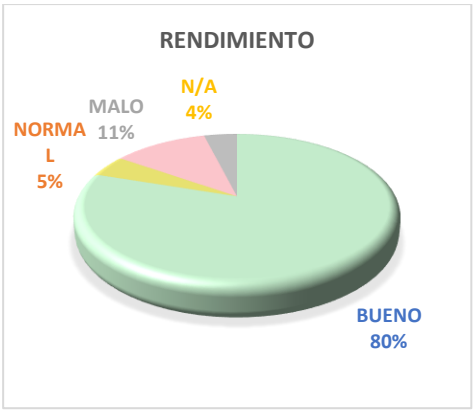


Ilustración 13. Atributo Rendimiento

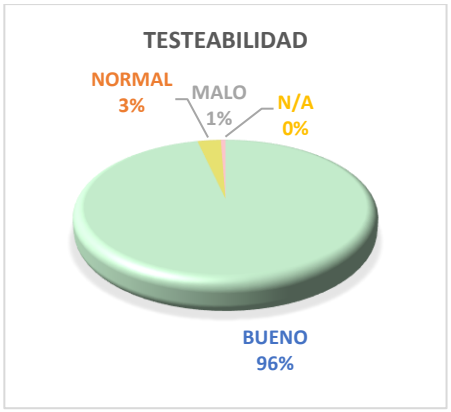


Ilustración 14. Sub-atributo Testeabilidad

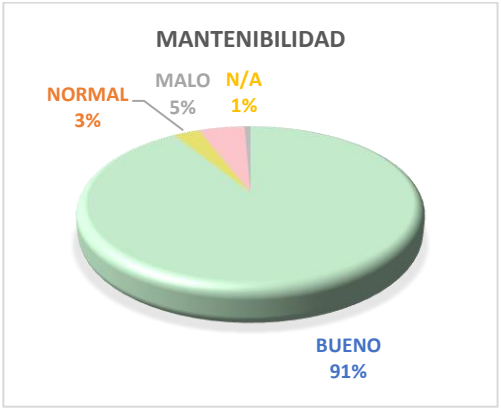


Ilustración 15. Atributo Mantenibilidad

De acuerdo con lo anterior, se entiende que el atributo de rendimiento es el que refleja más carencias en el proyecto Big Data: Modeling Adverse Drug Reactions, Koziol [120]. A continuación, la *Tabla 12* presenta los resultados obtenidos en la evaluación del caso de estudio: Big Data Analytics on Amazon Customer Reviews Using Hadoop and MapReduce, Gandhi [118].

5.3 Resultados Caso de estudio 2- Big-Data Analytics on Amazon Customer Reviews, Gandhi [118]

Tabla 12. Resultados a detalle de caso de estudio #2

DETALLES		MÉTRICAS		
No. Paquetes: 8		**PAQUETE**	**CLASE**	**MÉTODO**
No. Clases: 28		**A, CA, CE, DN, I**	**AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO**	**LOC, NEST, NPAR, V(G) (4)**
No. Métodos: 41		**MHF (6)**	**NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)**	
Líneas de código totales: 563		**RESULTADOS DE MÉTRICAS EVALUADAS**		

		CODIGO ESTATICO																							EJECUCIÓN		
ID	**ELEMENTOS REVISADOS:**	**A**	**AHF**	**CA**	**CBO**	**CE**	**DAC**	**DIT**	**Dn/D**	**I**	**LCOM**	**LOC**	**MHF**	**MPC**	**NMO-NORM**	**NEST**	**NA/NOF**	**NOC**	**NOM**	**NOPA**	**NOPM**	**NPAR**	**RFC**	**V(G)**	**WMC**	**ME**	**MTTF**	**MTTR**	**MTBT**
1	amazonAnalysis	0		0		1			0.0	1.0		69	1.00												22				
2	YearPartitioner		N/A		6		3	1			0.0	69		0	0		0	0	0	0	1		18		3	0			
3	YearPartitionMapper		N/A		3		0	2			0.0	12		0	1		2	0	1	0	0		8		2				
4	map ( )				3							9				2						3		2		1			
5	YearPartitionPartitioner		N/A		1		0	2			0.0	34		0	0		0	0	1	0	1		2		15				
6	getPartition ( )				1							33				2						3		15		0			
7	YearPartitionReducer		N/A		3		0	2			0.0	4		0	0		0	0	1	0	0		3		2				
8	reduce ( )				3							3				2						3		2		1	NO APLICA		
9	main ( )				6							18				2						1		3		1			
10	AverageProductRating	0		0		4			0.0	1.0		79	1.00												17				
11	CountAverageTuple		0.5		0		0	1			0.3	24		0	0		2	0	9	0	9		16		9				
12	CountAverageTuple( )				0							1				1						0		1		0			
13	CountAverageTuple ( )				0							3				1						2		1		0			
14	getAverage( )				0							2				1						0		1		0			
15	getCount( )				0							2				1						0		1		0			

Métricas más utilizadas [29] **Bueno/Común→** **Normal/Casual→** **Malo/Poco común→**

Continuación, … Tabla 12. Resultados a detalle de caso de estudio #2

DETALLES		MÉTRICAS		
No. Paquetes: 8		**PAQUETE**	**CLASE**	**MÉTODO**
No. Clases: 28		**A, CA, CE, DN, I**	**AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO**	**LOC, NEST, NPAR, V(G) (4)**
No. Métodos: 41		**MHF (6)**	**NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)**	
Líneas de código totales: 563		**RESULTADOS DE MÉTRICAS EVALUADAS**		
		CODIGO ESTATICO		EJECUCIÓN

ID	ELEMENTOS REVISADOS:	A	AHF	CA	CBO	CE	DAC	DIT	Dn/D	I	LCOM	LOC	MHF	MPC	NMO-NORM	NEST	NA/NOF	NOC	NOM	NOPA	NOPM	NPAR	RFC	V(G)	WMC	ME	MTTF	MTTR	MTBT
16	readFields ( )				0							3				1						1		1		1			
17	setAverage ( )				0							2				1						1		1		0			
18	setCount ( )				0							2				1						1		1		0			
19	toString( )				0							3				1						0		1		0			
20	write ( )				0							3				1						1		1		1			
21	ProductAverageRatingMain		N/A		5		3	1			0.0	24		0	0		0	0	0	0	1		18		2				
22	main ( )				5							22				2						1		2		1			
23	ProductAverageRatingMapper		N/A		4		1	2			0.0	15		3	1		2	0	1	0	1		13		3				
24	map ( )				4							12				3						3		3		1	NO APLICA		
25	ProductAverageRatingReducer		N/A		3		1	2			0.0	16		5	1		1	0	1	0	1		8		3				
26	reduce ( )				3							14				3						3		3		1			
27	BinReviewRatings	0		0		2			0.0	1.0		53	##												14				
28	BinningProductReviewRatingsMain		N/A		8		1	1			0.0	26		0	0		0	0	0	0	1		20		4				
29	main ( )				8							24				3						1		4		1			
30	BinningProductReviewRatingsMapper		0.5		5		0	2			0.0	27		0	3		1	0	3	0	0		13		10				
31	cleanup ( )				2							3				1						1		1		1			
32	map ( )				5							19				3						3		8		1			

Métricas más utilizadas [29] **Bueno/Común→** **Normal/Casual→** **Malo/Poco común→**

Continuación, … Tabla 12. Resultados a detalle de caso de estudio #2

DETALLES		MÉTRICAS		
No. Paquetes: 8	**PAQUETE**	**CLASE**	**MÉTODO**	
No. Clases: 28	**A, CA, CE, DN, I**	**AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO**	**LOC, NEST, NPAR, V(G) (4)**	
No. Métodos: 41	**MHF (6)**	**NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)**		
Líneas de código totales: 563	**RESULTADOS DE MÉTRICAS EVALUADAS**			
	CODIGO ESTATICO			EJECUCIÓN

ID	ELEMENTOS REVISADOS:	A	AHF	CA	CBO	CE	DAC	DIT	DnID	I	LCOM	LOC	MHF	MPC	NMO-NORM	NEST	NA/NOF	NOC	NOM	NOPA	NOPM	NPAR	RFC	V(G)	WMC	ME	MTTF	MTTR	MTBT
33	setup ( )				2							3				1						1		1		0			
34	invertedIndexPattern	0		0		3			0.0	1.0		62	0.8												10				
35	InvertedIndexMain		N/A		7		2	1			0.0	28		0	0		0	0	0	0	1		21		3				
36	main ( )				7							26				3						1		3		1			
37	InvertedIndexMapper		N/A		3		0	2			0.0	15		0	1		2	0	1	0	1		8		3				
38	map ( )				3							12				2						3		3		1			
39	InvertedIndexReducer		N/A		2		0	2			0.0	19		0	1		1	0	1	0	1		8		4				
40	reduce( )				2							17				4						3		4		1			
41	mahoutRecommendation	0		0		4			0.0	1.0		110	0.58												18		NO APLICA		
42	RecommendationDataMapper		N/A		4		0	2			0.0	13		0	1		0	0	1	0	0		9		3				
43	map()				4							12				3						3		3		1			
44	RecommendationMain		N/A		7		3	1			0.0	47		0	0		0	0	0	0	1		23		5				
45	main ( )				7							45				4						1		5		1			
46	RecommendationReducer		0.5		14		0	2			0.8	40		0	2		6	0	2	0	0		18		8				
47	reduce ( )				5							14				4						3		4					
48	setup ( )				7							19				2						1		4		1			
49	UserDataMapper		N/A		4		0	2			0.0	10		0	1		0	0	1	0	0		8		2				

Métricas más utilizadas [29] **Bueno/Común→** **Normal/Casual→** **Malo/Poco común→**

Continuación, … Tabla 12. Resultados a detalle de caso de estudio #2

DETALLES		MÉTRICAS		
No. Paquetes: 8		**PAQUETE**	**CLASE**	**MÉTODO**
No. Clases: 28		**A, CA, CE, DN, I**	**AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO**	**LOC, NEST, NPAR, V(G) (4)**
No. Métodos: 41		**MHF (6)**	**NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)**	
Líneas de código totales: 563		**RESULTADOS DE MÉTRICAS EVALUADAS**		
		CODIGO ESTATICO		EJECUCIÓN

ID	ELEMENTOS REVISADOS:	A	AHF	CA	CBO	CE	DAC	DIT	Dn/D	I	LCOM	LOC	MHF	MPC	NMO-NORM	NEST	NA/NOF	NOC	NOM	NOPA	NOPM	NPAR	RFC	V(G)	WMC	ME	MTTF	MTTR	MTBT
50	map ( )				4							9				2						3		2		1			
51	reduceSideInnerJoin	0		0		4			0.0	1.0		72	##												16				
52	JoinMain		N/A		7		3	1			0.0	28		0	0		0	0	0	0	1		18		3				
53	main ( )				7							26				3						1		3		1			
54	JoinReducer		N/A		2		0	2			0.0	18		0	1		0	0	1	0	0		9		9				
55	reduce ( )				2							17				5						3		9		1			
56	RatingsMapper		N/A		8		0	2			0.0	13		0	1		0	0	1	0	0		8		2				
57	map ( )				3							12				2						3		2		1			
58	TopProductsMapper		N/A		3		0	2			0.0	13		0	1		0	0	1	0	0		8		2				
59	map ( )				3							12				2						3		2		1			
60	topNProducts	0		0		4			0.0	1.0		69	##												14				
61	CountComparator		N/A		2		0	2			0.0	8		0	1		0	0	2	0	1		3		4				
62	compare ( )				2							5				1						2		3		0			
63	CountComparator ( )				0							2				1						0		1		0			
64	TopNProductsMain		N/A		7		3	1			0.0	32		0	0		0	0	0	0	1		25		3				
65	main ( )				7							30				3						1		3		1			
66	TopNProductsMapper		N/A		4		0	2			0.0	12		0	1		0	0	1	0	1		8		2				
67	map ( )				4							11				2						3		2		1			

NO APLICA

Métricas más utilizadas [29] **Bueno/Común→** **Normal/Casual→** **Malo/Poco común→**

Continuación, … Tabla 12. Resultados a detalle de caso de estudio #2

DETALLES		MÉTRICAS		
No. Paquetes: 8		**PAQUETE**	**CLASE**	**MÉTODO**
No. Clases: 28		**A, CA, CE, DN, I**	**AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO**	**LOC, NEST, NPAR, V(G) (4)**
No. Métodos: 41		**MHF (6)**	**NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)**	
Líneas de código totales: 563		**RESULTADOS DE MÉTRICAS EVALUADAS**		
		CODIGO ESTATICO		EJECUCIÓN

ID	ELEMENTOS REVISADOS:	A	AHF	CA	CBO	CE	DAC	DIT	Dn/D	I	LCOM	LOC	MHF	MPC	NMO-NORM	NEST	NA/NOF	NOC	NOM	NOPA	NOPM	NPAR	RFC	V(G)	WMC	ME	MTTF	MTTR	MTBT
68	TopNProductsReducer		0.3		4		0	2			0.5	17		0	2		2	0	2	0	1		7		5				
69	reduce ( )				3							11				4						3		4		1			
70	setup ( )				2							3				1						1		1		1			
71	totalProducts	0		0		3			0.0	1.0		49	1.00												8				
72	ProductCountMain		N/A		6		2	1			0.0	26		0	0		0	0	0	0	1		19		3				
73	main ( )				6							24				2						1		3		1			
74	ProductCountMapper		N/A		4		0	2			0.0	12		0	1		2	0	1	0	1		8		2				
75	map				3							9				2						3		2		1			
76	ProductCountReducer		N/A		3		0	2			0.0	11		0	1		0	0	1	0	1		5		3				
77	reduce ( )				3							10				3						3		3		1			
	Métricas más utilizadas [29]		**Bueno/Común→**								**Normal/Casual→**								**Malo/Poco común→**										

5.3.1 Análisis resultados-Caso de estudio 2

Tabla 13. Resultados Caso de Estudio 2

CASO DE ESTUDIO #2: Big-Data Analytics on Amazon Customer Reviews Using Hadoop and MapReduce, Gandhi [118]						
Tipo	**Métrica**	**Elementos Umbral**				**Observaciones**
		B	N	M	N/A	
Paquete(s) **(8)**	A	0	0	8	0	Las métricas afectadas son 3: Abstracción (A), Inestabilidad (I) y Factor de Ocultación de Métodos (MHF).
	CA	8	0	0	0	
	CE	8	0	0	0	
	DN	8	0	0	0	
	I	0	0	8	0	
	MHF	1	3	4	0	
Clase(s) **(28)**	AHF	0	0	4	24	Las métricas afectadas son 3: Factor de Ocultamiento de Atributos (AHF), Falta de Cohesión en los Métodos (LCOM), y Acoplamiento entre Clases de Objetos (CBO).
	CBO	14	16	1	0	
	DAC	23	5	0	0	
	DIT	28	0	0	0	
	LCOM	25	2	1	0	
	LOC	18	10	0	0	
	MPC	28	0	0	0	
	NMO	27	1	0	0	
	NA	27	1	0	0	
	NOC	28	0	0	0	
	NOM	27	1	0	0	
	NOPA	28	0	0	0	
	NOPM	27	1	0	0	
	RFC	5	23	0	0	
	WMC	27	1	0	0	
Método(s) **(41)**	LOC	19	22	0	0	Las métricas afectadas son 2: Nivel de Anidamiento (NEST), y Complejidad Ciclomática(V(G)).
	NEST	14	22	5	0	
	NPAR	23	18	0	0	
	V(G)	21	16	4	0	

El proyecto analizado está compuesto de 8 paquetes. A continuación, en *Ilustración 16* se presenta la distribución de los umbrales: Bueno, Normal y Malo, obtenidos para los elementos tipo paquete revisados, donde se refleja que existen carencias que abordar en un 42%.

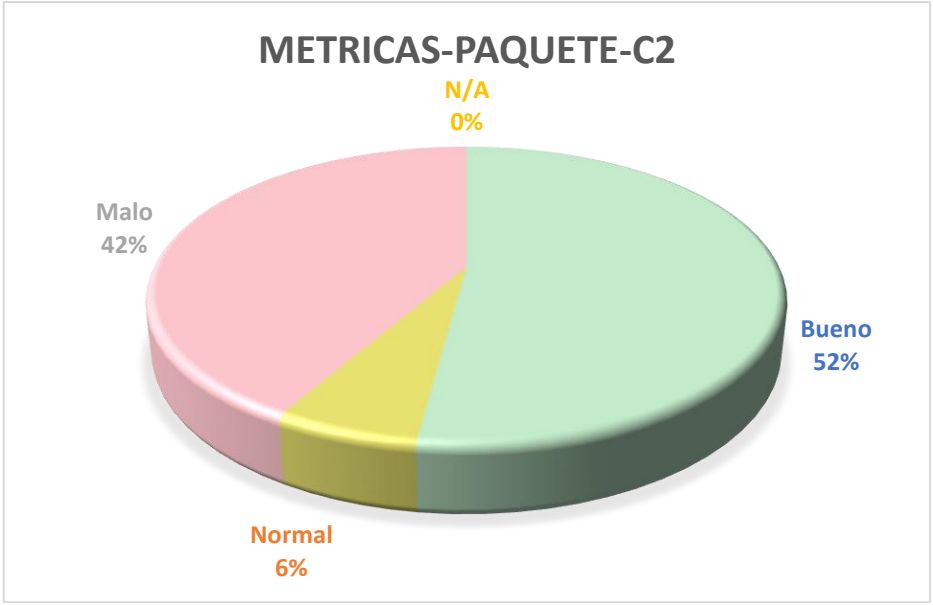


Ilustración 16. Gráfica de Resultados por Paquete del C2

La *Ilustración 17*,**¡Error! No se encuentra el origen de la referencia.** la gráfica hace referencia a los valores obtenidos en 28 clases java analizadas, en un 79 % respetan los umbrales utilizados en el modelo.

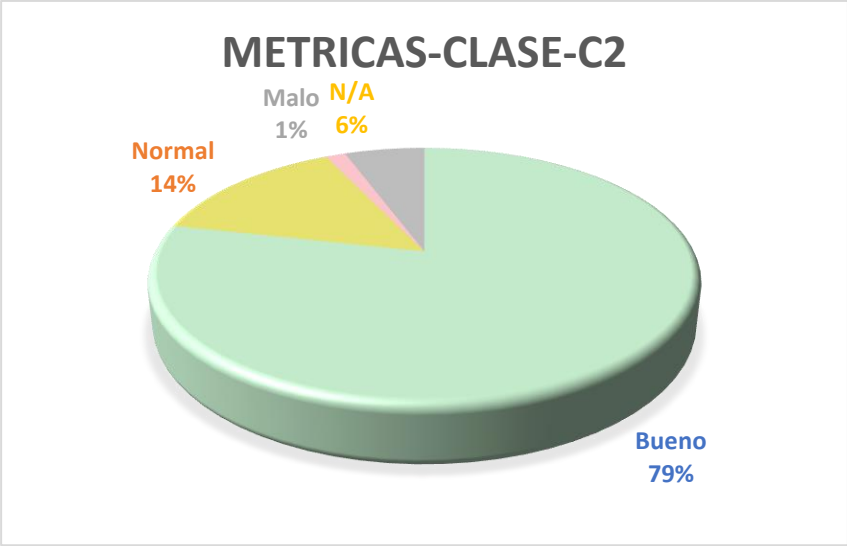


Ilustración 17. Gráfica de Resultados por Clase del C2

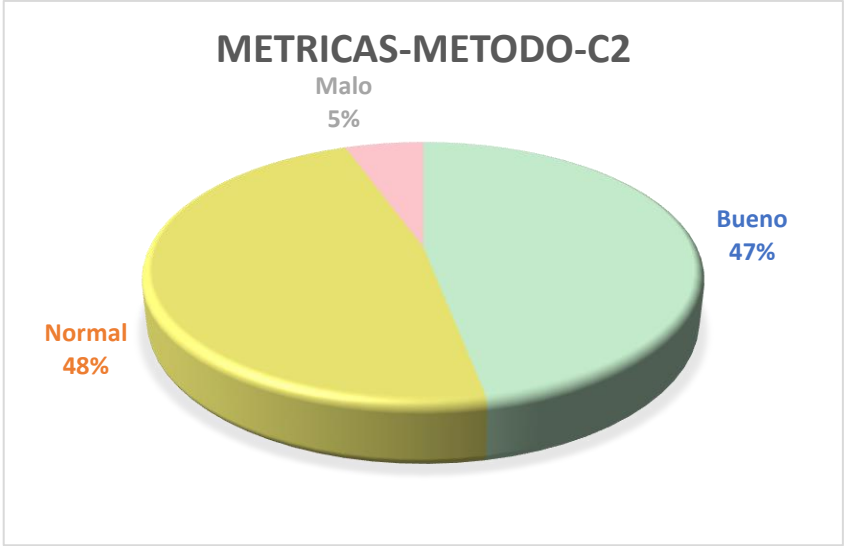


Ilustración 18. Gráfica de Resultados por Método del C2

La siguiente gráfica *Ilustración 18* muestra como 48% de los métodos revisados caen la clasificación "Normal", superando por 1% al umbral "Bueno" que sería lo esperado, es decir que si bien no se realizan mejoras al código (mantenimiento) para subir un nivel de mejora en las métricas: LOC, NEST, NPAR, V(G) y sería importante mantener estable y evitar caer al umbral "Malo".

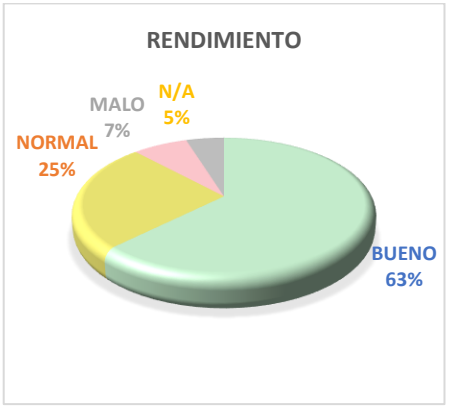


Ilustración 19,

Ilustración 20, *Ilustración 21*).

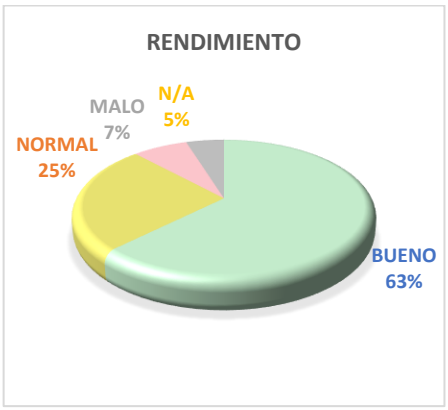


Ilustración 19. Atributo Rendimiento

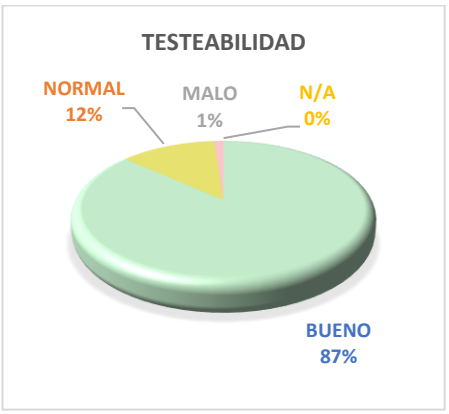


Ilustración 20. Sub-atributo Testeabilidad

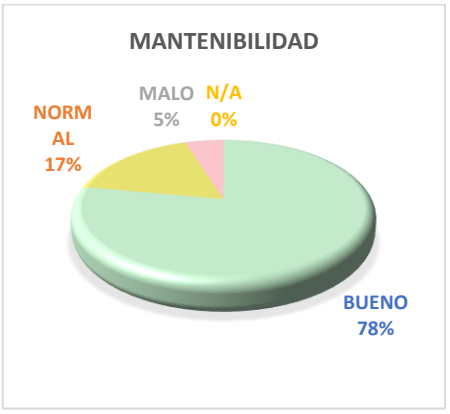


Ilustración 21. Atributo Mantenibilidad

Es importante mencionar que el modelo propuesto aborda la tolerancia a fallas a través de los atributos disponibilidad, rendimiento, mantenibilidad y el sub-atributo testeabilidad, donde de todos casos de estudio evaluados en este trabajo, se excluyó el atributo disponibilidad, ya que las métricas que lo conforman (MTTF, MTTR, MTBF) están enfocadas a mediciones en tiempo de ejecución o de lo contrario se requiere de documentación que refleje el control en relación a las fallas que presenta el sistema, es decir, tiempo de duración de la falla, frecuencia y el tiempo de reparación de estas.

A continuación, la *Tabla 14* presenta los resultados obtenidos en la evaluación del caso de estudio: Twitter Analyser.

5.4 Resultados Caso de estudio 3- Twitter Analyser, Mehrotra [119]

Tabla 14. Resultados a detalle de caso de estudio #3

DETALLES		MÉTRICAS		
No. Paquetes: 1		PAQUETE	CLASE	MÉTODO
No. Clases: 6		A, CA, CE, DN, I	AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO	LOC, NEST, NPAR, V(G) (4)
No. Métodos: 20		MHF (6)	NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)	
Líneas de código totales: 212		RESULTADOS DE MÉTRICAS EVALUADAS		
		CODIGO ESTATICO		EJECUCIÓN

ID	ELEMENTOS REVISADOS:	A	AHF	CA	CBO	CE	DAC	DIT	Dn/D	I	LCOM	LOC	MHF	MPC	NMO-NORM	NEST	NA/NOF	NOC	NOM	NOPA	NOPM	NPAR	RFC	V(G)	WMC	ME	MTTF	MTTR	MTBT
1	storm	0		0		5			0.0	1.0		212	0.9												44				
2	DriverTopology		N/A		11		4	1			0.0	33		2	0		0	0	0	0	1		17		6				
3	main()				11							32				3						1		6		1			
4	FileWriterBolt		0.3		4		0	3			0.9	37		0	0		8	0	5	0	4		17		9				
5	cleanup				0							4				1						0		1		0			
6	declareOutputFields				1							2				1						1		1		0			
7	execute				2							7				2						1		2		0			
8	FileWriterBolt				0							3				1						1		1		0	NO APLICA		
9	prepare				2							12				2						3		4		1			
10	LossyCountBolt		0.0		5		1	3			1.0	54		0	0		8	0	6	0	6		19		11				
11	declareOutputFields				1							2				1						1		1		0			
12	execute				2							9				2						1		2		0			
13	lossy_counting				1							19				3						1		4		0			
14	plain_counting				0							7				2						1		2		0			
15	prepare				2							3				1						3		1		0			

Métricas más utilizadas [29] — Bueno/Común→ — Normal/Casual→ — Malo/Poco común→

Continuación, … Tabla 14. Resultados a detalle de caso de estudio #3

DETALLES		MÉTRICAS																												
No. Paquetes: 1 No. Clases: 6 No. Métodos: 20 Líneas de código totales: 212		PAQUETE A, CA, CE, DN, I MHF (6)								CLASE AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)							MÉTODO LOC, NEST, NPAR, V(G) (4)																
		RESULTADOS DE MÉTRICAS EVALUADAS																												
		CODIGO ESTATICO																											EJECUCIÓN	
ID	ELEMENTOS REVISADOS:	A	AHF	CA	CBO	CE	DAC	DIT	Dn/D	I	LCOM	LOC	MHF	MPC	NMO-NORM	NEST	NA/NOF	NOC	NOM	NOPA	NOPM	NPAR	RFC	V(G)	WMC	ME	MTTF	MTTR	MTBT
16	prune				0							5				1						0		1		0				
17	LossyCountObject		1.0		0		0	1			0.0	5		0	0		4	0	0	4	0		0		0					
18	TweetExtractionBolt		0.0		8		0	3			1.0	15		0	0		1	0	3	0	3		10		4					
19	declareOutputFields				2							3				1						1		1		0				
20	execute				5							6				2						1		2		0				
21	prepare ( )				2							3				1						3		1		0				
22	TweetSpout		0.0		13		0	3			0.74	68		0	0		4	0	5	0	11		20		8					
23	close( )				1							4				1						0		1		0				
24	declareOutputFields ( )				2							3				1						1		1		0				
25	nextTuple( )				4							10				3						0		3		1				
26	open ( )				7							35				3						3		2		1				
27	TweetSpout ( )				1							6				1						1		1		0				
	Métricas más utilizadas [29]		Bueno/Común→									Normal/Casual→										Malo/Poco común→								

NO APLICA

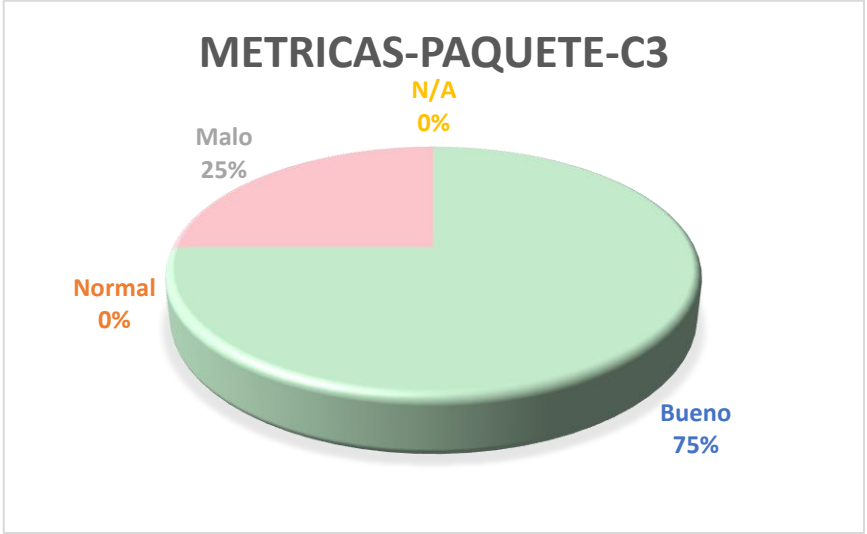
El proyecto está conformado por 1 paquete, 6 clases java y un total de 20 métodos es importante precisar que de los 4 casos de estudio considerados para probar el modelo, este es el más pequeño. Sin embargo, no está exento de presentar valores indeseables en las métricas revisadas. En seguida (ver *Tabla 15*) se presenta la cantidad de umbrales buenos, normales y malos que se identificaron para cada métrica.

