COMPONENTE SOFTWARE PARA EL CÁLCULO DE MÉTRICAS A PARTIR DE CÓDIGO FUENTE

INVESTIGADORES

Jesús David Escobar Lascarro

Carlos Andrés Quevedo Díaz

UNIVERSIDAD DE CARTAGENA

FACULTAD DE INGENIERÍA

PROGRAMA DE INGENIERÍA DE SISTEMAS

CARTAGENA DE INDIAS, 2019

COMPONENTE SOFTWARE PARA EL CÁLCULO DE MÉTRICAS A PARTIR DE CÓDIGO FUENTE

TÉSIS DE GRADO

E-Soluciones

Ingeniería De Software

INVESTIGADORES

Jesús David Escobar Lascarro

Carlos Andrés Quevedo Díaz

Director: Martín Monroy Ríos, Msc, PhD. (Universidad de Cartagena)

UNIVERSIDAD DE CARTAGENA

FACULTAD DE INGENIERÍA

PROGRAMA DE INGENIERÍA DE SISTEMAS

CARTAGENA DE INDIAS, 2019

UNIVERSIDAD DE CARTAGENA

Tesis de Grado: COMPONENTE SOFTWARE PARA EL CALCULO DE METRICAS A PARTIR DE CODIGO FUENTE

Autores: JESÚS DAVID ESCOBAR LASCARRO
CARLOS ANDRÉS QUEVEDO DÍAZ

Director: PhD. MARTÍN MONROY RÍOS

Nota de Aceptación

_____

_____

_____

_____

_____

Presidente del Jurado

_____

Jurado

_____

Jurado

Cartagena de Indias, ____ de _____ de 2019

CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INDICE DE ANEXOS

RESUMEN

Hoy día la calidad de un producto software juega un papel fundamental en el ámbito académico, por tal motivo se desarrollan métricas para evaluarlo en todas las etapas de desarrollo y así garantizar la calidad del mismo. El objetivo de este proyecto es proporcionar una herramienta que permita calcular métricas orientadas a objetos para apoyar el análisis de productos software y los procesos de enseñanza y aprendizaje en ambientes académicos. Se hace énfasis en las métricas relacionadas con acoplamiento y falta de cohesión en clases, estas permiten determinar si se deben hacer ajustes para mejorar los aspectos de calidad del producto durante la fase de implementación o mantenimiento.

Para la implementación de la herramienta se utilizó la metodología de desarrollo RUP y en la revisión de la literatura se empleó el método analítico, con el cual se obtuvo información relevante que fue usada como guía para el desarrollo de la herramienta.

El presente documento describe los resultados de analizar el código fuente de cuatro proyectos de acuerdo a las métricas propuestas por Hitz & Montazeri; y Robert C. Martin. Falta de Cohesión en las clases y acoplamiento entre clases respectivamente. Como resultado se presentan los detalles del análisis de cada código fuente. Respecto a la Falta de Cohesión es bien sabido que una clase con valor de LCOM>=2 indica que la clase tiene un problema. Sin embargo no necesariamente significa que la clase deba dividirse en otras más pequeñas puesto que existe la posibilidad de modificar y/o adecuar la estructura de los métodos de modo que se disminuya la falta de cohesión sin afectar el flujo lógico de la clase. Se deja a consideración del usuario si desea dividir la clase o modificarla.

Por otra parte, el análisis e interpretación de las métricas que muestra JQM3trics (herramienta implementada en este estudio) suponen un logro importante ya que permiten el apoyo de los procesos de enseñanza y aprendizaje en ingeniería de software.

En relación a los resultados obtenidos de acoplamiento Eferente, Aferente e Inestabilidad de una clase, se puede decir que estos coinciden con la base teórica encontrada en la revisión de la literatura.

ABSTRACT

Nowadays the quality of a software product plays a fundamental role in the academic field, for this reason, Object-Oriented metrics are built to evaluate the software product in all stages of development and thus guarantee the quality of it. The purpose of this project is to provide a tool for calculating object-oriented metrics to support the analysis of software products and teaching and learning processes in academic environments. Emphasis is placed on the metrics related to coupling and lack of cohesion in classes, because they allow determining if adjustments should be made to improve the quality aspects of the product during the implementation or maintenance phase.

For the implementation of the tool, the RUP development methodology was used, and in the literature review the analytical method was used, with which relevant information was obtained that was used as a guide for the development of the tool.

This document describes the results of analyzing the source code of four projects according to the metrics proposed by Hitz & Montazeri; and Abreu & Melo. Lack of cohesion in the classes and coupling factor (CF) respectively. As a result, the details of the analysis of each source code are presented. Regarding the lack of cohesion, it is well known that a class with LCOM value = 2 indicates that the class has a problem. However, it does not necessarily mean that the class should be divided into smaller ones since there is the possibility of modifying and / or adapting the structure of the methods so that cohesion is reduced without affecting the logical flow of the class. It is left to the user's consideration if he wishes to divide the class or modify it.

On the other hand, the analysis and interpretation of the metrics shown by JQM3trics represent an important achievement since they allow the support of teaching and learning processes in software engineering.

In relation to the results obtained of Efferent coupling, Afferent coupling and Unstability of a class, it can be conclude that these coincide with the theoretical base found in the literature review.

1. INTRODUCCIÓN

Este capítulo se subdivide en tres secciones. Inicialmente se tratan de forma concreta y especifica algunos antecedentes referentes a métricas orientadas objetos, seguidamente se describe la formulación del problema del presente proyecto; el ítem 1.3 hace referencia a la justificación del estudio y finalmente se detallan los objetivos y el alcance del estudio.

Es pertinente mencionar que el presente estudio verificó la literatura actual mediante las pruebas de caja negra realizadas a las métricas orientadas a objetos implementadas en la herramienta JQM3trics (ver secciones 4.6.5.3 y 4.6.5.4). Dichas pruebas permitieron verificar la base teórica descrita en los estudios mencionados en esta sección ya que los resultados obtenidos concuerdan con los resultados esperados establecidos de acuerdo a los cálculos manuales de las métricas.

Las métricas que se sometieron a prueba son Acoplamiento entre Clases, Falta de Cohesión en Métodos (LCOM[1]) y Factor de Acoplamiento (CF[2]). Para el caso de esta investigación se adaptó el enfoque descrito por (Martin C. Robert, 2006) para calcular el Acoplamiento entre Clases y para calcular la Falta de Cohesión en Métodos se siguieron los parámetros descritos por (Hitz & Montazeri, Measuring Coupling and Cohesion, 1995). El Acoplamiento entre Clases se divide en Acoplamiento Eferente y Aferente (ver Marco Conceptual) los cuales, una vez determinados, permiten calcular la Inestabilidad de una clase. El CF se implementó teniendo en cuenta la fórmula propuesta por (Abreu & Melo, 1996) en su estudio "*Evaluating the Impact of Object-Oriented Design on Software Quality*".

1.1 ANTECEDENTES

Luego de la etapa de crisis, la ingeniería de software ha presentado avances significativos. Sin embargo, estos han venido acompañados por un aspecto crucial que ha sido motivo de preocupación para el hombre a la hora de desarrollar productos software. Este aspecto es la calidad de software, la cual se define como el "proceso eficaz de software que se aplica de manera que crea un producto útil que proporciona valor medible a quienes lo producen y a quienes lo utilizan" (Pressman R. S., 2010, pág. 340) Para tratar de medir ese valor, existen distintas técnicas, entre las cuales las de mayor validez corresponden a las técnicas cuantitativas.

[1] Siglas de Lack Of Cohesion in Methods (Falta de Cohesión en Métodos)
[2] Siglas de 'Coupling Factor', que en español se traduce como Factor de Acoplamiento

Una técnica cuantitativa implica medir los atributos del software para poder calificarlo como bueno o malo. A esa medida del grado en el que se encuentra un atributo determinado, se le conoce como **métrica** (IEEE, 1993)**.**

Las métricas de software permiten asignar valores numéricos a los atributos de un sistema (Lincke, Lundberg, & Löwe, 2008), y algunas de ellas pueden calcularse directamente desde el código fuente.

Ahora bien, Chidamber & Kemerer (1994) propusieron en su momento 6 métricas orientadas a objetos que hoy día se pueden medir a partir del código fuente, como siguen: Métodos Ponderados por Clase (WMC), Profundidad de Árbol de Herencia (DIT), Número de Hijos[3] (NOC), Acoplamiento entre Clases de Objetos (CBO), Respuesta para una Clase (RFC) y Falta de Cohesión en Métodos (LCOM).

Hitz & Montazeri presentaron, en el estudio titulado "*Measuring Coupling and Cohesion In Object-Oriented System*", una versión teórica mejorada de LCOM en donde la definen como el "número de componentes conectados en una clase".

En 1996, (Abreu & Melo, 1996) presentaron un estudio en el que se evalúa experimentalmente el impacto del Diseño Orientado a Objetos en las características de calidad del software. En dicha investigación también se describe un conjunto de métricas para el diseño Orientado a Objetos denominado MOOD. El Factor de Acoplamiento (CF), una de las métricas MOOD, se implementó en la investigación descrita en este documento.

Métricas como CBO y LCOM se han implementado en herramientas como MASU en Japón (Higo, y otros, 2011); y RTTOOL, que fue desarrollada para apoyar un estudio en Brasil (Oliveira, Lima, Valente, & Serebrenik, 2014). Ambas son de uso libre pero no están enfocadas en la enseñanza de la ingeniería de software. Por otro lado, vale la pena mencionar que también existen herramientas de uso privado cuyo acceso es restringido para los interesados que carezcan de recursos, es por ello que el componente propuesto en este proyecto es de carácter OpenSource.

3 Número de subclases inmediatas subordinadas a una clase en la jerarquía de clases (Chidamber & Kemerer, 1994).

1.2 FORMULACIÓN DEL PROBLEMA

En las últimas décadas, la ingeniería de software ha evolucionado significativamente. Esto, en parte, gracias a que las grandes empresas, organizaciones y hasta los mismos ingenieros han venido invirtiendo grandes sumas de dinero, tanto en tecnología como en mano de obra calificada en aras de que sus productos software sean de calidad. Bessin, J. Citado en (Pressman R. S., 2010, pág. 340) define calidad de software como el "proceso eficaz de software que se aplica de manera que crea un producto útil que proporciona valor medible a quienes lo producen y a quienes lo utilizan". En ese sentido, en el contexto de la ingeniería de software es indispensable medir la calidad del producto; para ello existen distintas técnicas, entre las cuales las de mayor validez corresponden a las técnicas cuantitativas. Una técnica cuantitativa implica medir los atributos del software para poder calificarlo como bueno o malo. A esa medida del grado en el que se encuentra un atributo determinado, se le conoce como métrica (IEEE, 1993).

Según la definición de Lincke, Lundberg y Löwe (2008) una métrica de software es una definición matemática que asigna valores numéricos a las entidades de un sistema software. Siguiendo la esencia de esa afirmación, a un software se le pueden medir distintos niveles de abstracción. "El nivel de abstracción es el grado de cercanía existente entre la realidad y su representación con base en un lenguaje definido sobre el que es posible visualizar, construir y documentar los artefactos de un sistema software" (Monroy, Arciniegas, & Rodríguez, 2013), dichos niveles están representados por, Conceptos y Requerimientos (Modelo del Dominio), Arquitectura y Diseño (Modelo de Diseño) y el Código Fuente (Modelo de implementación) (Monroy, Arciniegas, & Rodríguez, 2013). Este estudio está enfocado en este último.

El código fuente escrito en cualquier lenguaje de programación bajo el paradigma orientado a objetos está compuesto por atributos, de los cuales algunos se pueden medir y otros no, debido a la complejidad que presentan. Entre los atributos que se pueden medir, se encuentran: cantidad de líneas de código, grado de cohesión de las clases, grado de acoplamiento, entre otras (Higo, y otros, 2011). Para completar tal fin se necesita una herramienta de análisis de código fuente. Se entiende por herramienta de análisis de código fuente como un programa que implementa un conjunto de funciones que permiten evaluar un sistema software de acuerdo con las métricas, extrayendo las entidades requeridas del sistema y proporcionándoles los valores correspondientes (Lincke, Lundberg, & Löwe, 2008)

Los desarrolladores pueden utilizar herramientas de análisis de código fuente para obtener una medida de la calidad del software que está siendo construido (Fuggetta & Di Nitto, 2014). Agregar este detalle a los procesos de desarrollo de software supone una mayor precisión a la hora de corregir fallos, y un significativo ahorro de tiempo. Sin duda alguna, también facilitaría el proceso de enseñanza (perspectiva del docente) y aprendizaje (perspectiva del estudiante) de la ingeniería de software.

A la fecha, se han ido desarrollando y mejorando una amplia gama de herramientas que calculan métricas a partir de código fuente, y en la revisión de la literatura se encontraron algunas, como por ejemplo MASU, la cual permite al usuario desarrollar plugins que calculen métricas propuestas por Chidamber & Kemerer (Higo, y otros, 2011). También se identificó RTTOOL, una herramienta OpenSource de medición de código fuente para la extracción de umbrales relativos, es decir valores que las métricas no pueden exceder (Oliveira, Lima, Valente, & Serebrenik, 2014). Estas herramientas fueron tomadas como referencia para este proyecto.

Si bien es cierto que muchas herramientas pueden calcular una buena cantidad de métricas, sólo se limitan a mostrar el resultado del cálculo, dificultando su interpretación para efectos de análisis más detallados. Por lo anterior, Lincke *et al* (2008) solicitan que se realice un estudio en profundidad que trate de explicar las diferencias en los resultados de la medición, posiblemente describiendo las métricas implementadas por unas herramientas determinadas (Lincke *et al*; 2008). La falta de descripción y, por consiguiente, la falta de interpretación de las métricas, suponen un obstáculo en procesos de desarrollo de software que requieran de análisis minucioso de código fuente, así como también en procesos de aprendizaje de la ingeniería de software, ya que no existe un reporte que describa detalladamente de dónde provienen los valores usados para calcular una determinada métrica.

Al contextualizar la problemática, se observó la necesidad de contar con un componente que sirva para apoyar los procesos de análisis de productos software y que sea de código abierto para que esté disponible a toda la comunidad desarrolladora.

Ante la necesidad mencionada anteriormente, surgió el siguiente interrogante: ¿Cómo realizar el cálculo de métricas sobre código fuente para apoyar el análisis de productos software, de modo que se brinde información detallada sobre sus atributos?

1.3 JUSTIFICACIÓN

La calidad es un factor preponderante y de sumo cuidado cuando de desarrollo de productos software se trata. Han sido cuantiosos los procesos de ingeniería de software cuyos resultados carecen de calidad debido a planeaciones pobres y malas prácticas. Estas carencias muchas veces se pasan por alto y los productos salen al mercado o son entregados en ambientes académicos con esos defectos, lo cual origina inconsistencias en los procesos de aprendizaje. De ahí, que sea indispensable medir, con base en métricas, algunas características inherentes del código fuente. La necesidad de tales métricas es particularmente aguda cuando una organización, para desarrollar sus productos, está adoptando una nueva tecnología para la cual aún no se han desarrollado prácticas establecidas (Chidamber & Kemerer, 1994). Por estas razones, fue necesario realizar una investigación de técnicas cuantitativas e implementarlas, de ese modo se brinda la información suficiente de las características del producto que está siendo construido y al mismo tiempo se apoyan los procesos de ingeniería de software.

La investigación descrita en este documento permitió la extracción de los aspectos fundamentales necesarios para desarrollar un componente que sirva de apoyo en los procesos de análisis al código fuente de manera sencilla, dinámica e intuitiva, ajustándose a los requerimientos actuales. La razón de lo anterior es que las herramientas existentes (identificadas en este estudio) que realizan cálculo de determinadas métricas no brindan una descripción detallada y concisa de cómo se llegó a dicho cálculo, impidiendo la correcta comprensión de las particularidades del software analizado. Esto a su vez puede conllevar a una falta de imprecisión en los procesos de desarrollo.

Autores como Lincke *et al* (2008), sugieren en sus investigaciones que se realicen estudios profundos en aras de que la descripción de las métricas implementadas por determinadas herramientas sea detallada. De lo anterior se infiere que en la actualidad existe una necesidad, tanto en las empresas productoras de software a la medida como en las entidades académicas, de una herramienta que les permita a las primeras guiar sus procesos de análisis de software con el fin de ofrecer mayor calidad, y a las segundas ofrecer una mejor enseñanza en el área de ingeniería de software (Lincke *et al*; 2008).

De ese modo, el componente para apoyar el análisis de productos software, supuso una propuesta pertinente y viable, no solamente por todos los beneficios expuestos anteriormente, sino, porque el desarrollo de este proyecto fue sin fines de lucro y es de código abierto.

Que el componente propuesto sea de código abierto supone una ventaja importante, puesto que está disponible para la comunidad en general, esto es, entidades académicas como las universidades e investigadores independientes. También tienen acceso las organizaciones de cualquier índole sin tener en cuenta su sector (industrial, comercial, entre otras) o fines. Es claro que la Universidad de Cartagena será la primera entidad en donde se ponga en marcha su utilización por ser la institución co-autora de la investigación en cuestión. Esto supondrá un apoyo en la enseñanza de algunas temáticas de la asignatura de ingeniería de software, mejorando así la dinámica de la misma. De igual manera, las empresas del sector industrial, financiero, entre otros, que opten por emplear la herramienta de análisis, se verán ampliamente beneficiadas porque tendrán la oportunidad de brindar una atención explicita a las características de la calidad de sus productos software. Estos beneficios conducirían a ahorros significativos en los costos del ciclo de vida (Boehm, Brown, & Lipow, 1976). Es en ese punto donde el cálculo de métricas juega un papel vital.

El carácter Open Source[4] de la herramienta, permite el acceso a la misma sin necesidad de asumir costos elevados, el cual es uno de los principales motivos por los cuales empresas, universidades y estudiosos del área decidan no incluir este tipo de herramientas en sus procesos de desarrollo, enseñanza e investigación respectivamente. La naturaleza académica e investigativa del proyecto justifica los bajos costos del mismo, ella se apoya en el área de ingeniería de software cuyo personal es altamente calificado y además es aportado por la Universidad de Cartagena, por lo tanto se evidencia que el proyecto es viable desde el punto de vista económico.

De igual forma, el proyecto es apoyado por sus propios autores, quienes contribuyeron con sus conocimientos en planeación de proyectos, diseño e implementación de software, con el fin de adquirir y mejorar competencias intelectuales y profesionales ya que el tema del presente proyecto es importante en la fabricación de software.

4 Es el término con el que se conoce al software distribuido y desarrollado libremente.

Lo que se propuso en esta investigación se traduce en una solución que contribuye al progreso de las organizaciones, sabiendo que el análisis pobre del software desemboca en grandes pérdidas económicas que pueden causar la quiebra de una corporación. Por tanto, la implementación de la herramienta para el análisis de productos software puede significar un gran ahorro de dinero para las organizaciones.

Por otro lado, construir una nueva forma de reportar los resultados del cálculo de métricas, constituye un aporte científico importante, ya que fue resultado de investigaciones y rigurosos estudios de herramientas y propuestas similares ya existentes. Lo anterior se complementó con la puesta en práctica del Proceso Unificado Racional (RUP) y la reutilización de código fuente.

En ese mismo sentido, el impacto tecnológico de esta investigación está dado por la parte diferencial e innovadora del software, ya que a nivel nacional no se han realizado proyectos que arrojen detalles concretos sobre las métricas calculadas. Ahora bien, aplicando la metodología RUP, que consta de modelo del negocio, especificación de requisitos, análisis, implementación y pruebas, se desarrolló un software de calidad que satisface las necesidades del medio, cumpliendo con las etapas del proceso y produciendo los artefactos pertinentes. Por tal motivo este proyecto pertenece al área de la Ingeniería de Software.

Finalmente, es pertinente agregar que el campo del cálculo de métricas y su aplicación en procesos de verificación y validación se están volviendo casi que imprescindibles por todo lo que significa para las organizaciones gozar de un software de calidad.

1.4 OBJETIVOS

A continuación se describe el Objetivo General y luego los objetivos específicos del estudio.

1.4.1 Objetivo General

Desarrollar un componente para el cálculo de métricas a partir de código fuente, con el fin de apoyar el análisis de productos software, aplicando la metodología RUP.

1.4.2 Objetivos específicos

- Construir el modelo de negocio para tener una visión holística del problema en cuestión.
- Identificar los requisitos del componente con el fin de establecer las funcionalidades y características del mismo.

- Construir los artefactos de diseño adecuados para el componente a desarrollar, mediante patrones arquitectónicos basados en los requerimientos funcionales.
- Implementar el componente software de cálculo de métricas mediante el lenguaje de programación JAVA.
- Realizar pruebas de carga de archivos y unitaria al componente previamente implementado, de modo que se verifique el correcto análisis de productos software. La prueba unitaria se realizará teniendo en cuenta la técnica de Caja Negra.

1.5 ALCANCE

El presente proyecto tuvo lugar en la ciudad de Cartagena de Indias y estuvo limitado al desarrollo, en un tiempo de aproximadamente 12 meses hábiles, de un componente software para el cálculo de métricas sobre código fuente Java. Dicho código está escrito bajo el paradigma de la POO[5]. El producto final será usado inicialmente por la Universidad de Cartagena. Por ser de carácter OpenSource, estará disponible a la comunidad hispanohablante desarrolladora de software y posiblemente a la comunidad global de esta área. El lenguaje de programación que se usó para su implementación fue JAVA de acuerdo a las necesidades. No obstante, el paradigma de programación fue orientado a objetos. Ahora bien, la utilidad del producto final es la de servir de apoyo en el análisis de productos software y en los procesos de enseñanza y aprendizaje de ingeniería de software. De ese modo, los interesados podrán beneficiarse reduciendo costos en sus ciclos de desarrollo (para el caso de empresas) y adquiriendo conocimientos (academia).

Es importante enfatizar que para realizar el cálculo de las métricas no se utilizaron librerías externas de apoyo, de ahí que el 95% de la codificación fue realizada desde cero. Se utilizaron algunas librerías externas en formato JAR para apoyar funciones secundarias como exportar resultados en PDF y mostrar una barra de progreso mientras se realiza una operación.

Las métricas que se propusieron e implementaron son las siguientes: Acoplamiento entre clases, Falta de cohesión en los métodos, Número de líneas de código, Número de clases y Número de métodos por clase. Se hace énfasis en los resultados arrojados por cada una de estas métricas, por tal motivo fueron tomados como variable de estudio. Además se implementaron métricas adicionales a las antes mencionadas, se trata de métricas derivadas como por ejemplo la

[5]Programación Orientada a Objetos

inestabilidad que es una métrica compuesta que se calcula a partir del acoplamiento eferente y aferente de una clase. Sin embargo, en la tabla 15 se puede apreciar que la métrica puede ser clasificada dentro de Acoplamiento entre clases. Otra métrica que no fue propuesta pero se implementó, es la cantidad de atributos por clase.

En esta investigación, la recolección de información se realizó a partir de fuentes secundarias, como son, artículos científicos, investigaciones, libros, proyectos previos, entre otros. Esto se hizo con el fin de identificar los requerimientos funcionales del componente.

2. MARCO DE REFERENCIA

En este apartado se detalla el estado actual del tema de métricas de código fuente y se define su marco teórico.

Inicialmente se describen los antecedentes a nivel de Norteamérica, y luego a nivel internacional, de Latinoamérica y a nivel nacional. Vale la pena aclarar que en el ámbito local no se identificaron investigaciones precedentes pero se encontró una tesis de pregrado en curso. Esta sección también cuenta con un marco teórico necesario para comprender lo referente a métricas de código fuente.

2.1 Estado del Arte

Aunque el primer libro dedicado a métricas de software no se publicó hasta 1976, el inicio de su historia se remonta a mediados de 1960 cuando la métrica Líneas de Código se utilizó como base para medir la productividad y el esfuerzo de la programación (Fenton & Neil, 2000), y fue hasta finales de los 70’s que se comenzó a hablar de métricas como un mecanismo para medir la calidad (McCall, Richards, & Walters, 1977) La revisión de la literatura disponible ha permitido identificar que hay varios tipos de métricas que se han utilizado en el desarrollo de proyectos software para gestionar, predecir y mejorar la calidad de los mismos (Rodríguez & Harrison, 2007). Al emprender un breve recorrido cronológico, en los siguientes párrafos se describen las investigaciones y aportes más significativos en la evolución del estudio acerca de métricas de código fuente y de otros niveles de abstracción, así como también las herramientas que se han implementado para realizar su cálculo:

Uno de los primeros acercamientos formales en el campo de métricas de software tuvo lugar en **Estados Unidos** y fue presentado en 1976 por Boehm, Brown y Lipow (1976). En su trabajo

muestran un marco conceptual y algunos resultados iniciales clave en el análisis de las características de la calidad del software. Además de llegar a la conclusión de que una mayor atención a las características de la calidad del software puede significar ahorros en los costos del ciclo de vida del software, desarrollan una jerarquía de cualidades bien definidas y bien diferenciadas de la calidad del software, cuyo nivel inferior está estrechamente relacionado con las evaluaciones de métricas de software que se pueden realizar. En ese sentido, también definen, clasifican y evalúan una cantidad significativa de métricas de evaluación de calidad de software, teniendo en cuenta sus beneficios potenciales, cuantificación y facilidad de automatización (Boehm, Brown, & Lipow, 1976). Si bien Boehm *et al* (1976) asumen que su investigación no es ciertamente completa, la han llevado a un punto suficiente para servir como una base viable para futuras mejoras y nuevos enfoques.

En 1990 se publica un estudio exhaustivo sobre las métricas basadas en la ciencia de software de Halstead[6] y la complejidad ciclomatica de McCabe[7], por parte de Coupal & Robillard (1990), los cuales presentan a su vez un procedimiento estadístico para validar dichas métricas. En este estudio también se propone una metodología para analizar grandes proyectos de software con un solo conjunto de métricas. Los resultados se basaron en 19 proyectos que constan de 14.348 rutinas, o bien lo que actualmente se conoce como métodos (Coupal & Robillard, 1990). Sin embargo, este estudio no se pudo tomar como referencia para la investigación puesto que no está disponible a la comunidad, por ende se concluye que no es de Código Abierto.

De acuerdo a lo anterior, las métricas desarrolladas en investigaciones realizadas durante el inicio de los 90's han contribuido a la comprensión del campo de los procesos de desarrollo de software, sin embargo han sido fuertemente criticadas debido a la falta de una base teórica bien fundamentada (Chidamber & Kemerer, 1994).

---

[6]La teoría de Halstead sobre la ciencia del software mide la complejidad del mismo con base en un conjunto de primitivas como son: el número de operadores y operandos distintos que operan en un programa y el número total de ocurrencias de los mismos. La teoría de Halstead se aplica sobre el producto software ya desarrollado (Alonso, Martínez, & Segovia, 2005).

[7]La medida de la complejidad ciclomática de McCabe V (g), representa el flujo de control de un programa mediante un grafo. Cada círculo del grafo representa una tarea de procesamiento y las flechas que unen los círculos son el flujo de control. La complejidad del software se basa en la complejidad ciclomática del grafo de programa de cada módulo (Alonso, Martínez, & Segovia, 2005).

Por fortuna, esa tendencia fue interrumpida por Chidamber & Kemerer (1994), con una investigación en la cual implementan un conjunto de métricas para el diseño Orientado a Objetos, con el fin de cubrir ciertas necesidades tales como el aumento de la demanda de medidas de software. En el estudio, que tuvo lugar en Estados Unidos, se desarrollaron seis métricas de diseño y luego se evaluaron analíticamente, las métricas son: Métodos Ponderados por Clase (WMC), Profundidad de Árbol de Herencia (DIT), Número de Hijos[8] (NOC), Acoplamiento entre Clases de Objetos (CBO), Respuesta para una Clase (RFC) y Falta de Cohesión en Métodos (LCOM). Teniendo en cuenta esto, se desarrolló una herramienta automatizada de recolección de datos para tomar una muestra empírica de estas métricas en dos sitios distintos, con el fin de demostrar su viabilidad y sugerir a los gerentes formas de emplearlas para la mejora de los procesos de desarrollo.

2.1.1 Métricas en Europa

En **Austria**, en el año 1995, Hitz & Montazeri presentaron un estudio titulado "Measuring Coupling and Cohesion in Object-Oriented Systems en el cual evaluaron los enfoques que se llevaban hasta ese entonces en materia de manejo del Acoplamiento y/o la Cohesión en sistemas orientados a objetos. Luego de la evaluación detectaron ciertas deficiencias y a raíz de ellas proporcionaron un marco integral para tratar con todos los tipos de acoplamiento, este marco hace distinción entre el acoplamiento a nivel de objeto y acoplamiento a nivel de clase. Dicha distinción hace referencia a dependencias dinámicas por un lado y dependencias estáticas entre implementaciones por el otro lado, que representan aspectos importantes de la calidad del software en tiempo de ejecución y durante la fase de mantenimiento respectivamente. En cuanto a la cohesión, realizan una análisis de la métrica LCOM presentada inicialmente por Chidamber & Kemerer; y reafirmada por Li & Henry, como resultado presentan una versión teórico-grafica mejorada de esta métrica.

Similarmente en Alemania fue publicado en 1996 un documento que describe los resultados de un estudio en el que se evalúa experimentalmente el impacto del diseño orientado a objetos en las características de calidad del software. Se adoptó un conjunto de métricas para el diseño Orientado a Objetos (OO) denominado MOOD (Metrics for Object Oriented Design), las

[8]Número de subclases inmediatas subordinadas a una clase en la jerarquía de clases (Chidamber & Kemerer, 1994).

métricas son Factor de Ocultamiento en métodos, Factor de Ocultamiento en atributos, Factor de Herencia en métodos, Factor de Herencia en atributos, Factor de Polimorfismo y Factor de Acoplamiento. Los datos obtenidos en el experimento muestran cómo los mecanismos de diseño OO, tales como la Herencia, el Polimorfismo, el Ocultamiento y el Acoplamiento pueden influir en las características de calidad como la confiabilidad o la capacidad de mantenimiento.

En el año 2000, tuvo lugar en el **Reino Unido** la publicación de un documento que pretende ser una hoja de ruta para métricas de software. La guía fue realizada por Fenton & Neil (2000), miembros del Departamento de Ciencias de La Computación de la Queen Mary and Westfield College, ubicada en Londres, Inglaterra. Fenton & Neil (2000), afirman que "la mayoría de las actividades de métricas de software no han abordado su requisito más importante: proporcionar información para apoyar la toma de decisiones de gestión cuantitativa durante el ciclo de vida del software" (Fenton & Neil, 2000). Además establecen que los enfoques de métricas tradicionales, a menudo impulsados por modelos basados en regresión para la estimación de costes y la predicción de defectos, no proporcionan el apoyo suficiente a los gestores que deseen utilizar la medición para analizar y minimizar el (Fenton & Neil, 2000). Por lo anterior recomiendan un enfoque en el cual se manejen los factores clave, generalmente omitidos de los enfoques de métricas habituales, esto es: causalidad, la incertidumbre y la combinación de pruebas diferentes. De ese modo, concluyen que el camino a seguir en la investigación de las métricas de software radica en el modelado causal (usando redes bayesianas), la ingeniería empírica de software y las ayudas de decisión multi-criterio.

Por otra parte, **en Holanda** se abordó uno de los problemas más importantes para los desarrolladores de librerías, la compatibilidad con versiones anteriores. En el estudio realizado en 2012, por parte del Grupo de Mejoramiento de Software de Amsterdam en conjunto con la Universidad de Delft de la misma ciudad, se evaluó la estabilidad de un conjunto de librerías de terceros de uso frecuente en términos de remoción de métodos, cambio de implementación y la proporción de métodos cambiados con relación al total total (Raemaekers, van Deursen, & Visser, 2012). En el artículo también se muestra un ejemplo de una empresa comercial que cuenta con varios problemas relacionados con el uso de bibliotecas de terceros. Ahora bien, para obtener dependencias de sistemas software se desarrolló un framework que extrae dependencias de los archivos de compilación de Maven y que analiza el código del sistema y de la librería. La

frecuencia de uso de los métodos de librería es un dato importante para la métrica final y se obtiene de un conjunto de datos de más de 2300 instantáneas de 140 sistemas Java industriales total (Raemaekers, van Deursen, & Visser, 2012).

2.1.2 Métricas en Asia

En 2011, se llevó a cabo en **Japón** uno de los aportes más completos, en los últimos años, sobre métricas de código fuente. Se trata de una herramienta para medir métricas de software a partir de código fuente. Específicamente, el estudio propone un nuevo mecanismo para medir una variedad métricas de código fuente a bajo costo (Higo, y otros, 2011). Se afirma que el mecanismo es bastante prometedor puesto que permite agregar nuevas métricas según la necesidad (Higo, y otros, 2011). Además, los usuarios no necesitan usar varias herramientas de medición en conjunto para medir múltiples métricas. El mecanismo propuesto fue desarrollado como una herramienta software MASU[9]. El documento de la investigación muestra cómo el uso de MASU hace que sea fácil y menos costoso desarrollar complementos de la suite de métricas CK[10] (Higo, y otros, 2011).

2.1.3 Métricas en Latinoamérica

En **Brasil** a principios de 2014, se publicó la investigación "*Extracción de umbrales relativos para las métricas de código fuente*", cuyo principal problema se describe así: "El establecimiento de umbrales creíbles es un desafío central para promover las métricas de código fuente como un instrumento eficaz para controlar la calidad interna de los sistemas de software (Oliveira, Valente, & Lima, Extracting relative thresholds for source code metrics, 2014). Para abordarlo, en el respectivo documento se propone el concepto de umbrales relativos para evaluar los datos de métricas después de distribuciones de cola pesada (Oliveira, Valente, & Lima, Extracting relative thresholds for source code metrics, 2014). Los umbrales que se proponen en el estudio son relativos porque suponen que los umbrales métricos deben ser seguidos por la mayoría de las entidades de código fuente, pero que también es natural tener un número de entidades en la "cola larga" que no siguen los límites definidos. En ese sentido, en el artículo se describe un método

[9]Plataforma Plugin de Evaluación de Métricas para Unidad de Software (Higo, y otros, 2011)

[10]Suite de métricas de Chidamber & Kemerer.

empírico para extraer umbrales relativos de sistemas reales (Oliveira, Valente, & Lima, Extracting relative thresholds for source code metrics, 2014).

Además de lo anterior, en ese mismo documento también se reporta la aplicación del método descrito en un corpus con 106 sistemas. Finalmente, en el documento se concluye que los umbrales propuestos expresan un equilibrio entre las prácticas de diseño real e idealizado (Oliveira, Valente, & Lima, Extracting relative thresholds for source code metrics, 2014).

En el mismo año 2014, en un artículo subsiguiente, Oliveira, Lima, Valente & Serebrenik (2014), afirman que a pesar del aumento constante de herramientas de medición de código fuente, ninguna herramienta disponible públicamente soporta la extracción de umbrales métricos (Oliveira, Lima, Valente, & Serebrenik, 2014). Por otro lado, estudios anteriores sugieren que en sistemas grandes, un número significativo de clases superan los umbrales métricos recomendados. Por esta razón, en el primer estudio se introdujo la noción de umbral relativo, que consiste en un límite que los valores métricos de un porcentaje clases no deben exceder. Teniendo en cuenta lo anterior, Oliveira, Lima, Valente & Serebrenik (2014) proponen la herramienta de código abierto RTTool, para extraer umbrales relativos de los datos de medición de un punto de referencia de sistemas de software.

2.1.4 Métricas a nivel nacional

En 2016, se publicó un artículo producto de una investigación titulada Apropiación y utilización de las métricas en medianas y pequeñas empresas de software de la ciudad de Medellín, cuyo objetivo fue diagnosticar la apropiación y utilización de las métricas en estas empresas, para realizar recomendaciones que contribuyan al fortalecimiento de la academia y del sector productivo (Metaute & Serna, 2016). Los autores abordaron diversas temáticas entre las cuales están, el tiempo de aplicación de métricas, su aplicación en diferentes fases, certificaciones en métricas, metodologías, herramientas utilizadas, procesos donde se aplican métricas, aportes al aplicarlas, tipos de prueba donde se aplican, entre otras. En el artículo se afirma que lo anterior desembocó en la identificación de hallazgos que crearon discusión argumentada desde las perspectivas de la normatividad, buenas prácticas y necesidades de los diferentes contextos donde se utilizan productos de software (Metaute & Serna, 2016).

La investigación también dio como resultado “conclusiones e implicaciones prácticas que permitieron realizar un diagnóstico sobre la apropiación de las métricas en medianas empresas de software de la ciudad de Medellín” (Metaute & Serna, 2016), así como también arrojó algunas sugerencias para la academia, orientadas al fortalecimiento de asignaturas que tienen como responsabilidad la generación de competencias en Ingeniería de software, especialmente en las métricas, buscando aportes significativos y contextualizados para este campo de la ingeniería.

2.1.5 Métricas a nivel local

La siguiente es una **tesis de pregrado en curso** localizada en Cartagena de Indias:

En Cartagena de Indias, Colombia, se está desarrollando una investigación por parte de los estudiantes Alexander Silvera Charris e Ismael Sayas Arrieta, titulada Elaboración de un componente de software para el cálculo de métricas de diseño a partir de XMI, cuyo objetivo es desarrollar un componente reutilizable que permita a los desarrolladores de software realizar cálculo de métricas de diseño a partir de diagramas en formato XMI. Los aportes de este estudio son considerablemente importantes, no solo por el carácter reutilizable del componente, sino también por la recuperación de métricas de un nivel de abstracción más elevado que el tratado en la presente investigación, ya que su obtención puede representar una mayor comprensión del modelo de diseño del producto software analizado. Muchas de las métricas de diseño están íntimamente relacionadas con métricas de código fuente, o bien son aplicables a ambos niveles de abstracción.

Este proyecto hace parte de la tesis doctoral titulada "Marco de referencia para la recuperación y análisis de vistas arquitectónicas de comportamiento", desarrollada por Martín Monroy Ríos (2016), profesor de la Universidad de Cartagena, en el que se hace uso del cálculo de métricas para analizar productos software (Monroy M. , 2016).

A pesar de que algunas de las investigaciones encontradas en la literatura han realizado aportes significativos en el área, ninguna se enfoca en apoyar los procesos de aprendizaje de la ingeniería de software a través del cálculo de métricas sobre código fuente. En ese sentido surgió la necesidad de realizar la investigación sobre métricas de código fuente utilizando como referencia algunos de los antecedentes descritos anteriormente.

Es pertinente aclarar que en la revisión de la literatura no se identificó alguna otra tesis de pregrado o postgrado que se haya realizado en la ciudad de Cartagena de Indias.

2.2 Marco Teórico

Es importante establecer una base teórica bien definida, clara y concisa con el fin de comprender lo referente a métricas de código fuente. Para ello es necesario abordar conceptos tales como ingeniería de software, calidad de software y factores de calidad; así como también los conceptos de RUP y UML. En ese sentido, este capítulo se desarrolla en forma Top-down, es decir desde lo más general hasta lo especifico.

2.2.1 Ingeniería de Software

Cientos de autores han establecido definiciones personales sobre la ingeniería de software, una de ellas es la siguiente: "La ingeniería de software es el establecimiento y uso de principios fundamentales de la ingeniería con objeto de desarrollar en forma económica software que sea confiable y que trabaje con eficiencia en máquinas reales." (Pressman R. S., Ingeniería de Software, 2010). Es una aseveración concreta pero en ella no se habla directamente de la satisfacción de los consumidores o de entregar el software a tiempo; además omite la medición y no se establece la importancia de un proceso eficaz.

El IEEE (IEEE, 1993) ha desarrollado una definición completa de ingeniería de software: "La ingeniería de software es: 1) La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software; es decir, la aplicación de la ingeniería al software. 2) El estudio de enfoques según el punto 1.". Aun con esto, el enfoque que se da "Sistemático, disciplinado y cuantificable" trabajado por un equipo de software puede ser algo tosco para otro. Se requiere disciplina, como también la adaptabilidad y agilidad.

2.2.2 Calidad de Software

Definiendo formalmente lo que es la calidad de software de manera general y tomada del libro (Pressman R. S., 2010), se define como: "Proceso eficaz de software que se aplica de manera que crea un producto útil que proporciona valor medible a quienes lo producen y a quienes lo utilizan".

2.2.3 Modelos, Factores y Criterios de Calidad

Los modelos de calidad son aquellos documentos que acoplan la mayor parte de las mejores prácticas, proponen temas de administración en los que cada organización deber hacer énfasis, componen varias prácticas dirigidas a procesos que son importantes y permiten medir los avances en la calidad. (Constanzo, 2014)

Los estándares de calidad permiten definir un conjunto de criterios de desarrollo que guían la forma en que se aplica la ingeniería de software. Los estándares da suministro a los medios para que todos los procesos se realicen de una misma forma y son una guía para lograr la productividad y la calidad, (Constanzo, 2014) La figura 1 muestra la estructura de un modelo de calidad.

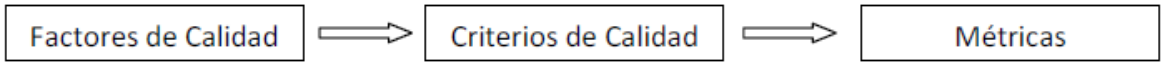


Figura 1. Estructura de un Modelo de Calidad. Tomada de *(Constanzo, 2014)*.

Los factores de calidad o atributos externos son características que forman la calidad, representan la calidad desde la perspectiva del usuario. Por otro lado, los factores de calidad o atributos internos son aquellos en los que se descomponen los diferentes factores, se representan por ser la calidad desde el punto de vista del producto software, son aspectos de la calidad que se asocian a cada factor. También se tienen las métricas que se definen para cada criterio de calidad, son medidas cuantitativas que muestran el grado en el que está presente un atributo en el software.

Para evaluar la calidad del producto software, existen distintos modelos, formados por factores y criterios asociados. Al evaluar estos factores de calidad se puede determinar la calidad del producto software.

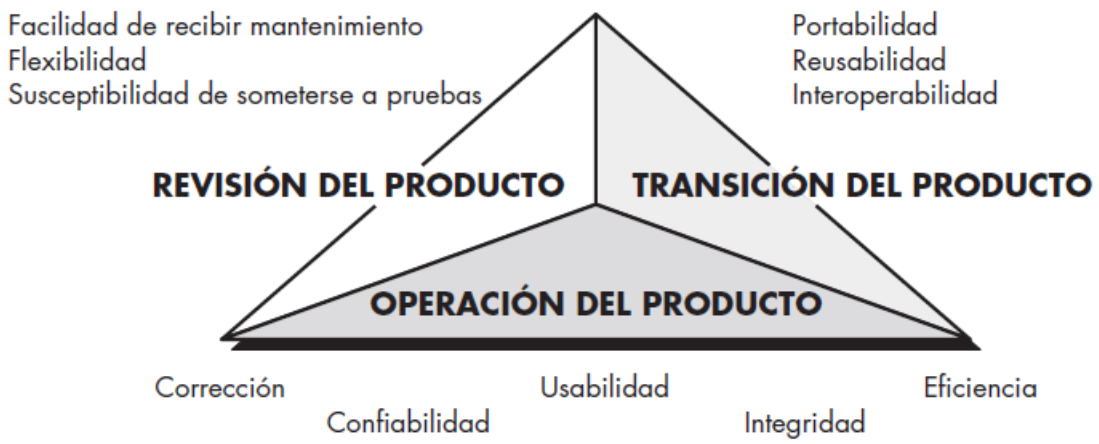


Figura 2. Factores de la calidad de McCall. Tomada de *(Pressman R. S., Factores de la calidad de McCall, 2010)*

2.3 Marco Conceptual

A continuación se define un conjunto de conceptos utilizados en este documento.

Componente: según Robert C. Martin, un componente es una unidad binaria desplegable (o que se puede desplegar) de forma independiente (Martin C. Robert, 2006).

Siendo elementos de vital importancia en grandes sistemas de software, los componentes permiten que dichos sistemas se descompongan en productos binarios más pequeños. Si las dependencias entre los componentes están bien administradas, es posible corregir errores y agregar funciones modificando solamente aquellos componentes que han cambiado. El diseño de grandes sistemas depende fundamentalmente de un buen diseño de componentes, de modo que esto permite enfocarse en componentes aislados en lugar de preocuparse por todo el sistema (Martin C. Robert, 2006).

Para el caso de esta investigación, el componente puede ser ejecutado como un JAR. (Ver SRS, sección 3.1 Interfaces Externas/ Interfaz del Software en la ruta \Componente Software para el Cálculo de Métricas a Partir de Código Fuente\4. Anexos\SRS - Componente Software para el Cálculo de Métricas a partir de Código Fuente)

Cohesión: se define como la relación funcional de los elementos de un modulo (Robert C. Martin, 2006). La anterior es la definición dada al Principio de Responsabilidad Única descrito en el trabajo de Tom DeMarco y Meillir Page-Jones, quienes definen responsabilidad como “Una

clase debería tener una única razón para cambiar". En ese sentido, se puede decir que una clase tiene más de una responsabilidad si tiene más de un motivo para cambiar en algún momento. O dicho de otra manera: si una clase asume más de una responsabilidad, esa clase tendrá más de un motivo para cambiar. En ese orden de ideas, **una clase es poco cohesiva cuando tiene más de una responsabilidad**.

Total de líneas de código: es una métrica que se calcula por cada unidad de compilación (archivo fuente .java). En este caso, la herramienta desarrollada en esta investigación cuenta las líneas de código incluyendo los comentarios, también se tienen en cuenta las líneas con caracteres de espacio en blanco. Es una métrica a nivel de unidad de compilación.

Número de clases: es la cantidad total de clases por unidad de compilación, es decir que por cada archivo fuente puede haber una o más clases. Es una métrica a nivel de unidad de compilación.

Complejidad Ciclomática de McCabe: cuenta el número de flujos en un fragmento de código. Cada vez que se produce una derivación (if, for, while, do, caso, catch y el operador Ternario, así como los operadores de lógica condicional && y || en las expresiones) esta métrica se incrementa en uno. La métrica se calcula para métodos.

Falta de cohesión: diversos autores han propuesto diferentes definiciones y formas de calcular la falta de cohesión; para el caso de esta investigación se escogió LCOM4 propuesta por Hitz & Montazeri, en su estudio 'Measuring Coupling and Cohesion In Object-Oriented Systems' y le llaman 'Improving LCOM'.

LCOM4 se define como el número de "componentes conectados" en una clase. Un componente conectado es aquel cuyos métodos y atributos están relacionados directa o indirectamente. Debe haber solo un componente de este tipo en cada clase. Si hay 2 o más componentes, la clase debe dividirse en clases más pequeñas (Hitz & Montazeri, 1995).

La Falta de cohesión (LCOM4), para el caso de este proyecto, se calcula de la siguiente forma**:** (Aivosto Oy, n.d.).

Por ejemplo, supóngase que hay dos métodos a y b; Los métodos a y b están relacionados si:

- ambos acceden al mismo atributo a nivel de clase, o

- a llama b, o b llama a.

LCOM4 es igual al número de grupos de métodos conectados.De acuerdo a eso se tiene:

- LCOM4 = 1 indica una clase cohesiva.
- LCOM4> = 2 indica un problema. La clase debe dividirse en tantas clases más pequeñas sean necesarias.
- LCOM4 = 0 sucede cuando no hay métodos en una clase. Esto sugiere una mala práctica en la codificación de la clase.

Detalles de implementación LCOM4: es una métrica a nivel de clase pero se calcula a partir de los métodos. Para el cálculo no se tienen en cuenta procedimientos (métodos) vacíos. Los métodos vacíos tienden a aumentar el valor de LCOM4 ya que no acceden a ninguna variable u otros métodos. En ese orden de ideas, una clase cohesiva con procedimientos vacíos tendría un alto LCOM4. También se ignoran los constructores y destructores. Los constructores y destructores frecuentemente establecen y borran todas las variables en la clase, haciendo que todos los métodos se conecten a través de estas variables, lo que aumenta la cohesión artificialmente (Hitz & Montazeri, 1995).

Número de métodos: Es la cantidad total de métodos definidos en una clase.

Acoplamiento entre clases: en este proyecto se utilizan 2 tipos de acoplamiento definidos por Robert Martin, que son Acoplamiento Aferente y Acoplamiento Eferente. Originalmente son métricas que se aplican a paquetes y/o componentes pero en esta investigación estos valores se calculan a nivel de clases.

Antes de definir los tipos de acoplamiento es pertinente saber qué es el Acoplamiento en general. Según Chidamber y Kemerer, citados en (Hitz & Montazeri, 1995), cualquier evidencia de un método de un objeto utilizando métodos o las variables de instancia de otro objeto constituyen acoplamiento. También se considera acoplamiento cuando se extiende y cuando se implementa.

Acoplamiento Eferente (Ce): Si bien es cierto que en esta investigación el acoplamiento eferente se calcula a nivel de clases, es una métrica que inicialmente fue definida a **nivel de componente** por Robert C. Martin como el número de dependencias salientes de un componente

software, es decir, es una medida que indica el número de usos de clases externas al componente. Para el caso específico de este proyecto, es la cantidad de clases de las que depende una clase X.