5.4.1 Análisis resultados-Caso de estudio 3

Tabla 15. Resultados Caso de Estudio 3

CASO DE ESTUDIO #3: Twitter Analyser, Mehrotra [119]						
Tipo	**Métrica**	**Elementos Umbral**				**Observaciones**
		B	N	M	N/A	
Paquete(s) **(1)**	A	0	0	1	0	Las métricas afectadas son 2: Abstracción (A) e Inestabilidad (I)
	CA	1	0	0	0	
	CE	1	0	0	0	
	DN	1	0	0	0	
	I	0	0	1	0	
	MHF	1	0	0	0	
Clase(s) **(6)**	AHF	1	0	1	1	Las métricas afectadas son 4: Factor de Ocultamiento de Atributos (AHF), Acoplamiento entre Clases de Objetos (CBO), Falta de Cohesión en los Métodos (LCOM), y Numero de Métodos Públicos (NOPM). Para el caso en que la métrica AHF tiene como valor N/A, se debe a que dicha clase no posee atributos públicos que puedan ser contemplados para su evaluación.
	CBO	1	3	2	0	
	DAC	5	1	0	0	
	DIT	2	4	0	0	
	LCOM	2	0	2	0	
	LOC	2	4	0	0	
	MPC	6	0	0	0	
	NMO	6	0	0	0	
	NA	2	4	0	0	
	NOC	6	0	0	0	
	NOM	6	0	0	0	
	NOPA	5	1	0	0	
	NOPM	2	3	1	0	
	RFC	5	1	0	0	
	WMC	6	0	0	0	
Método(s) **(20)**	LOC	16	4	0	0	La única métrica afectada es: Complejidad Ciclomática V(G).
	NEST	11	9	0	0	
	NPAR	16	4	0	0	
	V(G)	16	3	1	0	

Como resultado de la evaluación de las 6 métricas relacionadas a medir la calidad de paquetes, para este caso se identificó que el paquete (*lustración* 22) cumple un 75% con los umbrales especificados como deseables en un software.



lustración 22. Gráfica de Resultados por Paquete del C2

La gráfica, *Ilustración 23*, detalla los resultados obtenidos en la evaluación de las 6 clases del proyecto Big Data, donde el 68 % corresponde al umbral “Bueno” y solo el 7% “Malo”.

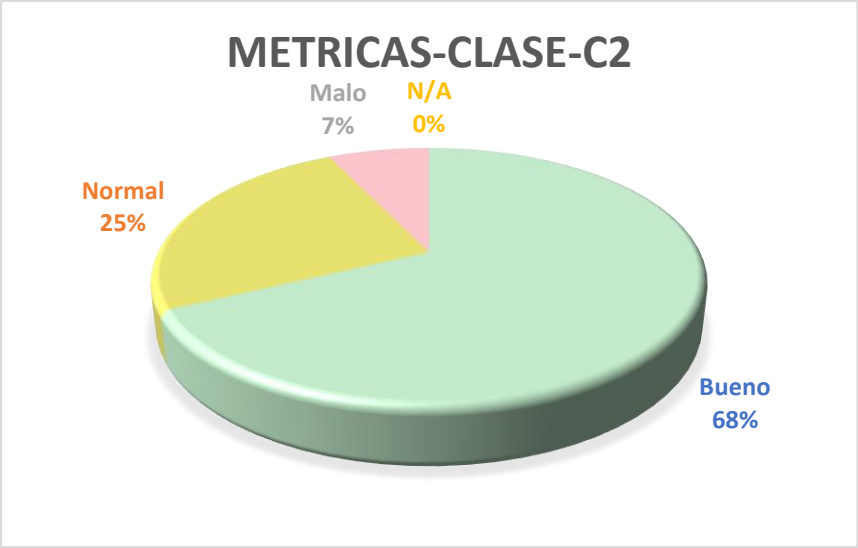


Ilustración 23. Gráfica de Resultados por Clase del C3

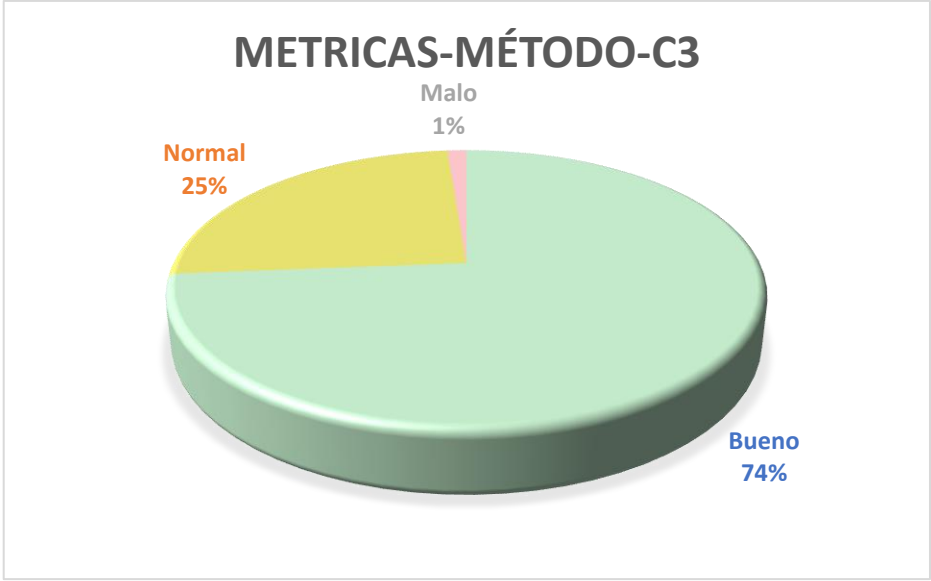


Ilustración 24. Gráfica de Resultados por Método del C3

A continuación, se muestra la *Ilustración 24* donde se entiende que solo el 1% de los 20 métodos implementados pertenece al umbral "Malo".

Por lo tanto, son los elementos tipo método para este caso de estudio, los que menos problemas presentan en los valores obtenidos en sus respectivas métricas involucradas durante la evaluación del código estático del proyecto.

En la *Ilustración 26* se puede observar que el al sub-atributo testeabilidad posee mayor vulnerabilidad, está conformado por 6 métricas: CBO, DIT, LCOM, NMO, NOM, WMC y al formar parte del atributo mantenibilidad afecta dicha propiedad. Sin embargo, la diferencia de mantenibilidad (*Ilustración 27*) con el atributo de rendimiento (*Ilustración 25*) es mínima 1%, por lo que sería importante abordar ambas propiedades.

Es importante destacar que este proyecto, a diferencia del resto, se conforma de un número de elementos reducido estructuralmente por lo que sería necesario prestar atención a la forma en que fue desarrollado o las prácticas implementadas, se esperaría que fuese el mejor evaluado debido al tamaño que posee.

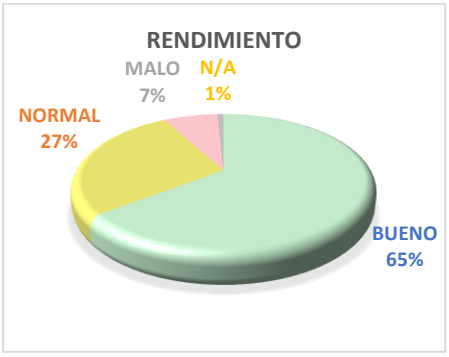


Ilustración 25. Atributo Rendimiento

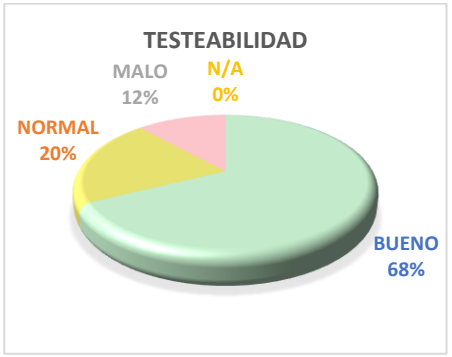


Ilustración 26. Sub-atributo Testeabilidad

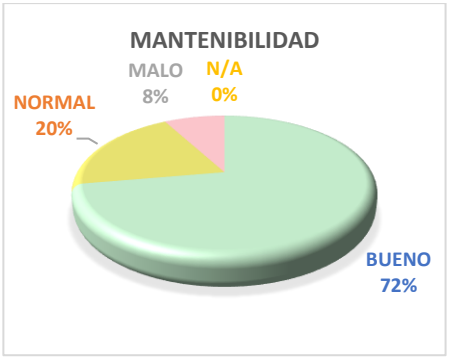


Ilustración 27. Atributo Mantenibilidad

Finalmente, como último caso de estudio utilizado para probar el modelo de calidad propuesto se presentan resultados del proyecto Big Data: CRIMEGRAPH, Marciani *et.al* [121], donde fueron analizados 272 elementos, 17 paquetes, 52 clases y 203 métodos. Cabe señalar que fue el proyecto más extenso (2,203 líneas de código) y sus detalles se presentan a continuación en la *Tabla 16*.

5.5 Resultados Caso de estudio 4- CRIMEGRAPH, Marciani *et.al* [121]

Tabla 16.Resultados a detalle de caso de estudio #4

DETALLES		MÉTRICAS		
No. Paquetes: 17 **No. Clases: 52** **No. Métodos: 203** **Líneas de código totales: 2203**		**PAQUETE** **A, CA, CE, DN, I** **MHF (6)**	**CLASE** **AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO** **NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)**	**MÉTODO** **LOC, NEST, NPAR, V(G) (4)**
		RESULTADOS DE MÉTRICAS EVALUADAS		
		CODIGO ESTATICO		EJECUCIÓN

ID	ELEMENTOS REVISADOS:	A	AHF	CA	CBO	CE	DAC	DIT	Dn/D	I	LCOM	LOC	MHF	MPC	NMO-NORM	NEST	NA/NOF	NOC	NOM	NOPA	NOPM	NPAR	RFC	V(G)	WMC	ME	MTTF	MTTR	MTBT
1	crimegraph	0		0		2			0.0	1.0		60	1.0												6				
2	CrimegraphTopology		N/A		20		21	1			0.0	32		11	0		0	0	0	0	1		33		3					
3	main( )				20							31				2						1		3		1			
4	CrimegraphTopologyMultiIndex		N/A		16		18	1			0.0	28		10	0		0	0	0	0	1		30		3					
5	main( )				16							27				2						1		3		1			
6	config	0		4		2			0.7	0.3		125	1.0												16				
7	AppConfiguration		0.0		6		5	1			0.5	70		2	0		15	0	3	15	3		3		3				
8	AppConfiguration ()				0							2				1						1		1		1			
9	copy()				0							16				1						1		1		1			
10	toDefault( )				0							16				1						0		1		1			
11	AppConfigurationService		0.0		7		3	1			0.8	46		9	0		0	0	0	1	11		23		13				
12	fromDefault( )				1							2				1						0		1		0			
13	fromJson()				3							3				1						1		1		1			
14	fromJsonResource()				3							6				2						1		1		1			
15	fromYaml()				3							3				1						1		1		1			
16	fromYamlResource()				3							6				2						1		1		1			

NO APLICA

Métricas más utilizadas [29] **Bueno/Común→** **Normal/Casual→** **Malo/Poco común→**

Continuación, … Tabla 16. Resultados a detalle de caso de estudio #4

DETALLES		MÉTRICAS																												
No. Paquetes: 17 No. Clases: 52 No. Métodos: 203 Líneas de código totales: 2203		PAQUETE A, CA, CE, DN, I MHF (6)									CLASE AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)					MÉTODO LOC, NEST, NPAR, V(G) (4)																	
		RESULTADOS DE MÉTRICAS EVALUADAS																												
		CODIGO ESTATICO																										EJECUCIÓN		
ID	ELEMENTOS REVISADOS:	A	AHF	CA	CBO	CE	DAC	DIT	Dn/D	I	LCOM	LOC	MHF	MPC	NMO-NORM	NEST	NA/NOF	NOC	NOM	NOPA	NOPM	NPAR	RFC	V(G)	WMC	ME	MTTF	MTTR	MTBT
17	loadDefault( )				1							2				1						0		1		0				
18	loadJson()				1							3				1						1		1		1				
19	loadJsonResource()				2							4				1						1		1		1				
20	loadYaml()				1							3				1						1		1		1				
21	loadYamlResource()				2							4				1						1		1		1				
22	getConfigurations()				1							6				4						0		3		0				
23	AppManifest		N/A		0		0	1			0.0	9		0	0		0	0	0	5	0		0		0					
24	serial	0		3		4			0.4	0.6		129	0.8												33				
25	AppConfigurationDeserializer		0.0		8		4	3			0.0	78		1	0		0	0	2	1	2		15		25					
26	AppConfigurationDeserializer()				0							2				1						0		1		0				
27	deserialize()				8							70				3						2		22		1				
28	getInstance()				0							4				2						0		2		0				
29	AppConfigurationJsonMapper		N/A		3		3	4			0.0	8		2	0		0	0	1	0	1		6		1					
30	AppConfigurationJsonMapper()				3							6				1						0		1		0				
31	AppConfigurationSerializer		0.0		4		3	3			0.0	35		0	0		0	0	2	0	2		7		6					
32	AppConfigurationSerializer( )				0							2				1						0		1		0				
33	serialize()				4							27				1						3		3		1				
34	getInstance( )				0							4				2						0		2		0				
35	AppConfigurationYamlMapper		N/A		3		3	5			0.0	8		2	0		0	0	1	0	1		6		1					
	Métricas más utilizadas [29]		Bueno/Común→									Normal/Casual→										Malo/Poco común→								

NO APLICA

Continuación, … Tabla 16. Resultados a detalle de caso de estudio #4

DETALLES		MÉTRICAS		
No. Paquetes: 17 **No. Clases: 52** **No. Métodos: 203** **Líneas de código totales: 2203**		**PAQUETE** **A, CA, CE, DN, I** **MHF (6)**	**CLASE** **AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO** **NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)**	**MÉTODO** **LOC, NEST, NPAR, V(G) (4)**
		RESULTADOS DE MÉTRICAS EVALUADAS		
		CODIGO ESTATICO		EJECUCIÓN

ID	ELEMENTOS REVISADOS:	A	AHF	CA	CBO	CE	DAC	DIT	Dn/D	I	LCOM	LOC	MHF	MPC	NMO-NORM	NEST	NA/NOF	NOC	NOM	NOPA	NOPM	NPAR	RFC	V(G)	WMC	ME	MTTF	MTTR	MTBT
36	AppConfigurationYamlMapper( )				3							6				1						0		1		0			
37	core	0		0		1			0.0	1.0		5	1.0												1				
38	CoreController		N/A		1		0	1			0.0	5		0	0		0	0	0	0	1		3		1				
39	splash( )				1							3				1						1		1		0			
40	db	0		15		1			0.9	0.1		532	1.0												58				
41	DbConfiguration		0.5		0		0	1			0.0	9		0	0		3	0	0	0	0		0		0				
42	Neo4JManager		N/A		19		3	1			0.0	379		0	0		0	0	0	0	35		64		58				
43	adamicAdar( )				6							10				2						3		2		0			
44	checkExtremes( )				6							11				2						3		2		0			
45	close( )				2							3				1						2		1		0	NO APLICA		
46	commonNeighbours( )				5							9				2						3		2		0			
47	commonNeighboursWithDegree( )				6							10				2						3		2		0			
48	commonNeighboursWithDegreeWithinDistance( )				6							11				2						4		2		0			
49	commonNeighboursWithinDistance( )				5							10				2						4		2		0			
50	countCommonNeighbours()				5							6				1						3		1		0			
51	empyting( )				4							8				1						1		1		0			
52	empyting( )				2							2				1						1		1		0			
53	gammaIntersection( )				6							12				2						3		2		0			
54	HDI( )				5							14				1						3		1		0			
55	hIntersection()				6							11				2						4		2		0			

Métricas más utilizadas [29] **Bueno/común→** **Normal/Casual→** **Malo/poco común→**

Continuación, … Tabla 16. Resultados a detalle de caso de estudio #4

DETALLES		MÉTRICAS		
No. Paquetes: 17	**PAQUETE**	**CLASE**	**MÉTODO**
No. Clases: 52	**A, CA, CE, DN, I**	**AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO**	**LOC, NEST, NPAR, V(G) (4)**
No. Métodos: 203	**MHF (6)**	**NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)**	
Líneas de código totales: 2203	**RESULTADOS DE MÉTRICAS EVALUADAS**		

		CODIGO ESTATICO																							EJECUCIÓN		
ID	ELEMENTOS REVISADOS:	A	AHF	CA	CBO	CE	DAC	DIT	Dn/D	I	LCOM	LOC	MHF	MPC	NMO-NORM	NEST	NA/NOF	NOC	NOM	NOPA	NOPM	NPAR	RFC	V(G)	WMC	ME	MTTF	MTTR	MTBT
56	HPI( )				5							14				1						3		1		0			
57	jaccard( )				5							9				1						3		1		0			
58	LHN1( )				5							14				1						3		1		0			
59	multiIndexTool( )				6							17				1						3		1		0			
60	neighbours( )				5							9				2						2		2		0			
61	neighboursWithDegree()				6							10				2						2		2		0			
62	neighboursWithDegreeWithinDistance( )				6							11				2						3		1		0			
63	neighboursWithinDistance( )				5							10				2						3		2		0			
64	open( )				8							7				1						1		1		0			
65	open( )				7							6				1						3		1		0	NO APLICA		
66	pairsToUpdate()				6							16				3						2		3		0			
67	pairsToUpdateTwice( )				6							16				3						3		3		0			
68	pairsToUpdateTwiceWithinDistance( )				6							19				3						4		3		0			
69	pairsToUpdateWithinDistance()				6							18				3						3		3		0			
70	preferentialAttachment()				5							10				1						3		1		0			
71	remove()				8							7				2						2		1	1	0			
72	remove( )				5							3				1						4		1		0			
73	resourceAllocation()				6							10				2						3		2		0			
74	salton()				5							14				1						3		1		0			
75	save()				7							11				2						3		2	1	0			

Métricas más utilizadas [29] **Bueno/común→** **Normal/Casual→** **Malo/poco común→**

Continuación, … Tabla 16. Resultados a detalle de caso de estudio #4

DETALLES		MÉTRICAS		
No. Paquetes: 17	**PAQUETE**	**CLASE**	**MÉTODO**	
No. Clases: 52	**A, CA, CE, DN, I**	**AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO**	**LOC, NEST, NPAR, V(G) (4)**	
No. Métodos: 203	**MHF (6)**	**NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)**		
Líneas de código totales: 2203	**RESULTADOS DE MÉTRICAS EVALUADAS**			
	CODIGO ESTATICO			EJECUCIÓN

ID	ELEMENTOS REVISADOS:	A	AHF	CA	CBO	CE	DAC	DIT	Dn/D	I	LCOM	LOC	MHF	MPC	NMO-NORM	NEST	NA/NOF	NOC	NOM	NOPA	NOPM	NPAR	RFC	V(G)	WMC	ME	MTTF	MTTR	MTBT
76	save()				8							15				3						2		2		0			
77	sorensen( )				5							14				1						3		1		0			
78	Neo4JQueries		N/A		0		0	1			0.0	144		0	0		0	0	0	23	0		0		0				
79	filter	0		0		2			0.0	1.0		16	1.0												4				
80	HiddenFilter		0.5		2		1	1			0.0	8		0	0		1	0	2	0	2		2		2				
81	filter()				2							3				1						1		1		1			
82	HiddenFilter( )				0							2				1						1		1		0			
83	PotentialFilter		0.5		2		1	1			0.0	8		0	0		1	0	2	0	2		2		2				
84	filter()				2							3				1						1		1		1			
85	PotentialFilter()				0							2				1						1		1		0			
86	keyer	0		2		2			0.5	0.5		8	1.0												2		NO APLICA		
87	NodePairScoreKeyer		N/A		3		1	1			0.0	4		0	0		1	0	1	0	1		1		1				
88	getKey()				3							3				1						1		1		1			
89	NodePairScoresKeyer		N/A		3		1	1			0.0	4		0	0		0	0	1	0	1		1		1				
90	getKey()				3							3				1						1		1		1			
91	metric	0		2		0			1.0	0.0		16	1.0												2				
92	HiddenMetrics		0.0		0		N/A	2			0.0	8		N/A	0		1	0	1	3	1		1		1				
93	HiddenMetrics()				0							2				1						1		1		0			
94	PotentialMetrics		0.0		0		N/A	2			0.0	8		N/A	0		1	0	1	3	1		1		1				
95	PotentialMetrics()				0							2				1						1		1		0			

Métricas más utilizadas [29] **Bueno/común→** **Normal/Casual→** **Malo/poco común→**

Continuación, … Tabla 16. Resultados a detalle de caso de estudio #4

DETALLES		MÉTRICAS		
No. Paquetes: 17 **No. Clases: 52** **No. Métodos: 203** **Líneas de código totales: 2203**		**PAQUETE** **A, CA, CE, DN, I** **MHF (6)**	**CLASE** **AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO** **NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)**	**MÉTODO** **LOC, NEST, NPAR, V(G) (4)**
		RESULTADOS DE MÉTRICAS EVALUADAS		
		CODIGO ESTATICO		EJECUCIÓN

ID	ELEMENTOS REVISADOS:	A	AHF	CA	CBO	CE	DAC	DIT	Dn/D	I	LCOM	LOC	MHF	MPC	NMO-NORM	NEST	NA/NOF	NOC	NOM	NOPA	NOPM	NPAR	RFC	V(G)	WMC	ME	MTTF	MTTR	MTBT
96	operator	0		2		8			0.2	0.8		434	1.0												96				
97	GraphUpdate		0.5		12		5	3			0.6	44		12	2		3	0	4	0	4		34		18				
98	close( )				1							3				1						0		1		1				
99	flatMap( )				8							31				3						2		15		0				
100	GraphUpdate()				1							2				1						1		1		0				
101	open( )				3							4				1						1		1		1				
102	GraphUpdateMultiIndex		0.5		16		5	3			0.7	60		11	2		5	0	4	0	4		38		18				
103	close()				1							3				1						0		1		1				
104	flatMap( )				13							42				3						2		15		0				
105	GraphUpdateMultiIndex()				1							2				1						1		1		0				
106	open()				3							6				1						1		1		1				
107	ScoreCalculator		0.5		12		6	3			0.6	45		5	2		3	0	4	0	4		24		10				
108	close()				1							3				1						0		1		1				
109	flatMap( )				8							32				3						2		7		0				
110	open()				3							4				1						1		1		1				
111	ScoreCalculator()				1							2				1						1		1		0				
112	ScoreCalculatorMultiIndex		0.5		12		6	3			0.5	90		18	2		2	0	4	0	4		27		9				
113	close()				1							3				1						0		1		1				
114	flatMap()				10							79				3						2		6		0				
115	open( )				2							3				1						1		1		1				

NO APLICA

Métricas más utilizadas [29] **Bueno/común→** **Normal/Casual→** **Malo/poco común→**

Continuación, … Tabla 16. Resultados a detalle de caso de estudio #4

DETALLES		MÉTRICAS		
No. Paquetes: 17		**PAQUETE**	**CLASE**	**MÉTODO**
No. Clases: 52		**A, CA, CE, DN, I**	**AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO**	**LOC, NEST, NPAR, V(G) (4)**
No. Métodos: 203		**MHF (6)**	**NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)**	
Líneas de código totales: 2203		**RESULTADOS DE MÉTRICAS EVALUADAS**		
		CODIGO ESTATICO		EJECUCIÓN

ID	ELEMENTOS REVISADOS:	A	AHF	CA	CBO	CE	DAC	DIT	Dn/D	I	LCOM	LOC	MHF	MPC	NMO-NORM	NEST	NA/NOF	NOC	NOM	NOPA	NOPM	NPAR	RFC	V(G)	WMC	ME	MTTF	MTTR	MTBT
116	ScoreCalculatorMultiIndex( )				1							2				1						1		1		0			
117	ScoreCalculatorMultiIndex2		0.5		16		6	3			0.8	93		5	2		4	0	5	0	4		36		17					
118	close( )				1							3				1						0		1		1			
119	flatMap( )				12							65				3						2		9		0			
120	isMalformed( )				4							5				2						2		5		0			
121	open()				3							5				1						1		1		1			
122	ScoreCalculatorMultiIndex2( )				1							2				1						1		1		0			
123	ScoreCalculatorTSteps		0.5		13		6	3			0.6	45		6	2		4	0	4	0	4		27		11					
124	close()				1							3				1						0		1		1			
125	flatMap()				9							26				4						2		8		0			
126	open()				3							7				1						1		1		1			
127	ScoreCalculatorTSteps()				1							3				1						2		1		0			
128	ScoreCalculatorTStepsWithWeights		0.4		13		6	3			0.6	47		6	2		5	0	4	0	4		28		11					
129	close( )				1							3				1						0		1		1			
130	flatMap()				9							26				4						2		8		0			
131	open( )				3							7				1						1		1		1			
132	ScoreCalculatorTStepsWithWeights()				1							4				1						3		1		0			
133	ScoreSplitter		N/A		3		2	1			0.0	10		0	0		0	0	1	0	1		4		2					
134	select()				3							9				2						1		2		0			
135	producer	0		0		1			0.0	1.0		19	1.0												3				
	Métricas más utilizadas [29]	**Bueno/común→**									**Normal/Casual→**											**Malo/poco común→**							