Acoplamiento Aferente (Ca): según Robert C. Martin, es el número de dependencias entrantes de un componente software, es decir, es una medida del número de clases externas que dependen de dicho componente. En el caso de este proyecto, es la cantidad de clases que dependen de una clase X.

Factor de Acoplamiento (CF): es una métrica a nivel de sistema y hace parte del conjunto de métricas MOOD (Métricas para diseño orientado a objetos) de (Abreu & Melo, 1996).

Se define como la relación entre el número real de acoplamientos del sistema y el número máximo posible de acoplamientos en el sistema (Rodríguez & Harrison, 2001). Formalmente se define así:

$$CF = \frac{\sum_{i=1}^{TC}\left[\sum_{j=1}^{TC} acop(C_i, C_j)\right]}{(TC^2 - TC)}$$

Donde,

Acop(Ci, Cj): Indica la cantidad de acoplamientos que tiene una clase Ci. ***El resultado es 1 si la clase Ci contiene al menos una referencia a un método o atributo de la clase Cj, de lo contrario el resultado es 0.***

$\sum_{i=1}^{TC}\left[\sum_{j=1}^{TC} acop(C_i, C_j)\right]$**: Representa el número total real de acoplamientos**

$(TC^2 - TC)$**: Número máximo de acoplamientos posibles para un sistema**

TC: cantidad total de clases del sistema

C: clase del sistema

En el contexto de esta investigación se hace el cálculo de CF de dos formas:

- Calculo de CF a partir de las relaciones entre clases
- Calculo de CF a partir de las relaciones entre paquetes

Vale la pena aclarar que la formula en ambos modalidades es la misma, lo único que cambia es el tipo de modulo que se usa como referencia (Clase o Paquete).

Calculo de CF a partir de las relaciones entre clases

$$CF = \frac{\sum_{i=1}^{TC}\left[\sum_{j=1}^{TC} acop(C_i, C_j)\right]}{(TC^2 - TC)}$$

Calculo de CF a partir de las relaciones entre paquetes

Nota: para el cálculo se tienen en cuenta los acoplamientos con paquetes (librerías) propias de Java.

$$CF = \frac{\sum_{i=1}^{n}\left[\sum_{j=1}^{n} acop(P_i, P_j)\right]}{(n^2 - n)}$$

Donde,

Acop(Pi, Pj): Indica la cantidad de acoplamientos que tiene un paquete Pi. ***El resultado es 1 si al menos una clase del paquete Pi tiene una referencia a un método o atributo de una clase perteneciente al paquete Pj, de lo contrario el resultado es 0.***

(n^2 – n): número máximo de acoplamientos del sistema

n: cantidad total de paquetes del sistema

P: paquete del sistema

Un proyecto de código fuente tiene un nivel normal de acoplamiento cuando el valor de CF es menor o igual a 33,3% s. Un grado de acoplamiento mayor a 33,3% y menor o igual a 50% indica que el sistema tiene un nivel medio de acoplamiento. Un grado de acoplamiento mayor a 50% y menor o igual a 99% indica un nivel de acoplamiento alto. Si todos los paquetes están interrelacionados entre sí el CF será igual a 100%, lo cual significa que el sistema tiene el máximo número de acoplamientos. En este documento se puede hacer referencia a factor de acoplamiento como CF o FA.

3. METODOLOGÍA

Inicialmente este proyecto fue desarrollado en la ciudad de Cartagena de Indias desde Enero de 2018 hasta Abril de 2019. Esta sección inicia con la descripción del tipo de investigación a la

cual pertenece este proyecto y las fuentes de información, luego se presentan las metodologías que se aplicaron a fin de cumplir con los objetivos propuestos. Se muestra la adaptación de RUP detallando las actividades a realizar en cada fase. También se explica la manera en la cual se trataron los datos obtenidos en la investigación.

3.1 Tipo y método de Investigación

El presente proyecto pertenece a un tipo de investigación bibliográfica y aplicada. Es bibliográfica porque se realizó teniendo en cuenta estudios anteriores y en desarrollo, cuya información fue proporcionada a través de artículos, libros, revistas y demás publicaciones de índole científica relacionadas con el tema de cálculo de métricas. Es de carácter aplicada puesto que se implementó como un componente de carácter OpenSource que calcula métricas de software definidas bajo un marco teórico.

En primer lugar se empleó el método analítico, con el cual se obtuvo información relevante que fue usada como guía y base para el desarrollo del componente de cálculo de métricas

3.2 Proceso de recolección de información

En esta sección se da una descripción detallada de los procesos de recolección de la información y lo que se hizo con la misma una vez obtenida. La validez y objetividad de la información recogida puede ser comparada con la revisión de la literatura (Véase 2.1 ESTADO DEL ARTE). Este proceso también se encuentra en el ítem 4.1.1 Revisión de la literatura que hace parte del capítulo 4. RESULTADOS.

3.2.1 Forma de recolección

La búsqueda de los estudios se realizó en las siguientes bases de datos bibliográficas: IEEExplore, ACM Digital Library, ScienceDirect, ComputerResearch, Wiley Online Library y además se aplicó también una búsqueda directamente en Google Scholar. En cada una de ellas se aplicaron las siguientes cadenas de búsqueda: “source code metrics”, “software metrics”, “source code metrics calculation”, “object oriented source code metrics”, “software object oriented metrics”, “software code metrics”, “code metrics”, “source code metrics Colombia”.

Esto implicó realizar 8 búsquedas en cada base de datos para un total de 48 que se realizaron en los campos: título, resumen, palabras clave y conclusiones (en algunos casos cuando la información del resumen no era suficiente), obteniendo un total de 387 documentos

encontrados. Se seleccionaron inicialmente los documentos que cumplieron con los siguientes criterios de inclusión: Capítulos de libros, reportes técnicos y artículos de revistas que presenten resultados de estudios empíricos publicados durante los últimos 5 años, sin embargo se seleccionaron algunos publicados en fechas anteriores por su grado de pertinencia y relevancia para el estudio actual.

3.2.1.1 Criterios de inclusión y exclusión

Respecto a los **criterios de inclusión**, se tienen:

- Se incluyeron aquellos documentos que obedecieron a la búsqueda definida por las siguientes cadenas de caracteres: “source code metrics”, “software metrics”, “source code metrics calculation”, “object oriented source code metrics”, “software object oriented metrics”, “software code metrics”, “code metrics”, “source code metrics Colombia”.
- Se incluyeron aquellos documentos cuyo tema principal era el cálculo de métricas sobre código fuente.

Para depurar el listado obtenido se aplicaron los siguientes **criterios de exclusión**:

- Documentos duplicados
- Documentos cuyo tema principal no era el cálculo de métricas sobre código fuente
- Documentos que no representan mayores aportes para este estudio.

En la tabla 1 se relaciona la cantidad de artículos seleccionados por cada base de datos consultada, los porcentajes son respecto del total de estudios seleccionados.

Tabla 1. Cantidad de artículos seleccionados por base de datos bibliográfica

Base de datos	Encontrados	Seleccionados	Porcentaje respecto del total seleccionado
IEEExplore	61	12	50%
ACM	27	5	21%
ScienceDirect	43	1	4%
ComputerResearch	29	1	4%
Wiley Online	32	2	8%

Library			
Búsqueda realizada directamente en Scholar	195	3	13%
Total	**387**	**24**	100%

Después de una segunda depuración, se encontró que los estudios más pertinentes para este proyecto son 7 en total. Lo anterior teniendo en cuenta el aporte conceptual y práctico que pudiesen representar dichos estudios para el proyecto actual, por tanto esto se define como un **criterio de inclusión**. En la tabla 2 se muestra la relación de la segunda depuración, los porcentajes son respecto del total de estudios seleccionados.

Tabla 2. Estudios seleccionados en la segunda depuración

Base de datos	Encontrados	Seleccionados	Porcentaje respecto del total seleccionado
IEEExplore	12	4	57%
ACM	5	1	14%
ScienceDirect	1	0	0%
ComputerResearch	1	1	14%
Wiley Online Library	2	0	0%
Scholar	3	1	14%
Total	**24**	7	

3.2.1.2 Análisis de los estudios seleccionados

El análisis de los estudios seleccionados se presenta en la tabla 3, utilizando las convenciones que se explican a continuación.

Tipos de estudio: No experimento (NE), reporte de experiencia (RE), estudio de caso (EC), cuasi-experimento (QE), estudio a partir de la observación (EO), revisión sistemática (RS) y experimento aleatorio (EA)

Propósito: Propuesta conceptual (PC), prueba de conceptos (PbC) comparación (C) o revisión de la literatura (RL).

Si el estudio no ofrece la herramienta en un sitio web, significa que es una herramienta privada.

Tipo de herramienta: Escritorio, Web, RIA, Servicio.

NOTA: Si el estudio no muestra una herramienta, las columnas relacionadas se dejan en blanco.

Si algún estudio no hace claridad en su arquitectura o uso de base de datos se coloca la notación NI que significa No Indica.

En la tabla 3 sólo se muestran 4 de los 7 estudios seleccionados ya que los 3 restantes son netamente conceptuales y no incluyen herramientas que calculen métricas sobre código fuente. Dichos artículos son:

- Comparison Study and Review on Object- Oriented Metrics
- A Metrics Suite for Object Oriented Design
- Medición en la Orientación a Objetos

Tabla 3. Análisis de estudios seleccionados

Estudio	RTTOOL : a tool for extracting relative thresholds for source code metrics	Quantitative Evaluation of Software Quality Metrics in Open-Source Projects	Experimental Assessment of Software Metrics Using Automated Refactoring	A Pluggable Tool for Measuring Software Metrics from Source Code
Tipo de estudio	RE	QE	EA	RE
Propósito del estudio	PbC	PbC	PbC, C	PC
Métricas	• Coupling Between Classes • Fan In • Fan Out • Hierarchy Nesting Level • Lack Of Cohesion In Methods • Number Of Attributes • Number Of Attributes Inherited • Number Of Children • Number Of Lines Of Code • Number Of Methods • Number Of Methods Inherited • Number Of Methods	Métricas que se usan en este experimento y que hacen parte de la herramienta **VizzAnalyzer** Métricas de **Chidamber & Kemerer**: • Combinación entre Objetos (CBO) • Profundidad del Árbol de la Herencia (DIT) • Falta de Cohesión en los Métodos (LCOM) • Número de Hijos (NOC) • Respuesta para una	Se tienen cinco métricas ampliamente utilizadas de cohesión LSCC, TCC, CC, SCOM y LCOM5. Se usa Code-Imp como plataforma de refactorización guiada para métricas basadas en la búsqueda.	• WMC (Weighted Methods per Class) • DIT (Depth of Inheritance Tree) • NOC (Number Of Children) • CBO (Coupling Between Object classes) • RFC (Response For a Class) • LCOM (Lack of Cohesion in Methods) • CYC (CYClomatic complexity)

	Overriden • Number Of Private Attributes • Number Of Private Methods • Number Of Public Attributes • Number Of Public Methods • Response For Class • Total Number Of Children • Weighted Method Count • Weight Of A Class	Clase (RFC) • Método Ponderado (WMC) usando la Complicidad Ciclomática de McCabe como peso para los métodos; **Li & Henry:** • Acoplamiento de abstracción de datos (DAC) • acoplamiento de paso de mensajes (MPC) • número de métodos locales (NOM) • número de atributos y métodos (NAM / SIZE2) **Bieman & Kang**: • Coherencia de Clase Estrecha (TCC) **Hitz & Montazeri**: • Lociedad de los datos (LD)		

		• Mejora de LCOM (ILCOM) • Y las métricas Líneas de código (LOC) • falta de documentación (LOD) • Longitud de los nombres de clase (LEN).		
Herramienta OpenSource (Sí o No)	Sí	Si, las herramientas que se definen en el experimento son de carácter opensource. Pero no se identifica si las herramientas que se desarrollaron por los autores sean libres, las cuales son **SF Extract** y **Extract SVN**	No se propone herramienta	Sí
Tipo de herramienta	Herramienta de escritorio	El analizador que se usa es de tipo escritorio		Escritorio
Arquitectura	N capas	NI		NI

usada				
¿Base de datos?	No	Si, para este experimento se tiene una base de datos que guarda los datos exportados por VizzAnalyzer, como las métricas para cada proyecto, los nombres y versión del proyecto, algunos metadatos y estadísticas adicionales		No

3.2.1.3 Herramientas de cálculo de métricas a partir código fuente

En la tabla 4 (que se observa abajo) se pueden observar las características de las herramientas relacionadas con métricas de código fuente encontradas en la revisión de la literatura.

	¿Muestra explicación de los resultados?	**Tipo de herramienta**	**Arquitectura usada**	**¿La herramienta utiliza base de datos? (Sí, No, NI)**	**¿Código fuente disponible?**
Gmetrics	No, solo muestra un reporte de resultados.	Escritorio	NI. Actúa como un plugin de complemento	No	Sí
MASU	No	Escritorio	Arquitectura basada en plugins.	No	Sí
RTTOOL	No (Esta herramienta no calcula las métricas)	Escritorio	N capas	No	Sí
Eclipse Metrics Plugin	No	Plug-in de Eclipse	NI	NI	Sí
VizzMaintenance	No (Por lo menos en la interfaz de resultados)	Escritorio	Variación de N capas	No	No

Nota: Las herramientas Gmetrics y Masu no suministran un manual de instalación para el usuario o una guía específica para ello. Por dichas razones, estas herramientas no se pudieron explorar.

3.3 Procedimiento de la investigación

En esta sección se muestra un resumen de cada paso en el desarrollo de la investigación.

Específicamente se muestra cómo fue aplicado el método de investigación analítico y la metodología RUP para la consecución de los objetivos específicos; el método analítico ayudó a cumplir en parte con el objetivo específico uno. El resto de objetivos y su cumplimiento fueron organizados de acuerdo a las fases de la metodología RUP: Fase inicial, Fase de Elaboración, Fase de Construcción y Cierre y Fase de Transición.

Puesta en práctica del método de investigación Analítico

Inicialmente se realizó una revisión minuciosa de la literatura y un análisis de herramientas existentes a fin de extraer bases teóricas y prácticas del cálculo de métricas y sus restricciones, así se comprendió la temática, se estableció el alcance del proyecto y se construyó un modelo de negocio que responde a los requerimientos identificados en la revisión bibliográfica. Por lo anterior, este método ayudó a cumplir en parte con el **objetivo específico número 1**.

Ajora bien, puesto que la investigación fue de tipo aplicada, se utilizó la metodología de desarrollo RUP para la construcción del componente, según se explica a continuación:

Fase inicial

Esta fase tuvo como resultado la construcción del modelo de negocio para dar solución al **primer objetivo específico**.

Para poder construir el modelo de negocio se identificó la esencia del problema a través del levantamiento de requerimientos, para esto se recolectó información acerca de la aplicación e implementación de las métricas establecidas (ver sección Alcance) a través de una revisión bibliográfica de investigaciones y software realizados con anterioridad.

Fase de Elaboración

En esta etapa también se cumplieron algunas actividades del **primer objetivo específico**, como lo son: elaboración del modelo del dominio, diagrama de actividades y diagrama de casos de uso, todos en el contexto del mundo real.

Además, se especificaron los requisitos funcionales y no funcionales a partir de la información obtenida en la fase inicial, así como también se elaboró el diagrama de casos de uso a nivel de requisitos. Esto con el propósito de cumplir con el **segundo objetivo específico**.

Una vez identificadas las funcionalidades del componente, se definió la arquitectura del mismo. A su vez, se crearon los artefactos UML correspondientes como lo son diagrama general de casos de uso, diagrama de clases, diagrama de componentes, entre otros. Todo esto con el fin de dar respuesta al **tercer objetivo específico**.

Actividades como elaboración del diagrama de despliegue y de paquetes también se ubican en esta fase, pero pertenecen al **cuarto objetivo específico**.

Fase de construcción y cierre

En esta fase se inició el proceso de codificación para materializar las funcionalidades y así se obtuvo una primera versión del componente deseado el cual fue mejorado progresivamente con el avance del tiempo.

Se realizó una programación bajo el paradigma orientado a objetos y las actividades de implementación necesarias para dar solución al **cuarto objetivo específico**. Vale la pena agregar que en esta etapa también se realizaron los manuales de usuario y del sistema del componente.

Fase de transición

Con el fin de cumplir con el **quinto y último objetivo específico**, en esta fase se diseñaron y aplicaron las respectivas pruebas de carga de archivos y Caja Negra para verificar la funcionalidad y usabilidad del componente software desarrollado. Con base en los resultados arrojados se redactó un informe donde se especifican los aspectos relevantes de la realización de las pruebas.

4. RESULTADOS

El capítulo correspondiente a los resultados está dividido en 5 grandes ítems. Cada ítem corresponde a un objetivo específico. El orden es el siguiente: Modelo de negocio, Requisitos, Modelo de diseño, Implementación y pruebas.

4.1 Modelo de Negocio

En este ítem se observan los resultados obtenidos a partir de la búsqueda bibliográfica realizada en bases de datos científicas. Luego se muestran una serie de tablas con la cantidad de estudios seleccionados teniendo en cuenta criterios de exclusión e inclusión. Inmediatamente después, se detallan un par de tablas con el análisis de los estudios y las herramientas de cálculo de métricas a partir de código fuente existentes.

Más adelante, se encuentra el diagrama de casos de uso del mundo real, el modelo de dominio y el diagrama de actividades.

4.1.1 Revisión de la literatura

La búsqueda de los estudios se realizó en las siguientes bases de datos bibliográficas: IEEExplore, ACM Digital Library, ScienceDirect, ComputerResearch, Wiley Online Library y además se aplicó también una búsqueda directamente en Google Scholar. En cada una de ellas se aplicaron las siguientes cadenas de búsqueda: “source code metrics”, “software metrics”, “source code metrics calculation”, “object oriented source code metrics”, “software object oriented metrics”, “software code metrics”, “code metrics”, “source code metrics Colombia”.

Esto implicó realizar 8 búsquedas en cada base de datos para un total de 48 que se realizaron en los campos: título, resumen, palabras clave y conclusiones (en algunos casos cuando la información del resumen no era suficiente), obteniendo un total de 387 documentos encontrados. Se seleccionaron inicialmente los documentos que cumplieron con los siguientes criterios de inclusión: Capítulos de libros, reportes técnicos y artículos de revistas que presenten resultados de estudios empíricos publicados durante los últimos 5 años, sin embargo se seleccionaron algunos publicados en fechas anteriores por su grado de pertinencia y relevancia para el estudio descrito en este documento.

4.1.2 Criterios de inclusión y exclusión

Respecto a los criterios de inclusión, se tienen:

- Se incluyeron aquellos documentos que obedecieron a la búsqueda definida por las siguientes cadenas de caracteres: “source code metrics”, “software metrics”, “source code metrics calculation”, “object oriented source code metrics”, “software object oriented metrics”, “software code metrics”, “code metrics”, “source code metrics Colombia”.
- Se incluyeron aquellos documentos cuyo tema principal era el cálculo de métricas sobre código fuente.

Para depurar el listado obtenido se aplicaron los siguientes **criterios de exclusión**:

- Documentos duplicados
- Documentos cuyo tema principal no era el cálculo de métricas sobre código fuente
- Documentos que no representan mayores aportes para este estudio.

En la tabla 5 se relaciona la cantidad de artículos seleccionados por cada base de datos consultada, los porcentajes son respecto del total de estudios seleccionados.

Tabla 4. Cantidad de artículos seleccionados por base de datos bibliográfica

Base de datos	Encontrados	Seleccionados	Porcentaje respecto del total seleccionado
IEEExplore	61	12	50%
ACM	27	5	21%
ScienceDirect	43	1	4%
ComputerResearch	29	1	4%
Wiley Online Library	32	2	8%
Búsqueda realizada directamente en Scholar	195	3	13%
Total	**387**	**24**	100%

Después de una segunda depuración, se encontró que los estudios más pertinentes para este proyecto son 7 en total. Lo anterior teniendo en cuenta el aporte conceptual y práctico que pudiesen representar dichos estudios para el proyecto actual, por tanto este se define como un **criterio de inclusión**. En la tabla 6 se muestra la relación de la segunda depuración, los porcentajes son respecto del total de estudios seleccionados.

Tabla 5. Estudios seleccionados en la segunda depuración

Base de datos	Encontrados	Seleccionados	Porcentaje respecto del total seleccionado
IEEExplore	12	4	57%
ACM	5	1	14%
ScienceDirect	1	0	0%
ComputerResearch	1	1	14%
Wiley Online Library	2	0	0%
Scholar	3	1	14%
Total	**24**	7	

4.1.3 Análisis de los estudios seleccionados

El análisis de los estudios seleccionados se presenta en la tabla 7, utilizando las convenciones que se explican a continuación.

Tipos de estudio: No experimento (NE), reporte de experiencia (RE), estudio de caso (EC), cuasi-experimento (QE), estudio a partir de la observación (EO), revisión sistemática (RS) y experimento aleatorio (EA)

Propósito: Propuesta conceptual (PC), prueba de conceptos (PbC) comparación (C) o revisión de la literatura (RL).

Si el estudio no ofrece la herramienta en un sitio web, significa que es una herramienta privada.

Tipo de herramienta: Escritorio, Web, RIA, Servicio.

NOTA: Si el estudio no muestra una herramienta, las columnas relacionadas se dejan en blanco.

Si algún estudio no hace claridad en su arquitectura o uso de base de datos se coloca la notación NI que significa No Indica.

En la tabla 7 sólo se muestran 4 de los 7 estudios seleccionados ya que los 3 restantes son netamente conceptuales y no incluyen herramientas que calculen métricas sobre código fuente. Dichos artículos son:

- Comparison Study and Review on Object- Oriented Metrics
- A Metrics Suite for Object Oriented Design
- Medición en la Orientación a Objetos

Tabla 6. Análisis de estudios seleccionados

Estudio	RTTOOL : a tool for extracting relative thresholds for source code metrics	Quantitative Evaluation of Software Quality Metrics in Open-Source Projects	Experimental Assessment of Software Metrics Using Automated Refactoring	A Pluggable Tool for Measuring Software Metrics from Source Code
Tipo de estudio	RE	QE	EA	RE
Propósito del estudio	PbC	PbC	PbC, C	PC
Métricas	• Coupling Between Classes • Fan In • Fan Out • Hierarchy Nesting Level • Lack Of Cohesion In Methods • Number Of Attributes • Number Of Attributes Inherited • Number Of Children • Number Of Lines Of Code • Number Of Methods • Number Of Methods	Métricas que se usan en este experimento y que hacen parte de la herramienta **VizzAnalyzer** Métricas de **Chidamber & Kemerer**: • Combinación entre Objetos (CBO) • Profundidad del Árbol de la Herencia (DIT) • Falta de Cohesión en los Métodos (LCOM) • Número de Hijos	Se tienen cinco métricas ampliamente utilizadas de cohesión LSCC, TCC, CC, SCOM y LCOM5. Se usa Code-Imp como plataforma de refactorización guiada para métricas basadas en la búsqueda.	• WMC (Weighted Methods per Class) • DIT (Depth of Inheritance Tree) • NOC (Number Of Children) • CBO (Coupling Between Object classes) • RFC (Response For a Class) • LCOM (Lack of Cohesion in Methods) • CYC (CYClomatic complexity)

	Inherited • Number Of Methods Overriden • Number Of Private Attributes • Number Of Private Methods • Number Of Public Attributes • Number Of Public Methods • Response For Class • Total Number Of Children • Weighted Method Count • Weight Of A Class	(NOC) • Respuesta para una Clase (RFC) • Método Ponderado (WMC) usando la Complicidad Ciclomática de McCabe como peso para los métodos; **Li & Henry:** • Acoplamiento de abstracción de datos (DAC) • acoplamiento de paso de mensajes (MPC) • número de métodos locales (NOM) • número de atributos y métodos (NAM / SIZE2) **Bieman & Kang**: • Coherencia de Clase Estrecha (TCC) **Hitz & Montazeri**:		

		• Lociedad de los datos (LD) • Mejora de LCOM (ILCOM) • Y las métricas Líneas de código (LOC) • falta de documentación (LOD) • Longitud de los nombres de clase (LEN).		
Herramienta OpenSource (Sí o No)	Sí	Si, las herramientas que se definen en el experimento son de carácter opensource. Pero no se identifica si las herramientas que se desarrollaron por los autores sean libres, las cuales son **SF Extract** y **Extract SVN**	No se propone herramienta	Sí
Tipo de herramienta	Herramienta de escritorio	El analizador que se usa es de tipo escritorio		Escritorio

Arquitectura usada	N capas	NI		NI
¿Base de datos?	No	Si, para este experimento se tiene una base de datos que guarda los datos exportados por VizzAnalyzer, como las métricas para cada proyecto, los nombres y versión del proyecto, algunos metadatos y estadísticas adicionales		No

4.1.4 Herramientas de cálculo de métricas a partir código fuente

Tabla 7. Compilado de herramientas seleccionadas

	¿Muestra explicación de los resultados?	**Tipo de herramienta**	**Arquitectura usada**	**La herramienta utiliza base de datos (Sí, No, NI)**	**Código fuente disponible**
Gmetrics	No, solo muestra un reporte de resultados.	Escritorio	NI. Actúa como un plugin de complemento	No	Sí
MASU	No	Escritorio	Arquitectura basada en plugins.	No	Sí
RTTOOL	No (Esta herramienta no calcula las métricas)	Escritorio	N capas	No	Sí
Eclipse Metrics Plugin	No	Plug-in de Eclipse	NI	NI	Sí
VizzMaintenance	No (Por lo menos en la interfaz de resultados)	Escritorio	Variación de N capas	No	No

Nota: Las herramientas Gmetrics y Masu no suministran un manual de instalación para el usuario o una guía específica para ello. Por dichas razones, estas herramientas no se pudieron explorar.