NO APLICA

Continuación, … Tabla 16. Resultados a detalle de caso de estudio #4

DETALLES		MÉTRICAS																												
No. Paquetes: 17 **No. Clases: 52** **No. Métodos: 203** **Líneas de código totales: 2203**		**PAQUETE** **A, CA, CE, DN, I** **MHF (6)**									**CLASE** **AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO** **NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)**					**MÉTODO** **LOC, NEST, NPAR, V(G) (4)**															
		RESULTADOS DE MÉTRICAS EVALUADAS																												
		CODIGO ESTATICO																										EJECUCIÓN		
ID	**ELEMENTOS REVISADOS:**	**A**	**AHF**	**CA**	**CBO**	**CE**	**DAC**	**DIT**	**Dn/D**	**I**	**LCOM**	**LOC**	**MHF**	**MPC**	**NMO-NORM**	**NEST**	**NA/NOF**	**NOC**	**NOM**	**NOPA**	**NOPM**	**NPAR**	**RFC**	**V(G)**	**WMC**	**ME**	**MTTF**	**MTTR**	**MTBT**
136	StringKafkaProducer		0.0		4		1	1			0.0	19		1	0		1	0	3	0	3		9		3				
137	close()				1							2				1						0		1		0			
138	send()				3							4				1						2		1		0			
139	StringKafkaProducer( )				1							11				1						1		1		0			
140	sink	0		2		6			0.3	0.8		149	1.0												30				
141	HiddenSink		0.5		9		5	3			0.6	25		5	2		4	0	4	0	4		25		5				
142	close()				1							3				1						0		1		1			
143	HiddenSink( )				1							3				1						2		1		0			
144	invoke( )				5							10				2						1		2		1			
145	open()				3							4				1						1		1		1			
146	LinkSink		0.5		5		2	3			0.6	21		3	2		5	0	4	0	4		22		4		NO APLICA		
147	close()				1							3				1						0		1		1			
148	invoke( )				2							3				1						1		1		1			
149	LinkSink()				0							4				1						3		1		0			
150	open( )				3							4				1						1		1		1			
151	MultiIndexSink		0.5		10		5	3			0.6	26		4	2		3	0	4	0	4		23		5				
152	close( )				1							3				1						0		1		1			
153	invoke( )				8							13				2						1		2		1			
154	MultiIndexSink( )				1							3				1						2		1		0			
155	open()				2							3				1						1		1		1			

Métricas más utilizadas [29] **Bueno/común→** **Normal/Casual→** **Malo/poco común→**

Continuación, … Tabla 16. Resultados a detalle de caso de estudio #4

DETALLES		MÉTRICAS																														
No. Paquetes: 17 **No. Clases: 52** **No. Métodos: 203** **Líneas de código totales: 2203**		**PAQUETE** **A, CA, CE, DN, I** **MHF (6)**									**CLASE** **AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO** **NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)**										**MÉTODO** **LOC, NEST, NPAR, V(G) (4)**										
		RESULTADOS DE MÉTRICAS EVALUADAS																														
		CODIGO ESTATICO																												EJECUCIÓN		
ID	**ELEMENTOS REVISADOS:**	**A**	**AHF**	**CA**	**CBO**	**CE**	**DAC**	**DIT**	**Dn/D**	**I**	**LCOM**	**LOC**	**MHF**	**MPC**	**NMO-NORM**	**NEST**	**NA/NOF**	**NOC**	**NOM**	**NOPA**	**NOPM**	**NPAR**	**RFC**	**V(G)**	**WMC**	**ME**	**MTTF**	**MTTR**	**MTBT**
156	MultiIndexSink2		0.5		14		6	3			0.7	33		3	2		4	0	4	0	4		29		6				
157	close( )				1							3				1						0		1		1				
158	invoke( )				11							17				3						1		3		1				
159	MultiIndexSink2()				1							2				1						1		1		0				
160	open()				3							5				1						1		1		1				
161	PotentialSink		0.5		9		5	3			0.6	25		5	2		4	0	4	0	4		25		5						
162	close( )				1							3				1						0		1		1				
163	invoke()				5							10				2						1		2		1				
164	open()				3							4				1						1		1		1				
165	PotentialSink()				1							3				1						2		1		0				
166	ToStringSink		0.5		1		0	3			0.5	19		0	2		2	0	4	0	4		14		5						
167	close()				0							4				1						0		1		1				
168	invoke()				0							4				1						1		1		1				
169	open( )				1							6				2						1		2		1				
170	ToStringSink()				0							2				1						1		1		1				
171	source	0		5		3			0.6	0.5		71	0.8												17						
172	KafkaProperties		N/A		0		0	4			0.7	19		0	0		0	0	5	0	5		6		5						
173	KafkaProperties()				0							2				1						0		1		0				
174	KafkaProperties()				0							5				1						3		1		0				
175	setBootstrapServers()				0							2				1						1		1		0				
	Métricas más utilizadas [29]	**Bueno/Común→**									**Normal/Casual→**											**Malo/Poco común→**									

NO APLICA

Continuación, … Tabla 16. Resultados a detalle de caso de estudio #4

DETALLES		MÉTRICAS		
No. Paquetes: 17 **No. Clases: 52** **No. Métodos: 203** **Líneas de código totales: 2203**		**PAQUETE** **A, CA, CE, DN, I** **MHF (6)**	**CLASE** **AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO** **NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)**	**MÉTODO** **LOC, NEST, NPAR, V(G) (4)**

RESULTADOS DE MÉTRICAS EVALUADAS

		CODIGO ESTATICO																						EJECUCIÓN		
ID	ELEMENTOS REVISADOS:	A	AHF	CA	CBO	CE	DAC	DIT	Dn/D	I	LCOM	LOC	MHF	MPC	NMO-NORM	NEST	NA/NOF	NOC	NOM	NOPA	NOPM	NPAR	RFC	V(G)	WMC	ME	MTTF	MTTR	MTBT
176	setGroupId()				0							2				1						1		1		0			
177	setZookeeperConnect()				0							2				1						1		1		0			
178	LinkDeserializationSchema		N/A		2		1	2			0.0	12		1	0		0	0	1	0	1		8		2				
179	deserialize( )				2							10				2						1		2		1			
180	LinkKafkaSource		N/A		1		2	6			0.0	3		0	0		0	0	1	0	1		1		1				
181	LinkKafkaSource()				1							2				1						2		1		0			
182	LinkSource		0.2		3		1	3			0.6	30		1	0		3	0	3	0	3		18		8				
183	cancel( )				1							10				3						0		3		1			
184	LinkSource( )				0							2				1						1		1		0			
185	run( )				3							13				3						1		4		1			
186	SourceType		0.0		0		N/A	2			0.0	7		N/A	0		1	0	1	2	1		1		1				
187	SourceType( )				0							2				1						1		1		0			
188	tuple	0		23		4			0.9	0.1		145	0.7												33				
189	Link		N/A		1		1	3			1.5	28		0	0		0	0	4	0	5		12		9				
190	Link( )				0							1				1						0		1		0			
191	Link( )				1							2				1						4		1		0			
192	Link( )				1							2				1						3		1		0			
193	valueOf()				1							11				1						1		4		1			
194	toString( )				1							6				2						0		2		0			
195	LinkType		0.0		0		N/A	2			0.0	21		N/A	0		1	0	1	16	1		1		1				

NO APLICA

Métricas más utilizadas [29] **Bueno/Común→** **Normal/Casual→** **Malo/Poco común→**

Continuación, … Tabla 16. Resultados a detalle de caso de estudio #4

DETALLES		MÉTRICAS		
No. Paquetes: 17		**PAQUETE**	**CLASE**	**MÉTODO**
No. Clases: 52		**A, CA, CE, DN, I**	**AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO**	**LOC, NEST, NPAR, V(G) (4)**
No. Métodos: 203		**MHF (6)**	**NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)**	
Líneas de código totales: 2203		**RESULTADOS DE MÉTRICAS EVALUADAS**		
		CODIGO ESTATICO		EJECUCIÓN

ID	ELEMENTOS REVISADOS:	A	AHF	CA	CBO	CE	DAC	DIT	Dn/D	I	LCOM	LOC	MHF	MPC	NMO-NORM	NEST	NA/NOF	NOC	NOM	NOPA	NOPM	NPAR	RFC	V(G)	WMC	ME	MTTF	MTTR	MTBT
196	LinkType()				0							2				1						1		1		0			
197	NodePair		N/A		1		1	3			1.5	20		0	0		0	0	3	0	4		10		6				
198	NodePair( )				0							1				1						0		1		0			
199	NodePair( )				1							2				1						3		1		0			
200	valueOf()				1							8				1						1		3		1			
201	toString( )				1							3				1						0		1		0			
202	NodePairScore		N/A		1		3	3			1.5	21		0	0		0	0	3	0	4		10		6				
203	NodePairScore( )				0							1				1						0		1		0			
204	NodePairScore()				1							2				1						4		1		0			
205	valueOf( )				1							9				1						1		3		1	NO APLICA		
206	toString( )				0							3				1						0		1		0			
207	NodePairScores		N/A		1		2	3			1.3	25		0	0		0	0	5	0	6		12		9				
208	addScore( )				1							2				1						2		1		0			
209	NodePairScores( )				0							1				1						0		1		0			
210	NodePairScores( )				0							2				1						3		1		0			
211	NodePairScores()				0							2				1						2		1		0			
212	valueOf()				0							12				2						1		4		1			
213	toString( )				0							3				1						1		1		0			
214	ScoreType		0.0		0		N/A	2			0.0	20		N/A	0		1	0	1	15	1		1		1				
215	ScoreType()				0							2				1						1		1		0			

Métricas más utilizadas [29] **Bueno/Común→** **Normal/Casual→** **Malo/Poco común→**

Continuación, … Tabla 16. Resultados a detalle de caso de estudio #4

DETALLES		MÉTRICAS		
No. Paquetes: 17 **No. Clases: 52** **No. Métodos: 203** **Líneas de código totales: 2203**	**PAQUETE** **A, CA, CE, DN, I** **MHF (6)**	**CLASE** **AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO** **NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)**	**MÉTODO** **LOC, NEST, NPAR, V(G) (4)**	

		RESULTADOS DE MÉTRICAS EVALUADAS																											
		CODIGO ESTATICO																										EJECUCIÓN		
ID	**ELEMENTOS REVISADOS:**	**A**	**AHF**	**CA**	**CBO**	**CE**	**DAC**	**DIT**	**Dn/D**	**I**	**LCOM**	**LOC**	**MHF**	**MPC**	**NMO-NORM**	**NEST**	**NA/NOF**	**NOC**	**NOM**	**NOPA**	**NOPM**	**NPAR**	**RFC**	**V(G)**	**WMC**	**ME**	**MTTF**	**MTTR**	**MTBT**
216	UpdateType		0.0		0		N/A	2			0.0	10		N/A	0		1	0	1	5	1		1		1				
217	UpdateType()				0							2				1						1		1		0			
218	tool	0		0		1			0.0	1.0		22	0.5												6				
219	DatasetFormatter		N/A		2		1	1			0.0	22		2	0		0	0	0	0	2		29		6				
220	main()				1							13				2						1		3		1			
221	writeDataset()				2							7				3						2		3		1			
222	io	0		1		1			0.5	0.5		25	0.8												8				
223	IOManager		N/A		1		0	1			0.0	25		0	0		0	0	0	0	6		19		8				
224	appendResource()				0							3				1						2		1		1			
225	getInputStream()				0							7				2						1		3		1			
226	getOutputStream()				0							3				1						1		1		1			
227	isWritableResource()				0							3				1						1		1		0			
228	readResource()				1							5				2						1		1		1			
229	writeResource()				1							3				2						2		1		1			
230	runtime	0		0		2			0.0	1.0		31	0.8												8				
231	DbRelease		0.5		1		0	1			0.0	7		0	0		1	0	1	0	1		2		1				
232	run( )				1							3				1						0		1		0			
233	RuntimeManager		N/A		2		0	1			0.0	24		0	0		0	0	0	0	4		21		7				
234	getCores( )				0							2				1						0		1		0			
235	getJvmName( )				0							1				1						0		1		0			

Métricas más utilizadas [29] **Bueno/Común→** **Normal/Casual→** **Malo/Poco común→**

NO APLICA

Continuación, … Tabla 16. Resultados a detalle de caso de estudio #4

DETALLES		MÉTRICAS		
No. Paquetes: 17		**PAQUETE**	**CLASE**	**MÉTODO**
No. Clases: 52		**A, CA, CE, DN, I**	**AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO**	**LOC, NEST, NPAR, V(G) (4)**
No. Métodos: 203		**MHF (6)**	**NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)**	
Líneas de código totales: 2203		**RESULTADOS DE MÉTRICAS EVALUADAS**		
		CODIGO ESTATICO		EJECUCIÓN

ID	ELEMENTOS REVISADOS:	A	AHF	CA	CBO	CE	DAC	DIT	Dn/D	I	LCOM	LOC	MHF	MPC	NMO-NORM	NEST	NA/NOF	NOC	NOM	NOPA	NOPM	NPAR	RFC	V(G)	WMC	ME	MTTF	MTTR	MTBT
236	registerPeriodic()				0							8				2						5		2		0			
237	registerShutdownHooks()				1							5				2						1		2		0			
238	run( )				2							6				1						1		1		1			
239	ui	0		0		2			0.0	1.0		416	0.1												63				
240	BaseOptions		0.0		2		0	2			1.0	280		0	0		0	0	24	0	1		35		26				
241	BaseOptions( )				1							47				1						0		1		0			
242	optConfig( )				2							9				1						0		1		0			
243	optDataset( )				2							9				1						0		1		0			
244	optEwmaFactor( )				2							9				1						0		1		0			
245	optHelp( )				2							7				1						0		1		0	NO APLICA		
246	optHiddenLocality( )				2							9				1						0		1		0			
247	optHiddenMetric( )				2							9				1						0		1		0			
248	optHiddenThreshold( )				2							9				1						0		1		0			
249	optHiddenWeights( )				2							9				1						0		1		0			
250	optKafkaBootstrap( )				2							9				1						0		1		0			
251	optKafkaGroup( )				2							9				1						0		1		0			
252	optKafkaTopic( )				2							9				1						0		1		0			
253	optKafkaZookeper( )				2							9				1						0		1		0			
254	optNeo4jHostname( )				2							9				1						0		1		0			
255	optNeo4JPassword( )				2							9				1						0		1		0			
	Métricas más utilizadas [29]	**Bueno/Común→**										**Normal/Casual→**										**Malo/Poco común→**							

Continuación, … Tabla 16. Resultados a detalle de caso de estudio #4

DETALLES		MÉTRICAS		
No. Paquetes: 17		**PAQUETE**	**CLASE**	**MÉTODO**
No. Clases: 52		**A, CA, CE, DN, I**	**AHF, CBO, DAC, DIT, LCOM, LOC, MPC, NMO**	**LOC, NEST, NPAR, V(G) (4)**
No. Métodos: 203		**MHF (6)**	**NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)**	
Líneas de código totales: 2203		**RESULTADOS DE MÉTRICAS EVALUADAS**		
		CODIGO ESTATICO		EJECUCIÓN

ID	ELEMENTOS REVISADOS:	A	AHF	CA	CBO	CE	DAC	DIT	Dn/D	I	LCOM	LOC	MHF	MPC	NMO-NORM	NEST	NA/NOF	NOC	NOM	NOPA	NOPM	NPAR	RFC	V(G)	WMC	ME	MTTF	MTTR	MTBT
256	optNeo4JUsername( )				2							9				1						0		1		0			
257	optOutput( )				2							9				1						0		1		0			
258	optParallelism( )				2							9				1						0		1		0			
259	optPotentialLocality( )				2							9				1						0		1		0			
260	optPotentialMetric( )				2							9				1						0		1		0			
261	optPotentialThreshold( )				2							9				1						0		1		0			
262	optPotentialWeights( )				2							9				1						0		1		0			
263	optSource( )				2							9				1						0		1		0			
264	optVersion( )				2							7				1						0		1		0	NO APLICA		
265	getInstance( )				0							4				2						0		2		0			
266	CliService		N/A		7		7	1			0.0	136		6	0		0	0	0	0	3		38		37					
267	getCommandLine()				6							9				2						1		2		1			
268	handleArguments()				2							104				3						1		31		1			
269	loadConfiguration()				0							3				2						1		1		1			
270	printHelp( )				2							9				1						0		1		0			
271	printSplash( )				0							5				1						0		1		0			
272	printVersion( )				0							4				1						0		1		0			

Métricas más utilizadas [29] **Bueno/Común→** **Normal/Casual→** **Malo/Poco común→**

5.5.1 Análisis resultados-Caso de estudio 4

Tabla 17. Resultados Caso de Estudio 4

CASO DE ESTUDIO #4: CRIMEGRAPH, Marciani *et.al* [121]						
Tipo	Métrica	Elementos Umbral				Observaciones
		B	N	M	N/A	
Paquete(s) (17)	A	0	0	17	0	Las métricas afectadas son: Abstracción (A), Distancia Normalizada (DN), Inestabilidad (I) y Factor de Ocultamiento de Métodos (MHF). La clasificación N/A, se debe a que el valor obtenido no empata con ninguno de los umbrales o rangos definidos.
	CA	15	2	0	0	
	CE	16	1	0	0	
	DN	7	0	3	7	
	I	3	0	7	7	
	MHF	2	3	12	0	
Clase(s) (52)	AHF	0	0	30	22	Las métricas afectadas son: AHF, CBO, DAC, DIT, LCOM, MPC NA, NPM, NOPA, NOPM, RFC, WMC Para la métrica DAC, la clasificación N/A corresponde al tipo de clase ENUM. Por otra parte, para la métrica AHF, la clasificación N/A se debe a que dicha clase no posee atributos públicos que puedan ser contemplados para su evaluación. El resto de N/A se debe a que el valor no empata con ninguno de los umbrales especificados.
	CBO	31	8	13	0	
	DAC	24	7	15	6	
	DIT	28	22	2	0	
	LCOM	30	15	3	4	
	LOC	27	25	0	0	
	MPC	45	0	1	6	
	NMO	52	0	0	0	
	NA	43	8	1	0	
	NOC	52	0	0	0	
	NOM	29	6	17	0	
	NOPA	46	2	4	0	
	NOPM	21	29	2	0	
	RFC	21	28	3	0	
	WMC	45	5	2	0	
Método(s) (203)	LOC	164	33	6	0	Las 4 métricas involucradas presentan vulnerabilidades.
	NEST	145	55	3	0	
	NPAR	167	35	1	0	
	V(G)	175	18	10	0	

Como se mencionó anteriormente se revisaron 17 paquetes, y según lo evaluado se logró identificar que no hay una estabilidad óptima en este proyecto, debido a que el umbral "Malo" esta tan presente como "Bueno" (ver *Ilustración 28*), por lo que los atributos rendimiento y mantenibilidad se ven afectados, mismos que tienen un impacto en la tolerancia a fallas por ser atributos relacionados.

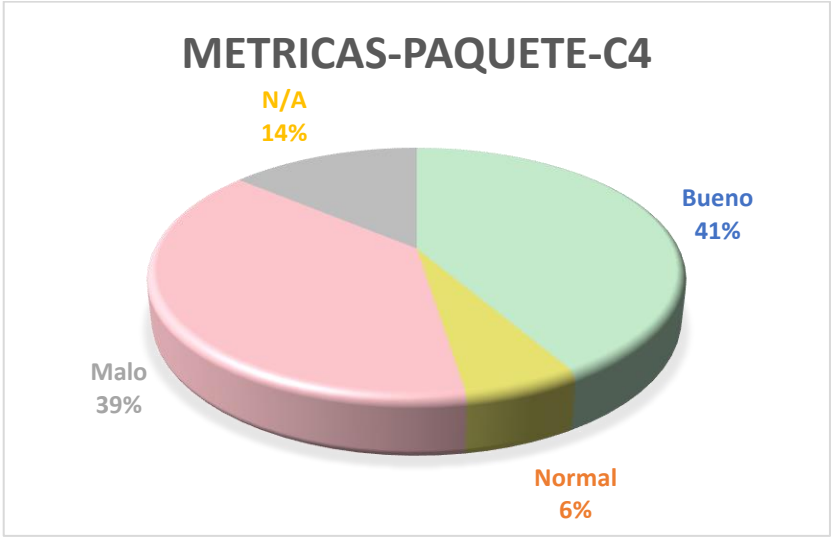


Ilustración 28. Gráfica de Resultados por Paquete del C4

A continuación (*Ilustración 29*) el resultado de la revisión de 52 elementos tipo clase (distribución por umbral).

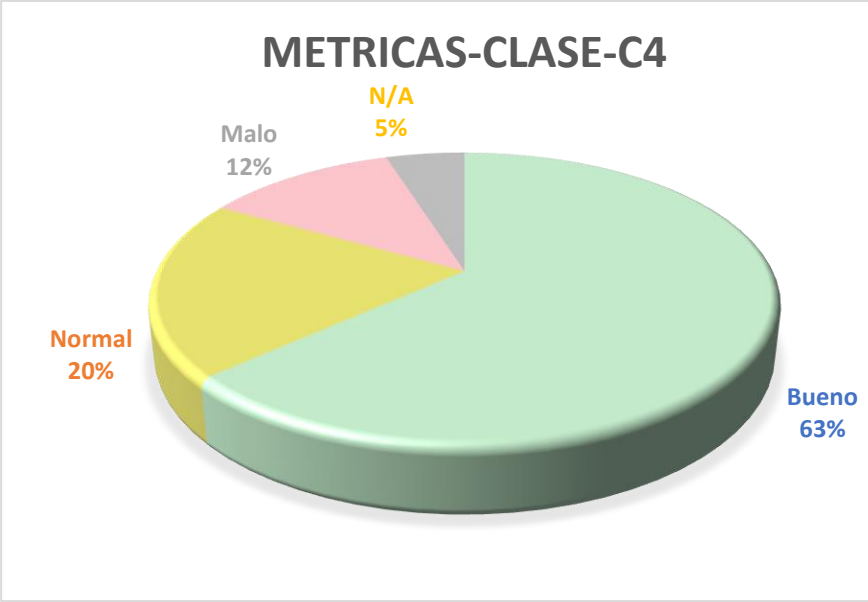


Ilustración 29. Gráfica de Resultados por Clase del C4

Como se puede observar en el gráfico, el umbral "Bueno" es el que predomina con una gran ventaja sobre el 12% "Malo". Ahora, si bien los resultados obtenidos indican que en su mayoría las métricas evaluadas corresponden al umbral "Bueno" es preciso destacar que basta con solo el porcentaje "Malo" para afectar a los atributos: rendimiento, mantenibilidad y sub sub-atributo testeabilidad y ser un futuro riesgo para la tolerancia a fallas, por lo que si la meta es mejorar la calidad de software es importante tomar medidas.

Por otra parte, como en el resto de casos de estudio los elementos tipo método, son quienes menos problemas presentaron en sus valores de acuerdo con la evaluación como se puede ver a continuación la *Ilustración 30*.

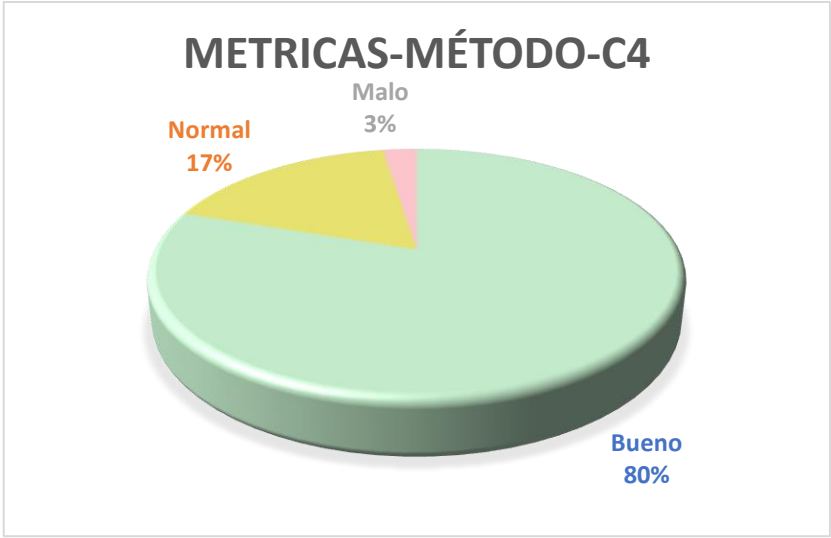


Ilustración 30. Gráfica de Resultados por Método del C4

A continuación, se presentan los resultados del proyecto, (ver *Ilustración 31*, *Ilustración 32*, *Ilustración 33*) agrupados por atributo de calidad, donde fueron considerados la mantenibilidad y el rendimiento.

Es importante destacar que este proyecto, a diferencia del resto, es mucho más extenso, y de acuerdo a los resultados mantenibilidad (atributo padre de testeabilidad) está en igualdad de condiciones respecto al atributo de rendimiento con un 10% en umbral "Malo", por lo que el factor de porcentaje afectado en conjunto incrementa a 20%, lo que representa un riesgo considerable.

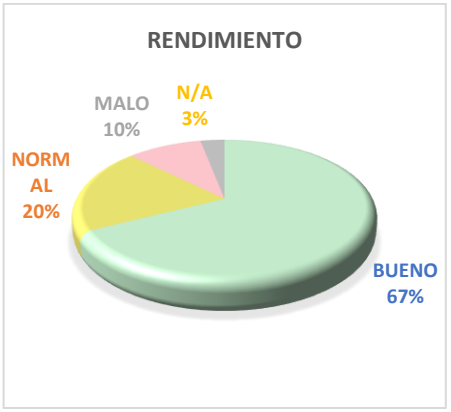


Ilustración 31. Atributo Rendimiento

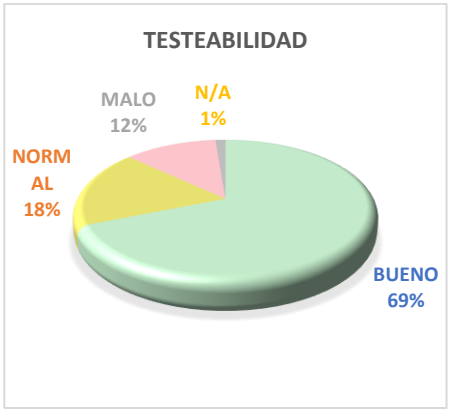


Ilustración 32. Sub-atributo Testeabilidad

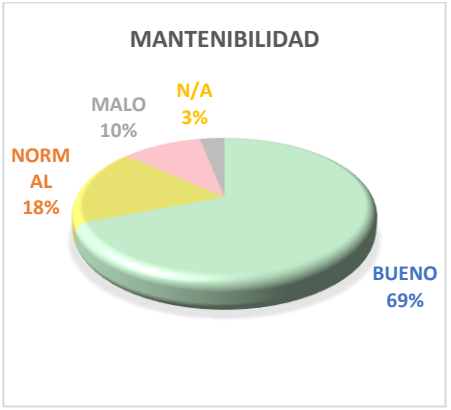


Ilustración 33. Atributo Mantenibilidad

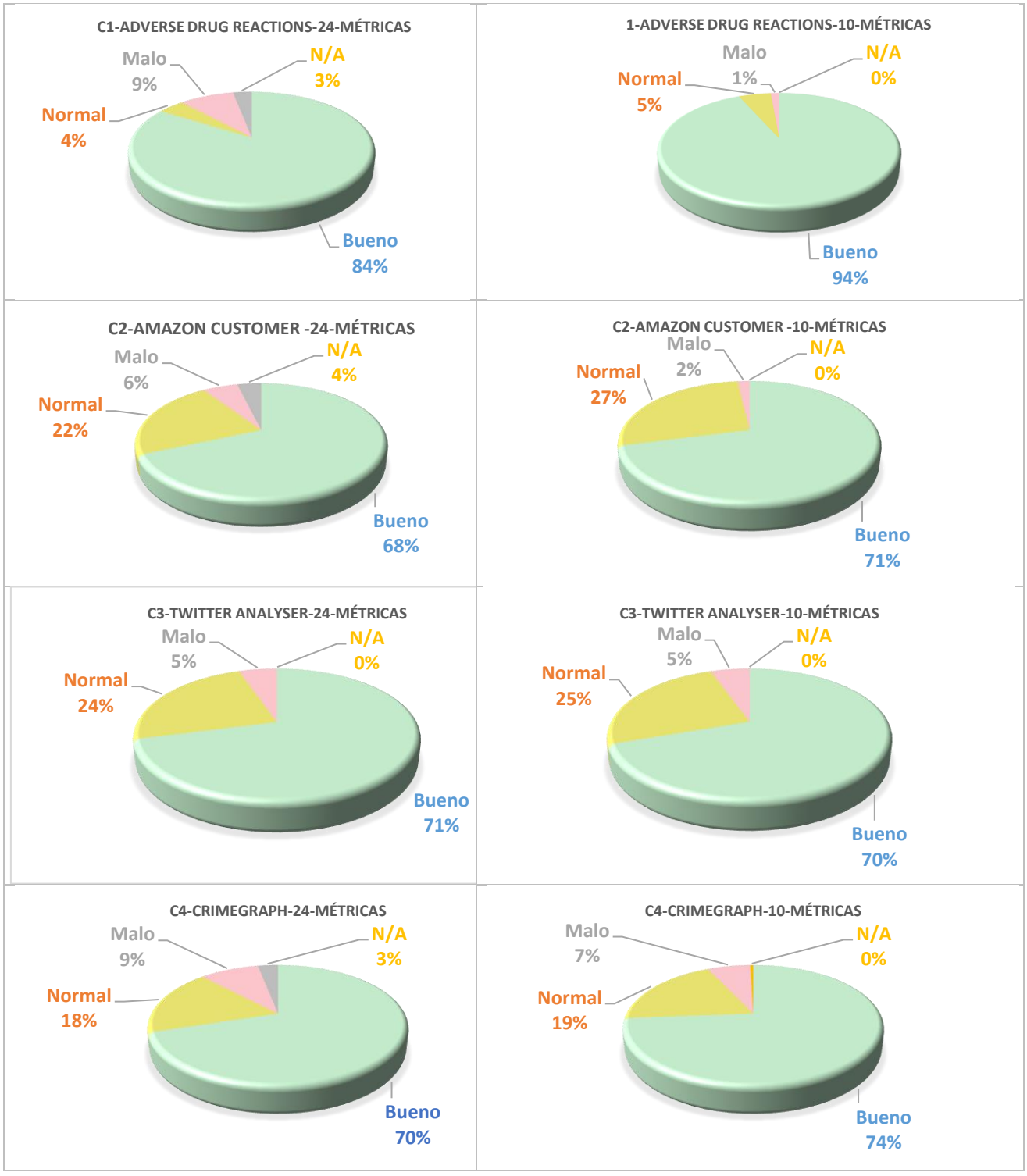
De forma general, a continuación (ver *Tabla 18*), se realiza una comparativa entre resultados de 4 casos de estudio revisados, utilizando la distribución de umbrales obtenidos en el cálculo de las métricas.