Los principales requerimientos que fueron extraídos a partir de la revisión de la documentación y herramientas existentes son:

- Calcular métricas a partir del código fuente.

4.1.5 Casos de uso del mundo real

Los casos de uso del mundo real se identificaron a partir de la revisión de documentación y herramientas que calculan métricas sobre código fuente. Encontrando la operación “Calcular métricas sobre el código fuente” como la más frecuente. Este caso de uso tiene dos extensiones, el caso de uso base toma el comportamiento de una y solo una de las extensiones dependiendo del siguiente condicional: si el Ingeniero de Software cuenta con una herramienta para apoyar el cálculo, ir a “Calcular métricas con ayuda de una herramienta”; si no, ir a “Calcular métricas en forma manual”.

Empleando el mayor nivel de abstracción se construyó el siguiente diagrama que consta de un actor y un caso de uso que tiene dos extensiones. Actor: Ingeniero de Software; casos de uso: Calcular métricas, extensiones: Calcular métricas con ayuda de una herramienta y Calcular métricas en forma manual.

Ingeniero de Software: es el actor encargado de iniciar el proceso de cálculo de métricas. Se encarga de seleccionar el código fuente del proyecto a analizar y define las métricas que serán calculadas. Es quien se encarga de calcular las métricas en forma manual si no cuenta con una herramienta para cálculo de métricas.

Herramienta para cálculo de métricas: se define como cualquier entidad que asiste al Ingeniero de Software en el cálculo de las métricas que este definió con antelación. Se encarga de aplicar, sobre el código fuente de un proyecto, las formulas correspondientes a cada métrica; finalmente muestra los resultados del cálculo. A partir de aquí se abrevia con “Herramienta”.

Entorno de trabajo: es el límite a nivel del mundo real y depende del condicional mencionado en el primer párrafo del ítem 4.1.5. Si el Ingeniero de Software cuenta con una herramienta, el Entorno de trabajo es la Herramienta. Si no, el Entorno de trabajo es el cálculo de las métricas en forma manual.

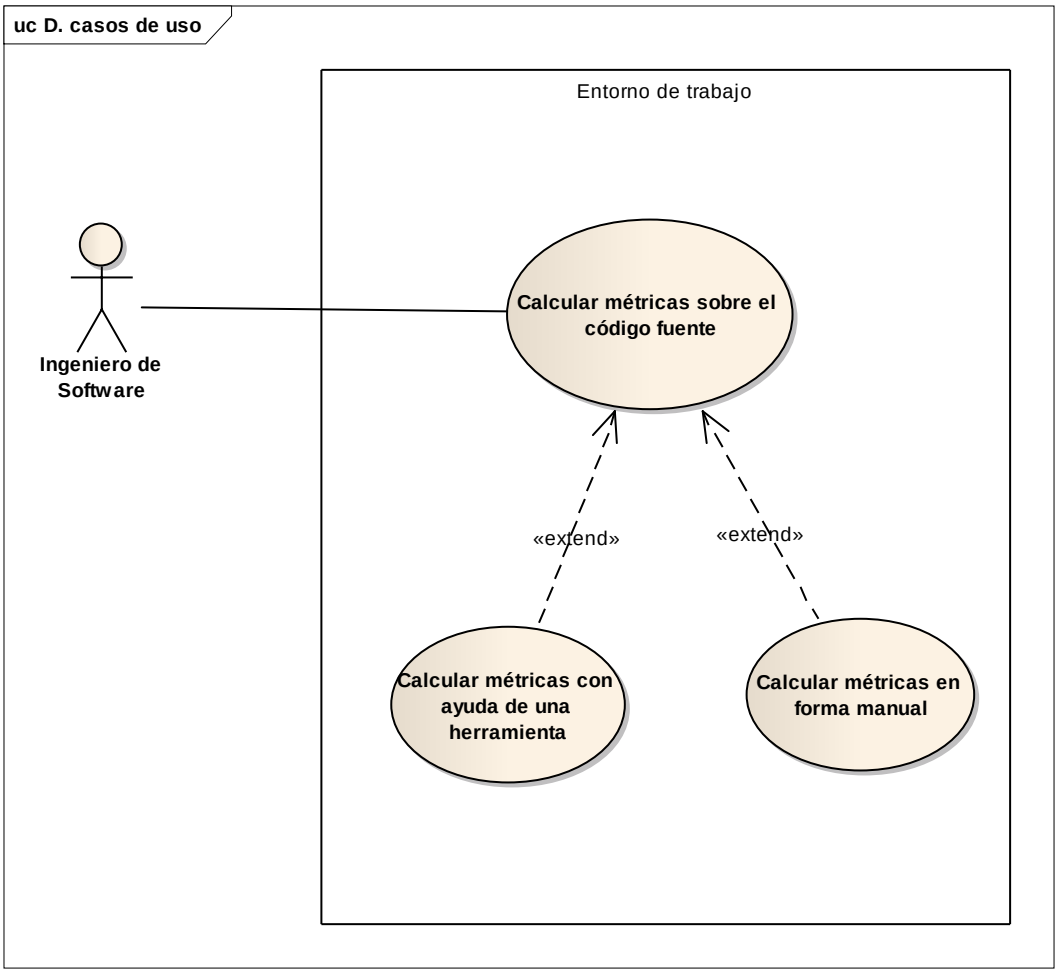


Figura 3. Diagrama de casos de uso del mundo real

4.1.5.1 Descripción de casos de uso del mundo real

CASO DE USO: CALCULAR MÉTRICAS SOBRE EL CÓDIGO FUENTE

Actor principal: Ingeniero de software

Personal involucrado e intereses:

- Arquitecto: está interesado en que la métrica sea acorde a lo proyectado en la arquitectura.
- Gerente de proyecto: está interesado en que el valor de la métrica sea el mejor para garantizar la calidad del producto.

- Programador: está interesado en que el valor de la métrica sea el mejor para garantizar la calidad del producto
- Gestor de calidad: está interesado en conocer el valor de la métrica para emitir un concepto sobre la calidad del producto.

Precondiciones: Disponibilidad del código fuente

Garantías de éxito (Post condiciones): El ingeniero de software realiza el análisis (extrae los parámetros) al código fuente y calcula (aplica las fórmulas de cada métrica) las métricas en forma manual, o apoyado con una herramienta para realizar el análisis al código fuente y calcular las métricas, según sea el caso.

Escenario principal de éxito (Flujo Básico):

1. El ingeniero de software selecciona una carpeta de proyecto.
2. El Ingeniero de software verifica que el código fuente sea el deseado.
3. El ingeniero de software selecciona el código fuente del proyecto seleccionado.
4. ¿El ingeniero de software cuenta con una herramienta para apoyar el análisis sobre el código fuente?
5. Fin.

Extensiones (Flujos alternativos):

2. a El código fuente no es el deseado por el Ingeniero de software

    1. Ir a “Calcular métricas sobre el código fuente”/Flujo Básico/Ítem 1

4. a Sí cuenta con una herramienta:

    1. Ir a “Calcular métricas con ayuda de una herramienta”

4. b No cuenta con una herramienta:

    1. Ir a “Calcular métricas en forma manual”

Requisitos especiales: Ninguno

Lista de tecnología y variaciones de datos: Ninguno

Frecuencia: Cada vez que sea necesario. Si el Ingeniero de Software desea calcular las métricas al mismo proyecto de código fuente o uno distinto, se retorna al flujo básico del presente caso de uso.

Temas abiertos: Ninguno

CASO DE USO: CALCULAR MÉTRICAS CON AYUDA DE UNA HERRAMIENTA

Actor principal: Herramienta para cálculo de métricas

Personal involucrado e intereses:

- Arquitecto: está interesado en que la métrica sea acorde a lo proyectado en la arquitectura.
- Gerente de proyecto: está interesado en que el valor de la métrica sea el mejor para garantizar la calidad del producto.
- Programador: está interesado en que el valor de la métrica sea el mejor para garantizar la calidad del producto
- Gestor de calidad: está interesado en conocer el valor de la métrica para emitir un concepto sobre la calidad del producto.

Precondiciones: Disponibilidad del código fuente deseado.

Garantías de éxito (Post condiciones): La herramienta realiza el análisis al código fuente y calcula las métricas adecuadamente.

Escenario principal de éxito (Flujo Básico):

1. El Ingeniero de Software inicia el análisis en la herramienta.
2. La herramienta extrae, del código fuente, los parámetros que serán utilizados en las fórmulas de cada métrica.
3. La herramienta aplica las fórmulas de las métricas (determinadas por el Ingeniero de Software) sobre el código fuente.
4. La herramienta finaliza el cálculo de las métricas sobre código fuente.
5. La herramienta muestra los resultados de las métricas.
6. Fin.

Extensiones (Flujos alternativos):

2. a La Herramienta no puede interpretar el código fuente (Análisis no exitoso).

1. El ingeniero vuelve a seleccionar el código fuente (Ir a "Calcular métricas sobre el código fuente"/Flujo Básico/Ítem 1).

5. a El ingeniero de software quiere calcular las métricas de un proyecto diferente o el mismo.

1. Ir a "Calcular métricas sobre el código fuente"/Flujo Básico/Ítem 1).

Requisitos especiales: Ninguno

Lista de tecnología y variaciones de datos:

- Computadora Pentium III o superior con mínimo 256 MB de RAM y 250 MB mínimo de espacio en disco duro. Además, con una versión de Windows XP o superior.
- Máquina donde se hospeda la Herramienta.

Frecuencia: Continuo

Temas abiertos: Ninguno

CASO DE USO: CALCULAR MÉTRICAS EN FORMA MANUAL

Actor principal: Ingeniero de Software

Personal involucrado e intereses:

- Arquitecto: está interesado en que la métrica sea acorde a lo proyectado en la arquitectura.
- Gerente de proyecto: está interesado en que el valor de la métrica sea el mejor para garantizar la calidad del producto.
- Programador: está interesado en que el valor de la métrica sea el mejor para garantizar la calidad del producto
- Gestor de calidad: está interesado en conocer el valor de la métrica para emitir un concepto sobre la calidad del producto.

Precondiciones: Disponibilidad del código fuente deseado.

Garantías de éxito (Post condiciones): El Ingeniero de software realiza el análisis al código fuente y calcula las métricas adecuadamente.

Escenario principal de éxito (Flujo Básico):

1. El ingeniero de software determina las métricas que serán calculadas: número de líneas de código, número de clases, número de métodos por clase, grado de acoplamiento y falta de cohesión en una clase.
2. El ingeniero de software extrae, del código fuente, los parámetros necesarios para cada fórmula correspondiente a una determinada métrica.
3. El Ingeniero de software reemplaza los parámetros en las fórmulas de cada métrica
4. El Ingeniero de software calcula las métricas.
5. El Ingeniero de software termina de calcular las métricas del código fuente.
6. El Ingeniero de software documenta los resultados de las métricas.
7. Fin.

Extensiones (Flujos alternativos):

2. a El Ingeniero de software no puede interpretar el código fuente.

    1. El ingeniero vuelve a seleccionar el código fuente (Ir a “Calcular métricas sobre el código fuente”/Flujo Básico/Ítem 1).

6. a El ingeniero de software quiere calcular las métricas de un proyecto diferente o el mismo.

    1. Ir a “Calcular métricas sobre el código fuente”/Flujo Básico/Ítem 1).

Requisitos especiales: Ninguno

Lista de tecnología y variaciones de datos: Ninguno

Temas abiertos: Ninguno

4.1.6 Modelo de Dominio

Empleando el mayor nivel de abstracción, se identificaron las clases conceptuales trascendentales del problema y se representaron de dos maneras: Modelo de dominio con apoyo de Herramienta y Modelo de dominio sin apoyo de Herramienta.

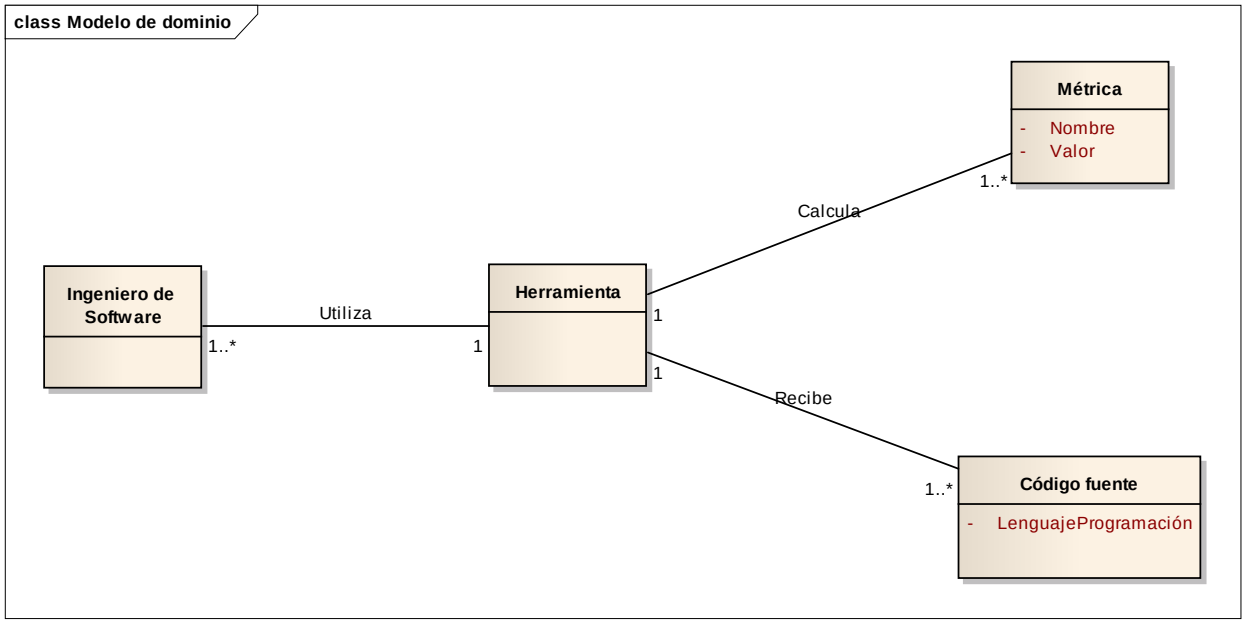


Figura 4. Modelo de dominio con apoyo de herramienta

Teniendo en cuenta que la lectura de un modelo de dominio es en sentido izquierda-derecha, abajo-arriba; se puede observar que existe un (o varios) Ingeniero de Software que utiliza una (1) Herramienta, éste recibe uno (1) o muchos Código fuente y a partir de esto, la Herramienta calcula una (1) o muchas métricas**.**

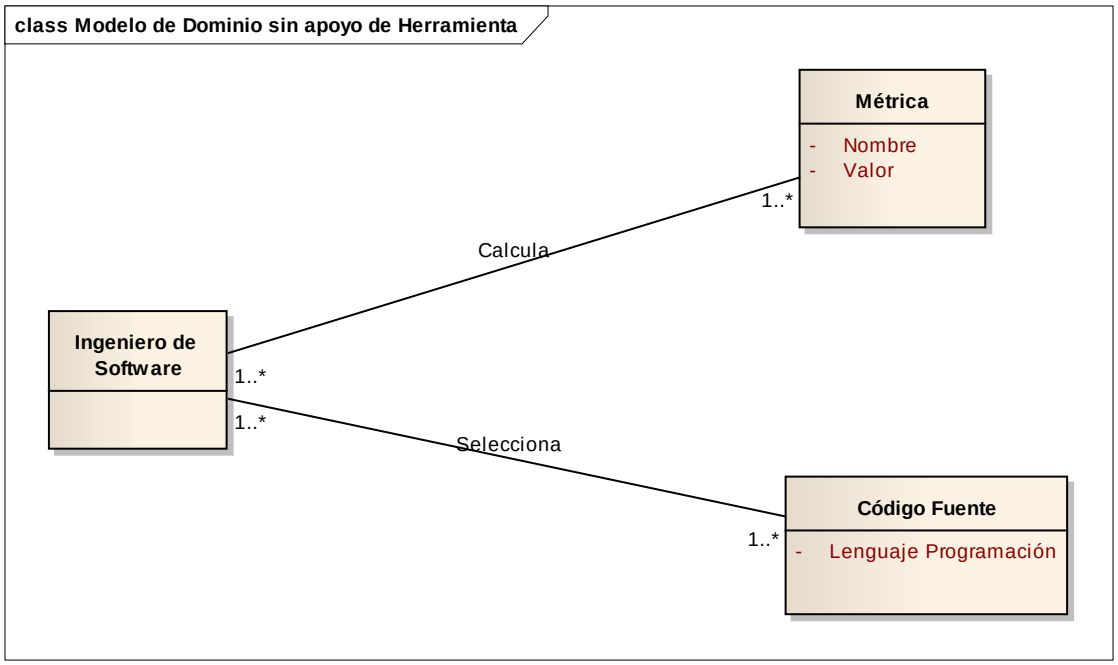


Figura 5. Modelo de dominio sin apoyo de Herramienta

Teniendo en cuenta que la lectura de un modelo de dominio es en sentido izquierda-derecha, abajo-arriba; se puede observar que existe uno (o varios) Ingeniero de Software que selecciona uno (1) o muchos Código fuente y a partir de esto calcula 1 o muchas métricas**.**

4.1.7 Procesos de negocio

En la figura. 6 se observa el diagrama de actividades. Este se elaboró a partir de los casos de uso del mundo real. En esta ocasión cada caso de uso NO equivale a una actividad. Cuente o no con una herramienta, el Ingeniero de software es quien inicia el proceso de negocio con la actividad "Seleccionar código fuente", seguidamente se encuentra el condicional "¿Es el código fuente deseado?, si la respuesta es negativa el Ing. De Software vuelve a seleccionar el código fuente, de lo contrario se continúa con el flujo básico, el cual se determina, a partir de aquí, de acuerdo al condicional "¿El ingeniero de software cuenta con una herramienta para apoyar el análisis sobre el código fuente?", de acuerdo a la respuesta, el flujo se realiza o en el Ingeniero de software o en la herramienta. Se puede evidenciar la coherencia con el modelo de dominio y casos de uso.

Existen dos actores: **Ingeniero de Software** y **Herramienta**. El Ingeniero inicia la actividad, una vez hecho esto procede a seleccionar el código fuente, si el código fuente es el deseado, se

sigue con el flujo básico. Si ocurre lo contrario, el ingeniero vuelve a seleccionar el código fuente hasta que encuentre el deseado. Si el ingeniero no cuenta con una herramienta para apoyar el análisis, el flujo se desarrolla en la calle del Ingeniero de Software, si el ingeniero cuenta con una, el flujo continuará pero en este caso en la calle de la Herramienta. Si ocurre algo al finalizar el cálculo (cálculo incluye análisis) se procede a seleccionar otro o el mismo código fuente. Si no es así, entonces se muestran los resultados del cálculo. La actividad termina si el ingeniero no quiere repetir el proceso con otro código fuente. Para ver el diagrama de actividades con mayor detalle se puede hacer clic en el siguiente vínculo o acceder a la ruta \Componente Software para el Cálculo de Métricas a Partir de Código Fuente\4. Anexos\Diagrama de actividades:

ANEXO A. Diagrama de Actividades

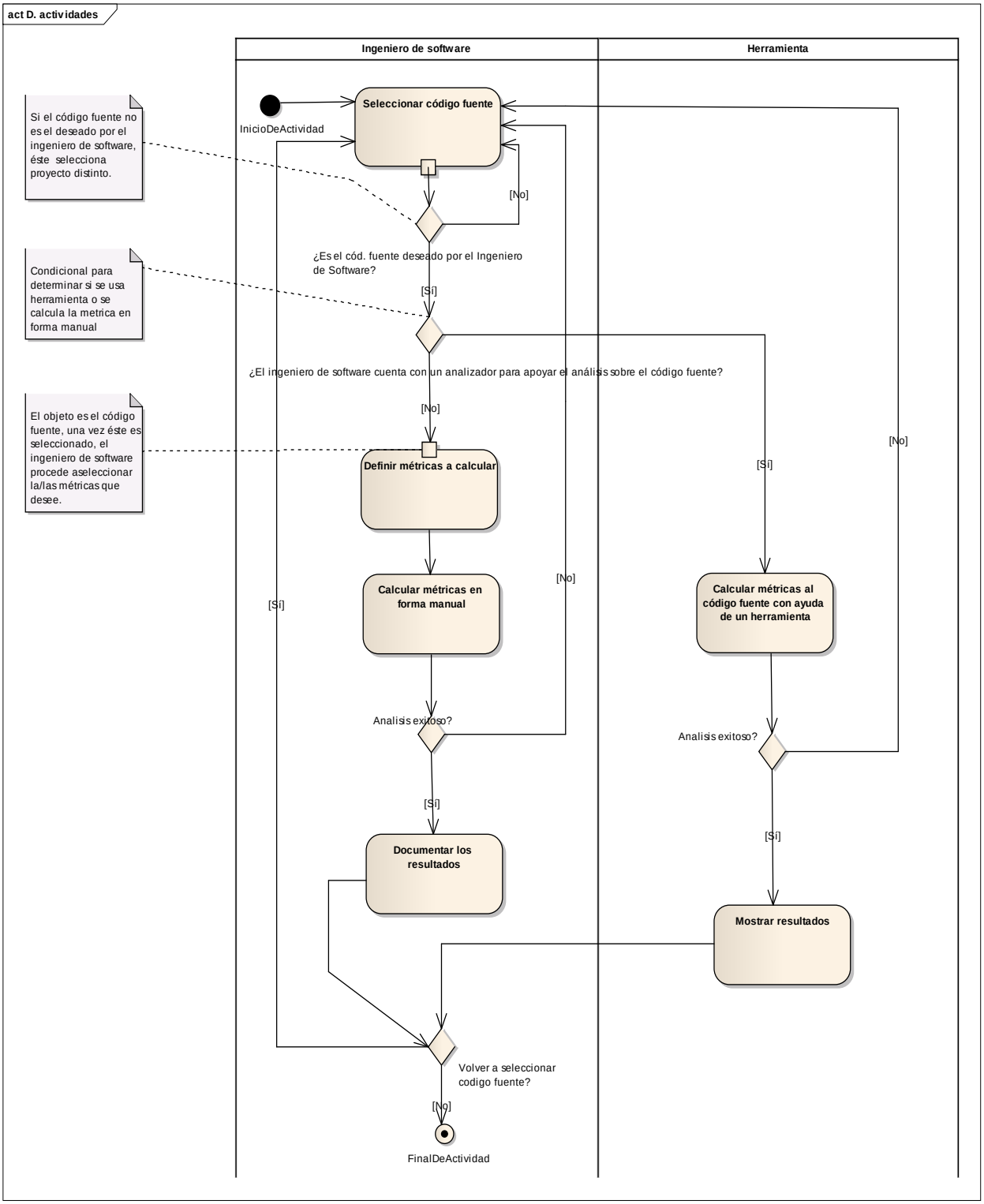


Figura 6. Diagrama de actividades

4.2 Requisitos

En esta sección se divide en Identificación de requisitos, luego se continúa con el documento SRS que se encuentra como un archivo anexo y por último se observa el diagrama de casos de uso a nivel de requisito.

4.2.1 Identificación de requisitos

Para identificación de los requisitos se utilizó la técnica para la revisión de la bibliografía disponible, aplicando la propuesta de Bárbara Kitchenham.

Se recopiló información relacionada con el objeto de estudio, a partir de fuentes secundarias como textos físicos, artículos, documentos publicados en la web (tesis de grado y postgrado), resultados de estudios anteriores publicados, entre otras revisiones bibliográficas. Luego se sistematizó y se realizó una reducción de la información obtenida. A este proceso se le conoce en ingeniería de software como Revisión Sistemática de la Literatura (Kitchenham, y otros, 2008).

A continuación se muestran los requisitos resultantes de la identificación de los requerimientos realizada en la revisión de la literatura.

Requisito funcional: Calcular métricas del código fuente JAVA

En primera instancia, el ingeniero de software selecciona el código fuente (formato JAVA, nombre y ruta correcta).

El código fuente JAVA comienza a ser analizado bajo las métricas establecidas. Si se producen problemas al calcular las métricas, el sistema lo notifica y permite volver a seleccionar el proyecto.

Al finalizar el análisis sobre el código fuente se muestran los resultados. Además, se muestran las fórmulas usadas para calcular las métricas y también los valores de las variables que componen a cada una de las fórmulas.

4.2.2 SRS (Software Requirements Specifications)

El archivo de especificación de requerimientos de software se encuentra en la carpeta Anexos. Para visualizar el SRS se puede hacer clic en el siguiente vínculo o acceder a la ruta \Componente Software para el Cálculo de Métricas a Partir de Código Fuente\4. Anexos\ SRS - Componente Software para el Cálculo de Métricas a partir de Código Fuente:

ANEXO B. SRS - Componente Software para el Cálculo de Métricas a partir de código Fuente.

4.2.3 Diagrama de casos de uso a nivel de requisito (Perspectiva de software y no de mundo real)

El único requisito base identificado es Calcular métricas del código fuente JAVA. Este requisito es mostrado únicamente con el actor Ingeniero de Software como actor principal, pues es el actor directamente implicado. El sistema es la Herramienta, esta apoya al Ingeniero de Software en el cálculo de las métricas sobre código fuente escrito en Lenguaje de programación JAVA.

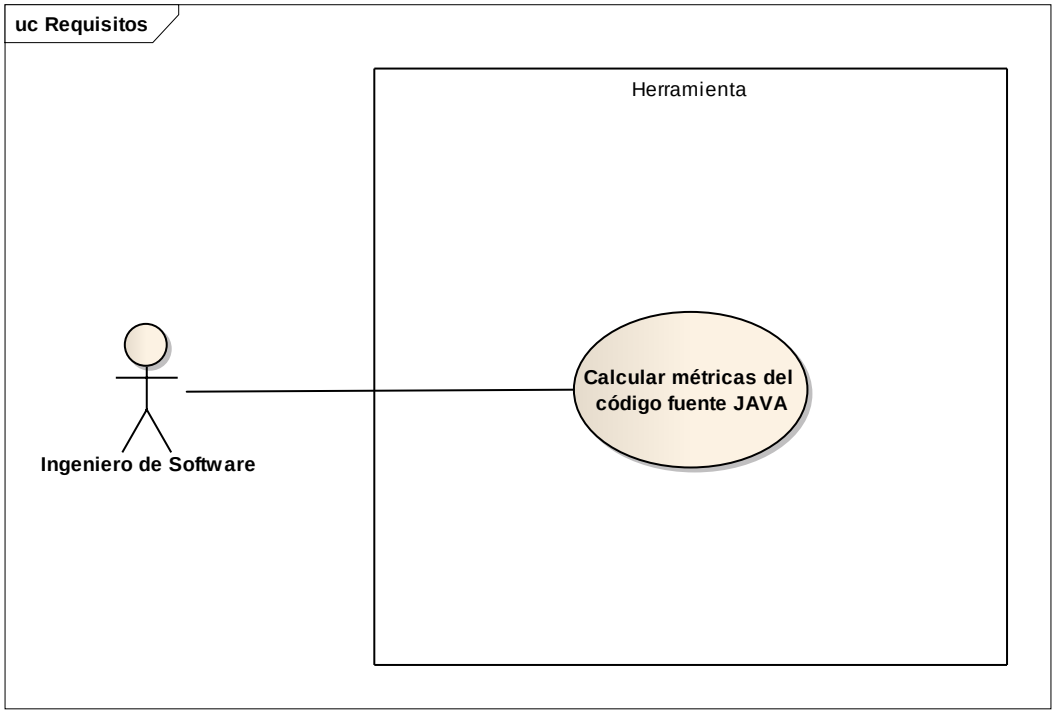


Figura 7. Diagrama de casos de uso del requisito principal

4.3 Modelo de Diseño

Se hace apertura de esta sección describiendo las razones que llevaron a la implementación del patrón arquitectónico de Programación por Capas en la herramienta desarrollada en esta investigación. Seguidamente se describen las distintas vistas correspondientes a nivel de diseño, como son: Vista de escenarios, compuesta por diagrama de caso de uso y descripción de casos de uso; Vista lógica, compuesta por diagrama de componentes y diagrama de clases; y Vista de procesos, compuesta por los diagramas de secuencia.

4.3.1 Definición de arquitectura

Para definir la arquitectura se tuvo en cuenta que la herramienta sería implementada como un componente. Un componente necesitar ser invocado por una capa superior o inferior en un determinado sistema, es decir interactúa a nivel de capas, de ahí se infiere que un componente juega el papel de capa o nivel en otro sistema, o bien puede ser parte de una capa del sistema donde será integrado. El ejemplo más sencillo es invocar las funciones del componente desde una Interfaz de Usuario (capa superior). Por las razones anteriores, la arquitectura que se definió para esta herramienta fue Programación por Capas. El tipo de arquitectura implementada se explica con más detalle en la sección 4.3.4.1 Diagrama de componentes.

4.3.2 Vista de Escenarios

Este ítem está compuesto en primer lugar por el diagrama general de casos de uso y a seguir, se encuentra la descripción general de casos de uso.

4.3.2.1 Diagrama de casos de uso

En esta sección se muestra como el Ingeniero de Software(Actor principal) se apoya en la Herramienta(sistema) para llevar a cabo el cálculo de métricas del código fuente JAVA (Caso de uso).

Diagrama general de casos de uso

En la figura 8 se observa que el Ingeniero de Software es quién inicia la operación de Calcular las métricas del Código fuente JAVA, esta acción es apoyada por la herramienta (sistema). La descripción del caso de uso ‘Calcular métricas del código fuente JAVA’ se encuentra en la sección 4.3.2.2

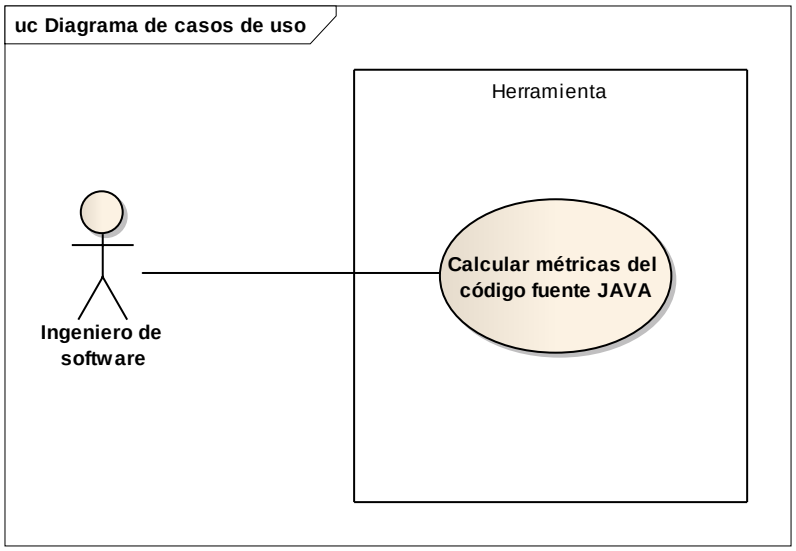


Figura 8. Diagrama general de casos de uso

4.3.2.2 Descripción general de casos de uso

CASO DE USO: CALCULAR MÉTRICAS DEL CÓDIGO FUENTE

Actor principal: Ingeniero de software

Personal involucrado e intereses:

- Arquitecto: está interesado en que la métrica sea acorde a lo proyectado en la arquitectura.
- Gerente de proyecto: está interesado en que el valor de la métrica sea el mejor para garantizar la calidad del producto.
- Programador: está interesado en que el valor de la métrica sea el mejor para garantizar la calidad del producto
- Gestor de calidad: está interesado en conocer el valor de la métrica para emitir un concepto sobre la calidad del producto.

Precondiciones: Disponibilidad del código fuente deseado.

Garantías de éxito (Post condiciones): El Ingeniero de Software realiza el análisis al código fuente y calcula las métricas del código fuente adecuadamente.

Escenario principal de éxito (Flujo Básico):

1. El ingeniero de software presiona el botón “Seleccionar proyecto” de la interfaz gráfica de usuario.
2. El sistema despliega un cuadro de diálogo que muestra los directorios locales de la computadora.
3. El ingeniero de software selecciona un directorio que contiene la carpeta de un proyecto y presiona el botón “Abrir”.
4. En la pestaña “detalles de carga” de la interfaz gráfica, el sistema muestra las rutas de los archivos .java contenidos en el proyecto seleccionado.
5. El ingeniero de software inicia, por medio del botón “Calcular métricas”, la acción de calcular las métricas de los archivos mostrados.
6. El sistema muestra un cuadro de dialogo con el interrogante: “¿Seguro que son los archivos que desea analizar?”
7. El ingeniero de software presiona “Sí”.
8. El sistema invoca el método analizar de la clase AnalizadorClase
9. El sistema calcula las métricas sobre el código fuente de los archivos JAVA, aplicando las fórmulas de cada métrica.
10. El sistema finaliza el análisis y la interfaz gráfica muestra el mensaje: “El análisis ha finalizado”.
11. En la interfaz gráfica se muestran los resultados de las métricas en el Panel de Cálculo
12. Fin.

Extensiones (Flujos alternativos):

1. a Se genera un error y no se puede desplegar la ventana con los directorios locales

1. Aparece un cuadro de dialogo con el mensaje “"Ha ocurrido un error en el buscador de archivos!".

2. Ir a “Calcular métricas del código fuente”/Flujo Básico/ítem 1.

2. a El Ingeniero de Software presiona “Cancelar”.

1. Aparece un cuadro de dialogo con el mensaje “Se ha cancelado la búsqueda de archivos”

2. Ir a “Calcular métricas del código fuente”/Flujo Básico/ítem 1.

3. a. La carpeta seleccionada no contiene archivos .java.

1. Se muestra el mensaje “No se encontraron archivos de código fuente Java”

2. Ir a “Calcular métricas del código fuente”/Flujo Básico/ítem 1.

6.a El Ingeniero de Software presiona “No”.

1. Ir a “Calcular Métricas del Código Fuente/Flujo Básico/Ítem 2

6.b Ocurre un error cuando se intenta verificar si existe por lo menos una clase en cada archivo JAVA

1. El sistema muestra el mensaje "Error al iniciar verificar el archivo, en el método 'existeClaseEnArchivo'!”

2. Ir a “Calcular métricas del código fuente”/Flujo Básico/ítem 3

6. c Ocurre un error cuando se intenten identificar los nombres de las clases de los archivos JAVA

1. El sistema muestra el mensaje: “Ups! Algo pasó”. Por esto, Puede que luego se muestren los resultados incompletos.

2. Ir a “Calcular métricas del código fuente”/Flujo Básico/ítem 2

11. a El Ingeniero de Software quiere calcular las métricas, con ayuda de la herramienta para cálculo de métricas, de un proyecto distinto o el mismo.

1. Ir a “Calcular métricas del código fuente”/Flujo Básico/ítem 1

Requisitos especiales: Ninguno

Lista de tecnología y variaciones de datos:

- Computadora Pentium III o superior con mínimo 256 MB de RAM y 250 MB mínimo de espacio en disco duro. Además, con una versión de Windows XP o superior.
- Máquina virtual de JAVA.

Frecuencia: cuando se presente la necesidad.

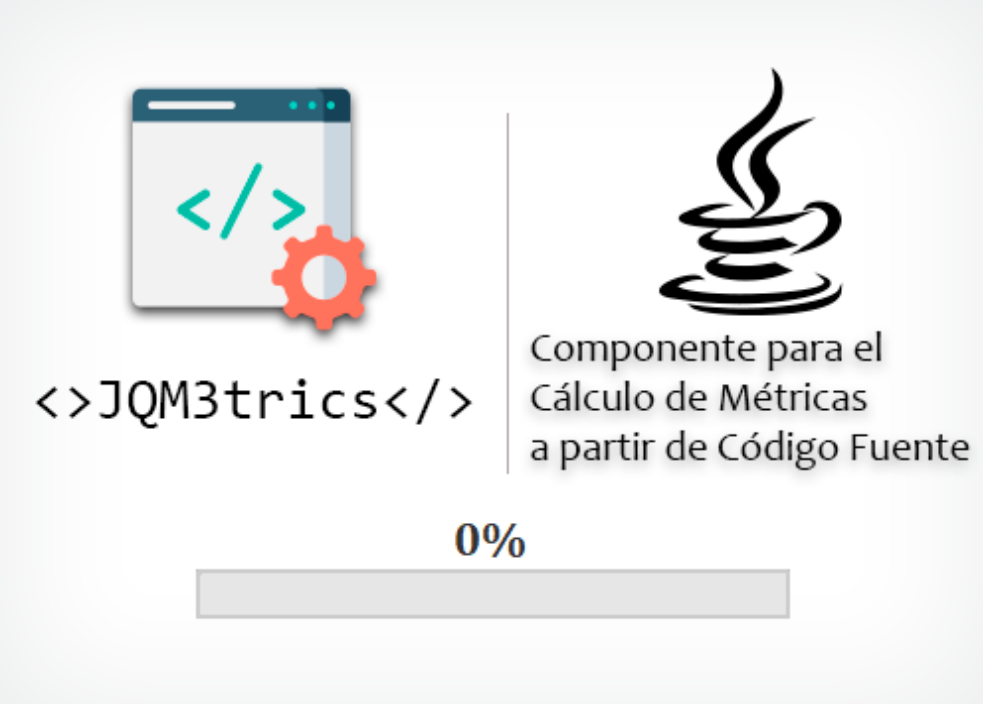
Temas abiertos: Lenguaje de programación del proyecto a analizar.

4.3.3 Interfaz de Usuario (GUI)

A continuación se explican los elementos encontrados en la interfaz de usuario y seguidamente se describen los principios de diseño de IU aplicados.

4.3.3.1 Elementos de la Interfaz de Usuario

Antes de continuar es importante aclarar que una vez se lanza (ejecuta) JQM3trics, se muestra una pantalla de inicio o Splash Screen que se puede observar en la siguiente imagen:



Se construyó una Interfaz de Usuario (IU) para invocar al componente de Cálculo de métricas. La IU está compuesta por cuatro áreas (enumeradas en la figura 9).

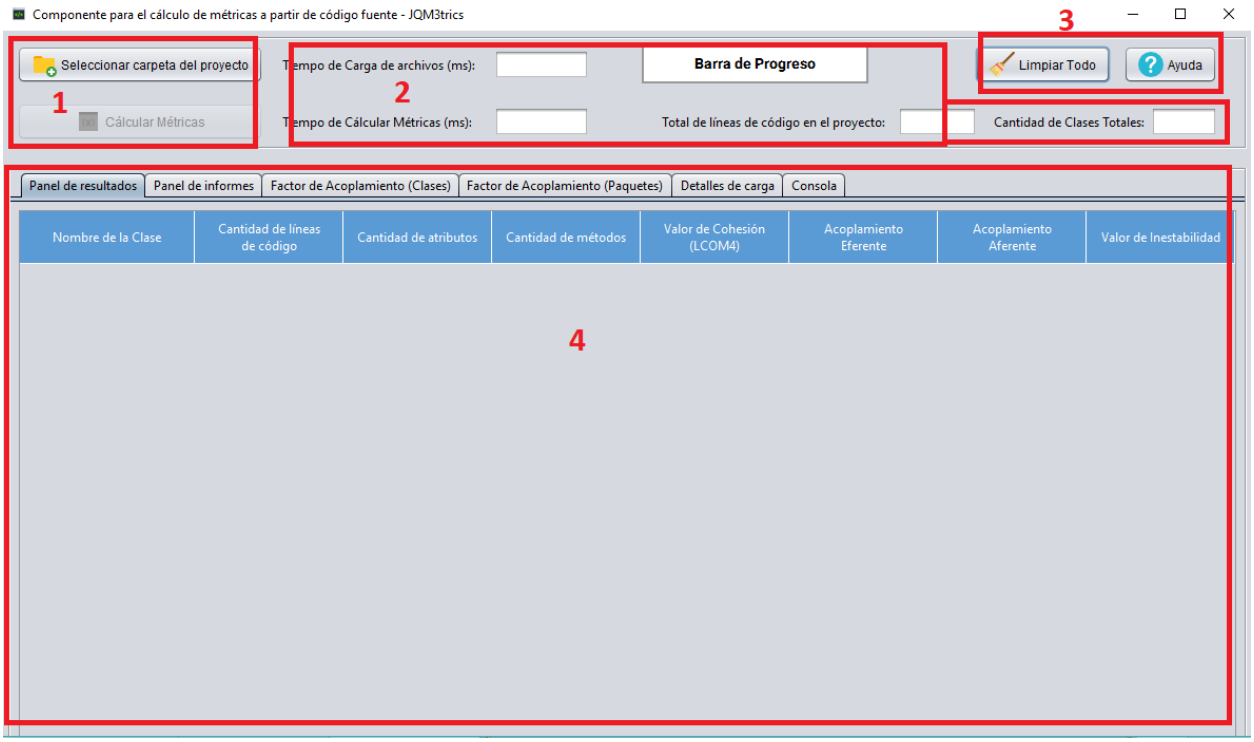


Figura 9. Áreas de IU enumeradas

Las áreas se han nombrado de la siguiente forma:

Área de control (número 1): está conformada por dos botones que manejan el flujo principal del sistema, los botones son “Seleccionar carpeta del proyecto” y “Calcular Métricas”. También es llamada área de control puesto que aquí se encuentra el botón que maneja la funcionalidad principal (requisito principal) del sistema, que es Calcular Métricas.

Área de datos generales (número 2): muestra el Tiempo, en milisegundos, que le toma al sistema cargar los archivos Java de un proyecto de código fuente, el Tiempo (milisegundos) que tarda el sistema en Calcular las Métricas a un proyecto de código fuente y una barra de progreso que mantiene al usuario informado mientras se lleva a cabo un proceso. Además muestra la cantidad de líneas de código del proyecto y la cantidad total de clases del mismo.

Área complementaria (número 3): está compuesta por dos botones que son “Ayuda” y “Limpiar Todo”. El botón “Ayuda” despliega un popup sencillo (ver Figura 10) que cuenta con un Combo Box cuyas opciones permiten consultar cómo usar la aplicación y consultar la explicación de cómo el sistema realiza el cálculo de cada una de las métricas implementadas.

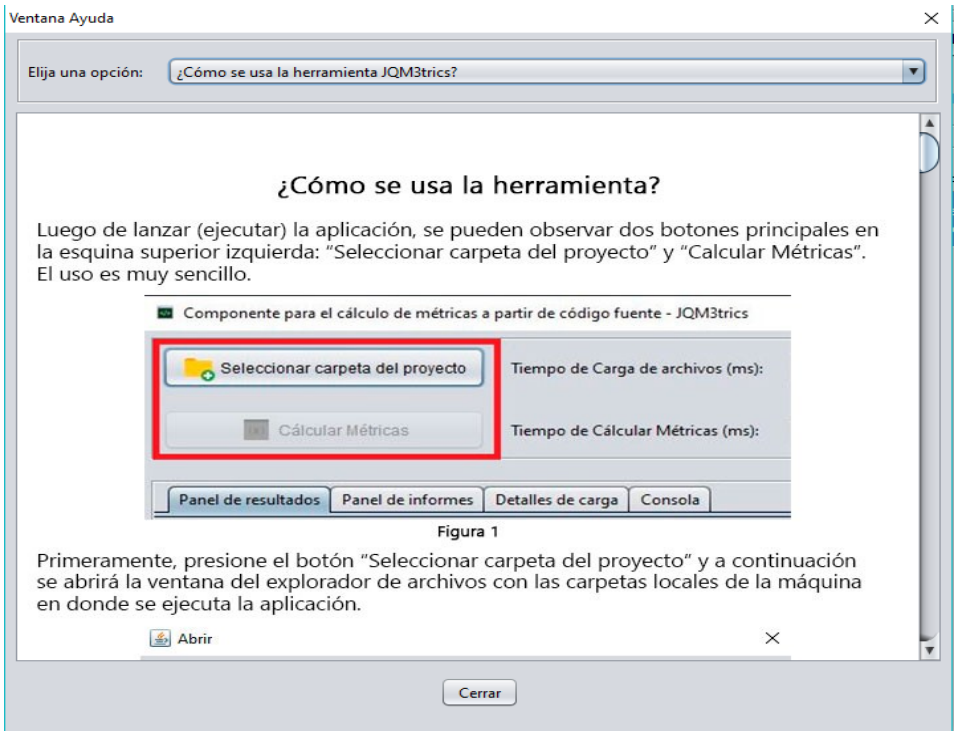
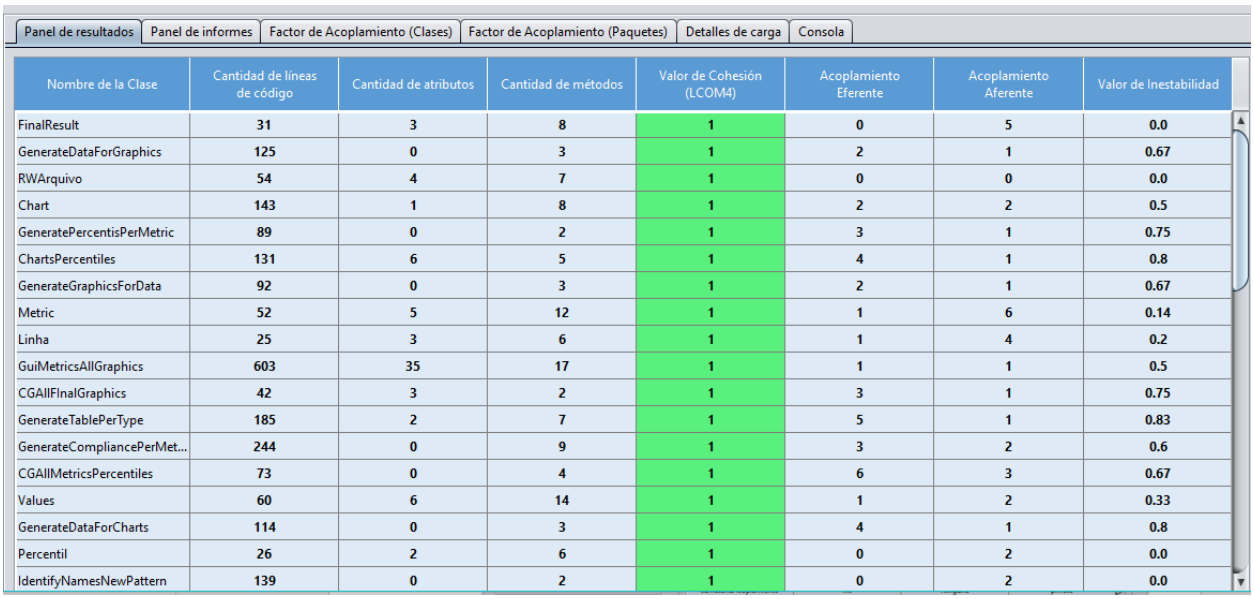


Figura 10. Ventana Ayuda

Área de trabajo (número 4): es la más importante después del área de control ya que permite al usuario visualizar los resultados del cálculo de métricas. El área de trabajo se divide en seis pestañas que son "Panel de resultados", "Panel de informes", "Factor de acoplamiento (clases)", "Factor de acoplamiento (paquetes)", "Detalles de carga" y "Consola".

- **Panel de resultados**. Esta pestaña muestra los resultados del cálculo de métricas. Específicamente son mostrados los valores de las métricas para cada archivo Java (ver Figura 11), las métricas mostradas en este panel son: Cantidad de líneas de código, Cantidad de atributos, Cantidad de métodos, Valor de Cohesión (LCOM4), Acoplamiento Eferente, Acoplamiento Aferente y Valor de Inestabilidad.