5.6 Comparación de resultados entre casos de estudio

Para la comparativa, son contemplados 2 escenarios: 1) Resultados considerando las métricas propuestas en el modelo (excepto las del atributo de disponibilidad- MTTR, MTTF, MTBF)

y 2) Resultados desde la perspectiva de las métricas más utilizadas por estudios presentes en la literatura, Nuñez *et.al* [29].

Tabla 18. Distribución de Umbral



De acuerdo a las gráficas, en los 4 proyectos predomina el umbral “Bueno”, y considerando las métricas que conforman la taxonomía de tolerancia a fallas (excepto las del atributo de disponibilidad), sería el caso de estudio: Adverse Drug Reactions, Koziol [120] con 84% es el que mayor porcentaje posee en dicha propiedad si solo se considera el umbral “Bueno”, y se obtiene el mismo resultado, si solo se contemplan las 10 métricas más comunes.

Sin embargo, el umbral “Normal” también es aceptable, de esta forma el porcentaje de tolerancia a fallas utilizando las métricas de la taxonomía propuesta seria:

- Adverse Drug Reactions 88%
- Amazon Customer 96%,
- Twitter Analyser 95 %
- CRIMEGRAPH 88%.

Por lo tanto, el modelo propuesto puede ser implementado de cualquiera de las dos formas, es decir, o bien considerando todas las métricas de la taxonomía o contemplando únicamente las 10 métricas más utilizadas en la literatura, Nuñez *et.al* [29]. Esto se debe a que en las gráficas anteriores se observa que no existe diferencia significativa entre los resultados y el impacto que representa en el resto de métricas es bajo. Por lo que todo depende del nivel de completes que se requiera en el análisis de tolerancia a fallas para un proyecto. Sin embargo, es conveniente continuar con la realización de más estudios de casos con fines de evaluar el comportamiento que presenten otros casos con el modelo de 10 o 28 métricas y considerar si se conservan o no las métricas propuestas en la taxonomía.

Finalmente, la *Tabla 19* presenta un vistazo general y refleja a que métricas se debería prestar atención, así como identificar qué tipo de elementos (paquetes, clases y métodos) resultan más afectados. Además, presenta en cuantos casos de estudio aparece la métrica con valores clasificados como umbral “Malo”, esto con el objetivo de tener una referencia de la probabilidad que existe encontrar en un futuro caso de estudio esa métrica, nuevamente con valores negativos.

Tabla 19. Resultados finales de los casos de estudio

Tipo	Métrica	Casos de estudio				Apariciones
		C1	C2	C3	C4	
Paquete(s)	A	✓	✓	✓	✓	4
	CA					0
	CE					0
	DN	✓			✓	2
	I	✓	✓	✓	✓	4
	MHF	✓	✓		✓	3
Clase(s)	AHF	✓	✓	✓	✓	4
	CBO		✓	✓	✓	3
	DAC				✓	1
	DIT				✓	1
	LCOM	✓	✓	✓	✓	4
	LOC					0
	MPC				✓	1
	NMO					0
	NA	✓			✓	2
	NOC					0
	NOM				✓	1
	NOPA				✓	1
	NOPM			✓	✓	2
	RFC				✓	1
	WMC				✓	1
Método(s)	LOC				✓	1
	NEST		✓		✓	2
	NPAR				✓	1
	V(G)		✓	✓	✓	3

De manera general se puede concluir que las métricas A, I, AHF, LCOM (sombreadas en rojo) están presentes en los 4 casos de estudio con valores fuera de los umbrales señalados como aceptables, afectando a paquetes y clases, de modo que es importante prestar atentación a la situación.

El siguiente conjunto de métricas (sombreadas en naranja) compuesto por: MHF, CBO Y V(G), aparece con valores "no aceptables" en 3 de los casos de estudio.

En contraste a lo anterior, las métricas Ca, Ce, LOC, NMO y NOC (sombreadas en verde) no presentan problemas en ninguno de los casos de estudio.

Finalmente es importante aclarar que la métrica cualitativa ME- Manejo de Excepciones (presente en la taxonomía), es implementada en todos los casos de estudio revisados.

La siguiente sección corresponde a las conclusiones para este trabajo, hallazgos de la investigación y algunos trabajos futuros.

Capítulo 6. Conclusiones y trabajos futuros

Esta sección presenta las conclusiones de la investigación, hallazgos encontrados durante el desarrollo del trabajo de investigación y los beneficios que aporta el modelo de calidad propuesto. Además, se proponen trabajos futuros para mejorar en el ámbito de tolerancia a fallas específicamente en sistemas de Big Data.

A lo largo de la investigación se identificó que no existe en la literatura alguna forma explícita para medir la tolerancia a fallas en los sistemas Big Data, una de las causas probables de esto es que, es un problema extenso y compuesto de diversos factores, como por ejemplo fallas no solo en el software, si no en la infraestructura del hardware y errores provocados por humanos. Además, de acuerdo a lo encontrado en la literatura, hasta este momento, se tienen identificadas las posibles fallas presentes en un sistema de tipo Big Data , Cao *et.al* [2], sin embargo, no se presenta la forma de abordar la corrección de estas fallas. Por lo tanto, resulta complejo proponer una solución que abarque todo lo necesario para cubrir la extensión de la tolerancia a fallas desde la prevención hasta la corrección.

Como resultado de lo anterior, se procedió a investigar las formas de medir la tolerancia a fallas de en sistemas de software convencionales para posteriormente aplicarlo a un contexto de software Big Data. Con esta base, la investigación se dirigió principalmente hacia los se indago sobre que atributos están relacionados a la tolerancia a fallas y durante la revisión se lograron identificar y recopilar a partir de diversos trabajos, los siguientes atributos:

- Disponibilidad
- Rendimiento
- Mantenibilidad
- Testeabilidad (sub-atributo de Mantenibilidad)

También se identificó que no existe un consenso entre autores que puntualice concretamente que atributos considerar. Las propuestas relacionadas al tema son escazas y además si bien mencionan su relación, no se especifica cómo medirlos o utilizarlos.

En el entorno de aplicación real, es inevitable la presencia de diversos tipos de fallas, sin importar el tipo de sistema. Específicamente en sistemas Big Data, las fallas pueden surgir por 6 aspectos principales, Cao *et.al* [2]:

- Falla de hardware
- ***Defecto de código***
- Limitación de diseño del sistema
- Falla de operación

- Mala configuración
- Falla del software de aplicación

Si bien, se conocen las causas que provocan fallas en sistemas Big Data, resulta complejo integrar una solución que contemple todos los escenarios, debido a que su vez, estos son originados/ocasionados por eventos lo que extiende el problema. Por lo tanto, este trabajo aborda el atributo de tolerancia a fallas única y específicamente desde la perspectiva de "defectos de código".

Así mismo, dado que actualmente la generación de información se masifica y crece rápidamente, provocara cada vez más desarrollos Big Data por lo que sería conveniente formular otras propuestas dirigidas al resto de tipos de fallas y contribuir a lograr sistemas más robustos.

Por otra parte, del catálogo de métricas disponibles en la literatura para medir atributos de calidad, es extenso, al menos 300 métricas, Nuñez *et.al* [29], sobre este volumen se identificaron 34 con mayor ocurrencia de las cuales se excluyeron 8 donde:

- Para el caso: FANIN, FANOUT fueron descartadas debido a que de acuerdo a lo encontrado en la literatura son consideradas métricas equivalentes a Ca (Acoplamiento Aferente), Ce (Acoplamiento Eferente).
- Por otro lado, LCOM2, LCOM3, LCOMM forman parte de las variantes existentes para la métrica LCOM por lo tanto no fueron contempladas en el modelo propuesto.
- Respecto a la métrica Esfuerzo (E) es una métrica de proceso (se concluyó que no se puede medir a partir del código fuente) y se relaciona con las horas hombre que se emplean en el desarrollo de software, motivo por el cual fue excluida.

En conclusión, el modelo de calidad propuesto se conforma de 28 métricas cuantitativas.

Para realizar el proceso de medición, se evaluó el modelo propuesto en los casos de estudio presentados, utilizando las herramientas: IDE Eclipse, Plugin CodeMR, Plugin VizzMaintenance, etc., para más detalles consultar el *Anexo 2, pág. 144*. Posteriormente, se contrastaron los valores obtenidos por cada métrica con los umbrales encontrados en la literatura y se clasificaron en: Bueno/común, Normal/casual, Malo/poco común, lo que permite identificar en qué medida se realiza un proceso de construcción de sistemas de

software de calidad. Además, respecto a los umbrales, es importante precisar que el problema radica en el número de umbrales disponibles en la literatura, para una misma métrica, no existe un consenso, por lo que se encontrar una forma correcta para describir, utilizar y calcular la métrica se vuelve tedioso. Por otro lado, también se presenta el caso, en que algunas métricas carecen de información que las describa.

Respecto a los complementos utilizados para calcular las métricas (*plugin*) detecto que son sumamente escazos, específicas y en algunos casos desactualizadas. Además, durante su implementación se observó que los resultados de los plugin utilizados arrojaron mediciones distintas entre ellos para una misma métrica, en varios casos, esto se debe a que cada desarrollo de los plugin, utiliza distintos métodos de cálculo de métricas y umbrales, por lo que resulta importante estandarizar los umbrales de las métricas y evitar ambigüedades. Además, en ciertos casos los plugin se reservan la información sobre como evalúan los proyectos.

El modelo de calidad propuesto contribuye en los aspectos: aportar métricas de código fuente que se pueden considerar antes o después de la construcción del software (programación), y técnicas de tolerancia a fallas que son una extensión para fortalecer la evaluación del software tolerante a fallas dado que es necesario implementar mecanismos que permitan a un sistema Big Data afrontar las fallas que se presenten. En relación con las técnicas de tolerancia a fallas consideradas en el modelo de calidad propuesto, es necesario precisar que son una recopilación de las técnicas más representativas y de uso común como medida de tolerancia a fallas.

Los 4 casos de estudio presentados en este documento, fueron evaluados desde la perspectiva: métricas cuantitativas (métricas de código fuente) utilizando el código estático del proyecto tomado de repositorios GitHub[2] y, por otro lado, las métricas cualitativas (técnicas de tolerancia a fallas) se omitieron, debido a que no fue posible obtener documentación del proyecto que permitiera valorar si el ambiente Big Data utilizaba mecanismos de tolerancia a fallas.

Finalmente, de acuerdo al análisis de resultados obtenidos, se identificó que el tamaño del proyecto es un factor importante, debido a que a mayor cantidad de elementos que conforman

[2] https://www.github.com

el sistema (paquetes, clases y métodos) mayor será el número de métricas que nos indiquen el estatus del proyecto en cuanto a tolerancia a fallas.

De acuerdo con los resultados obtenidos:

- A (Abstracción), I (Inestabilidad), AHF (Factor de Ocultación de Atributos), LCOM (Falta de cohesión en los Métodos), en todos los casos evaluados, arrojaron resultados que en conjunción pudiese reflejar baja tolerancia a fallas. Sin embargo, se debe continuar con el análisis de dichas métricas, la relación con tolerancia a fallas y su comportamiento en otros casos de estudio con un análisis comparativo.
- Ca (Acoplamiento Aferente), Ce (Acoplamiento Eferente), LOC (Líneas de Código), NMO (Número de Métodos) y NOC (Número de Hijos) obtuvieron mejores resultados en todos los casos, esto podría significar que estas métricas son las más presentes a conciencia en la práctica de los desarrolladores del software.

En conclusión, el objetivo general establecido para el proyecto, que consiste en definir un "Modelo de calidad para medición de la tolerancia a fallas en los sistemas de Big Data", se cumplió al establecer un conjunto de atributos de calidad, que utilizando métricas de código fuente para su evaluación, permiten identificar el estado de tolerancia a fallas que posee el software revisado. Sin embargo, al no encontrar en la literatura un trabajo similar, es decir, que aborde específicamente la problemática planteada no hay posibilidad de comparar los resultados obtenidos dado que este trabajo es precursor de la medición de tolerancia a fallas para código estático en sistemas Big Data.

6.1 Aportaciones y beneficios

La realidad es que, no tener a disposición información de cómo evitar o resolver cierto problema, provoca que se continúen realizando malas prácticas. En el desarrollo de sistemas Big Data, la deficiencia en el proceso de diseño y construcción de software a futuro ocasiona la presencia de fallas en él sistema, de modo que el desarrollo de un Modelo de calidad para medición de la tolerancia a fallas en sistemas de Big Data, representa una forma de evaluar objetivamente los desarrollos de software Big Data, su implementación permitirá identificar las carencias que posee el sistema, y conocer el nivel de tolerancia a fallas presente en el software. Por otro lado, un aspecto importante es, que el trabajo propuesto puede auxiliar a

nuevos desarrolladores de software que recién se incorporen a un proyecto, a conocer y poner énfasis en los aspectos que influyen en la tolerancia a fallas, con el objetivo de generar software de mejor calidad.

6.2 Publicaciones

Se realizó la publicación de un artículo en el volumen 8 de la revista Jornada de Ciencia y Tecnología Aplicada. Este artículo, Métricas de tolerancia a fallas: una taxonomía para sistemas Big Data (Guadarrama, Rojas, López, Fragoso y Hernández, 2022), describe varias métricas oculares, su generación y su aplicación en un contexto de evaluación de la experiencia de usuario.

6.3 Trabajos futuros

Como trabajos futuros, se propone continuar con la evaluación de más casos de estudios y determinar puntualmente el catálogo de métricas enfocado a la medición del atributo tolerancia a fallas. Esto incluye la evaluación de las técnicas de tolerancia a fallas, de tal forma que puedan ser interpretadas como métricas cualitativas de aquellos sistemas que están diseñados con el propósito de ser tolerante a fallas. Además, un aspecto importante sería fijar un porcentaje de impacto a los atributos relacionados a tolerancia a fallas, así como a las métricas que los conforman.

Por último, es importante destacar que el escenario ideal sería probar el modelo de calidad en un sistema Big Data en ejecución para así considerar la totalidad de atributos relacionados, recordando que este trabajo excluye la disponibilidad en sus evaluaciones.

Referencias

En estas secciones presentan todos los trabajos consultados y utilizados durante el proceso de investigación, dichas investigaciones son el sustento teórico de este trabajo de investigación

[1] I. Lee, “Big data: Dimensions, evolution, impacts, and challenges,” *Bus. Horiz.*, vol. 60, no. 3, pp. 293–303, 2017.
[2] R. Cao and J. Gao, “Research on reliability evaluation of big data system,” in *2018 IEEE 3rd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, Apr. 2018, pp. 261–265. doi: 10.1109/ICCCBDA.2018.8386523.
[3] C. Arcila-Calderón, E. Barbosa-Caro, and F. Cabezuelo-Lorenzo, “Técnicas big data: análisis de textos a gran escala para la investigación científica y periodística,” *Prof. la Inf.*, vol. 25, no. 4, pp. 623–631, 2016, doi: 10.3145/epi.2016.jul.12.
[4] E. Raguseo, “Big data technologies: An empirical investigation on their adoption, benefits and risks for companies,” *Int. J. Inf. Manage.*, vol. 38, no. 1, pp. 187–195, Feb. 2018, doi: 10.1016/j.ijinfomgt.2017.07.008.
[5] C. A. Salma, B. Tekinerdogan, and I. N. Athanasiadis, “Feature Driven Survey of Big Data Systems,” in *Proceedings of the International Conference on Internet of Things and Big Data*, 2016, pp. 348–355. doi: 10.5220/0005877503480355.
[6] S. Alkatheri, S. Abbas, and M. Siddiqui, “A Comparative Study of Big Data Frameworks,” *Int. J. Comput. Sci. Inf. Secur.*, vol. 17, Jan. 2019.
[7] A. Mohamed, M. K. Najafabadi, Y. B. Wah, E. A. K. Zaman, and R. Maskat, “The state of the art and taxonomy of big data analytics: view from new big data framework,” *Artif. Intell. Rev.*, vol. 53, no. 2, pp. 989–1037, Feb. 2020, doi: 10.1007/s10462-019-09685-9.
[8] K. N. Singh, R. K. Behera, and J. K. Mantri, “Big Data Ecosystem: Review on Architectural Evolution,” 2019, pp. 335–345. doi: 10.1007/978-981-13-1498-8_30.
[9] M. Callejas-Cuervo, A. C. Alarcón-Aldana, and A. M. Álvarez-Carreño, “Modelos de calidad del software, un estado del arte,” *ENTRAMADO*, vol. 13, no. 1, pp. 236–250, 2017, doi: 10.18041/entramado.2017v13n1.25125.
[10] A. J. Capistran Abundez, “Selección de un Método Ágil para el desarrollo de Sistemas Big Data,” Centro nacional de investigación y desarrollo tecnológico, 2020.
[11] D. Ramírez Gervacio, “Estudio de mapeo sistemático en el problema de la variedad en sistemas Big Data,” Centro nacional de investigación y desarrollo tecnológico, 2020.
[12] M. López Sanz, “Tema 2: Ciclo de vida del software.” 2009. [Online]. Available: http://cotana.informatica.edu.bo/downloads/2.1 Ciclo de VidaSW1.pdf
[13] P. Berander *et al.*, “Software quality attributes and trade-offs,” *Blekinge Inst. Technol.*, vol. 97, no. 98, p. 19, 2005.
[14] ISO/IEC, “La familia de normas ISO/IEC 25000.” 2011. [Online]. Available: https://iso25000.com/index.php/normas-iso-25000
[15] I. O. for Standardization, *Systems and Software Engineering: Systems and Software Quality Requirements and Evaluation (SQuaRE): Measurement of System and Software Product Quality*. ISO, 2016.
[16] J. Bermeo Conto, J. J. Maldonado, M. Sánchez, and J. P. Carvallo, “Modelos de calidad de software en la práctica: Mejorando su construcción con el soporte de modelos conceptuales,” 2016.
[17] M. J. Blas, S. Gonnet, and H. Leone, “Una Taxonomía de Atributos de Calidad para la Evaluación de Arquitecturas de Software por medio de Simulación,” 2014. doi: 10.13140/2.1.5067.8560.

[18] M. Al-Kuwaiti, N. Kyriakopoulos, and S. Hussein, “A comparative analysis of network dependability, fault-tolerance, reliability, security, and survivability,” *IEEE Commun. Surv. Tutorials*, vol. 11, no. 2, pp. 106–124, 2009, doi: 10.1109/SURV.2009.090208.

[19] C. Pan, M. Lu, H. Zhang, and B. Xu, “Qualitative Software Reliability Requirements: Concept, Classification and Practical Elicitation Methods,” in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, Jul. 2018, pp. 164–171. doi: 10.1109/QRS-C.2018.00040.

[20] D. Poola, M. A. Salehi, K. Ramamohanarao, and R. Buyya, “A taxonomy and survey of fault-tolerant workflow management systems in cloud and distributed computing environments,” in *Software architecture for big data and the cloud*, Elsevier, 2017, pp. 285–320.

[21] F. C. Gärtner, “Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments,” *ACM Comput. Surv.*, vol. 31, no. 1, pp. 1–26, Mar. 1999, doi: 10.1145/311531.311532.

[22] N. Fenton and J. Bieman, *Software metrics: a rigorous and practical approach*. CRC press, 2019.

[23] R. S. Samosir, H. L. Hendric, F. L. Gaol, E. Abdurachman, and B. Soewito, “Measurement Metric Proposed For Big Data Analytics System,” in *Proceedings of the 2017 International Conference on Computer Science and Artificial Intelligence*, 2017, pp. 265–269.

[24] H. Zhou, J.-G. Lou, H. Zhang, H. Lin, H. Lin, and T. Qin, “An Empirical Study on Quality Issues of Production Big Data Platform,” in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, May 2015, pp. 17–26. doi: 10.1109/ICSE.2015.130.

[25] Ho-Won Jung, Seung-Gweon Kim, and Chang-Shin Chung, “Measuring Software Product Quality: A Survey of ISO/IEC 9126,” *IEEE Softw.*, vol. 21, no. 05, pp. 88–92, Sep. 2004, doi: 10.1109/MS.2004.1331309.

[26] P. Zhang, X. Zhou, W. Li, and J. Gao, “A survey on quality assurance techniques for big data applications,” in *2017 IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService)*, 2017, pp. 313–319.

[27] I. S. Association and others, “Systems and software engineering—Vocabulary ISO/IEC/IEEE 24765: 2010,” *Iso/Iec/Ieee*, vol. 24765, pp. 1–418, 2010.

[28] N. Falih, B. Hendradjaya, and W. D. Sunindyo, “Quality measurement for Web GIS using object-oriented development,” in *2016 6th International Annual Engineering Seminar (InAES)*, 2016, pp. 144–149.

[29] A. S. Nuñez-Varela, H. G. Pérez-Gonzalez, F. E. Martínez-Perez, and C. Soubervielle-Montalvo, “Source code metrics: A systematic mapping study,” *J. Syst. Softw.*, vol. 128, pp. 164–197, Jun. 2017, doi: 10.1016/j.jss.2017.03.044.

[30] H. Singhani and P. R. Suri, “Testability assessment model for object oriented software based on internal and external quality factors,” *Glob. J. Comput. Sci. Technol.*, 2015.

[31] M. Nabi, M. Toeroe, and F. Khendek, “Availability in the cloud: State of the art,” *J. Netw. Comput. Appl.*, vol. 60, pp. 54–67, Jan. 2016, doi: 10.1016/j.jnca.2015.11.014.

[32] P. C. Neves, B. R. Schmerl, J. Cámara, and J. Bernardino, “Big Data in Cloud Computing: Features and Issues.,” in *IoTBD*, 2016, pp. 307–314.

[33] N. Zanoon, A. Al-Haj, and S. M. Khwaldeh, “Cloud computing and big data is there a relation between the two: a study,” *Int. J. Appl. Eng. Res.*, vol. 12, no. 17, pp. 6970–

6982, 2017.
[34] M. A. Mukwevho and T. Celik, “Toward a Smart Cloud: A Review of Fault-Tolerance Methods in Cloud Systems,” *IEEE Trans. Serv. Comput.*, vol. 14, no. 2, pp. 589–605, Mar. 2021, doi: 10.1109/TSC.2018.2816644.
[35] M. A. Shahid, N. Islam, M. M. Alam, M. S. Mazliham, and S. Musa, “Towards Resilient Method: An exhaustive survey of fault tolerance methods in the cloud computing environment,” *Comput. Sci. Rev.*, vol. 40, p. 100398, May 2021, doi: 10.1016/j.cosrev.2021.100398.
[36] M. Hasan and M. S. Goraya, “Fault tolerance in cloud computing environment: A systematic survey,” *Comput. Ind.*, vol. 99, pp. 156–172, Aug. 2018, doi: 10.1016/j.compind.2018.03.027.
[37] International Organization for Standardization, *ISO/IEC/IEEE 12207:2017 Systems and software engineering — Software life cycle processes*, 1st ed. 2017.
[38] H. Al-Kilidar, K. Cox, and B. Kitchenham, “The use and usefulness of the ISO/IEC 9126 quality standard,” in *2005 International Symposium on Empirical Software Engineering, 2005.*, pp. 122–128. doi: 10.1109/ISESE.2005.1541821.
[39] Li Qian, Xiuming Chen, Jiahua Wan, Lu Chen, Yuanyuan Zhou, and Chunjiao Yin, “The research of software reliability based on Failure Data,” in *2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, Aug. 2016, pp. 880–883. doi: 10.1109/ICSESS.2016.7883206.
[40] V. C. Storey and I.-Y. Song, “Big data technologies and Management: What conceptual modeling can do,” *Data Knowl. Eng.*, vol. 108, pp. 50–67, Mar. 2017, doi: 10.1016/j.datak.2017.01.001.
[41] J. Merino, I. Caballero, B. Rivas, M. Serrano, and M. Piattini, “A Data Quality in Use model for Big Data,” *Futur. Gener. Comput. Syst.*, vol. 63, pp. 123–130, Oct. 2016, doi: 10.1016/j.future.2015.11.024.
[42] V. Garises and J. Quenum, “An Evaluation of Big Data Architectures,” in *Proceedings of the 8th International Conference on Data Science, Technology and Applications*, 2019, pp. 152–159. doi: 10.5220/0007840801520159.
[43] S. Reddivari and J. Raman, “Software Quality Prediction: An Investigation Based on Machine Learning,” in *2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI)*, Jul. 2019, pp. 115–122. doi: 10.1109/IRI.2019.00030.
[44] B. Khan *et al.*, “Software Defect Prediction for Healthcare Big Data: An Empirical Evaluation of Machine Learning Techniques,” *J. Healthc. Eng.*, vol. 2021, pp. 1–16, Mar. 2021, doi: 10.1155/2021/8899263.
[45] A. Davoudian and M. Liu, “Big Data Systems,” *ACM Comput. Surv.*, vol. 53, no. 5, pp. 1–39, Oct. 2020, doi: 10.1145/3408314.
[46] S. Martinez-Fernandez, A. Jedlitschka, L. Guzman, and A. M. Vollmer, “A Quality Model for Actionable Analytics in Rapid Software Development,” in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Aug. 2018, pp. 370–377. doi: 10.1109/SEAA.2018.00067.
[47] P. Kasu, P. Hamandawana, and T.-S. Chung, “DLFT: Data and Layout Aware Fault Tolerance Framework for Big Data Transfer Systems,” *IEEE Access*, vol. 9, pp. 22939–22954, 2021.
[48] A. Rosa, L. Y. Chen, and W. Binder, “Failure Analysis and Prediction for Big-Data Systems,” *IEEE Trans. Serv. Comput.*, vol. 10, no. 6, pp. 984–998, Nov. 2017, doi:

10.1109/TSC.2016.2543718.
[49] M. Dyavanur and K. Kori, “Fault tolerance techniques in big data tools: a survey,” *Int. J. Innov. Res. Comput. Commun. Eng. IJIRCCE*, vol. 2, 2014.
[50] Y. Fang, Q. Chen, and N. Xiong, “A multi-factor monitoring fault tolerance model based on a GPU cluster for big data processing,” *Inf. Sci. (Ny).*, vol. 496, pp. 300–316, Sep. 2019, doi: 10.1016/j.ins.2018.04.053.
[51] X. Wu, Z. Du, S. Dai, and Y. Liu, “The Fault Tolerance of Big Data Systems,” 2017, pp. 65–74. doi: 10.1007/978-981-10-3996-6_5.
[52] T. Cowsalya and S. R. Mugunthan, “Hadoop architecture and fault tolerance based hadoop clusters in geographically distributed data center,” *ARPN J. Eng. Appl. Sci.*, vol. 10, no. 7, pp. 2818–2821, 2015.
[53] M. Saadoon, S. H. Ab. Hamid, H. Sofian, H. H. M. Altarturi, Z. H. Azizul, and N. Nasuha, “Fault tolerance in big data storage and processing systems: A review on challenges and solutions,” *Ain Shams Eng. J.*, Jul. 2021, doi: 10.1016/j.asej.2021.06.024.
[54] A. Boranbayev, S. Boranbayev, K. Yersakhanov, A. Nurusheva, and R. Taberkhan, “Methods of Ensuring the Reliability and Fault Tolerance of Information Systems,” 2018, pp. 729–730. doi: 10.1007/978-3-319-77028-4_93.
[55] Y. A. Alshehri, K. Goseva-Popstojanova, D. G. Dzielski, and T. Devine, “Applying Machine Learning to Predict Software Fault Proneness Using Change Metrics, Static Code Metrics, and a Combination of Them,” in *SoutheastCon 2018*, Apr. 2018, pp. 1–7. doi: 10.1109/SECON.2018.8478911.
[56] S. R. Aziz, T. A. Khan, and A. Nadeem, “Experimental Validation of Inheritance Metrics’ Impact on Software Fault Prediction,” *IEEE Access*, vol. 7, pp. 85262–85275, 2019, doi: 10.1109/ACCESS.2019.2924040.
[57] S. R. Aziz, T. A. Khan, and A. Nadeem, “Exclusive use and evaluation of inheritance metrics viability in software fault prediction—an experimental study,” *PeerJ Comput. Sci.*, vol. 7, p. e563, 2021.
[58] F. Ullah and M. Ali Babar, “Architectural Tactics for Big Data Cybersecurity Analytics Systems: A Review,” *J. Syst. Softw.*, vol. 151, pp. 81–118, May 2019, doi: 10.1016/j.jss.2019.01.051.
[59] N. Akhtar, F. Parwej, and Y. Perwej, “A perusal of big data classification and hadoop technology,” *Sci. Educ.*, vol. 4, no. 1, pp. 26–38, 2017.
[60] M. Lackovic, D. Talia, R. Tolosana-Calasanz, J. A. Banares, and O. F. Rana, “A taxonomy for the analysis of scientific workflow faults,” in *2010 13th IEEE International Conference on Computational Science and Engineering*, 2010, pp. 398–403.
[61] M. Elattar, V. Wendt, A. Neumann, and J. Jasperneite, “Potential of multipath communications to improve communications reliability for internet-based cyberphysical systems,” in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016, pp. 1–8.
[62] “IEEE Standard Dictionary of Measures of the Software Aspects of Dependability,” *IEEE Std 982.1-2005 (Revision IEEE Std 982.1-1988)*, pp. 1–41, 2006, doi: 10.1109/IEEESTD.2006.215280.
[63] Software and systems engineering, *ISO/IEC 9126-1:2001 Software engineering — Product quality — Part 1: Quality model*, 1st ed. 2001.
[64] A. Capiluppi, N. Ajienka, and S. Counsell, “The effect of multiple developers on

structural attributes: a study based on java software," *J. Syst. Softw.*, vol. 167, p. 110593, 2020.
[65] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, Jun. 1994, doi: 10.1109/32.295895.
[66] S. K. Dubey and A. Rana, "Assessment of maintainability metrics for object-oriented software system," *ACM SIGSOFT Softw. Eng. Notes*, vol. 36, no. 5, pp. 1–7, 2011.
[67] R. Goyal, P. Chandra, and Y. Singh, "Fuzzy inferencing to identify degree of interaction in the development of fault prediction models," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 29, no. 1, pp. 93–102, Jan. 2017, doi: 10.1016/j.jksuci.2014.12.008.
[68] P. Kumari and P. Kaur, "A survey of fault tolerance in cloud computing," *J. King Saud Univ. Inf. Sci.*, vol. 33, no. 10, pp. 1159–1176, 2021.
[69] G. K. Saha, "Software-implemented self-Healing system," *CLEI Electron. J.*, vol. 10, no. 2, 2007.
[70] S. Almugrin, W. Albattah, O. Alaql, M. Alzahrani, and A. Melton, "Instability and abstractness metrics based on responsibility," in *2014 IEEE 38th Annual Computer Software and Applications Conference*, 2014, pp. 364–373.
[71] S. Hyrynsalmi and V. Leppänen, "A Validation of Martin's Metric," in *Proceedings of 11th Symposium on Programming Languages and Software Tools and 7th Nordic Workshop on Model Driven Software Engineering*, 2009, pp. 87–101.
[72] L. Al-Zobaidy and K. Ibrahim, "Existing Object Oriented Design Metrics a Study and Comparison," *AL-Rafidain J. Comput. Sci. Math.*, vol. 10, no. 1, pp. 137–147, Feb. 2013, doi: 10.33899/csmj.2013.163431.
[73] K. E. M. Sabri, "REVERSE SOFTWARE ENGINEERING AND REENGINEERING TO DETECT PLAGIARISM IN JAVA PROGRAMS," Citeseer, 2003.
[74] D. Rodr\'\iguez and R. Harrison, "Medición en la orientación a objetos," *Sch. Comput. Sci. Cybern. \& Electron. Eng. Univ. Reading, UK*, 2007.
[75] A. K. Sharma, A. Kalia, and H. Singh, "Metrics identification for measuring object oriented software quality," *Int. J. soft Comput. Eng.*, vol. 2, no. 5, pp. 255–258, 2012.
[76] K. Kuk, P. Milić, and S. Denić, "Object-oriented software metrics in software code vulnerability analysis," in *2020 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*, 2020, pp. 1–6.
[77] S. O. Hasoon and F. M. R. Younis, "Application of Fuzzy Logic Model for Software Quality Measurements," *Comput. Sci.*, vol. 17, no. 3, pp. 1227–1240, 2022.
[78] T. G. S. Filó, M. Bigonha, and K. Ferreira, "A catalogue of thresholds for object-oriented software metrics," *Proc. 1st SOFTENG*, pp. 48–55, 2015.
[79] L. C. Briand and J. W. Daly, "A unified framework for coupling measurement in object-oriented systems," *IEEE Trans. Softw. Eng.*, vol. 25, no. 1, pp. 91–121, 1999, doi: 10.1109/32.748920.
[80] H. A. Sahraoui, R. Godin, and T. Miceli, "Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. IEEE Transactions on Software Engineering, 20(6), 476–493. doi:10.1109/32.295895," 2000.
[81] H. A. Sahraoui, R. Godin, and T. Miceli, "Can metrics help bridging the gap between the improvement of OO design quality and its automation," 2000.
[82] J. P. Antony, "Predicting reliability of software using thresholds of CK metrics," *Int. J. Adv. Netw. Appl.*, vol. 4, no. 6, p. 1778, 2013.
[83] M. Choinzon and Y. Ueda, "Detecting Defects in Object Oriented Designs Using

Design Metrics.," 2006, pp. 61–72.
[84] K. K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Empirical Study of Object-Oriented Metrics.," *J. Object Technol.*, vol. 5, no. 8, pp. 149–173, 2006.
[85] D. Glasberg, K. El-Emam, W. Memo, and N. Madhavji, "Validating Object-Oriented Design Metrics on a Commercial Java Application," Jan. 2000, doi: 10.4224/8914217.
[86] M. R. Shaheen and L. du Bousquet, "Relation between Depth of Inheritance Tree and Number of Methods to Test," in *2008 1st International Conference on Software Testing, Verification, and Validation*, Apr. 2008, pp. 161–170. doi: 10.1109/ICST.2008.34.
[87] E. Chandra and P. E. Linda, "Class break point determination using CK metrics thresholds," *Glob. J. Comput. Sci. Technol.*, 2010.
[88] S. C. Misra and V. C. Bhavsar, "Relationships Between Selected Software Measures and Latent Bug-Density: Guidelines for Improving Quality," 2003, pp. 724–732. doi: 10.1007/3-540-44839-X_76.
[89] Refactorit, "RefactorIT User Manual." pp. 1–116, 2004. [Online]. Available: https://manualzz.com/doc/7049317/refactorit-user-manual
[90] A. M. Garc\'\ia Sánchez, "Evaluación de métricas de calidad del software sobre un programa Java," 2010.
[91] W. Li and S. Henry, "Object-oriented metrics that predict maintainability," *J. Syst. Softw.*, vol. 23, no. 2, pp. 111–122, 1993.
[92] D. K. Kim, "Finding bad code smells with neural network models," *Int. J. Electr. Comput. Eng.*, vol. 7, no. 6, p. 3613, 2017.
[93] A. Madi, O. K. Zein, and S. Kadry, "On the improvement of cyclomatic complexity metric," *Int. J. Softw. Eng. Its Appl.*, vol. 7, no. 2, pp. 67–82, 2013.
[94] S. Herbold, J. Grabowski, and S. Waack, "Calculation and optimization of thresholds for sets of software metrics," *Empir. Softw. Eng.*, vol. 16, no. 6, pp. 812–841, 2011.
[95] M. O. Elish, A. H. Al-Yafei, and M. Al-Mulhem, "Empirical comparison of three metrics suites for fault prediction in packages of object-oriented systems: A case study of Eclipse," *Adv. Eng. Softw.*, vol. 42, no. 10, pp. 852–859, Oct. 2011, doi: 10.1016/j.advengsoft.2011.06.001.
[96] K. K. Ganguly, M. S. Siddik, R. Islam, and K. Sakib, "An environment aware learning-based self-adaptation technique with reusable components," *Int. J. Mod. Educ. Comput. Sci.*, vol. 11, no. 6, p. 53, 2019.
[97] R. V Hudli, C. L. Hoskins, and A. V Hudli, "Software metrics for object-oriented designs," in *Proceedings 1994 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 1994, pp. 492–495.
[98] Dart Code Metrics, "Maximum Nesting." 2021. [Online]. Available: https://dartcodemetrics.dev/docs/metrics/maximum-nesting-level
[99] Ndepend, "Code Metrics Definitions." 2021. [Online]. Available: https://www.ndepend.com/docs/code-metrics#ILNestingDepth
[100] M. Genero, E. Manso, A. Visaggio, G. Canfora, and M. Piattini, "Building measure-based prediction models for UML class diagram maintainability," *Empir. Softw. Eng.*, vol. 12, no. 5, pp. 517–549, Sep. 2007, doi: 10.1007/s10664-007-9038-4.
[101] C. López, E. Manso, and Y. Crespo, "Un caso de estudio sobre la identificación de valores umbrales para medidas de código," *XVI Jornadas Ing. del Softw. y Bases Datos.. La Coruna*, pp. 471–485.
[102] S. Kaur and R. Maini, "Analysis of various software metrics used to detect bad

smells," *Int J Eng Sci*, vol. 5, no. 6, pp. 14–20, 2016.
[103] F. Brito e Abreu, "Object-Oriented Software Engineering: Measuring and Controlling the Development Process," Jan. 1997.
[104] A. B. Binkley and S. R. Schach, "Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures," in *Proceedings of the 20th international conference on Software engineering*, 1998, pp. 452–455.
[105] A.-J. Molnar, A. Neamtu, and S. Motogna, "Longitudinal Evaluation of Software Quality Metrics in Open-Source Applications.," in *ENASE*, 2019, pp. 80–91.
[106] F. A. Fontana, V. Ferme, M. Zanoni, and R. Roveda, "Towards a prioritization of code debt: A code smell intensity index," in *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*, 2015, pp. 16–24.
[107] F. Pecorelli, F. Palomba, D. Di Nucci, and A. De Lucia, "Comparing heuristic and machine learning approaches for metric-based code smell detection," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, 2019, pp. 93–104.
[108] R. Malhotra and A. Sharma, "Estimating the threshold of software metrics for web applications," *Int. J. Syst. Assur. Eng. Manag.*, vol. 10, no. 1, pp. 110–125, 2019.
[109] K. A. M. Ferreira, M. A. S. Bigonha, R. S. Bigonha, L. F. O. Mendes, and H. C. Almeida, "Identifying thresholds for object-oriented software metrics," *J. Syst. Softw.*, vol. 85, no. 2, pp. 244–257, Feb. 2012, doi: 10.1016/j.jss.2011.05.044.
[110] F. N. Topuz and others, "Empirical Evidence of the Consequences of Bad Smells in Software," 2022.
[111] M. Zhang, T. Hall, and N. Baddoo, "Code Bad Smells: a review of current knowledge," *J. Softw. Maint. Evol. Res. Pract.*, vol. 23, no. 3, pp. 179–202, Apr. 2011, doi: 10.1002/smr.521.
[112] S. R. Chidamber and C. F. Kemerer, "Towards a metrics suite for object oriented design," in *Conference proceedings on Object-oriented programming systems, languages, and applications*, 1991, pp. 197–211.
[113] C. W. Butler, "Software Analytics of Open Source Business Software," *J. Softw. Eng. Appl.*, vol. 15, no. 5, pp. 150–164, 2022.
[114] M. Shepperd, "A critique of cyclomatic complexity as a software metric," *Softw. Eng. J.*, vol. 3, no. 2, p. 30, 1988, doi: 10.1049/sej.1988.0003.
[115] A. H. Watson, D. R. Wallace, and T. J. McCabe, *Structured testing: A testing methodology using the cyclomatic complexity metric*, vol. 500, no. 235. US Department of Commerce, Technology Administration, National Institute of~…, 1996.
[116] I. Heitlager, T. Kuipers, and J. Visser, "A Practical Model for Measuring Maintainability," in *6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007)*, Sep. 2007, pp. 30–39. doi: 10.1109/QUATIC.2007.8.
[117] S. Talwani and J. Singla, "Comparison of various fault tolerance techniques for scientific workflows in cloud computing," in *2019 International conference on machine learning, big data, cloud and parallel computing (COMITCon)*, 2019, pp. 454–459.
[118] A. Gandhi, "Big-Data-Analytics-on-Amazon-Customer-Reviews." [Online]. Available: https://github.com/AmiGandhi/Big-Data-Analytics-on-Amazon-Customer-Reviews-Using-Hadoop-and-MapReduce

[119] M. Mehrotra, “TwitterAnalyser_StormBoi.” [Online]. Available: https://github.com/mehrotrasan16/TwitterAnalyser_StormBoi
[120] K. Koziol, “OpenFDA BigData Pipeline.” [Online]. Available: https://github.com/koziolk/openfda-bigdata-pipeline
[121] B. Marciani and M. Porretta, “Crimegraph.” [Online]. Available: https://github.com/braineering/crimegraph
[122] I. C. Society, “IEEE Standard Classification for Software Anomalies.” IEEE Std, 1993.
[123] CodeMR Team, “CodeMR Static Code Analyser.” 2018. [Online]. Available: https://marketplace.eclipse.org/content/codemr-static-code-analyser
[124] Applied Research in Systems Analysis ARiSA, “VizzMaintenance.” 2014. [Online]. Available: http://www.arisa.se/products.php?lang=en
[125] Eclipse Foundation, “Eclipse IDE 2018-09 R Packages.” [Online]. Available: https://www.eclipse.org/downloads/packages/release/2018-09/r
[126] Microsoft, “Visual Studio Code,” 2022. https://visualstudio.microsoft.com/es/
[127] Python Software Foundation, “Python,” 2022. https://www.python.org
[128] Microsoft, “Excel,” 2022. https://www.microsoft.com/es-mx/microsoft-365/excel
[129] R. W. White *et al.*, “Early identification of adverse drug reactions from search log data,” *J. Biomed. Inform.*, vol. 59, pp. 42–48, Feb. 2016, doi: 10.1016/j.jbi.2015.11.005.
[130] U.S. Department of Health and Human Services Food and Drug Administration, “Drug Adverse Event Overview.” [Online]. Available: https://open.fda.gov/apis/drug/event/
[131] A. Boucher and M. Badri, “Software metrics thresholds calculation techniques to predict fault-proneness: An empirical comparison,” *Inf. Softw. Technol.*, vol. 96, pp. 38–67, 2018.
[132] R. C. Martin, *Agile Software Development: Principles, Patterns, and Practices*. Pearson Education, 2003.
[133] F. B. Abreu and R. Carapuça, “Object-oriented software engineering: Measuring and controlling the development process,” in *Proceedings of the 4th international conference on software quality*, 1994, vol. 186.
[134] J. Al-Ja’Afer and K. Sabri, “Metrics for object oriented design (MOOD) to assess Java programs,” *King Abdullah II Sch. Inf. Technol. Univ. Jordan, Jordan*, 2004.
[135] J. Gorman, “OO design principles \& metrics,” *Online verfügbar unter http//www. parlezuml. com/metrics/OO\% 20Design\% 20Principles\% 20\&\% 20Metrics. pdf, zuletzt geprüft am*, vol. 15, p. 2009, 2006.
[136] M. V. Kosti, A. Ampatzoglou, A. Chatzigeorgiou, G. Pallas, I. Stamelos, and L. Angelis, “Technical debt principal assessment through structural metrics,” in *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2017, pp. 329–333.
[137] G. Kaur and B. Singh, “Improving the quality of software by refactoring,” in *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*, Jun. 2017, pp. 185–191. doi: 10.1109/ICCONS.2017.8250707.
[138] F. Rabbi, S. S. Hossain, and M. M. S. Arefin, “SCMA: A Lightweight Tool to Analyze Swift Projects”.
[139] M. Thapaliyal and G. Verma, “Software defects and object oriented metrics-an empirical analysis,” *Int. J. Comput. Appl.*, vol. 9, no. 5, pp. 41–44, 2010.

[140] A. Kumar, “Measure Code Quality using Cyclomatic Complexity.” 2022. [Online]. Available: https://vitalflux.com/cyclomatic-complexity-used-measure-code-quality/
[141] J. Ramirez, G. Gil, and C. Reyes, “Umbrales sugeridos para promedios de métricas de diseño de una aplicación en Java,” 2015.
[142] T. Nguyen, P. Vu, and T. Nguyen, “Recommending exception handling code,” in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2019, pp. 390–393.
[143] T. Montenegro, H. Melo, R. Coelho, and E. Barbosa, “Improving developers awareness of the exception handling policy,” in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018, pp. 413–422.
[144] J. S. Challa, A. Paul, Y. Dada, V. Nerella, P. R. Srivastava, and A. P. Singh, “Integrated software quality evaluation: a fuzzy multi-criteria approach,” *J. Inf. Process. Syst.*, vol. 7, no. 3, pp. 473–518, 2011.
[145] H. Shah, C. Gorg, and M. J. Harrold, “Understanding exception handling: Viewpoints of novices and experts,” *IEEE Trans. Softw. Eng.*, vol. 36, no. 2, pp. 150–161, 2010.
[146] M. H. Bejarano and L. E. B. Rey, *Estructuras de datos: Fundamentación práctica*. Ediciones de la U, 2021.

Anexos

Este capítulo corresponde a información complementaria y asociada a las características de una falla, la especificación de las herramientas utilizadas para lograr el proceso de medición de las métricas para código fuente relacionadas a tolerancia a fallas y su descripción detallada. Además, se presentan los 4 casos de estudio que fueron evaluados con la propuesta de solución resultante.

Anexo 1. Atributos de una falla y sus detalles

A continuación, la representación gráfica de los atributos correspondientes a una falla (ver *Ilustración 34).*

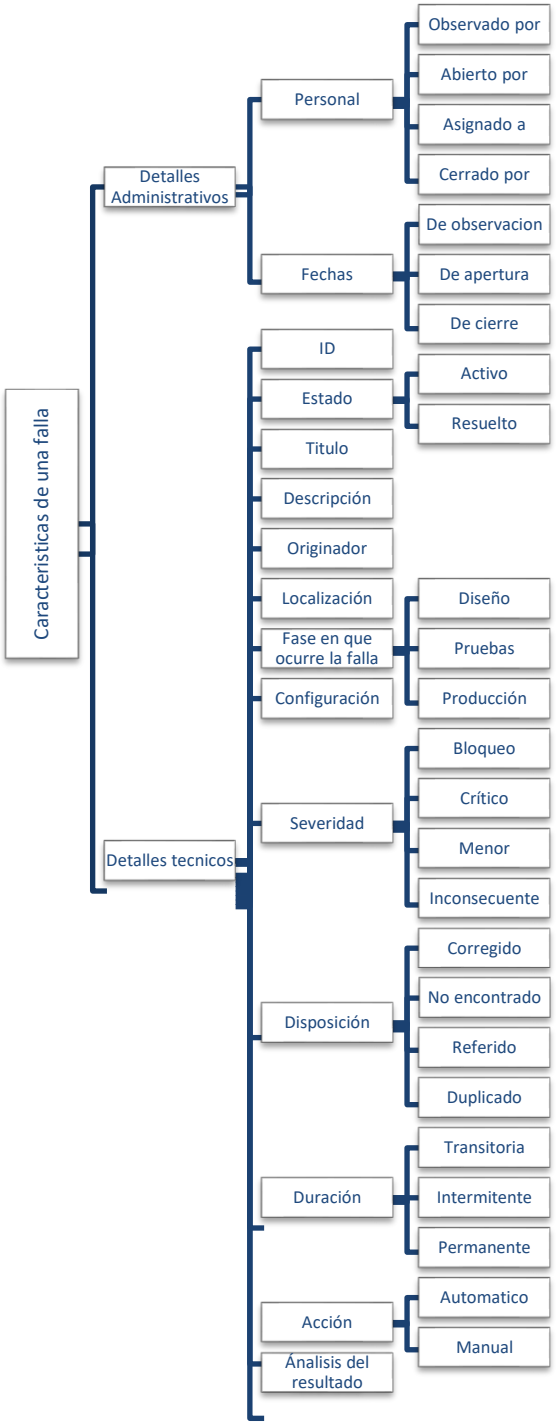


Ilustración 34. Atributos de una falla [122][60].

La *Tabla 20* presenta los valores que puede tomar cada uno de los atributos (administrativas y técnicas)y descripciones que conforman una falla resultado del análisis de los trabajos de IEEE [122] y Lackovic *et.al* [60] con la finalidad de conocer y facilitar el control de las fallas que se presentan en un sistema Big data y en caso de ser necesario aplicar las acciones para resolverlas. Tener información relacionada a la forma de resolver cierto problema sirve como medio para identificar las deficiencias que comúnmente presenta el sistema, así como ahorrar tiempo en la reparación de dicha falla.

Nota: Lo resaltado en color gris corresponde a los detalles administrativos de una falla, el resto son detalles técnicos.

Tabla 20. Atributos de una falla, adaptado de [122][60].

No.	Atributos de falla	Valor/descripción			
1	ID	Identificador único de la falla			
2	Estado	Activa: Se prevén acciones futuras		Resuelta: No se prevén más acciones	
3	Detalles del personal	Observado por Nombre:	Abierto por Nombre:	Asignada a Nombre:	Cerrado por Nombre:
4	Detalles de fechas	Fecha de observación	Fecha de apertura		Fecha de cierre
5	Titulo	Breve descripción de la falla para fines de informes resumidos.			
6	Descripción	Descripción completa del comportamiento anómalo y las condiciones bajo las cuales ocurrió, incluida la secuencia de eventos y/o acciones del usuario que precedieron a la falla.			
7	Originador	Se refiere al componente dentro del sistema responsable de haber causado la falla.			
8	Localización de la falla	Se refiere a la parte del sistema donde se manifiesta una falla.			
9	Fase en que ocurre la falla	Se refiere al ciclo de vida del flujo de trabajo (diseño, prueba o producción) durante el cual se introdujo una falla en el sistema.			
		Diseño	Pruebas	Producción	
10	Configuración	Detalles de configuración, incluidos los identificadores de versión y producto relevantes			

Continuación, ... Tabla 20. Atributos de una falla, adaptado de [122][60].

No.	Atributos de falla	Valor/descripción				
11	**Severidad**	**Bloqueo**: Las pruebas se inhiben o suspenden en espera de la corrección o identificación de una solución alternativa adecuada	**Critico:** Las operaciones esenciales se interrumpen inevitablemente, la seguridad se pone en peligro y la seguridad se ve comprometida	**Importante:** Las operaciones esenciales se ven afectadas, pero pueden continuar	**Menor:** Las operaciones no esenciales se interrumpen	**Inconsecuente:** Sin impacto significativo en las operaciones.
12	**Análisis**	Resultados finales del análisis causal sobre la conclusión de la investigación de fallas.				
13	**Disposición**	**Corregido**: El defecto fue corregido/eliminado.	**No encontrado:** No se encontró ningún defecto. La falla no se pudo reproducir o el comportamiento informado es realmente el comportamiento previsto.	**Referido:** El defecto está contenido en el activo de otra organización y se remitió a esa organización para su corrección.	**Duplicado:** El informe de defectos es un duplicado	
14	**Duración**	La duración se refiere a la frecuencia de una falla y está relacionada con el tiempo de incidencia e incluye: fallas transitorias, intermitentes y permanentes.				
		Transitoria	Intermitente	Permanente		
15	**Acción**	Como consecuencia de una falla, se puede realizar una acción o un conjunto de acciones en el lado del cliente, y se puede realizar de forma automática o manual.				
		Automático		Manual		

Anexo 2. Herramientas utilizadas

Los valores de las métricas propuestas en la taxonomía se obtuvieron realizando una evaluación del código estático de cada uno de los casos de estudio seleccionados anteriormente. El proceso de revisión del código se realizó en 3 etapas y se describen a continuación:

1. Como primera opción, se añadieron los complementos "*plugin"* para agregar facilidades al IDE Eclipse, fueron utilizados para analizar los archivos que conforman el proyecto y obtener un reporte, del cual solo se toman los valores correspondientes a métricas consideradas en la taxonomía de tolerancia a fallas.

    a. *Plugin* **CodeMR Static Code Analyser** [123], con esta herramienta se calcularon las métricas: A, CA, CBO, CE, DIT, DN, I, LCOM, LOC, NEST, NMO, NA, NOC, NOM/NM, NPAR, RFC, V(G), WMC.
    b. *Plugin* **VizzMaintenance 2.0** [124], se utilizó para detalles de las métricas: DAC y MPC.

2. Manual, este proceso consistió en la revisión los elementos del proyecto (paquetes, clases y métodos) y de acuerdo a la descripción de la métrica se realiza su cálculo, esta técnica se empleó para las métricas: NOPA, NOPM, ME.
3. Finalmente, en vista del esfuerzo y tiempo que se requiere para realizar la medición, se definió una herramienta simple (desarrollada con el lenguaje Python) que se puede utilizar para evaluar las métricas MHF, AHF.

Los resultados obtenidos de cada una de las etapas fueron capturados en la herramienta Excel. A continuación, en *Tabla 21*, se presentan las herramientas utilizadas, seguido del proceso de instalación.