Nombre de la Clase	Cantidad de líneas de código	Cantidad de atributos	Cantidad de métodos	Valor de Cohesión (LCOM4)	Acoplamiento Eferente	Acoplamiento Aferente	Valor de Inestabilidad
FinalResult	31	3	8	1	0	5	0.0
GenerateDataForGraphics	125	0	3	1	2	1	0.67
RWArquivo	54	4	7	1	0	0	0.0
Chart	143	1	8	1	2	2	0.5
GeneratePercentisPerMetric	89	0	2	1	3	1	0.75
ChartsPercentiles	131	6	5	1	4	1	0.8
GenerateGraphicsForData	92	0	3	1	2	1	0.67
Metric	52	5	12	1	1	6	0.14
Linha	25	3	6	1	1	4	0.2
GuiMetricsAllGraphics	603	35	17	1	1	1	0.5
CGAllFInalGraphics	42	3	2	1	3	1	0.75
GenerateTablePerType	185	2	7	1	5	1	0.83
GenerateCompliancePerMet...	244	0	9	1	3	2	0.6
CGAllMetricsPercentiles	73	0	4	1	6	3	0.67
Values	60	6	14	1	1	2	0.33
GenerateDataForCharts	114	0	3	1	4	1	0.8
Percentil	26	2	6	1	0	2	0.0
IdentifyNamesNewPattern	139	0	2	1	0	2	0.0

Figura 11. Panel de Resultados

- **Panel de informes**. Muestra la interpretación de los resultados obtenidos en la pestaña Panel de resultados, concretamente se muestra la lista de atributos y métodos por clase seleccionada (ver Figura 12), se pueden apreciar los datos acerca de las líneas de código, líneas de comentario y líneas vacías (ver Figura 13); también cuenta con dos pestañas correspondientes a las métricas de Acoplamiento a nivel de clases (ver Figura 14) y Falta de cohesión (ver Figura 15). El usuario tiene la opción de escoger la clase de la cual desee visualizar información por medio del Combo Box “Seleccionar Clase” (ver Figura 12).

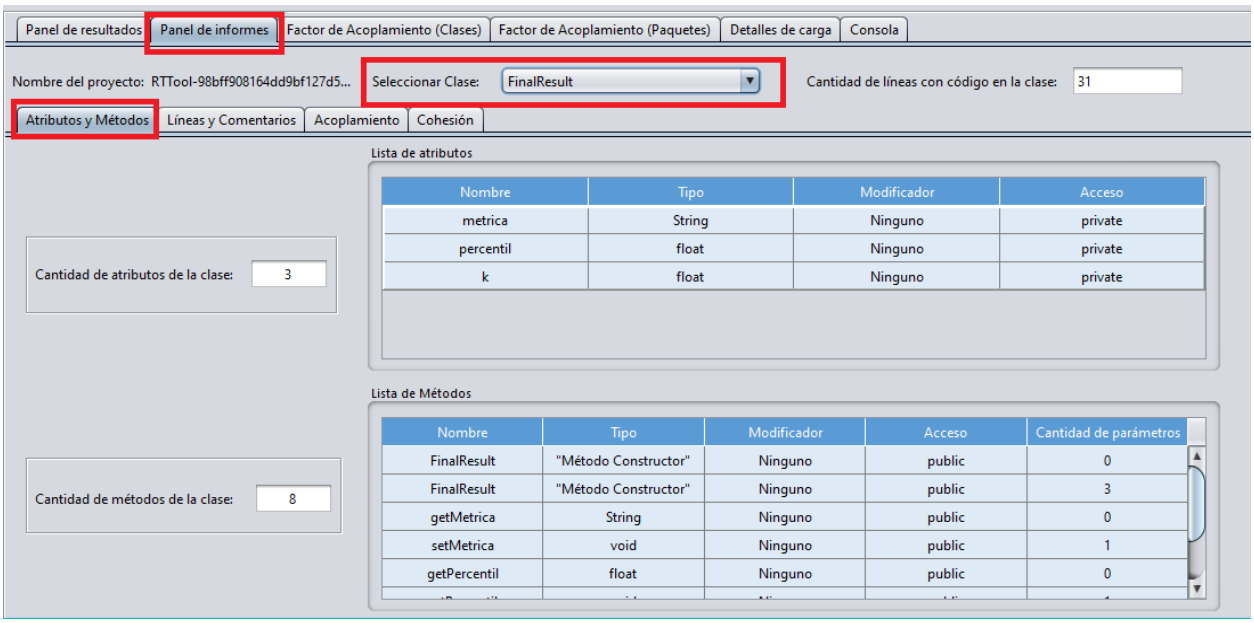


Figura 12. Panel de Informes - Atributos y Métodos

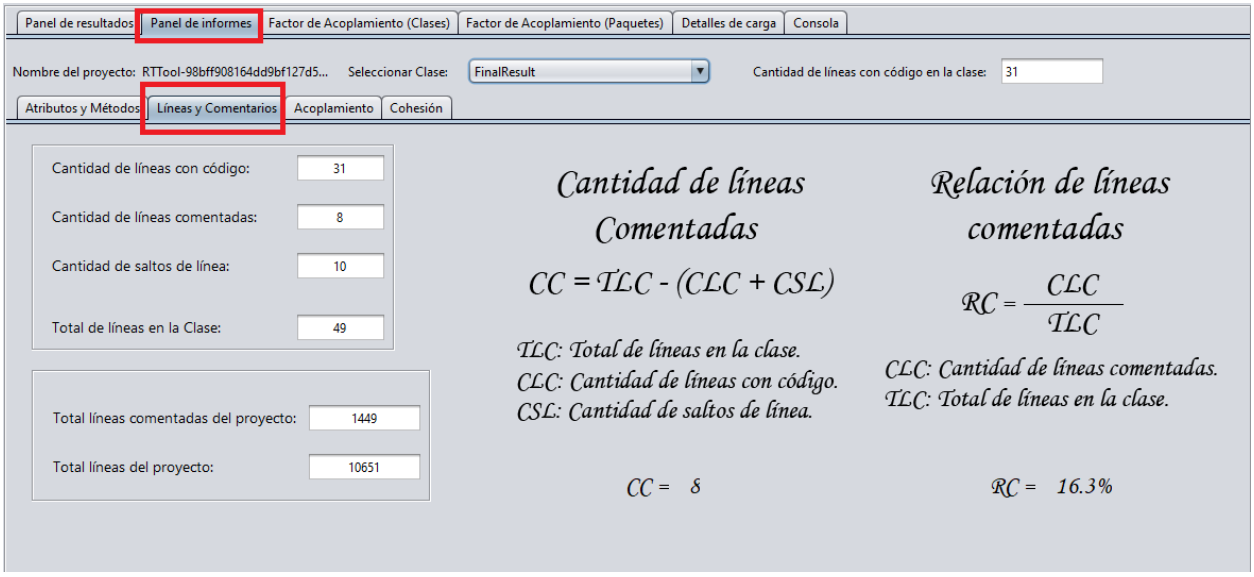


Figura 13. Panel de informes – Líneas y comentarios

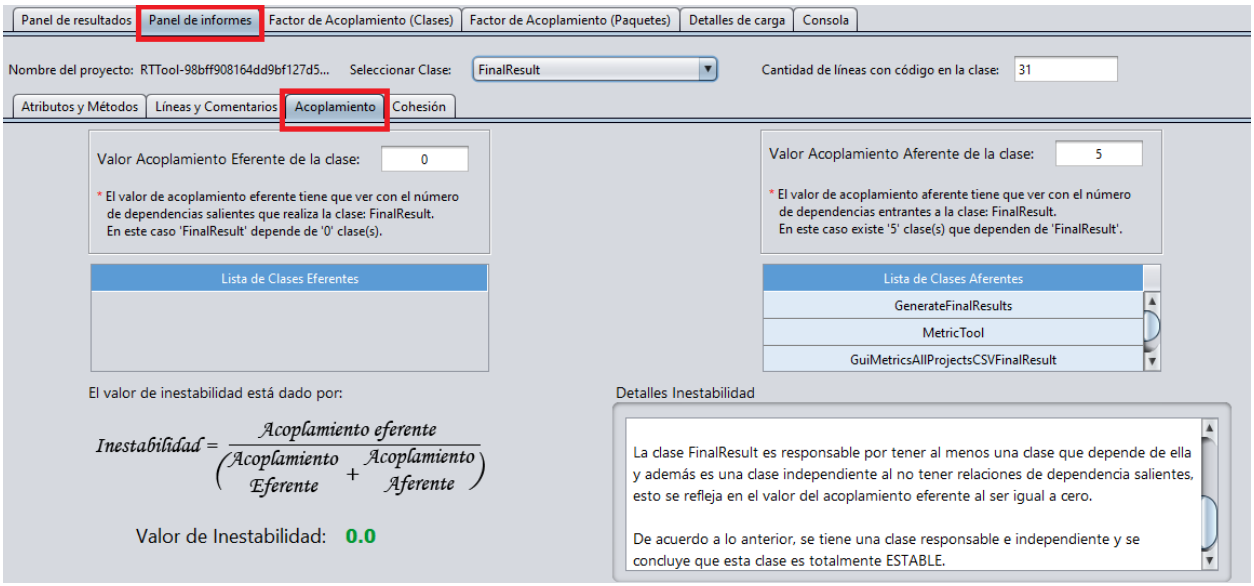


Figura 14. Panel de Informe – Acoplamiento a nivel de clases

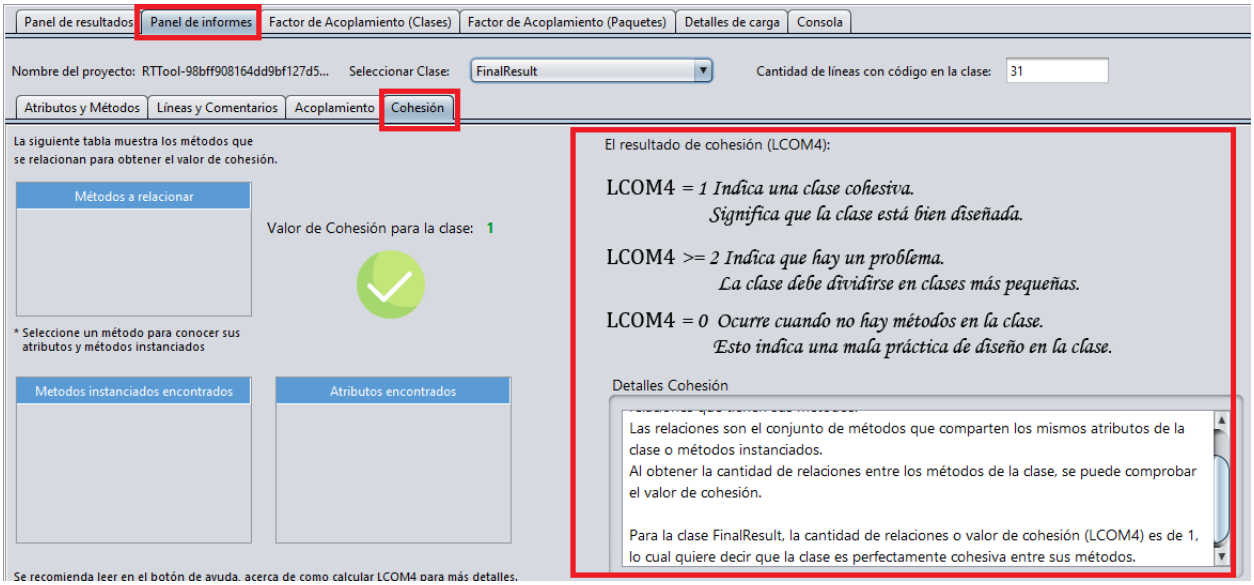


Figura 15. Panel de Informe - Cohesión

- **Detalles de Carga**. Muestra la ruta local de cada uno de los archivos Java de un determinado proyecto de código fuente luego de presionar “Seleccionar carpeta del proyecto” (ver Figura 16). Además, después de calcular las métricas de cada uno de los archivos, en esta pestaña se muestra el número total de archivos .java y el número total de líneas del proyecto (ver Figura 17).

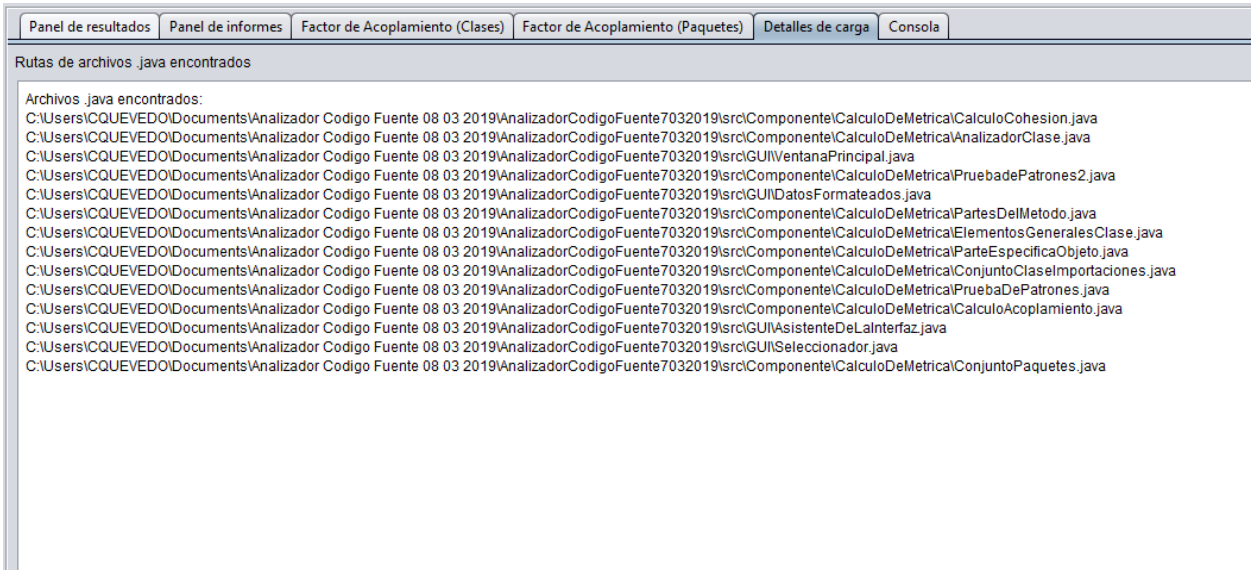


Figura 16. Detalles de carga antes de calcular las métricas

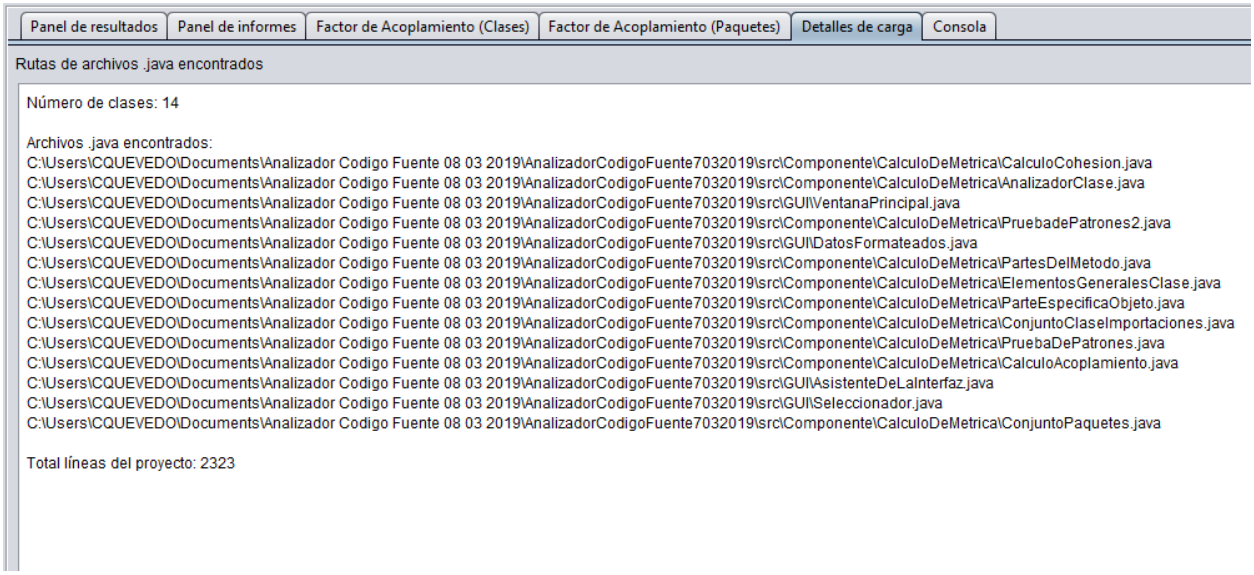


Figura 17. Detalles de Carga luego de calcular las métricas

- **Consola**. Puesto que la aplicación está dirigida a Ingenieros de Software o personas con conocimientos en el campo de cálculo de métricas, la interfaz cuenta con una consola donde se observan los atributos y métodos de cada una de las clases (ver Figura 18); también muestra la cantidad de paquetes del proyecto analizado, las clases contenidas en

cada paquete y los nombres de dichos paquetes y clases (ver Figura 19); finalmente muestra de quién depende una clase (Relaciones Eferentes) y quién depende de dicha clase (Relaciones Aferentes). (ver Figura 20).

Específicamente en esta pestaña se detalla el acceso de los atributos, si tiene modificador y el tipo de valor del atributo; de igual manera se muestra el Acceso, Modificador, Parámetros recibidos y Tipo de valor de retorno de cada uno de los métodos (Por ejemplo: Entero, String, etc.)

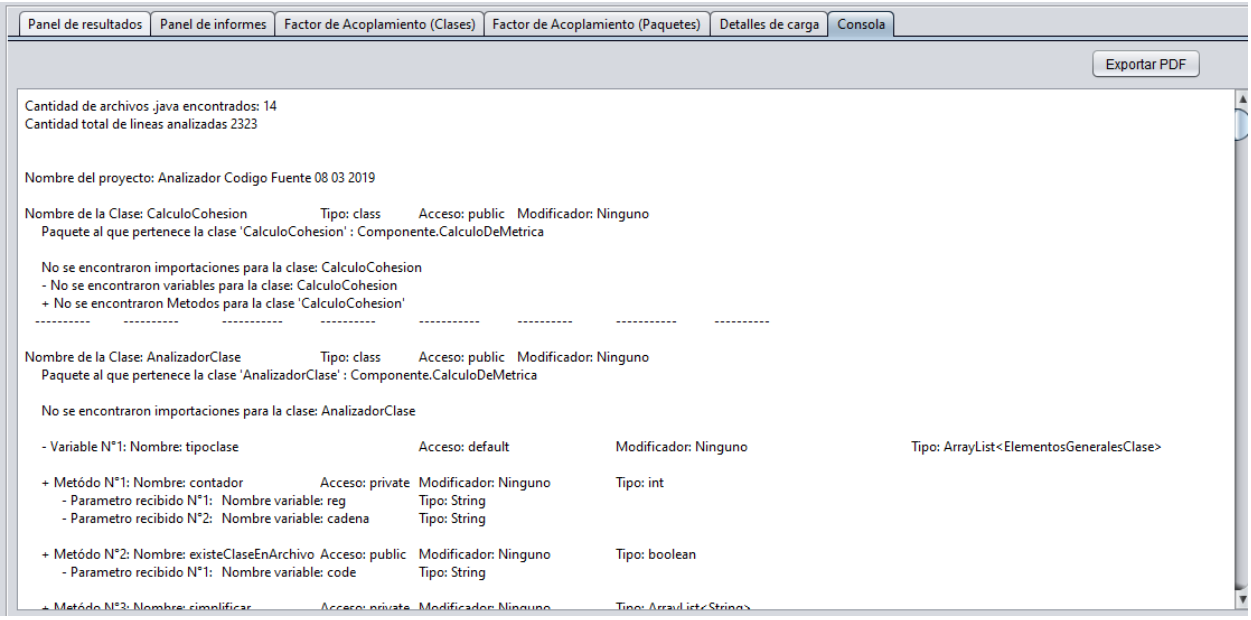


Figura 18. Consola, atributos y métodos de cada clase

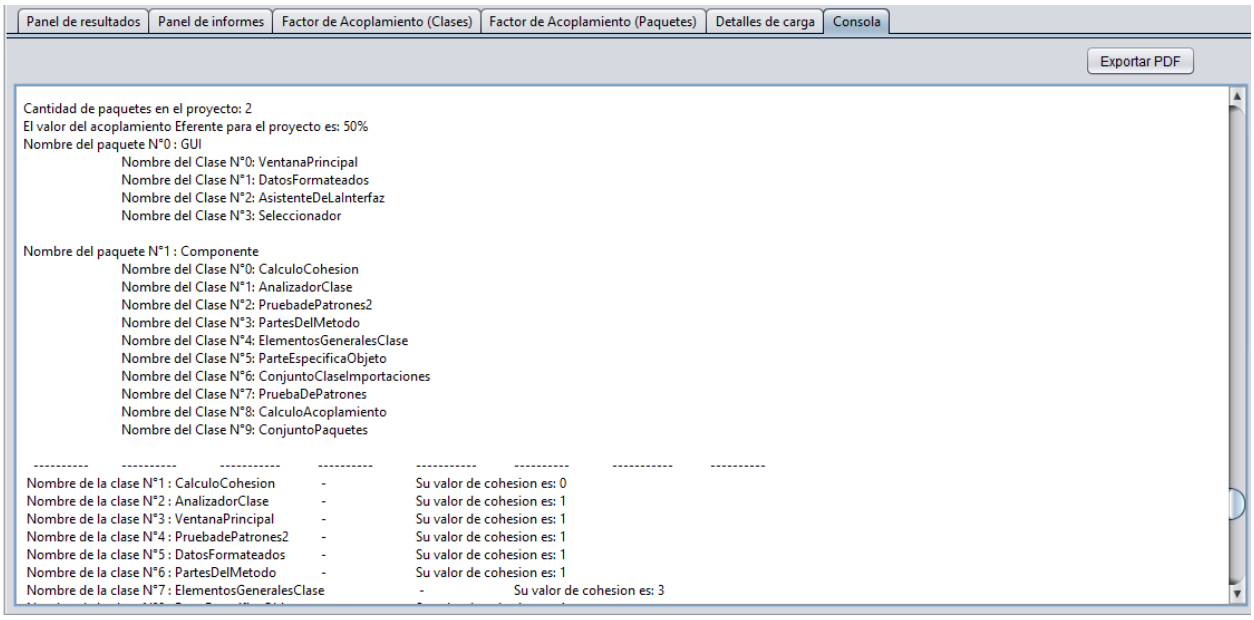


Figura 19. Paquetes y clases contenidas en cada paquete

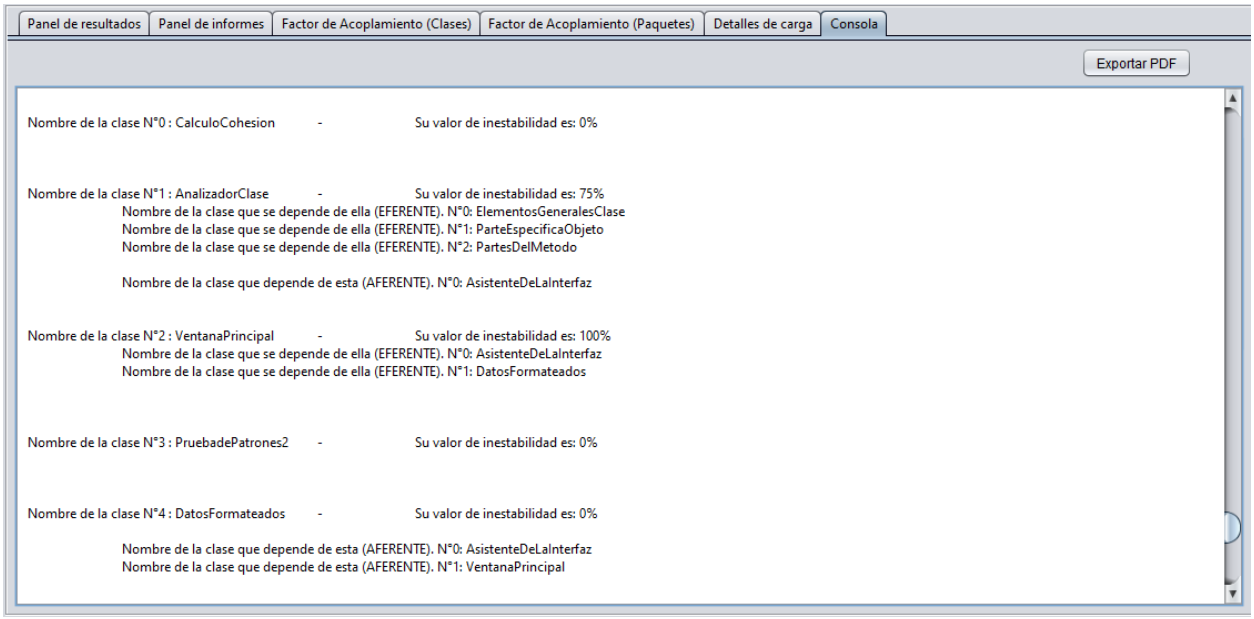


Figura 20. Relaciones Aferentes, Eferentes e Inestabilidad de cada clase

- **Factor de acoplamiento (clases)**

  En esta pestaña se muestra el resultado del cálculo de la métrica Factor de acoplamiento a partir de clases (ver Figura 21). La pestaña se divide en cuatro secciones: en la sección superior izquierda se observa el nombre del proyecto, la cantidad total de clases, la

cantidad total de acoplamientos entre clases del mismo y el valor del Factor de Acoplamiento del proyecto.

En la sección inferior izquierda se muestra la fórmula utilizada para calcular la métrica acompañada de la descripción de cada uno de sus parámetros.

En la sección superior derecha se puede observar un Combo Box que contiene las clases del proyecto, al elegir una de ellas es posible visualizar la cantidad de acoplamientos de la clase y el paquete al cual pertenece.

Así mismo, en el cuadrante inferior derecho se aprecia la descripción (detalles) de acuerdo al resultado de la métrica obtenido.

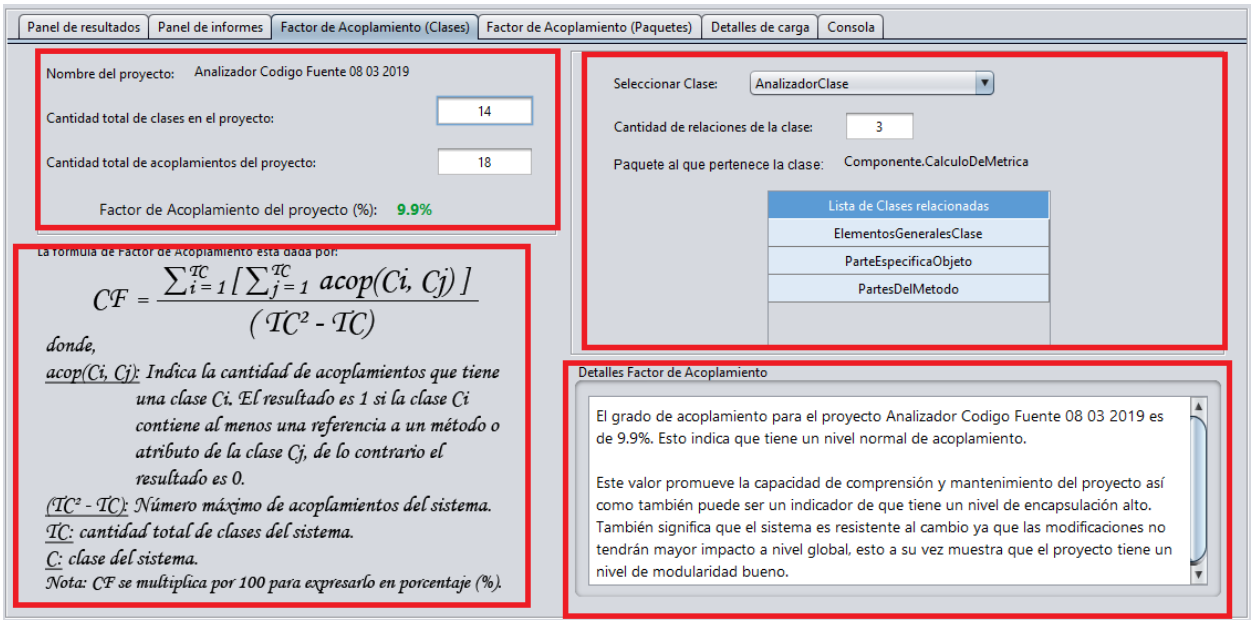


Figura 21. Pestaña Factor de acoplamiento (clases)

- **Factor de acoplamiento (paquetes)**

  Muestra el resultado del cálculo de la métrica Factor de acoplamiento a partir de paquetes (ver Figura 22). La pestaña se divide en cuatro secciones: en la sección superior izquierda se observa el nombre del proyecto, la cantidad total de clases, la cantidad total de acoplamientos entre clases del mismo y el valor del Factor de Acoplamiento del proyecto.

  En la sección inferior izquierda se muestra la fórmula utilizada para calcular la métrica junto con la descripción de cada parámetro de la misma.

En la sección superior derecha se puede observar un ComboBox con los paquetes del proyecto, al elegir uno de ellos es posible visualizar la cantidad de acoplamientos del paquete, los paquetes con los que está acoplado y las clases pertenecientes al paquete seleccionado.

Así mismo, en el cuadrante inferior derecho se aprecia la descripción (detalles) del resultado de la métrica obtenido.

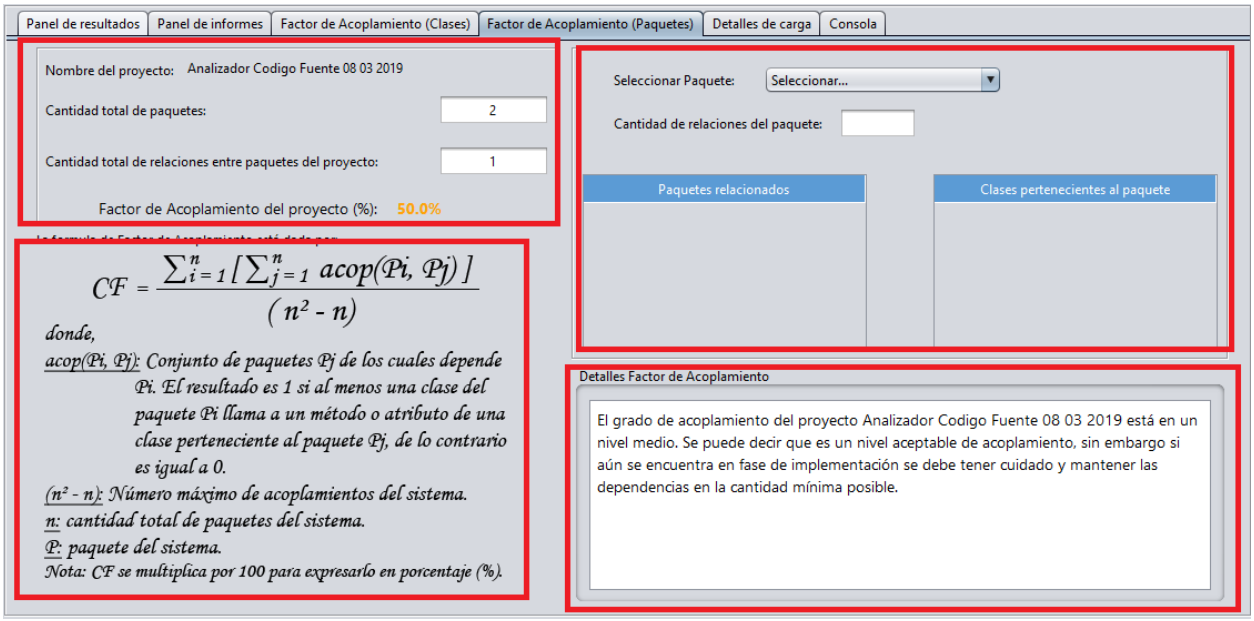


Figura 22. Pestaña Factor de Acoplamiento (paquetes)

Botón "Seleccionar carpeta del proyecto"

El botón "Seleccionar carpeta del proyecto" ejecuta la acción de abrir un jFileChooser (ventana con directorios locales de la computadora donde se ejecute la herramienta) para poder seleccionar una carpeta que contenga un proyecto con archivos .java, si la carpeta no contiene archivos .java se muestra el mensaje "No se encontraron archivos de código fuente java" y luego se regresa a la ventana principal para poder seleccionar otro directorio. Ver Figura 23.

Si la carpeta seleccionada tiene archivos .java, las rutas de los mismos serán cargadas en la pestaña "Detalles de carga" (ver Figura 24). Cuando los archivos son cargados se muestra una barra de progreso que indica que la operación de carga de archivos está en curso.

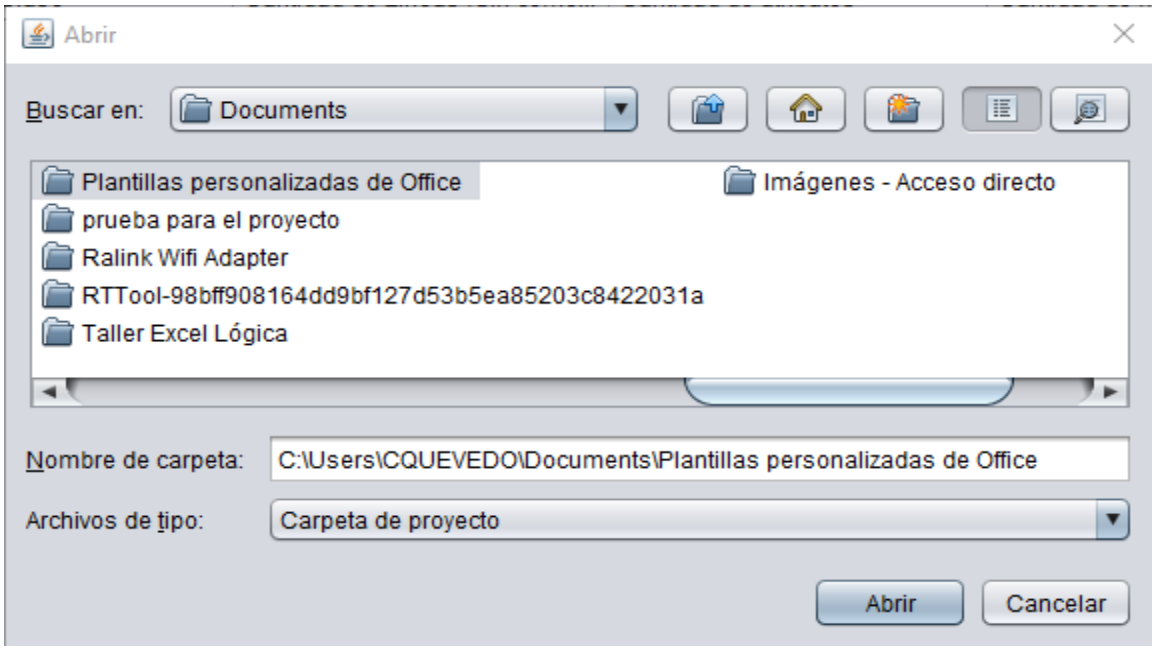


Figura 23. Directorios

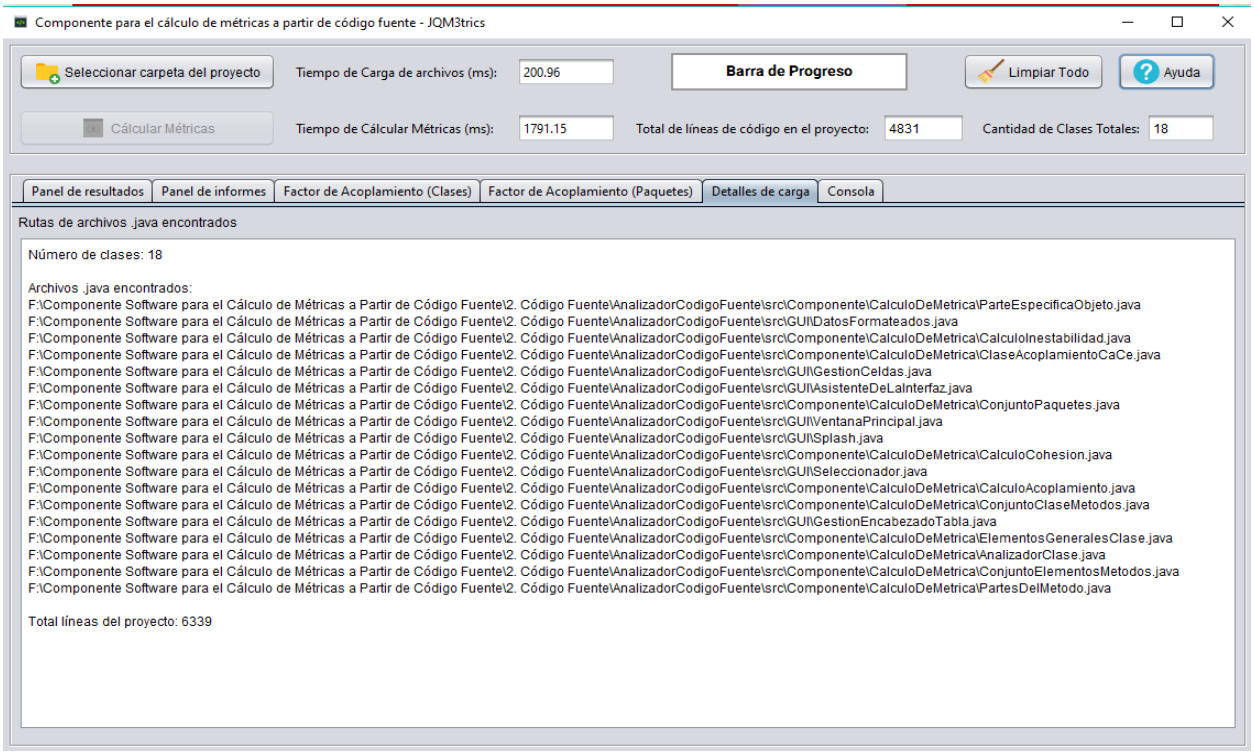
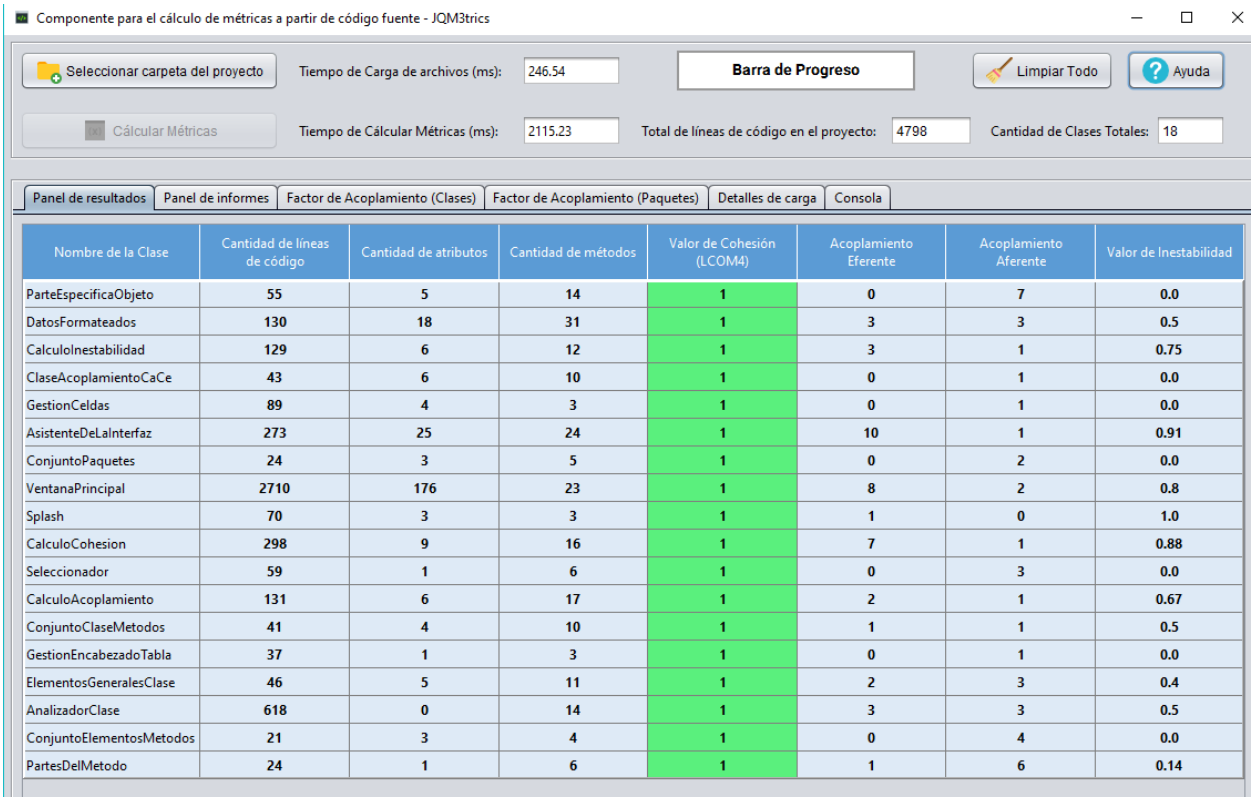


Figura 24. Carga de archivos

Botón “Calcular Métricas”

Con este botón se inicia el cálculo de las métricas de los archivos .java previamente cargados. Inicialmente se debe presionar el botón “Calcular métricas”, luego aparece un mensaje de estado con la siguiente pregunta “¿Seguro que son los archivos que desea analizar?”, al presionar “Sí”, inicia el proceso de análisis del código fuente, extracción de parámetros y cálculo de las métricas con dichos parámetros. La barra de progreso indica al usuario que se está llevando a cabo el cálculo de las métricas sobre el código fuente seleccionado.

Una vez el proceso de cálculo ha finalizado se mostrará el mensaje de estado: “El análisis ha finalizado”, este mensaje viene con un botón “Aceptar”, al pulsar dicho botón se muestran inmediatamente las métricas calculadas para cada archivo .java (ver Figura 25, Resultados del cálculo de métricas).



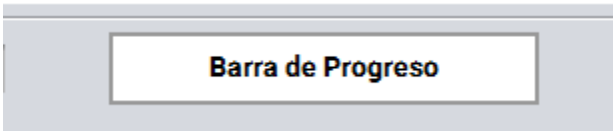
Nombre de la Clase	Cantidad de líneas de código	Cantidad de atributos	Cantidad de métodos	Valor de Cohesión (LCOM4)	Acoplamiento Eferente	Acoplamiento Aferente	Valor de Inestabilidad
ParteEspecificaObjeto	55	5	14	1	0	7	0.0
DatosFormateados	130	18	31	1	3	3	0.5
CalculoInestabilidad	129	6	12	1	3	1	0.75
ClaseAcoplamientoCaCe	43	6	10	1	0	1	0.0
GestionCeldas	89	4	3	1	0	1	0.0
AsistenteDeLaInterfaz	273	25	24	1	10	1	0.91
ConjuntoPaquetes	24	3	5	1	0	2	0.0
VentanaPrincipal	2710	176	23	1	8	2	0.8
Splash	70	3	3	1	1	0	1.0
CalculoCohesion	298	9	16	1	7	1	0.88
Seleccionador	59	1	6	1	0	3	0.0
CalculoAcoplamiento	131	6	17	1	2	1	0.67
ConjuntoClaseMetodos	41	4	10	1	1	1	0.5
GestionEncabezadoTabla	37	1	3	1	0	1	0.0
ElementosGeneralesClase	46	5	11	1	2	3	0.4
AnalizadorClase	618	0	14	1	3	3	0.5
ConjuntoElementosMetodos	21	3	4	1	0	4	0.0
PartesDelMetodo	24	1	6	1	1	6	0.14

Figura 25. Interfaz de Usuario

4.3.3.2 Principios de diseño de Interfaz de Usuario (IU) aplicados

Teniendo en cuenta los principios de diseño de IU establecidos por Jakob Nielsen, se aplicaron los siguientes a la IU (Martínez, 2017):

- **Coherencia y estándares**: los botones cumplen con las acciones que deben realizar y cada panel muestra lo que debe mostrar. Hay uniformidad en los colores y elementos gráficos; no se manejan distintos elementos para una misma acción.
- **Prevención de errores**: la herramienta muestra preguntas de advertencia antes de proseguir con cualquier acción. Por ejemplo, antes de calcular las métricas.
- **Correspondencia entre el sistema y el mundo real**: la interfaz tiene un diseño sencillo y claro. Los íconos de los botones de Seleccionar y Calcular son lógicos e intuitivos.
- **Eficiencia de uso**: la interfaz es sencilla y no está sobrecargada, de modo que visualmente es agradable al ojo humano. Además, todos los controles de la aplicación están visibles en la ventana.
- **Reconocimiento en vez de recordar**: los iconos permiten que la interfaz sea intuitiva.
- **Visibilidad del estado del sistema**: Con la interfaz desarrollada, el usuario puede saber en todo momento que es lo qué está pasando y obtiene una respuesta lo más rápida posible a sus acciones. Por ejemplo, la herramienta muestra una barra de progreso (ver imagen abajo) cuando la herramienta está cargando los archivos y cuando está realizando el cálculo de métricas.



- **Control de usuario y libertad**: El usuario tiene la opción de re-elegir un directorio de proyecto si escoge uno equivocado. La interfaz también cuenta con el botón ‘Limpiar Todo’ que permite deshacer cualquier operación que se haya realizado hasta el momento.
- **Diseño estético**: la interfaz desarrollada no contiene información innecesaria que pueda distraer al usuario.
- **Ayuda y documentación**: la interfaz de usuario posee un botón de ‘Ayuda’ donde se explica el uso básico de la herramienta e información sobre cómo se calculan las métricas establecidas.

4.3.4 Vista Lógica

Esta sección está conformada por el diagrama de componentes y el diagrama de clases. Cada ítem está acompañado de su respectiva descripción.

4.3.4.1 Diagrama de componentes

Teniendo en cuenta los requisitos arquitectónicos, la funcionalidad, el tipo de aplicación y los componentes que integran el software; se aplicó el patrón arquitectónico de programación por capas. El cual permite separar la capa de la Vista (interfaz gráfica de usuario) de la capa Componente, en donde se encuentra la lógica de negocio (Cálculo de métricas), permitiendo así la reutilización y optimización de código a futuro.

La capa más alta no maneja ningún tipo de lógica relacionada con el cálculo de métricas. Lo anterior aumenta el potencial reutilizable, de ahí que la capa Vista pueda sustituirse en el futuro.

Cabe anotar que JQM3trics usa tres librerías complementarias que son AbsoluteLayout, iText-5.0.5 y RSProgressBar. La primera es usada para el diseño de la Splash Screen (Ver sección 4.3.3 Interfaz de Usuario), la segunda es utilizada para la exportación de los datos de la consola en un documento PDF y la tercera es empleada para la barra de progreso de la IU.

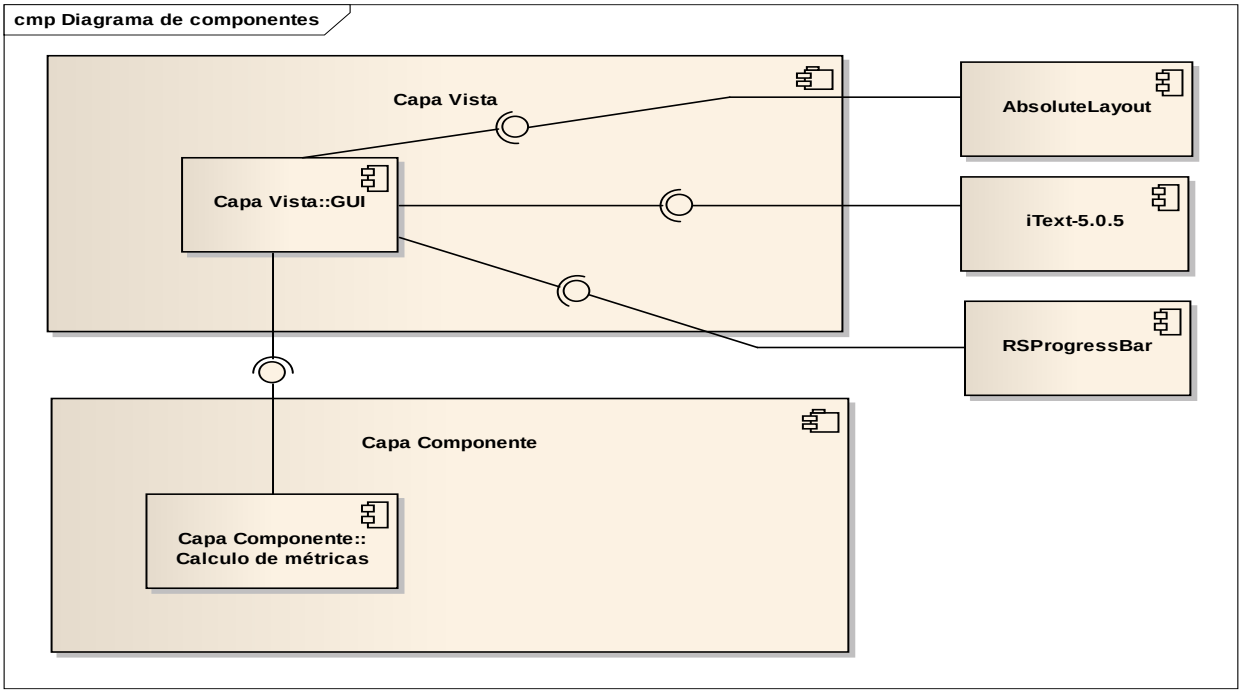


Figura 26. Diagrama de componentes

4.3.4.2 Diagrama de clases

En el diagrama de clases de la Figura 27 se evidencia el patrón arquitectónico de Capas. En el paquete de Cálculo de métricas se encuentran las clases encargadas de la lógica del negocio, es decir, el proceso de análisis que se ejecuta sobre el código fuente. En el paquete GUI se observa la clase AsistenteDeLaInterfaz, la cual asiste a la VentanaPrincipal de la Capa Vista, es decir, es la responsable de invocar la funcionalidad de calcular métricas del paquete Cálculo de métricas cuando se requiera. La clase AsistenteDeLaInterfaz hace las veces de un **Controlador** (patrón de diseño) quien recibe-maneja los eventos del sistema y define qué métodos del paquete CalculoDeMetricas se utilizan.