Tabla 21. Herramientas utilizadas

Herramientas utilizadas		
IDE (Eclipse) [125] 	**Plugin (CodeMR)** [123] 	**Plugin (VizzMaintenance)** [124] 
Editor de código (Visual Studio Code)[126] 	**Lenguaje de programación (Python)** [127] 	**Hoja de calculo (Excel)** [128] 

Detalles de herramientas utilizadas

Se utilizó el IDE Eclipse, en su versión 2018-09 (4.9.0)[125] → Ir a Eclipse IDE for Java EE Developers, se descarga, descomprime JAR y ejecuta el archivo eclipse.exe (para su correcta instalación se deben seguir las indicaciones). La *Ilustración 35*, corresponde al IDE funcionando correctamente

Ilustración 35. Eclipse versión 2018-09

El plugin **CodeMR Static Code Analyser**, es una herramienta de análisis de código estático y de calidad de software en varios idiomas (Java y C++), para extraer valores de diversas métricas enfocadas a : Métricas de proyecto, Métricas de paquete, Métricas de clase, Métricas de método, donde para este modelo, solo se toman algunas de las incluidas en la taxonomía propuesta en la sección: 4.2.4 : A, CA, CBO, CE, DIT, DN, I, LCOM, LOC, NMO, NA, NOC, NOM/NM, NPAR, RFC, V(G), WMC.

Para instalar esta herramienta, ir al menú Ayuda → seleccionamos la opción Elipse Marketplace, como se muestra a continuación en la *Ilustración 36*.

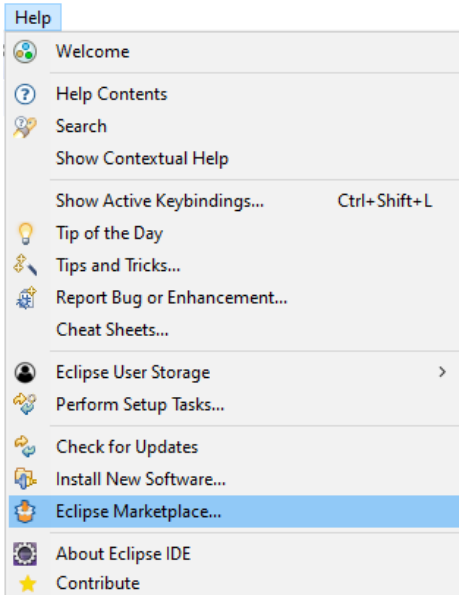


Ilustración 36. Marketplace Eclipse

En la ventana de Marketplace *(Ilustración 37)* → Ir a la sección buscar → ingresar CodeMR o la url [123] que corresponde al plugin deseado → elegir la opción install → Aceptar términos e instalar.

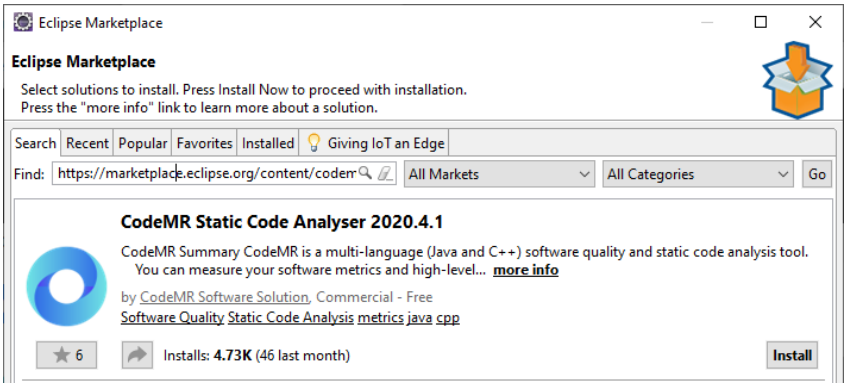


Ilustración 37. Plugin CodeMR

Para ejecutar el plugin y obtener resultados de las métricas: De los proyectos cargados en el IDE eclipse, seleccionar el que desea evaluar→ click derecho y navegar a la opción CodeMR Analysis→ Extract Model. A continuación (ver *Ilustración 38*), seleccionar las siguientes características.

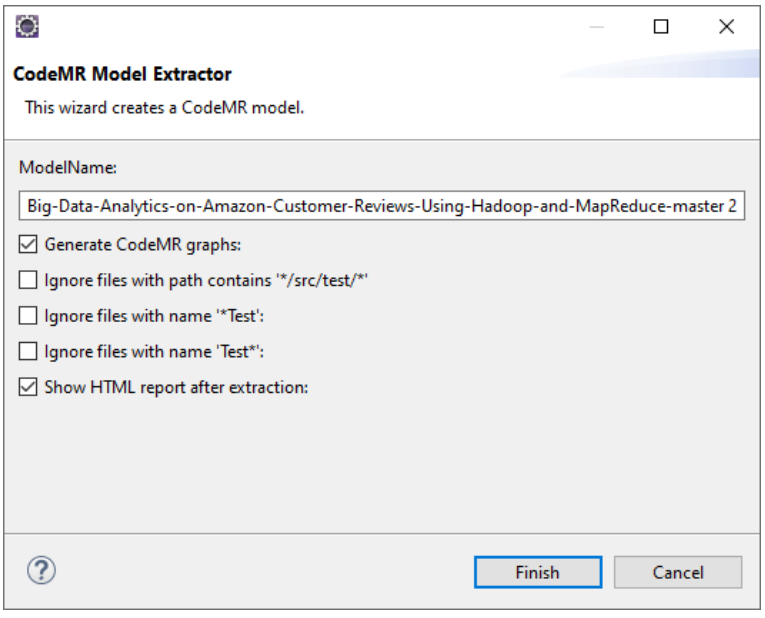


Ilustración 38. Características para generar reporte CodeMR

Al dar clic en finalizar se procesa la información, en la *Ilustración 39*, se observan los resultados obtenidos, correspondientes al proyecto evaluado.

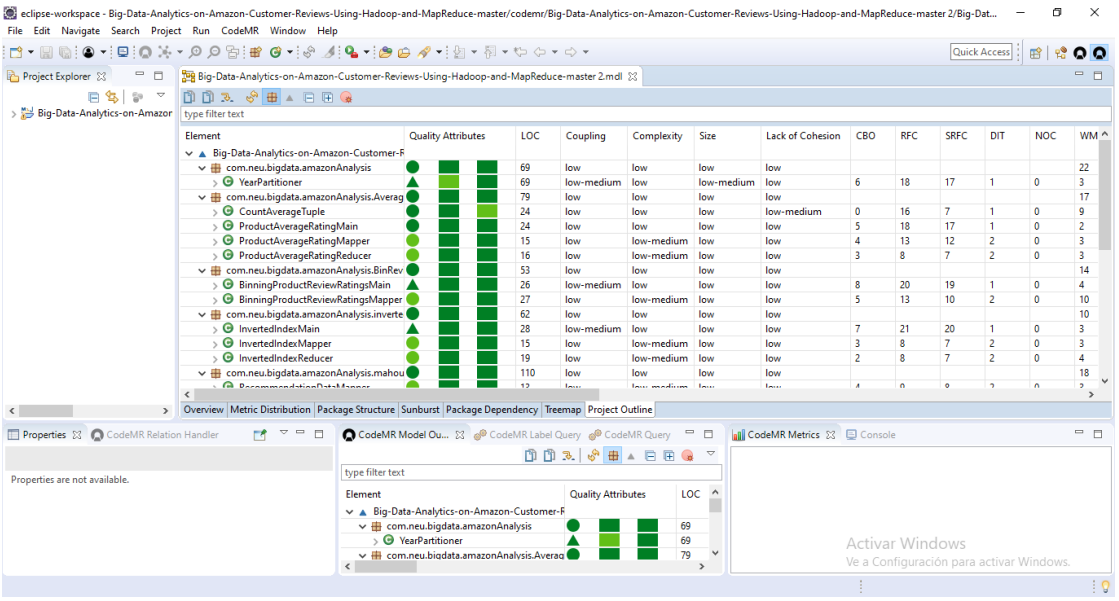


Ilustración 39. Resultados para métricas por CodeMR

Para el uso del plugin **VizzMaintenance 2.0,** que es un complemento de Eclipse compatible con Eclipse 3.5 o superior para calcular 17 métricas de calidad de software discutidas en la literatura es necesario descargarlo de: http://www.arisa.se/products.php?lang=en [124] y registrarse (*Ilustración 40*) para poder usarlo durante 90 días.

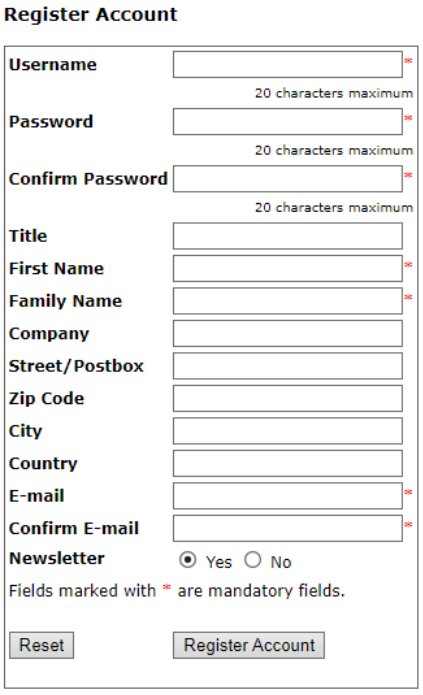


Ilustración 40. Registro

Una vez realizado el registro, se obtendrán credenciales para acceder (ver *Ilustración 41*) a la plataforma por lo que es necesario autenticarse como se observa a continuación.

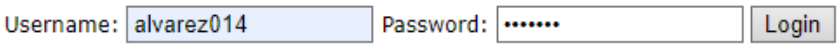


Ilustración 41. Ventana de autenticación de la plataforma VizzMaintenance 2.0

Ahora, es posible utilizar el plugin en el IDE Eclipse, para ello es necesario copiar la licencia (*Ilustración 42*) proporcionada por la plataforma.

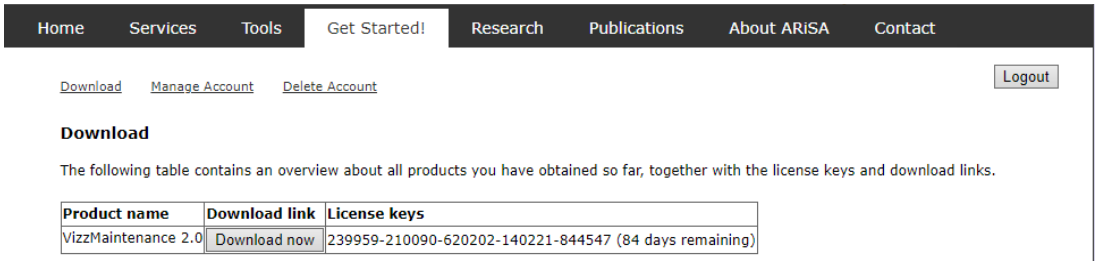


Ilustración 42. Licencia del plugin VizzMaintenance 2.0

Para hacer uso de la herramienta en Eclipse ver *Ilustración 43*, Ir al menú Window→ Show View→ Other y verificamos se encuentre VizzMaintenance:

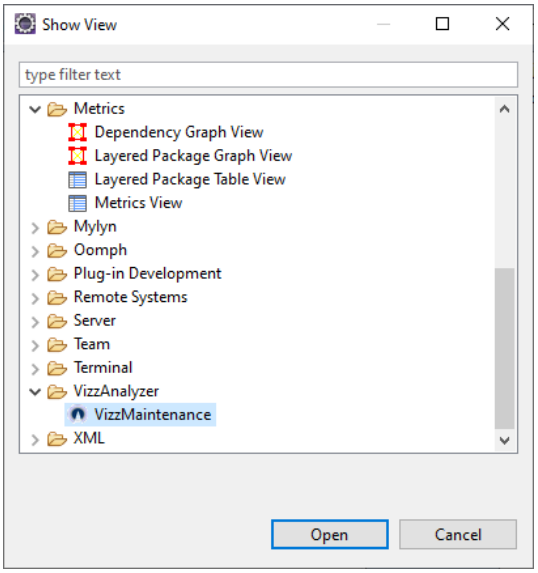


Ilustración 43.Plugin VizzMaintenance

Al dar click en open, abrirá el siguiente panel (ver *Ilustración 44*), donde se debe ir al menú y seleccionar Register Licence.

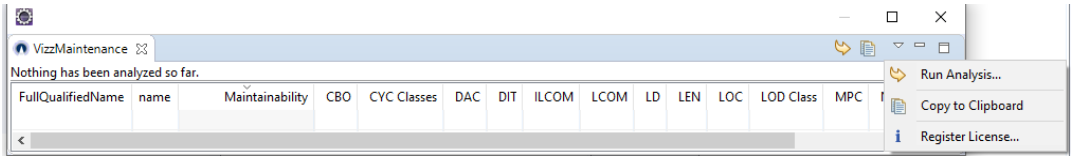


Ilustración 44. Agregar Licencia

Finalmente, para obtener resultados se debe seleccionar el proyecto de interés (ver *Ilustración 45*), dar click al símbolo y se generara un reporte.

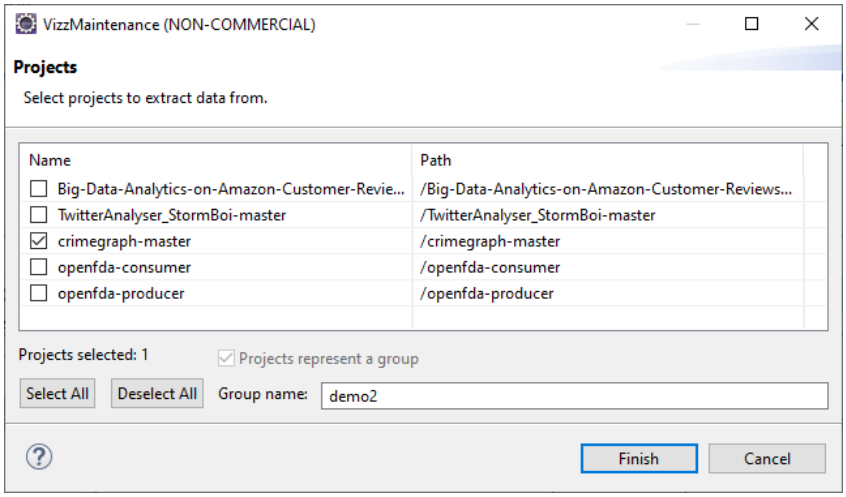


Ilustración 45. Selección de proyecto

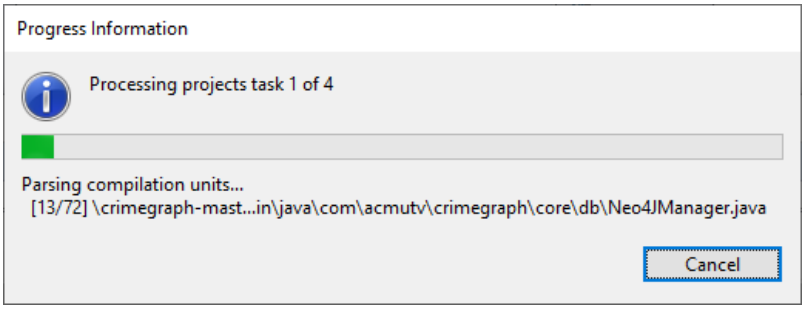


Ilustración 46. Generación de reporte

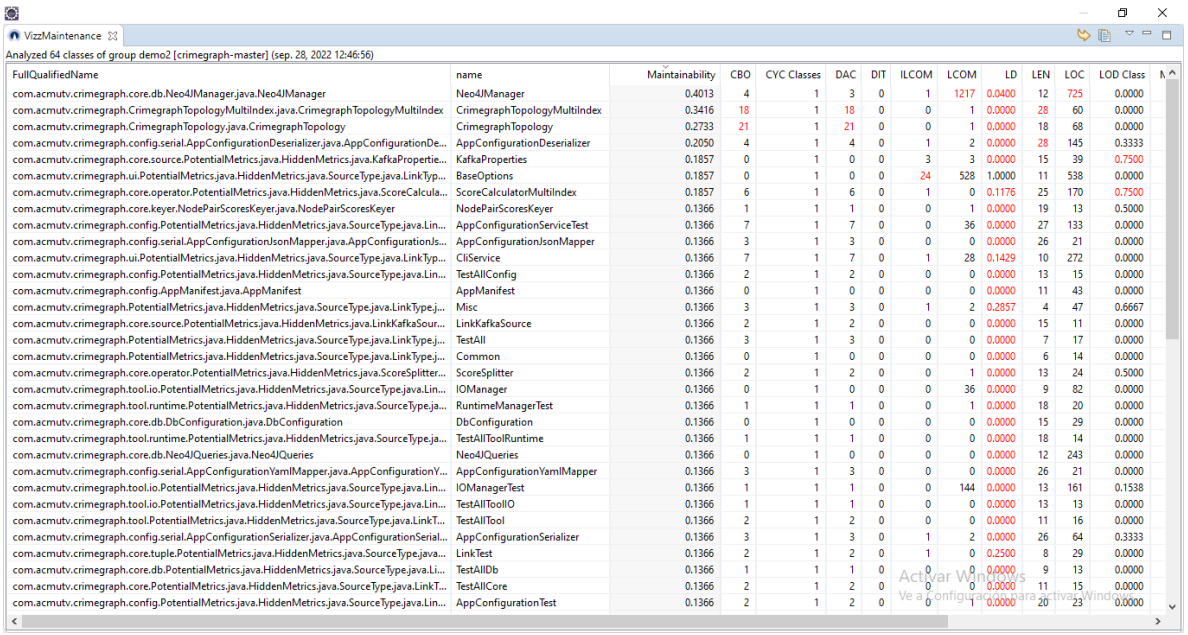


Ilustración 47. Reporte de métricas VizzMaintenance

Una vez realizados los cálculos de las métricas, sus resultados fueron capturados en un documento **Excel** (ver *Ilustración 48*), donde para cada métrica se asigna un *identificador*, el

valor obtenido, y además esta es clasificada por *tipo de elemento* (paquete, método y clase) y *umbral obtenido* (bueno/común, normal/casual y malo/poco común), detalles que permitirán conocer si el resultado del cálculo se encuentra dentro del rango permitido y además identificarlo fácilmente del resto de los elementos.

"Modelo de Calidad para la Medición de Tolerancia a Fallas en Sistemas de Big Data"

CASO DE ESTUDIO #1: Modeling Adverse Drug Reactions Github

OBJETIVO: OpenFDA Big Data Pipeline permite la recopilación, el procesamiento y la presentación en tiempo real de datos sobre eventos adversos de medicamentos de la BD de openFD.

DETALLES	MÉTRICAS		
No. Paquetes: 12	PAQUETE	CLASE	MÉTODO
No. Clases: 28	A, CA, CE, DN, I	AHF,CBO, DAC, DIT, LCOM, LOC, MPC, NMO	LOC, NEST, NPAR, V(G) (4)
No. Metodos: 18	MHF (6)	NA, NOC, NOM, NOPA, NOPM, RFC, WMC, ME (16)	
Lineas de codigo totales: 305	RESULTADOS DE MÉTRICAS EVALUADAS		

		CODIGO ESTATICO																								EJECUCIÓN		
ID	ELEMENTOS REVISADOS:	A	AHF	CA	CBO	CE	DAC	DIT	Dn/D	I	LCOM	LOC	MHF	MPC	NMO-NORM	NEST	NAN/OF	NOC	NOM	NOPA	NOPM	NPAR	RFC	V(G)	WMC	ME	MTTF	MTTR	MTBT
1	consumer	0		0		1			0.0	1.0		8	10												1				
2	OpenfdaConsumerApplication		N/A		1		0	1			0.0	6		0	0		0	0	0	0	1		2		1				
3	main ( )				1							2				1						1		1		0			
4	OpenfdaConsumerApplicationTests		N/A		0		0	1			0.0	2		0	0		0	0	0	0	4		0		0				
5	config	0		0		1			0.0	1.0		33	10												2				
6	KafkaConsumerConfig		0.5		8		1	1			1.0	33		0	0		4	0	0	0	2		10		2				
7	consumerFactory ( )				4							13				1						0		1		0			
8	drugAdverseEventKafkaListenerContainerFactory ( )				4							9				1						0		1		0			
9	kafka	0		0		1			0.0	1.0		16	N/A												1				
10	DrugAdverseEventKafkaConsumer		0.0		2		3	1			0.0	16		1	0		1	0	1	0	0		4		1				
11	handleMessage ( )				2							11				1						3		1		0			
12	mongo	0.7		1		3			0.4	0.8		24	10												2				
13	DrugAdverseEventDocument		0.3		1		0	1			0.0	18		0	0		12	0	0	0	0		0		0				
14	DrugAdverseEventMapper		N/A		2		2	1			0.0	5		0	0		0	0	2	0	0		2		2				
15	map(Collection)				0							1				1						1		1		0			
16	map(DrugAdverseEvent)				2							1				1						1		1		0			
17	DrugAdverseEventRepository		N/A		0		1	1			0.0	1		0	0		0	0	0	0	0		0		0				
18	kafka	0		1		0			1.0	0.0		16	N/A												0				
19	DrugAdverseEvent		0.2		0		0	1			0.0	16		0	0		11	0	0	0	0		0		0				
	Metricas mas utilizadas [1]	Bueno/Común→									Normal/Casual→										Malo/Poco común→								

NO APLICA

Ilustración 48. Reporte de resultados (plantilla)

Anexo 3. Casos de estudio – Descripción

A continuación, se describe el enfoque de cada uno de los proyectos utilizados como casos de estudio.

BIG-DATA ANALYTICS ON AMAZON CUSTOMER REVIEWS

El análisis de Amazon es una parte muy importante cuando se trata de encontrar una forma eficiente de obtener información sobre las opiniones de los clientes sobre diferentes productos. Por lo tanto, este proyecto tiene como objetivo principal analizar grandes datos y producir un resultado informativo sobre las opiniones de los clientes sobre el producto Cámara presente en Amazon utilizando la arquitectura Hadoop, MapReduce y Apache-Pig.

A continuación en *Ilustración 49*, se observa la estructura del proyecto revisado, Gandhi [118].

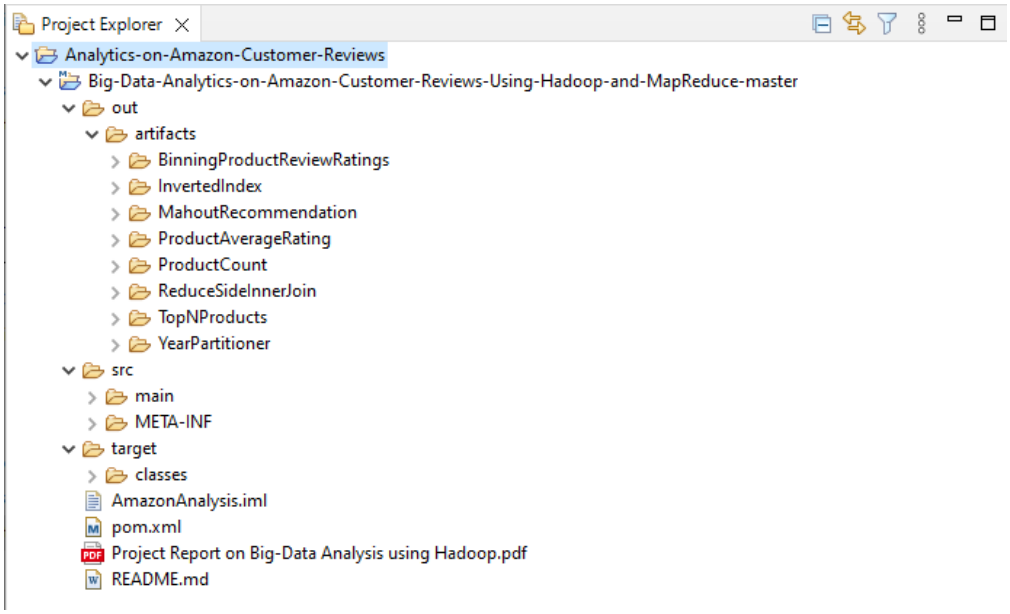


Ilustración 49. Estructura de archivos del proyecto

TWITTER ANALYSER STORMBOI

Twitter es un servicio de redes sociales en línea que proporciona una plataforma para enviar y recibir tweets de usuarios y el hashtag se utiliza para categorizar tweets por palabra clave agregando # antes de la palabra clave relevante.

Se implementó un algoritmo de conteo con pérdida para determinar los hashtags de mayor tendencia utilizando la API de Twitter para obtener un flujo continuo de tweets. Este es un proyecto (ver *Ilustración 50*) que consiste en encontrar el hashtag más utilizado por tema en tiempo real.

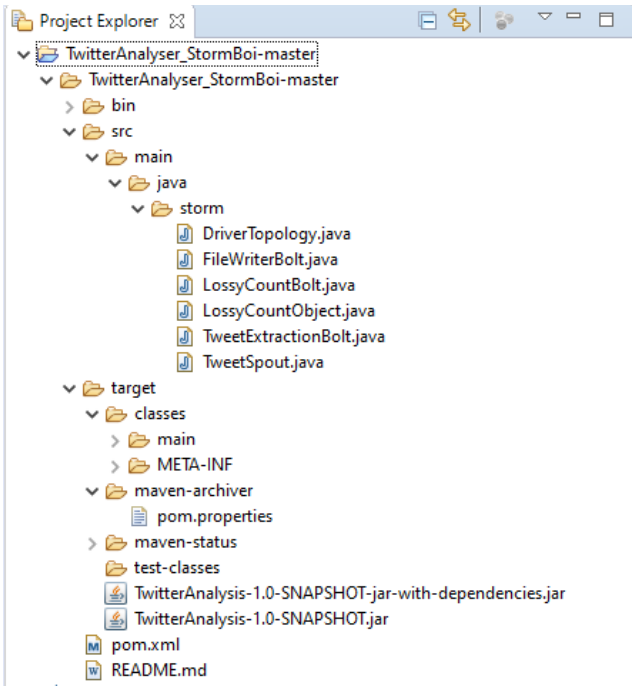


Ilustración 50. Estructura de archivos del proyecto.

MODELING ADVERSE DRUG REACTIONS

La identificación oportuna y precisa de las reacciones adversas a los medicamentos (RAM) después de la aprobación del fármaco es un desafío de salud pública grave y persistente [129]. La API de eventos adversos de medicamentos de openFDA devuelve datos que se recopilaron del Sistema de informes de eventos adversos (FAERS) de la FDA, en el caso de los medicamentos, esto incluye los efectos secundarios graves de los medicamentos, los errores en el uso del producto, los problemas de calidad del producto y las fallas terapéuticas de los medicamentos recetados o de venta libre y los medicamentos administrados a pacientes hospitalarios o en centros de infusión para pacientes ambulatorios. Los informes de eventos adversos son realizados directamente de profesionales de la salud (como médicos, farmacéuticos, enfermeras y otros) y consumidores (como pacientes, familiares, abogados y otros) [130]. A continuación, en la *Ilustración 51*, se presenta la estructura del proyecto revisado [120].

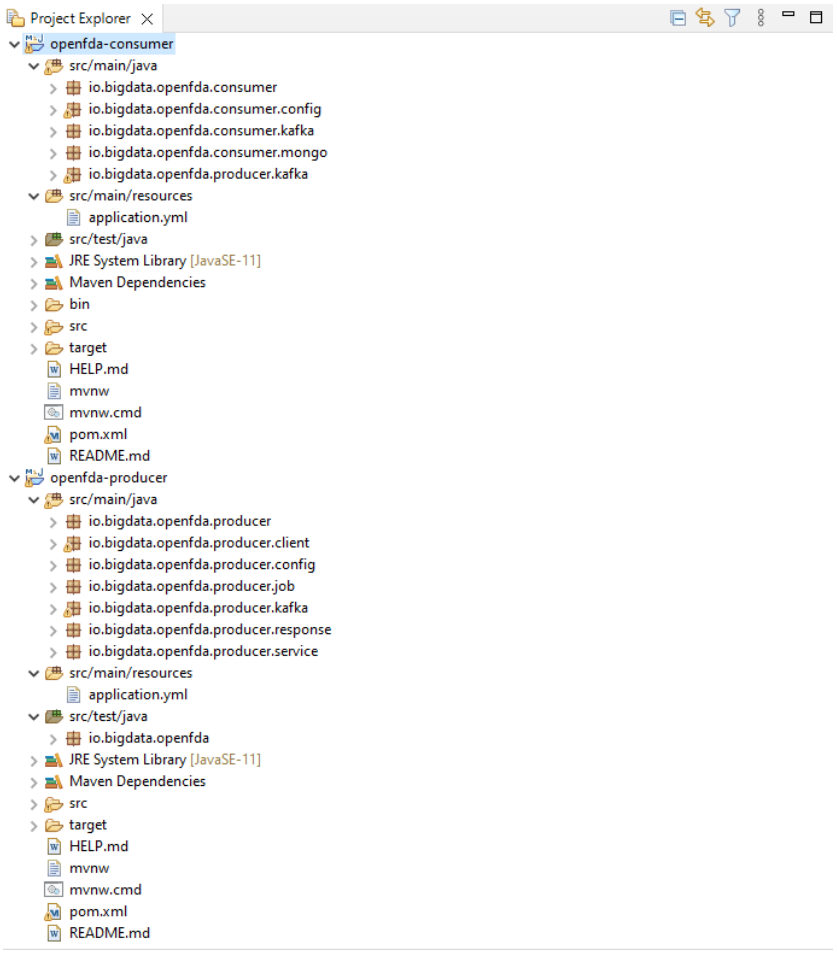


Ilustración 51.Estructura de archivos del proyecto

CRIMEGRAPH

Las redes criminales pueden generar rápidamente una cantidad masiva y compleja de datos, algunos ejemplos bien conocidos del escenario criminal general monitoreado diariamente son: relaciones entre presuntos terroristas, los registros de escuchas telefónicas, las transacciones monetarias, el tráfico de armas y drogas. Sin embargo, la red criminal es un caso especial de red social. La capacidad de descubrir patrones de interés (vínculos ocultos y potenciales) casi en tiempo real es de suma importancia para la detección y predicción de enlaces en una red criminal en evolución. Se desarrollo una aplicación procesamiento de flujo de datos flexible, en el framework Apache Flink para explotar un enfoque de procesamiento de flujo para extraer información de interés tan pronto como se generan los datos, en la era de Big Data, estos conjuntos de datos brindan una valiosa oportunidad para la lucha contra el crimen organizado basada en datos. A continuación, (ver *Ilustración 52*)se presenta la estructura del proyecto.

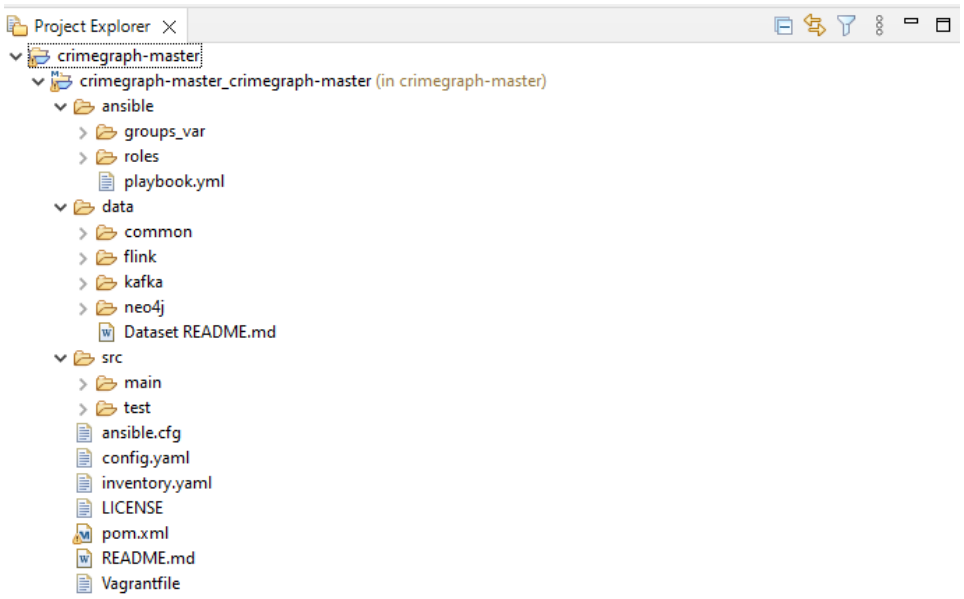


Ilustración 52. Estructura de archivos del proyecto

Anexo 4. Umbrales de métricas seleccionadas

A continuación, se presenta una breve descripción de cada una de las métricas utilizadas para la medición de los sub-atributos relacionados a tolerancia a fallas, así como el umbral permitido (valor), extraídos de diversos trabajos de investigación. El valor que obtenga la métrica aplicada, será clasificado de acuerdo a los umbrales como: 1) bueno/común, 2) normal/casual y 3) malo/poco común, lo que permitirá identificar qué aspectos cumplen satisfactoriamente con el atributo de tolerancia a fallas, y las deficiencias que presenta el sistema Big Data, para más detalles consultar la *Tabla 7 (página 33)*.

En el trabajo [131], se menciona que los umbrales propuestos en su investigación no se pueden generalizar a todos los proyectos, ya que diferentes estilos de programación y tamaños de sistemas probablemente generarán diferentes distribuciones de métricas de código fuente, por lo que algunos valores de umbral quedarán obsoletos para ciertos sistemas. Por supuesto, los valores de umbral de algunos sistemas podrían reutilizarse para otros sistemas, pero esto no se puede generalizar. Según algunos estudios [2, 6], los umbrales de las métricas deben calcularse para cada proyecto e incluso para cada versión de un sistema [131].

(A)-Abstracción

La abstracción, pertenece a las ocho métricas de Robert C. Martin [132] que calculan diferentes características de un paquete: acoplamiento eferente (Ce), acoplamiento aferente (Ca), inestabilidad (I), número de clases abstractas (Na), número de clases (Nc), abstracción (A), la distancia de la secuencia principal (D) y el normalizado distancia de la secuencia principal (Dn).

La abstracción satisface el principio de diseño "Abierto/Cerrado" en el que las clases son flexibles para extenderse sin modificaciones [70]. El rango de A es [0,1], donde cero indica que el paquete no tiene partes abstractas [71] y por lo tanto 1 indica un paquete abstracto.

Las dos medidas de tamaño con las que se miden la abstracción son N_c y N_a. La abstracción A del paquete es:

$$A = \frac{N_a}{N_c} \quad (2)$$

Donde: N_c es el número de clases en el paquete y N_a es el número de clases o interfaces abstractas en el paquete.

Para este trabajo se interpreta un valor de abstracción de 0 como malo/poco común y un valor de 1 como bueno/común.

(MHF)-Factor de ocultación del método

MHF forma parte del conjunto de métricas propuestas para abordar la encapsulación (mecanismo estructural básico en el paradigma de la orientación a objetos [74]), proceso de ocultar todos los detalles de un objeto que no contribuye a sus características esenciales [72], MHF se puede aplicar a un conjunto de clases (Class Cluster) existente o cualquier combinación de ellos [133].

Se ha demostrado empíricamente que cuando se incrementa MHF, la densidad de defectos y el esfuerzo necesario para corregirlos debería disminuir [74]. De acuerdo al trabajo [133] su valor va de 0 a 1, un valor de MHF alto indica que todos los métodos son privados, revelando poca funcionalidad y haciendo imposible reutilizar dichos métodos . Por el contrario, MHF con un valor bajo indica que todos los métodos son públicos, lo que significa que la mayoría de los métodos no están protegidos [72]. Por lo tanto, debido a lo mencionado anteriormente, una clase que posee todos sus métodos públicos y/o privados es indeseable, por lo que para este trabajo, los valores considerados se presentan en tres rangos: Bueno/Común (MHF = 0.8 [76]), Regular/Casual (0.6< MHF ≤0.79) y Malo/Poco común (0< MHF ≤0.59, 0.79 ≤ MHF ≤1). El Factor de ocultación del método se calcula [77]:

$$MHF = \frac{\sum_{i=1}^{TC}\left[\sum_{i=1}^{Nd(ei)}(1-V(N_{mi})\right]}{\sum_{i=1}^{TC} N_d(E_i)} \quad (3)$$

Donde: TC = Número total de clases, $N_d(E_i)$ = Numero de métodos definidos en E_i, $V(N_{mi})$ = Valor de visibilidad del método para todos los E_i, un valor de método privado = 1, público = 0, protegido = Tamaño del árbol de herencia / Número de clases.

(AHF)-Factor de ocultación de atributos

AHF es la medida de invisibilidades de los atributos en una clase, idealmente esta métrica debe de ser siempre 100% implementada, intentando ocultar todos los atributos para que solo sean accedidos mediante los métodos de clase correspondientes [72]. Los valores muy bajos de AHF deberían llamar la atención de los diseñadores (no se debe emplear atributos públicos, ya que se considera que violan los principios de encapsulación al exponer

la implementación de las clases [74] y teniendo atributos desprotegidos [72]). En general, a medida que aumenta la AHF, disminuye la complejidad del programa [134].

Para mejorar el rendimiento, en ocasiones se evita el uso de métodos que acceden o modifican atributos (métodos get/set) accediendo a ellos directamente. Esta práctica debe extremarse la prudencia y evaluar si realmente los pros son mayores que los contras [74].

El valor de AHF va de 0 a 1 de acuerdo al trabajo [75], por lo que los valores considerados, se presentan en tres rangos: Bueno/Común (AHF=1), Regular/Casual (0.79< MHF ≤0.99 [76]) y Malo/Poco común (≤0 MHF ≤0.79[76]). El Factor de ocultación de atributos se calcula [77]:

$$AHF = \frac{\sum_{i=1}^{TC}\left[\sum_{i=1}^{Ad(ei)}(1-V(A_{mi})\right]}{\sum_{i=1}^{TC} A_d(E_i)} \quad (4)$$

Donde: TC = Número total de clases, $A_d(E_i)$ = Numero de atributos definidos en E_i, $V(A_{mi})$ = Un valor de atributo privado = 1, público = 0, protegido = Tamaño del árbol de herencia / Número de clases.

(CBO)- Acoplamiento entre clases de objetos

Esta métrica revela la dependencia de una clase sobre otras clases en el diseño, mediante el mecanismo de transmisión de mensajes o la herencia y se estima sumando el número de clases distintas no relacionadas con la herencia junto con otras clases [67].

Un objeto de una clase está acoplado a otro, si los métodos de una clase usan métodos o atributos de la otra [79], un CBO alto es indeseable debido a que el acoplamiento excesivo entre clases de objetos es perjudicial para el diseño modular y evita la reutilización, cuanto más independiente sea una clase, más fácil será reutilizarla en otra aplicación. Para promover la encapsulación y mejorar la modularidad las parejas de clases entre objetos deben mantenerse al mínimo, cuanto mayor sea el número de parejas, mayor será la sensibilidad a los cambios en otras partes del diseño, y por lo tanto el mantenimiento es más difícil [80]. Se ha encontrado que un alto acoplamiento indica propensión a fallas, por lo que se requiere de pruebas rigurosas [81]. Un CBO>14 se considera alto, el valor deseable para la métrica CBO en el trabajo [87] va de 0 a 8.

Definición (Medida CBO) [79]:

$$CBO(c) = |\{d \in C - \{c\} \mid usa\ (c,d) \vee usa\ (d,c)\} \quad (5)$$

Donde: La clase *c* está acoplada a la clase *d* si usa *d* o está siendo usada por *d*. La definición de *usos* de predicados *(c, d)* exige que tanto el método de uso de la clase *d* como el método o atributo *usado* de la clase *d* se implementen en sus clases [79].

A continuación, un ejemplo grafico de la métrica CBO (*Ilustración 53*).

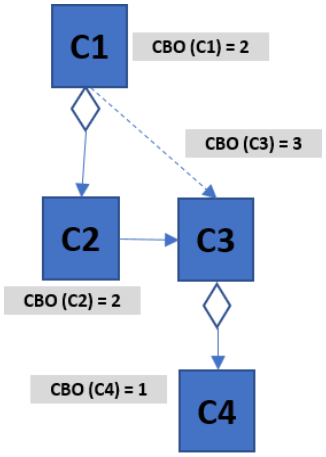


Ilustración 53. Representación gráfica de CBO

Para este trabajo se interpreta un valor de abstracción de 0 como malo/poco común y un valor de 1 como bueno/común.

Para este modelo un valor en CBO≤3 se considera bueno/común y CBO≥9 normal/casual, como se propone en el trabajo [82].

(Ca)-Acoplamiento aferente

CA mide el número de clases en otros paquetes que dependen de las clases dentro del paquete específico. Cuanto mayor sea el valor de CA, mayor será la responsabilidad de ese paquete y mayor su relevancia dentro del software. Un paquete con muchas dependencias externas se convierte en un artefacto de riesgo, sabiendo que un cambio en él puede impactar directa e indirectamente en muchas clases.

Partiendo de la definición de CA, una clase aferente es aquella clase que pertenece a un paquete específico y es utilizada en otros paquetes del sistema:

$$clase\ aferente = \{x |\ x \in P\ \wedge\ x \in O\} \quad (6)$$

Donde: x = es una clase, P = conjunto de clases del paquete especifico y O = conjunto de clases en otros paquetes.

De esta manera se puede calcular el CA utilizando la fórmula:

$$CA = \sum_{i=1}^{OP} \sum_{j=1}^{TC_i} C_j(C) (7)$$

Donde: CA = Acoplamiento aferente de un paquete, C = Clases del paquete principal, OP = Total de otros paquetes, TC_i = Número total de clases en el paquete i y C_j = Clase del paquete externo.

En este sentido, los umbrales identificados son útiles para definir cuando el CA tiene un valor alto, basado en los estándares de calidad del software que ha sido desarrollados. Saber cuál es un valor alto de CA puede ayudar a identificar la necesidad de redistribución de responsabilidades de un paquete demasiado influyente en un conjunto de paquetes más cohesivos. Los valores para CA, se presentan en tres rangos: Bueno/Común (CA≤ 7), Regular/Casual (CA ≤ 39) y Malo/Poco común (CA> 39) [78]. En la siguiente *Ilustración 54*, hay una sola clase: Clase A, que depende de clase B, por lo tanto, el acoplamiento aferente que corresponde es 1.

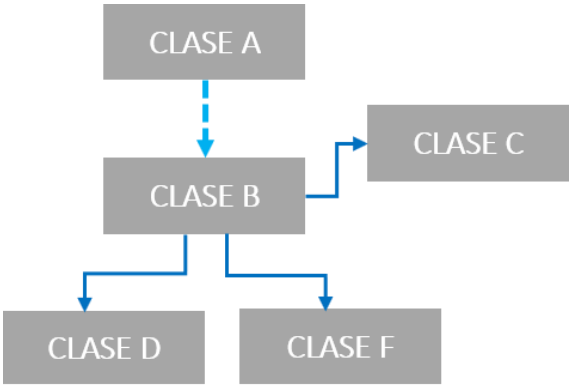


Ilustración 54.Ejemplo de Acoplamiento Aferente y Eferente

(Ce)-Acoplamiento eferente

CE es el número de clases internas en un paquete que dependen de las clases externas de este paquete. Un valor alto de CE significa que el paquete depende en gran medida de otras clases de otros paquetes, lo que lo convierte en un artefacto más inestable, dado el alto grado de clases dependientes. Mantener un bajo grado de CE significa conseguir un paquete con mayor independencia.

Partiendo de la definición de CE, una clase eferente es aquella clase que pertenece a un paquete especifico y utiliza otros paquetes del sistema:

$$clase\ eferente = \{x |\ x \in P\ \wedge\ O \in x\}(8)$$

Donde: x = Es una clase, P = El conjunto de clases del paquete especifico y O = El conjunto de clases en otros paquetes.

De esta manera para calcular el Ce de un paquete, se cuenta el número de clases que dependen de otros paquetes dentro del paquete analizado utilizando la fórmula:

$$CE = \sum_{i=1}^{OP} \sum_{j=1}^{TC_i} C(C_j)(9)$$

Donde: CE = Acoplamiento eferente de un paquete, C = Clases del paquete principal, OP = Total de otros paquetes, TC_i = Número total de clases en el paquete i, C_j = Clase del paquete externo.

Los umbrales identificados muestran que la mayoría de los paquetes de software OO se han desarrollado con hasta 6 clases dependientes, ocasionalmente tienen de 7 a 16 clases dependientes y rara vez más de 16. Los valores para CE, se presentan en tres rangos: Bueno/Común (CE≤ 6), Regular/Casual (CE ≤ 16) y Malo/Poco común (CE> 16) [78]. En la anterior Ilustración 54, el acoplamiento aferente que corresponde a la clase B, es 3, debido a que Clase A depende de otras clases (C, D, F).

(DAC)-Acoplamiento de abstracción de datos

Esta es una de las métricas consideradas como importantes para la evaluación de los atributos de calidad del software testeabilidad (capacidad de prueba) y la detección de fallas [83]. La abstracción de datos es una técnica de creación de nuevos tipos de datos adecuados para la programación de una aplicación, brinda la capacidad de crear tipos de datos definidos por el usuario llamados tipos de datos abstractos (ADT) [84], un ADT se define en una clase *c*, si es el tipo de un atributo de clase *c*, DAC es el número de atributos no heredados que tienen una clase como tipo (ADT) [79].

Se define como número de ADT definidos en una clase, y la medida DAC se representa[79]:

$$DAC(c) = |\{a\ |a \in A_I(c) \wedge T(a) \in C\}|\ (9)$$

Donde: $a \in A_I(c)$, se refiere a los atributos abstractos de una clase c y $T(a) \in C$, se refiere a los atributos que tiene una clase como tipo.

Los limites malos señalados en el trabajo [83] son : 3-4 poco mal , 5-6 mal, 6 muy mal. De acuerdo a lo encontrado, a continuación, se concluyen los umbrales definidos para este

modelo son: Bueno/Común (DAC ≤ 2), Regular/Casual (3 ≤ DAC ≤ 4) y Malo/Poco común (DAC>4).

En la *Ilustración 55,* hay dos ADT en la clase Libro, pub y Mercado. Para la clase Libro, DAC es 2 [84].

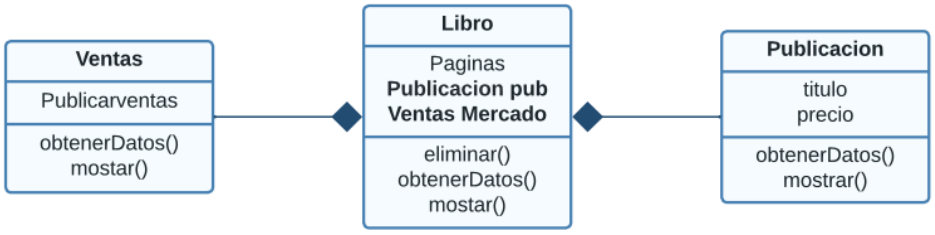


Ilustración 55. Diagrama de clases, sistema de ventas (métrica DAC)

(DIT)- Profundidad del árbol de herencia

Esta métrica muestra el nivel de herencia en el diseño de la clase y denota la longitud de la ruta más larga desde una clase determinada hasta la clase raíz en la jerarquía de herencia. Sin embargo, la herencia fomenta la reutilización de una clase, pero hace que el mantenimiento y la depuración sean más complejos [67].

Se ha encontrado que un DIT alto aumenta las falla, no obstante, no necesariamente las clases más profundas en la jerarquía de clases son las que tienen más fallas si no que, se presentan en aquellas clases que están en el medio del árbol [85] ya que las clases raíz y más profunda se consultan con frecuencia y, existe una familiaridad, por lo que tienen una baja propensión a fallas en comparación de las clases intermedias.

Un sistema orientado a objetos consta de un conjunto de clases, C. Donde, para cada clase $c \in C$, tenemos $Ancestros\ (c) \subset C$ el conjunto de clases de las que c hereda directa o indirectamente. La siguiente definición formal de profundidad del árbol de herencia [86]:

$$DIT(c) = |\ Ancestros\ (c)| \quad (9)$$

Sea: *Ancestros (c), C* sea el conjunto de clases de antepasados de la clase c [79].

En el trabajo [82] proponen un valor de 1 a 6, lo que coincide con el valor del umbral presente en [87], en [78] se considera que un DIT>4 es alto. En [88] según un estudio de 30 proyectos de C++, relacionan el aumento en DIT con acrecentar la densidad de errores y disminuye la

calidad. Para este trabajo, se considera lo propuesto en [78]: Bueno/Común (DIT ≤ 2), Regular/Casual (2 < DIT ≤ 4) y Malo/Poco común (DIT > 4).

(DN/D)- Distancia Normalizada desde la Secuencia Principal

Esta métrica se aplica a los paquetes, la razón principal es que la abstracción y la estabilidad de los paquetes están estrechamente relacionadas, es decir, cuanto más abstracto es un paquete, más estable debe ser, ya que debe tener muchos clientes que dependen de sus abstracciones. El caso ideal, todos los paquetes se encuentran en los puntos:

- Paquetes en el punto (0,1), máximamente estables (I=0) y máximamente abstractos (A=0).
- Paquetes en el punto (1,0), es decir, completamente inestables (I=1) y concretos (A = 0).

Sin embargo, en el mundo real existen diferentes grados de abstracción y estabilidad, por lo que no todos los paquetes residirán en los puntos mencionados, pero deben ubicarse en la secuencia principal (línea que conecta (1,0) con (0,1) ver *Ilustración 56*) debido a que es la posición ideal para los paquetes y se encuentran con las porciones correctas de abstracción y estabilidad [70], [71].

Ejemplos:

En un paquete con abstracción A = 0, I = 0 resulta Dn = -1, por lo tanto, el paquete es máximamente estable y concreto. Tal paquete no es deseable, ya que es rígido, no puede extenderse porque no es abstracto. Además, es difícil de cambiar porque es estable (por lo tanto, tiene clientes que dependen de él).

Un paquete con A = 1, I = 1, resulta Dn = 1, siendo máximamente estable y concreto. Tal paquete no es deseable: es rígido porque las abstracciones son imposibles de extender [89].

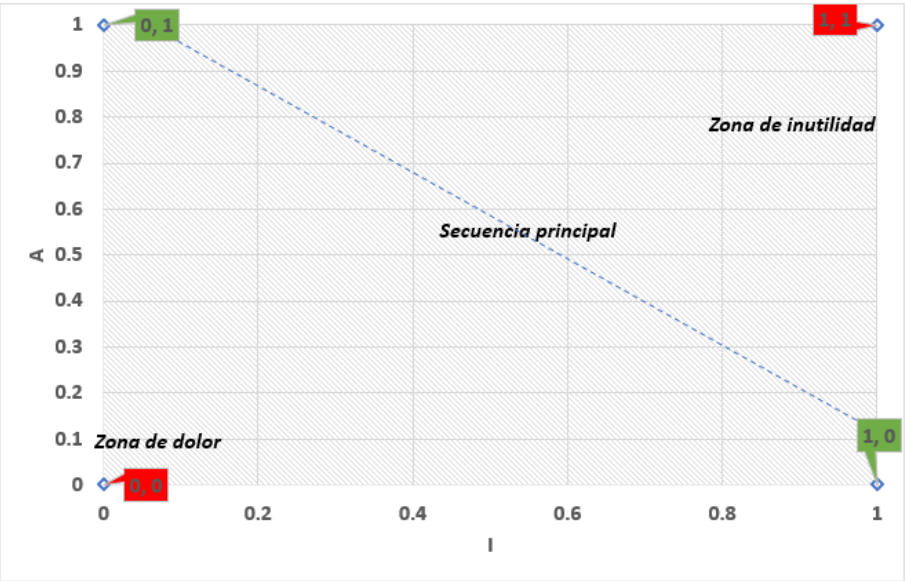


Ilustración 56. Grafica de Inestabilidad-Abstracción [71]

La métrica Dn mide el cuadrado de la distancia del paquete desde la Secuencia Principal [89].

$$Dn = |A + I| - 1 \qquad (11)$$

Donde : $A = Abstracción\ e\ I = Inestabilidad$, lo ideal es $Dn = 0$.

Existen tres probables resultados (ver *Tabla 22*) -1, 0, 1:

Tabla 22. Valores para la métrica Distancia Normalizada

Dn = |0 + 0| - 1 =**-1**	Dn = |0 + 1| - 1 =**0**	Dn = |1 + 0| - 1 =**0**	Dn = |1 + 1| - 1 =**1**
No deseable	**Deseable**	**Deseable**	**No deseable**

Sin embargo, para este trabajo y con el propósito de reducir la escala de resultados se propone trabajar con la siguiente formula que tiene dos probables resultados 0 o 1 (ver *Tabla 23*), donde 0 es mejor valor.

$$Dn = | \, |A + I| \, | - 1 \qquad (12)$$

Tabla 23. Valores absolutos para la métrica Distancia Normalizada

Dn = ||0 + 0|| - 1 =**1**	Dn = ||0 + 1|| - 1 =**0**	Dn = ||1 + 0|| - 1 =**0**	Dn = ||1 + 1|| - 1 =**1**
No deseable	**Deseable**	**Deseable**	**No deseable**

(I) Inestabilidad

La métrica de inestabilidad se utiliza para medir la susceptibilidad relativa de la clase a los cambios. Se define como la relación de dependencias salientes a todas las dependencias del paquete [123], métrica comprendida entre [0,1], siendo 0 la máxima estabilidad y 1 máxima inestabilidad. Robert C. Martin propuso el Principio de Dependencias Estables, que establece que las dependencias entre paquetes en un diseño deben estar en la dirección de la estabilidad de los paquetes. En otras palabras, un paquete debe depender solo de paquetes que sean más estables de lo que es [89]. La inestabilidad del paquete se calcula:

$$I = \frac{C_e}{C_e + C_a} \quad (13)$$

Donde: Acoplamiento eferente C_e es el número de paquetes de los que dependen las clases en P, y acoplamiento aferente C_a, son el número de paquetes que tienen clases que dependen de P.

A continuación (ver *Ilustración 57*), un ejemplo de la inestabilidad presente en los paquetes A, B, y C.

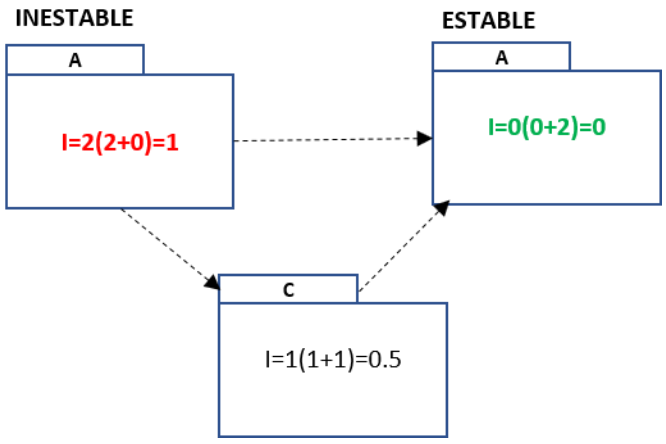


Ilustración 57. -Representación de la métrica inestabilidad, de la referencia [135].

Para este trabajo se interpreta un valor de inestabilidad de 0 como bueno y un valor de 1 como malo.

(LCOM)- Falta de cohesión en los métodos

La métrica LCOM mide la falta de cohesión de una clase, que se refiere al grado de relación de los métodos locales con las variables de instancia locales de la clase [91]. Un valor alto de LCOM generalmente señala una clase poco cohesiva [87], indica que una clase debe ser considerada, debido a que el diseño de la clase no es ideal y es posible que deba refactorizarse

en dos o más clases [82]. El principio de responsabilidad única establece que una clase no debe tener más de una razón para cambiar [87].

Los módulos altamente cohesivos son más fáciles de desarrollar, mantener y reutilizar. Si los métodos y las variables de una clase no se relacionan entre sí (es decir, tienen un bajo grado de cohesión), además, la baja cohesión a nivel de clase también puede indicar que se encapsula demasiada funcionalidad en una sola clase, lo que resulta en una clase innecesariamente compleja. Del mismo modo, la alta cohesión promueve la encapsulación de clases. En el trabajo [82], indican la posibilidad para considerar que una clase con un valor LCOM pequeño es menos propensa a errores, mientras que una clase con un valor LCOM grande es más propensa a errores. LCOM toma sus valores en el rango de 0 a 1 y su cálculo es el siguiente:

$$LCOM = 1 - \sum MF|(M * F) \quad (14)$$

Donde:

• *M* es el número de métodos en la clase (se cuentan tanto los métodos estáticos como los de instancia, incluye también constructores, getters/setters de propiedades, métodos de agregar/quitar eventos).

• *F* es el número de campos de instancia en la clase.

• *MF* es el número de métodos de la clase que acceden a un campo de instancia en particular.

• *Sum (MF)* es la suma de *MF* sobre todos los campos de instancia de la clase.

Para este trabajo el umbral para esta métrica seria de LCOM 0,167 a 0,7251 como se define en [78].

(LOC)-Líneas de código

La métrica se utiliza para contar las líneas del código fuente independientemente de si contienen líneas código, comentarios o líneas en blanco. LOC es una métrica de software típica que se utiliza para medir el tamaño del programa. Muchos hallazgos de investigación demuestran que los valores LOC más grandes toman más tiempo para desarrollar y mantener un programa [92]. En [89] se menciona que un número de 1000 suele ser el máximo en una clase o archivo, en [93] recomiendan para LOC, la longitud del archivo debe ser de 4 a 400 líneas de programas, una función debe ser de 4 a 40 líneas y que al menos el 30 por ciento y como máximo el 75 por ciento de un archivo deben ser comentarios, el umbral de métrica LOC [94] 24 como valor mínimo y 500 [28], [94] como valor máximo. Por otra parte en [78]

sugieren un umbral para líneas de código por método MLOC va de 10 a 30. Un método que incluye muchas líneas de código es una señal de olor a código de método largo. Los métodos más largos son difíciles de leer. Cuando se necesita un cambio en un método, un desarrollador debe comprender cada línea de código. A medida que el método se hace más largo, se vuelve más complicado. Eso haría que el método fuera difícil de mantener [110]. Para este trabajo, se considera: Bueno/Común (LOC≤ 24), Regular/Casual (24 < LOC ≤ 400) y Malo/Poco común (LOC > 400) como valores para evaluar las clases y Bueno/Común (LOC≤ 10), Regular/Casual (24 < LOC ≤ 40) y Malo/Poco común (LOC > 40) para las líneas de código presentes en un método. Dada su definición LOC, es calculado como:

$$LOC = \sum lineasContables \quad (15)$$

Donde: LíneasContables = líneas de código (exceptuando comentarios y saltos de línea).

(MPC)-Acoplamiento de paso de mensajes

Se utiliza para medir la complejidad del paso de mensajes entre clases , el número de mensajes enviados desde una clase puede indicar cuán dependiente es la implementación de los métodos locales, de los métodos de otras clases [91].

La métrica MPC solo cuenta las invocaciones de métodos de otras clases, y no las invocaciones de métodos propios de la clase. Además, solo se cuentan las declaraciones de envío en "métodos locales" [79].

Se define como:

$$MPC(c) = \sum_{m \in M_I(c)} \sum_{m' \in SIM(m) - M_I(c)} NSI(m, m') \quad (16)$$

Donde: $SIM(m)$, es el conjunto de métodos invocados estáticamente de m, $NSI(m, m')$, corresponde al número de invocaciones estáticas de m′ por m y $M_I(c)$, el conjunto de métodos implementados en c.

Por ejemplo, si dos métodos diferentes en la clase A acceden al mismo método en la clase B, entonces MPC = 2. En la *Ilustración 58*, el valor de MPC para la clase Libro es 4, ya que los métodos de la clase Libro llaman a Ventas::obtenerDatos(), Ventas::mostrar(), Publicación::obtenerDatos(), Publicación::mostrar() [95].

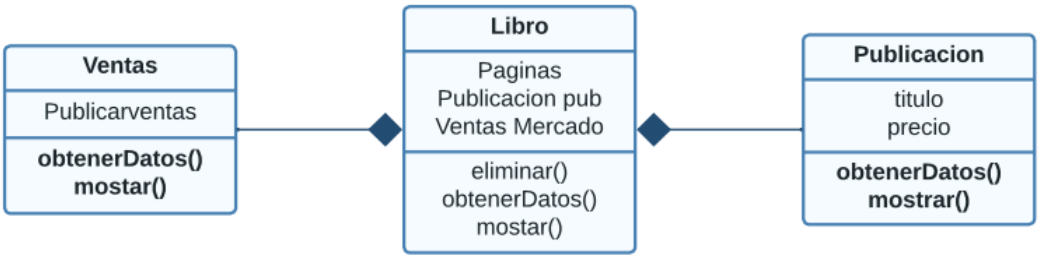


Ilustración 58. Diagrama de clases, sistema de ventas (métrica MPC)

Los umbrales para la métrica MPC, como valor mínimo 0, y como valor máximo 17, Cuanto mayor sea el MPC, mayor será la clase dependiente de otras clases, por lo tanto, menor será la reutilización , un MPC bajo indican que se requiere menos esfuerzo para personalizar y reutilizar el componentes, [96].

MPC evidencia: 1) la dificultad de aplicar cambios en un módulo dado debido a la cantidad de dependencias, y 2) la posibilidad de que una clase cambie debido a efectos dominó, en el sentido de que es un intermediario (proxy) de la fuerza de la dependencia [136].

(NMO/NORM)-Número de métodos anulados

Cuando las clases se derivan de sus superclases, algunos de los métodos derivados pueden redefinirse. Los métodos pueden anularse por extensión, restricción, optimización o conveniencia. El número de dichos métodos anulados (NMO) para la clase es una indicación de los cambios en la semántica de los métodos derivados. El número también indica el grado en que se han especializado los métodos heredados de la clase. Cada vez que la subclase redefine un método, el código escrito para ese método en la superclase se "desecha". Una superclase bien diseñada que sea genérica tendría menos métodos modificados por las clases derivadas [97]. Los valores para NMO considerados para este trabajo, se presentan en tres rangos: Bueno/Común (NMO≤ 2), Regular/Casual (NMO ≤ 4) y Malo/Poco común (NMO> 4) [78]. Dada la definición, NMO se calcula:

$$\sum_{i=1}^{n} metodosAnulados_i \quad (17)$$

Donde: metodosAnulados, son métodos derivados que pueden anularse por extensión, restricción, optimización o conveniencia.

(NEST/ NBD)- Nivel de Anidamiento

Es el nivel máximo de anidamiento de bloques/estructuras de control que están presentes en un método (o función) [98], ayuda a identificar que si un método o clase cumple más de un propósito, seguirá agregando LOC a la clase/método versión tras versión, lo que finalmente lo hará inmanejable después de un tiempo [137]. Bajar el NBD, es una clase más manejable. La profundidad del bloque anidado aumenta la complejidad del código y, por lo tanto, aumenta la capacidad de mantenimiento del código. Simplificar bloques anidados y reemplazarlos con clases heredadas ayuda a mantener la simplificación.

El código con un nivel de anidamiento profundo suele ser complejo y difícil de mantener. En general, los bloques con declaraciones if, else, else if, do, while, for, switch, catch, etc. son parte de bucles anidados [98].

En [99], se refiere que el nesting depth sea de 0 a 4, más de eso lo hará demasiado complejo y difícil de leer y se recomienda dividir el método en partes, en [78] de los umbrales identificados para distintas métricas, presentan para NEST los siguientes rangos : Bueno/Común (NEST≤ 1), Regular/Casual (NEST ≤ 3) y Malo/Poco común (NEST > 3), mismos que serán considerados en el modelo propuesto.

(NOA/NA/NOF)- Numero de Atributos

Cuenta el número total de atributos definidos en una clase [84] sin incluir atributos heredados o atributos definidos dentro de los métodos [100]. En el trabajo [101], se presentan valores mínimo (0) y máximo (5) para esta métrica, en [102] indican que su rango normal esta entre 2 y 5, y que un número de atributos (NOA>10) indica el diseño de mal olor (smell desing), finalmente los valores considerados para este trabajo, se presentan en tres rangos: Bueno/Común (NOA≤ 3), Regular/Casual (3 < NOA ≤ 8) y Malo/Poco común (NOA > 8 para la métrica NOA [78]. El número de atributos definidos en la clase C_i es dado por:

$$A_d(C_i) = A_v(C_i) + A_h(C_i)$$ [103]*(18)*

Donde: $A_v(C_i)$- número de atributos visibles en la clase C_i y $A_h(C_i)$ - número de atributos ocultos en la clase C_i

(NOC)- Número de hijos

Esta métrica se relaciona con la función de herencia, similar a DIT, y se calcula contando el número de clases secundarias inmediatas heredadas de una clase determinada [67], parece lógico que cuantos más hijos directos tenga una clase, más clases puede afectar potencialmente debido a la herencia. Por ejemplo, si hay muchas subclases de la clase que dependen de algunos métodos o variables de instancia definidas en la superclase, cualquier cambio en estos métodos o variables puede afectar a las subclases. Uno puede intuir que cuanto mayor sea la métrica NOC, más difícil será mantener la clase [91].

El cálculo de NOC es el siguiente:

NOC = número de subclases directas, que va de 0 a N; donde N es un entero positivo [91]. Un valor alto de NOC mejora la reutilización, pero hace que una clase sea difícil de probar [67].

En él trabajo [104] se encontró entre otras métricas que NOC, no se correlaciona significativamente con el número de fallas. Sin embargo, dado que el enfoque de la investigación es comprender los factores determinantes que permitan conocer el estado de tolerancia a fallas que posee un sistema Big Data y la métrica NOC forma parte del catálogo de métricas seleccionadas que permitirán medir los sub-atributos relacionados (en este caso afecta la mantenibilidad y el rendimiento), esta métrica si es incluida para este modelo. En [78] el rango para un NOC adecuado va de 11 a 28, considera que un NOC > 28 es alto, sin embargo el trabajo [87] trabaja con los valores para NOC de 0 a 6. Este trabajo el umbral para esta métrica seria de NOC 1 a 3 como se define en [82]. Partiendo de la definición de la métrica, NOC, se calcula:

$$\sum_{i=1}^{n} subclaseDirecta_i \quad (19)$$

Donde: subclaseDirecta = es una clase que hereda de una clase padre principal, i= número de subclaseDirecta y n= número total de subclasesDirectas de una clase.

(NOM/NM)-Numero de métodos

Mide el número de métodos definidos localmente por una clase. No cuenta los métodos heredados ni los constructores [105]. Según Lehman, las funcionalidades que ofrece un sistema deben incrementarse continuamente para mantener la satisfacción del usuario. Si este crecimiento se hace de forma descontrolada, las clases crecerán cada vez más añadiendo

métodos y campos que cumplan con las crecientes expectativas del usuario. Si no se hace este crecimiento de forma diseñada, se deteriorará la calidad del software, ya que las clases se vuelven grandes y complejas, para este trabajo el umbral para esta métrica que va de 6 hasta 14, seria: Bueno/Común (NOM ≤6), Regular/Casual ((6 < NOM ≤ 10) y Malo/Poco común (NOM > 14) [78].

$$\sum_{i=1}^{n} metodoLocal_i \quad (20)$$

Donde: metodoLocal = un método definido localmente por una clase (sin contar métodos heredados ni los constructores).

(NOPA)-Número de atributos públicos

Los datos de una clase deben ser privados, de lo contrario se está violando el principio de ocultación de información a esta práctica se le conoce como código smell y aparece en casos donde una clase expone sus atributos. Los desarrolladores normalmente no reconocen la presencia de este tipo de código y lo consideran menos dañino que otros para la mantenibilidad. Si NOPA (Número de atributos públicos de la clase medida) es superior a 10, se considera como smell [107], el trabajo [106] propone umbrales por defecto para todos los smells, entre estos la métrica NOPA asignando los valores: Muy bajo→1 Bajo→2 Medio→3 Alto→5 Muy alto→12. De acuerdo a lo encontrado, los valores para NOPA considerados para este trabajo, se presentan tres rangos: Bueno/Común (NOPA ≤ 3), Regular/Casual (NOPA ≤ 10) y Malo/Poco común (NOPA > 10) [106] [107].

$$\sum_{i=1}^{n} atributosPublicos_i \quad (21)$$

Donde: atributosPublicos, cuenta todos los atributos de una clase que se declaran como públicos.

(NOPM/NPM)-Numero de métodos públicos

Esta métrica es el número total de métodos públicos definidos en una clase, se considera un efectivo indicador para las predicciones de fallas y particularmente útil para software de tamaño mediano y grande, sirve de apoyo para estimar la cantidad de trabajo para desarrollar una clase o subsistema [108],[109]. Se encontró que un umbral apropiado para el límite inferior de NPM es 1, dado que una clase debe definir al menos un único método que la aplicación pueda utilizar [108]. De acuerdo al trabajo [109], un buen rango de número de métodos públicos es de 0 a 10, lo que refleja que una clase debe tener pocas responsabilidades, siendo lo más idóneo y lo más común de encontrar de acuerdo a los

hallazgos, reflejando que las clases brindan pocos servicios. El rango medio es para clases con 10 a 40 métodos públicos, es raro de encontrar, y finalmente la probabilidad de que una clase tenga más de 40 métodos públicos es bastante baja.

Las clases con valores de NPM más altos, se sugiere que se puede dividir para un rendimiento óptimo. Un valor alto de NPM puede estar asociado con la complejidad de la clase (tener demasiadas responsabilidades) o sirve como un indicador de una clase que está altamente acoplada con otras partes del software [108].

Dada la definición de NPM, se puede calcular como:

$$\sum_{i=1}^{n} \text{metodosPublicos}_i \quad (22)$$

Donde: metodosPublicos, cuenta todos los métodos de una clase que se declaran como públicos.

Los valores para NOPM considerados para este trabajo, se presentan tres rangos: Bueno/Común (NOPM ≤ 1), Regular/Casual (1< NOPM ≤ 10) y Malo/Poco común (NOPM > 10) [108],[109].

(NPAR)- Número de Parámetros

Esta métrica cuenta el número de parámetros de un método o un constructor. En general, los métodos y constructores con más de tres parámetros ($NPAR \geq 3$) son difíciles de recordar y usar. El código fuente con invocaciones de métodos que tienen largas listas de parámetros es difícil de comprender. Además, la invocación de métodos con formas similares es propensa a errores [89], y es clasificado como código con malos olores (Code Bad Smells) ya que un método no necesita demasiados parámetros [111]. En el trabajo [138], indican que la función no debe exceder de los 10 parámetros, mientras que en el trabajo [110] se considera lista larga de parámetros donde: $PAR \geq 5$, en [101] presentan como valor máximo 0 y mínimo 4 para esta métrica, por otra parte [78], presenta para la métrica *NPAR* los siguientes umbrales: Bueno/Común (NPAR ≤ 2), Regular/Casual (2 < NPAR ≤ 4) y Malo/Poco común (NPAR > 4), valores que serán considerados para este trabajo y se calcula como:

$$\sum_{i=1}^{n} \text{parametros}_i \quad (23)$$

Donde: parametros, cuenta todos los valores que recibe un método.

(RFC)- Respuesta para una clase

El RFC es el conteo del conjunto de todos los métodos que se pueden invocar en respuesta a un mensaje a un objeto de la clase o por algún método en la clase. Esto incluye todos los métodos accesibles dentro de la jerarquía de clases. Pressman, afirma que, dado que RFC aumenta, el esfuerzo requerido para la prueba también aumenta porque la secuencia de prueba crece. Si RFC aumenta, la complejidad general del diseño de la clase aumenta y se vuelve difícil de entender [139].

Se ha encontrado un RFC grande para indicar más fallas.

Definición [112]:

$$RFC = |\,RS\,| \quad (24)$$

Sea RS el conjunto de respuestas para la clase. El conjunto de respuestas para la clase se puede expresar como:

$$RS = \{M_i\} \cup \{R_i\} \quad (25)$$

Donde: M_i= todos los métodos de la clase y R_i = conjunto de métodos llamados por M_i

Un estudio de 30 proyectos de C++ sugiere que un aumento en RFC aumenta la densidad de errores y disminuye la calidad [88] . Los valores considerados para el umbral de esta métrica presentados en [87] va de 0 a 35. Sin embargo, para este trabajo el umbral para esta métrica seria de 6 a 36, como se propone en el trabajo [78].

V(G)- Complejidad Ciclomática

La complejidad Ciclomática es el miembro más utilizado de un conjunto de métricas de software estáticas, se puede considerar como medida amplia de solidez y confianza para un programa, esta métrica mide el número de caminos linealmente independientes a través de un código. V(G) es una medida de la complejidad del flujo de control de un método o constructor que cuenta el número de ramas en el cuerpo del método.

En general cuanto más complejos son los métodos de un programa, más difícil es probarlo, afectando negativamente a la fiabilidad, respecto al acoplamiento, alude a qué tan relacionadas están dos clases/módulos y qué tan subordinadas están entre sí. Una alta complejidad Ciclomática, por ejemplo, un aumento en v(g) > 10 es negativo y las unidades de software con mayor v(g) se consideran riesgosas, tienden a ser programas difíciles de entender y mantener el software, lo que afecta indirectamente a la seguridad al dificultarlo

y/o requiere mucho tiempo para encontrar y/o corregir vulnerabilidades. En consecuencia, es más fácil introducir vulnerabilidades, por lo tanto, tienen una mayor probabilidad de contener defectos [89], [76], [113].

La complejidad Ciclomática se calcula: $v(G) = e - n + 1$ *(26)*

Donde: e, es el número de aristas y n es el número de nodos.

De acuerdo a lo anterior, el siguiente grafo (ver *Ilustración 59*) tiene una complejidad Ciclomática de 4 [114].

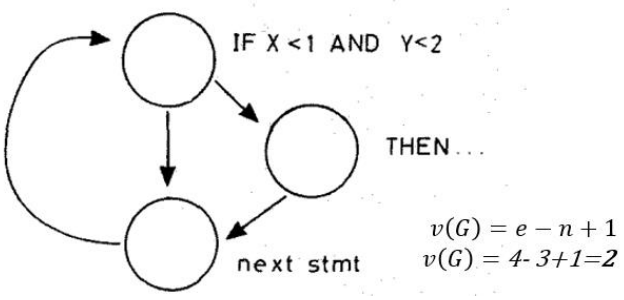


Ilustración 59. Derivación de v(G) [114].

Aplicar la fórmula original de la complejidad a mano es tedioso y propenso a errores, una forma de simplificar el cálculo esta dado en la fórmula: $v(G) = p + 1$ *(27)* (ver *Ilustración 60*), que consiste en contar predicados.

Donde: p, es un predicado de decisión binario, aparece en el gráfico de flujo de control como un nodo del que salen exactamente dos aristas. Si todas las decisiones son binarias y hay p predicados de decisión binarios $v(G) = p + 1$ [115].

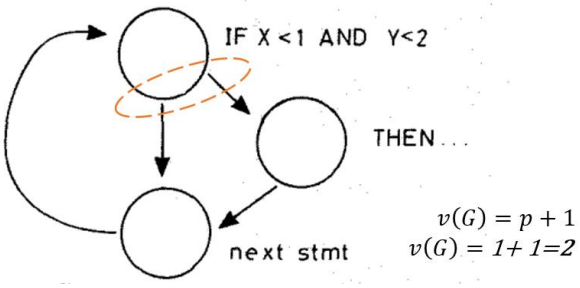


Ilustración 60. Derivación de v(G), adaptado de [114].

De acuerdo al trabajo [93], una manera más simple, se encuentra determinando el número de declaraciones de decisión que son causadas por declaraciones condicionales en un programa y se calcula como:

$$CC = numero\ de\ declaraciones\ de\ decision + 1 \quad (28)$$

Donde hay cuatro reglas básicas que se pueden utilizar para calcular CC:

• Contar el número de declaraciones *si/entonces* en el programa. • Encontrar declaraciones de selección (o cambio) y contar el *número de casos* dentro de ellas, encontrar el total de los casos en todas las declaraciones seleccionadas combinadas. No considerar el caso predeterminado o "si no". • Contar todos los *bucles* del programa. • Contar todas las sentencias *try/catch*.

El resultado de la suma de los 4 pasos anteriores + 1, da como resultado, la complejidad Ciclomática del programa. Básicamente, se trata de encontrar el número de caminos de decisión, es decir las siguientes palabras clave: if, while, for, &&, ||, catch, case etc. +1 [140]. A continuación, en la *Ilustración 61*, se muestra un ejemplo aplicando las tres fórmulas mencionadas anteriormente, donde se puede observar que en todos los casos se llega al mismo resultado, a pesar de calcular la complejidad usando un grafo o el código fuente.

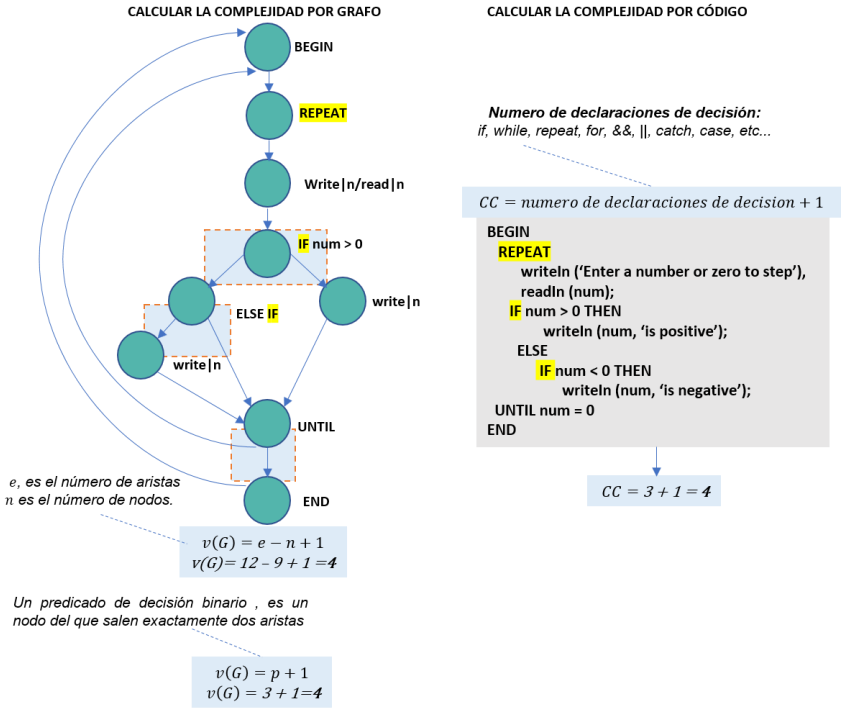


Ilustración 61. Calculo de la complejidad Ciclomática [114], [115], [93].

El conjunto de valores umbral para comprar la complejidad Ciclomática de un programa considerados en los trabajos [116], [89] son :

Evaluación de riesgos V(G)

1 - 10 un programa sencillo, sin mucho riesgo

11 - 20 programa más complejo, riesgo moderado

21 - 50 programa complejo, alto riesgo

Además, en el trabajo [78], manejan el umbral para desde tres perspectivas :
Bueno/Común (V(G)≤ 2), Regular/Casual (2 < V(G) ≤ 4) y Malo/Poco común (V(G)> 4), mientras que en [138] consideran 20 como límite de la métrica, por lo que para este trabajo se considera como correcto una complejidad Ciclomática que va de 1 a 10 basado en las propuestas revisadas, Bueno/Común (V(G)≤ 4), Regular/Casual (4 < V(G) ≤ 10) y Malo/Poco común (V(G) > 10) [116], [89], [101].

(WMC)- Métodos ponderados por clase

Esta métrica se relaciona con los métodos implementados en una clase y se determina sumando la complejidad Ciclomática de todos los métodos implementados en una clase [67], indica cuánto esfuerzo será necesario para mantener y desarrollar una clase particular de software [76]. De acuerdo con el trabajo [112], es probable que los objetos con una gran cantidad de métodos sean más específicos de la aplicación, lo que limita la posibilidad de reutilización.

Para WMC, se han definido diferentes límites para el número de métodos en una clase, por ejemplo en [76] proponen un umbral de 100 para WMC, en [87] si el número de métodos en una clase es ≤15, entonces se puede considerar que la clase tiene una complejidad normal. A continuación, en la *Tabla 24*, se presentan los umbrales identificados en el trabajo [141].

Tabla 24. Valores propuestos en otros trabajos como umbral para WMC [141]

Métrica	Shatnawi	Rosenberg	Lanza y Marinescu	Herbold et al.
WMC	20	100	47	100

En el trabajo [87],el umbral para la métrica WMC va de 0-15, [82],presenta una recopilación de valores de umbral para las métricas de CK en la literatura y revisar otras propuestas. Como puede observarse, las cifras postuladas son muy disímiles. Por lo que para este trabajo el umbral para esta métrica seria de 11 a 34, como se propone en el trabajo [78].

Los métodos ponderados por clase se definen como sigue [112]:

$$WMC = \sum_{i=1}^{n} c_i \quad (29)$$

Donde una clase C_i, con métodos M_1, hasta M_n que son definidos en la clase, donde C_i, hasta C_n es la complejidad de los métodos [65].

(ME)-Manejo de excepciones

Las excepciones son errores inesperados que ocurren en el tiempo de ejecución de los sistemas de software, que pueden provocar fallas graves del sistema, como fallas o bloqueos si se manejan incorrectamente. Para evitar tales fallas, los lenguajes y marcos de programación populares a menudo incluyen mecanismos avanzados de manejo de excepciones [142] como medio para mejorar la solidez del software [143], y mejorando la tolerancia a fallas, la presencia de mecanismos indica tolerancia a fallas es muy alta y, de lo contrario, es moderada [144]. Los expertos emplean manejo de excepciones para transmitir mensajes de falla decentes cuando se encuentran situaciones inesperadas o cuando se encuentran situaciones excepcionales esperadas, pasando mensajes a través de excepciones al usuario para informarle qué salió mal, además su uso también se da para evitar daños en los datos y estados corruptos, controlar el flujo de ejecución y mostrar mensajes más atractivos [145].

El manejo de excepciones de Java consta de cinco componentes o palabras clave: try, catch, throw, throws and finally, entre las excepciones más comunes se encuentran: IOException, FileNotFoundException, NullPointerException, IllegalArgumentException, etc.) para manejar excepciones [146], [142].

(MTTF) -Tiempo medio hasta el fallo

El sistema funciona de manera normal, cuanto tiempo en promedio debería pasar para que el sistema falle desde el estado inicial hasta que se presente el 1er falla. Debería ser lo más alto posible para considerar que el sistema es tolerante a fallas [18].

$$MTTF = \frac{\sum_i^n t_i}{n} \quad (30)$$

Donde: $t_i$es el i-ésimo tiempo entre fallas, ignorando los tiempos de reparación, para $i \geq 1$, y n es el número de veces entre fallas, $t_i$es el i-ésimo tiempo entre fallas de una versión o módulo y n es el número de veces entre fallas, ISO/IEC/IEEE 24765 [27].

(MTTR) -Tiempo medio de reparación

Se refiere a la duración esperada u observada necesaria para que un sistema o componente que funciona mal vuelva a funcionar con normalidad, ISO/IEC/IEEE 24765 [27], lo mejor para este caso es que la reparación se demore lo menos posible para poderlo considerar tolerante a fallas [18].

$$MTTR = \frac{\sum_i^n R_i}{n} \quad (31)$$

Donde: R_i, es el i-ésimo tiempo de reparación de una liberación o módulo y n, es el número de veces de reparación [27]**.**

(MTBF)- Tiempo medio entre fallas

El tiempo esperado u observado entre fallas consecutivas en un sistema o componente, ISO/IEC/IEEE 24765 [27], y representa la suma de tiempo medio de reparación y tiempo medio hasta el fallo [31].

$$MTBF = MTTR + MTTF \quad (32)$$

Donde: MTTF es el Tiempo medio hasta el fallo) y MTTR el Tiempo estimado o promedio para reparar.

El tiempo para que ocurra una falla debería ser lo más alto posible para considerar que el sistema es tolerante a fallas [18].