La clase Seleccionador permite la detección de archivos JAVA con sus rutas correspondientes, además permite seleccionarlos para su posterior análisis. Esta funcionalidad es invocada desde la GUI.

AnalizadorClase es la columna vertebral del análisis y cálculo de las métricas sobre el código fuente. En ella se encuentra la lógica para el cálculo de las líneas de código y realiza los ajustes necesarios sobre el código fuente para que métricas como Factor de Acoplamiento, Falta de cohesión en métodos, Acoplamiento Eferente y Acoplamiento Aferente sean calculadas apropiadamente.

La herencia presente entre la clase ParteEspecificaObjeto (clase padre) y las clases PartesDelMetodo y ElementosGeneralesClase (hijas) se debe a que los procesos a realizar varían según los objetos que se pueden encontrar en un archivo JAVA (Clase, Método, Atributo). La clase ParteEspecificaObjeto puede ser cualquier objeto (un método, una clase, un atributo, etc). Teniendo en cuenta esto, a futuro es posible agregar nuevas clases que hereden de ParteEspecificaObjeto sin afectar en gran medida el funcionamiento. Lo anterior evidencia el uso del patrón de dieños **Variaciones Protegidas**. ElementosGeneralesClase se refiere a los distintos tipos de clase que existen (Clase abstracta, Interface, entre otras).

La aplicación de Variaciones Protegidas también se evidencia en el hecho que a medida que ha ido evolucionando el proyecto se han ido agregando métricas hasta completar las cinco métricas propuestas sin causar gran impacto en el código fuente y su funcionamiento.

Es fundamental agregar que la creación de instancias de objetos es responsabilidad de las clases que cuentan con la información necesaria para hacerlo. Esto desemboca en un **encapsulamiento de la información.**

La clase AsistenteDeLaInterfaz evidencia el uso del patrón **Adapter**, ya que permite a la clase AnalizadorClase y Seleccionador interactuar por medio de ella. Esto también contribuye al desacoplo de las partes. El patrón Adapter también está presente en la interacción entre VentanaPrincipal y CalculoDeMetrica, la cual se realiza por medio de AsistenteDeLaInterfaz.

El patrón **Facade** fue utilizado en la clase AsistenteDeLaInterfaz puesto que esta crea una interfaz simplificada para tratar con otra parte del sistema más compleja: CalculoDeMetrica.

El Diagrama de Clases se encuentra como Anexo en la ruta \Componente Software para el \Cálculo de Métricas a Partir de Código Fuente\4. Anexos, el nombre del archivo es Diagrama de Clases. Para ir directamente a la imagen se puede hacer clic en el siguiente vínculo o abrir el archivo AnalizadorCodigoFuente.eap:

ANEXO C. Diagrama de Clases

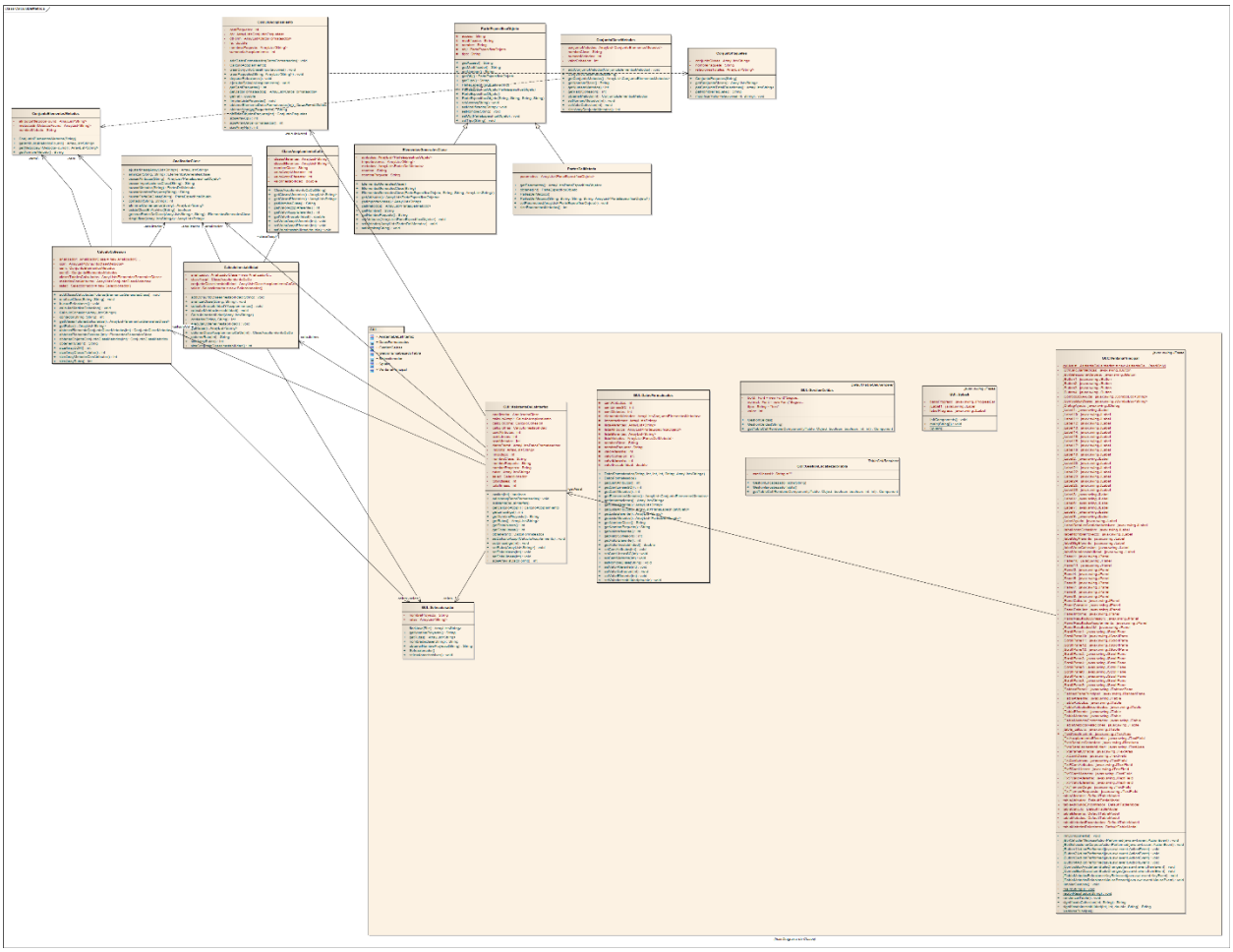


Figura 27. Diagrama de Clases

4.3.5 Vista de procesos

En esta sección se muestra, a través de tres diagramas de secuencia, el comportamiento del sistema en los tres momentos del Caso de Uso “Calcular Métrica”. Los momentos ocurren de forma secuencial en el siguiente orden respectivamente: “Seleccionar código fuente”, “Calcular métricas” y “Mostrar resultados”.

La figura 28 muestra el proceso de selección del código fuente: en el diagrama se pueden observar los métodos que son activados una vez el Ingeniero de Software presiona el botón “Seleccionar código” desde la interfaz de usuario; así mismo se muestran los mensajes que envían las clases involucradas en este primer momento del caso de uso Calcular Métricas. Para ver este momento del diagrama con mayor detalle se debe abrir el archivo AnalizadorCodigoFuente.eap ubicado en la ruta \Componente Software para el Cálculo de

Métricas a Partir de Código Fuente\1. Documentación\ AnalizadorCodigoFuente o hacer clic en el siguiente vínculo:

ANEXO D. Diagrama de secuencia, Seleccionar Carpeta del Proyecto

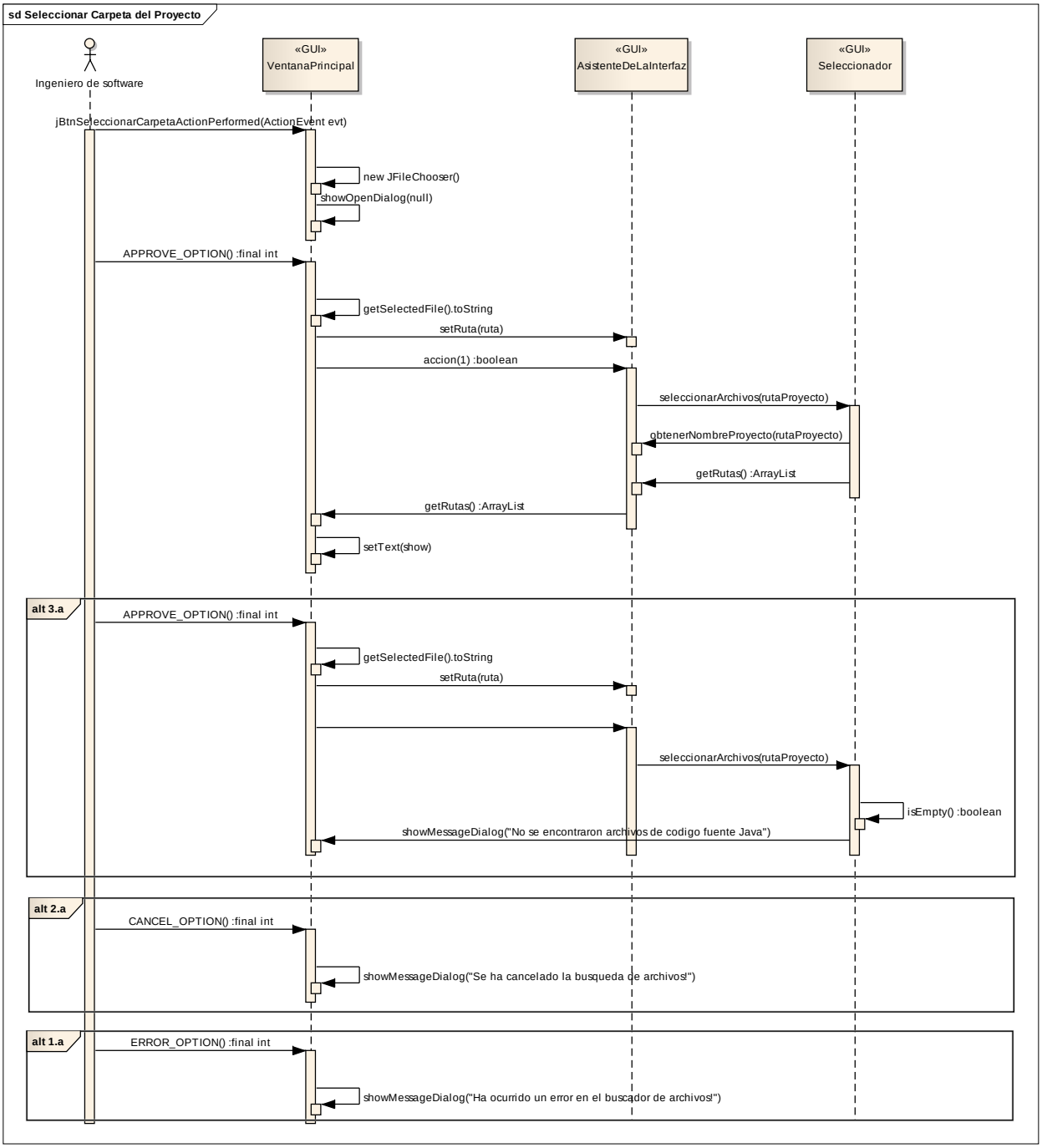


Figura 28. Diagrama de secuencia parte 1, Seleccionar carpeta del proyecto

A continuación se observa el segundo diagrama que describe el cálculo de las métricas:

De forma similar al momento anterior, la figura 29 muestra los métodos involucrados en durante el cálculo de métricas. Se pueden observar los métodos que se activan una vez el botón “Calcular Métricas” es presionado desde la GUI por el Ingeniero de Software. A través del diagrama se observa la comunicación que hay entre las Clases VentanaPrincipal, Asistente de la Interfaz y las clases contenidas en el componente CalculoDeMetrica. Para ver este momento del diagrama con más detalle se debe abrir el archivo AnalizadorCodigoFuente.eap ubicado en la ruta Componente Software para el \Cálculo de Métricas a Partir de Código Fuente\1. Documentación\ AnalizadorCodigoFuente o hacer clic en el siguiente vínculo:

ANEXO E. Diagrama de Secuencia, Calcular Métricas

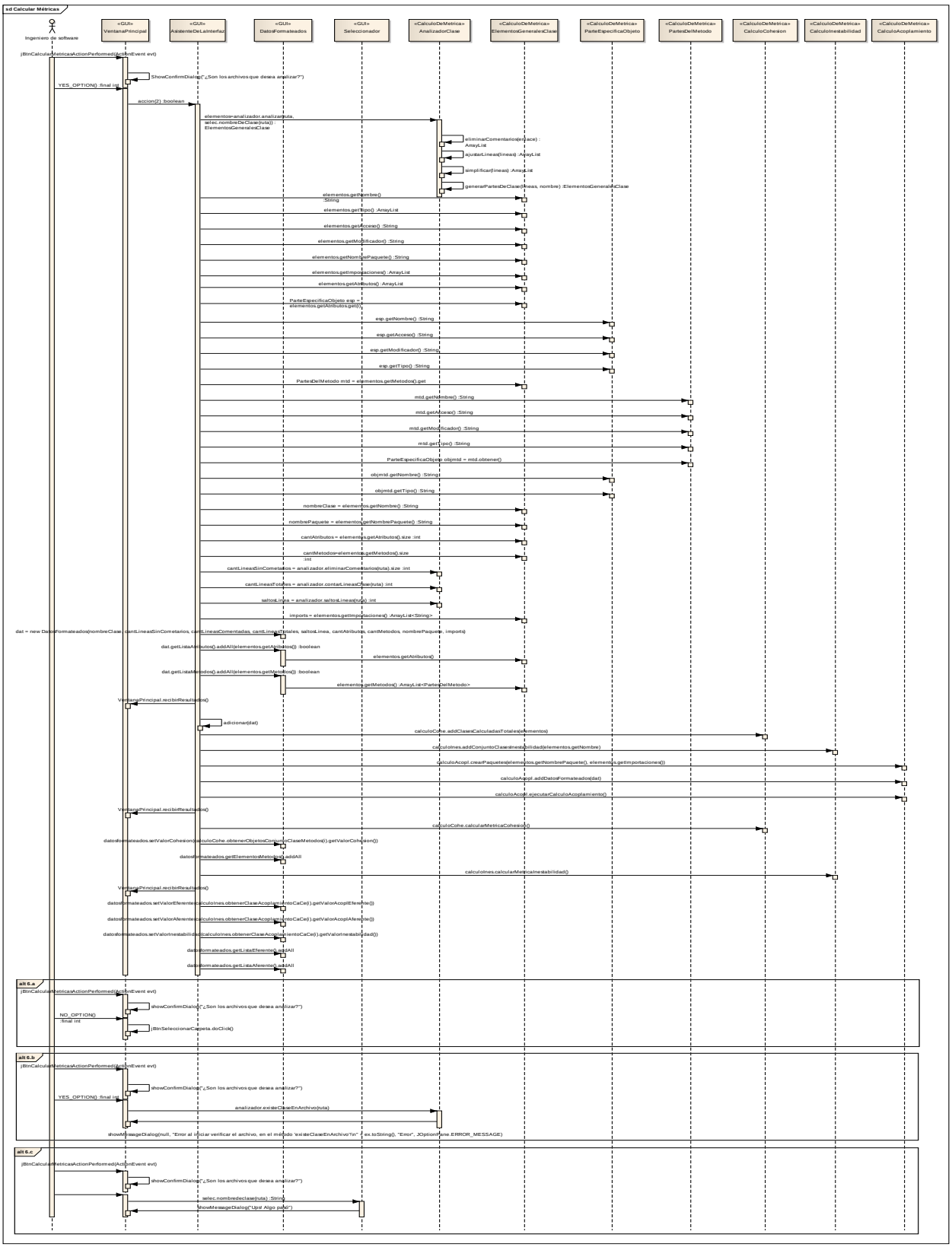


Figura 29. Diagrama de secuencia parte 2, Calcular Métricas

A continuación se muestra el tercer momento del caso de uso, el cual describe la muestra de resultados:

En la figura 30 se encuentra el momento final del caso de uso “Calcular Métricas”, este momento es conocido como “Mostrar Resultados” y se activa una vez el sistema finaliza de calcular las métricas. Para el caso de esta investigación, se construyó una interfaz de usuario que invoca las funciones del componente y por medio de la misma se muestran los resultados del cálculo de las métricas.

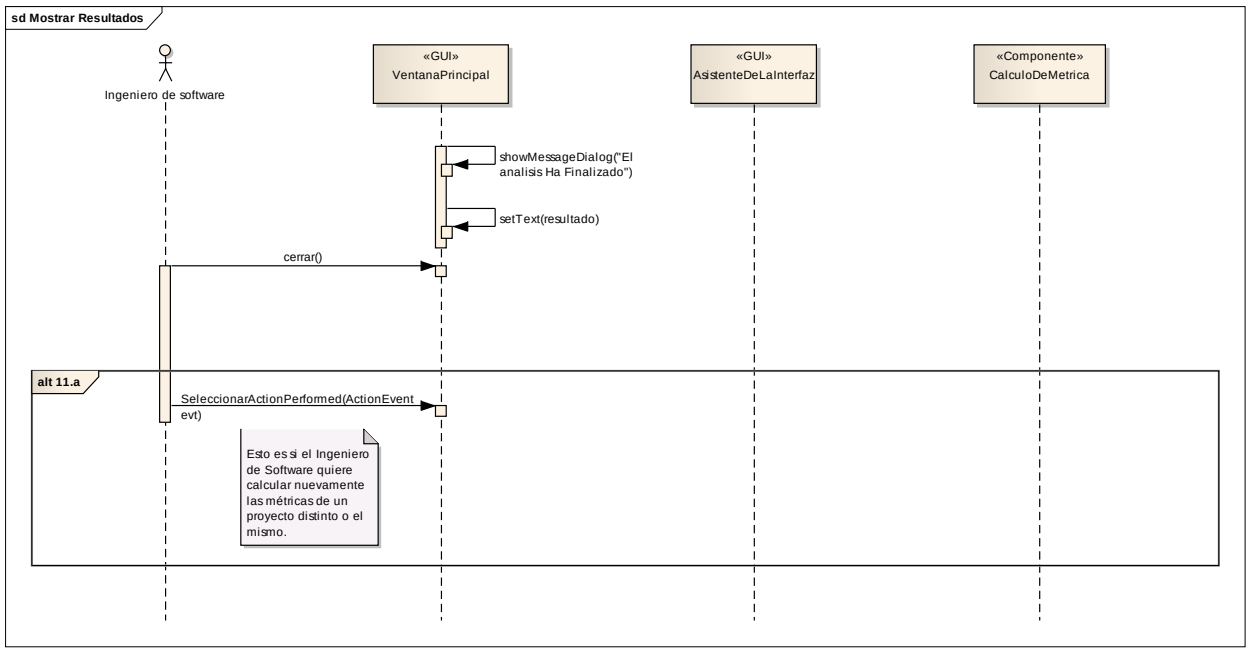


Figura 30. Diagrama de secuencia parte 3, Mostrar Resultados

4.4 Modelo de implementación

En esta sección se describen las tecnologías utilizadas para el desarrollo de la solución y seguidamente se muestran las vistas de desarrollo y física.

Para el desarrollo de esta solución se utilizó el lenguaje de programación JAVA puesto que se definió como una restricción del proyecto. Como Entorno de desarrollo se utilizaron los IDE de Netbeans 8.0 y 8.2.

4.4.1 Vista de desarrollo

El diagrama de la Figura 31 es similar al de componentes con la particularidad de que este está relacionado con los recursos (iconos de la IU, imágenes) utilizados por la IU.

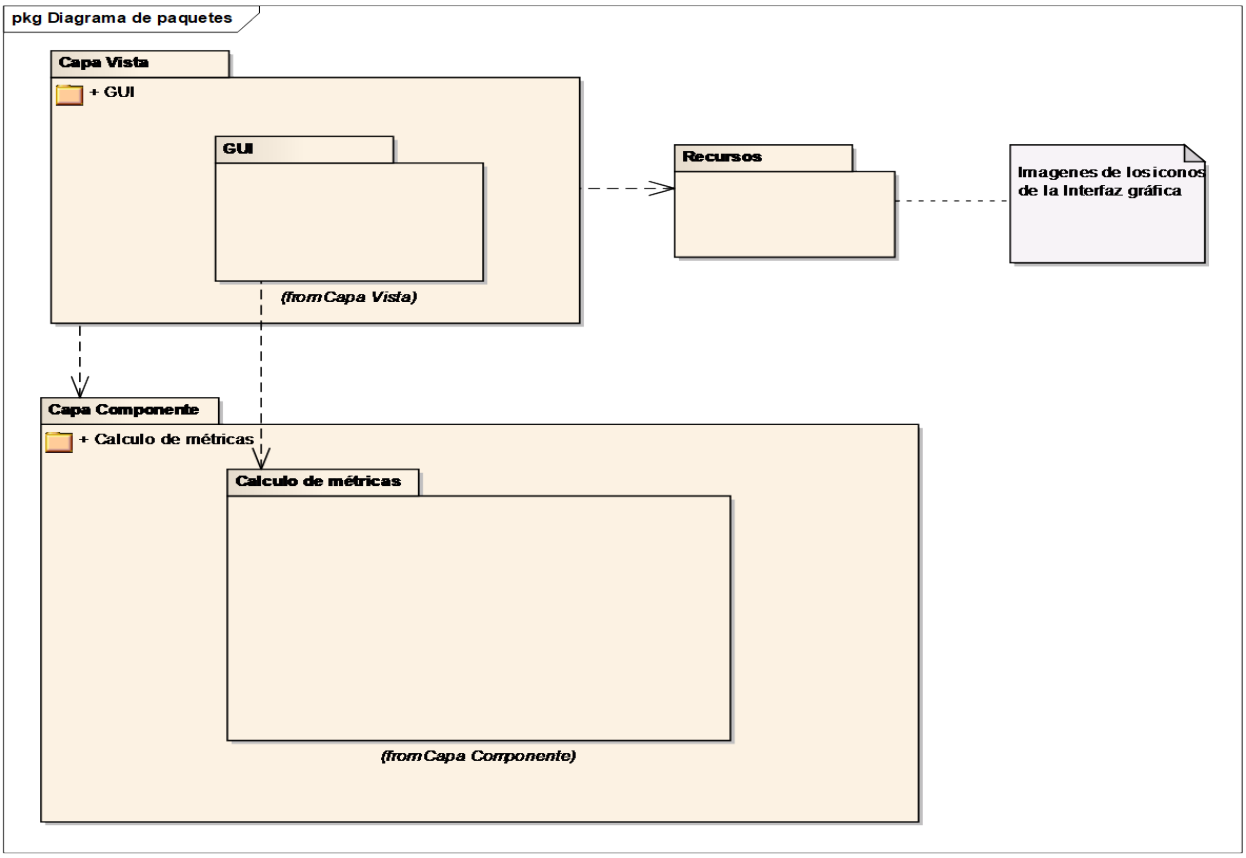


Figura 31. Vista de desarrollo

4.4.2 Vista física

En este apartado se muestra el diagrama de despliegue como la interacción entre el software y las características hardware de la máquina donde se ejecuta dicho software. De antemano es importante mencionar que para el despliegue del sistema es necesario que la computadora donde se ejecute la herramienta tenga la máquina virtual de JAVA, así como también es necesario que tenga un sistema operativo Windows XP o una versión superior.

4.4.2.1 Diagrama de despliegue

En la Figura 32 se puede apreciar el entorno de ejecución del sistema completo. Tanto la Interfaz de Usuario Gráfica como el Componente se ejecutan localmente en la misma máquina (PC), que, entre otras cosas, debe tener un sistema operativo Windows XP o superior. Puesto que la herramienta es nativa de JAVA, debe ser ejecutada sobre la máquina virtual de JAVA la cual está contenida en el Entorno de Ejecución de JAVA (JRE). En el diagrama también se aprecia el uso que el componente hace del código fuente cuyos archivos deben ser formato .java

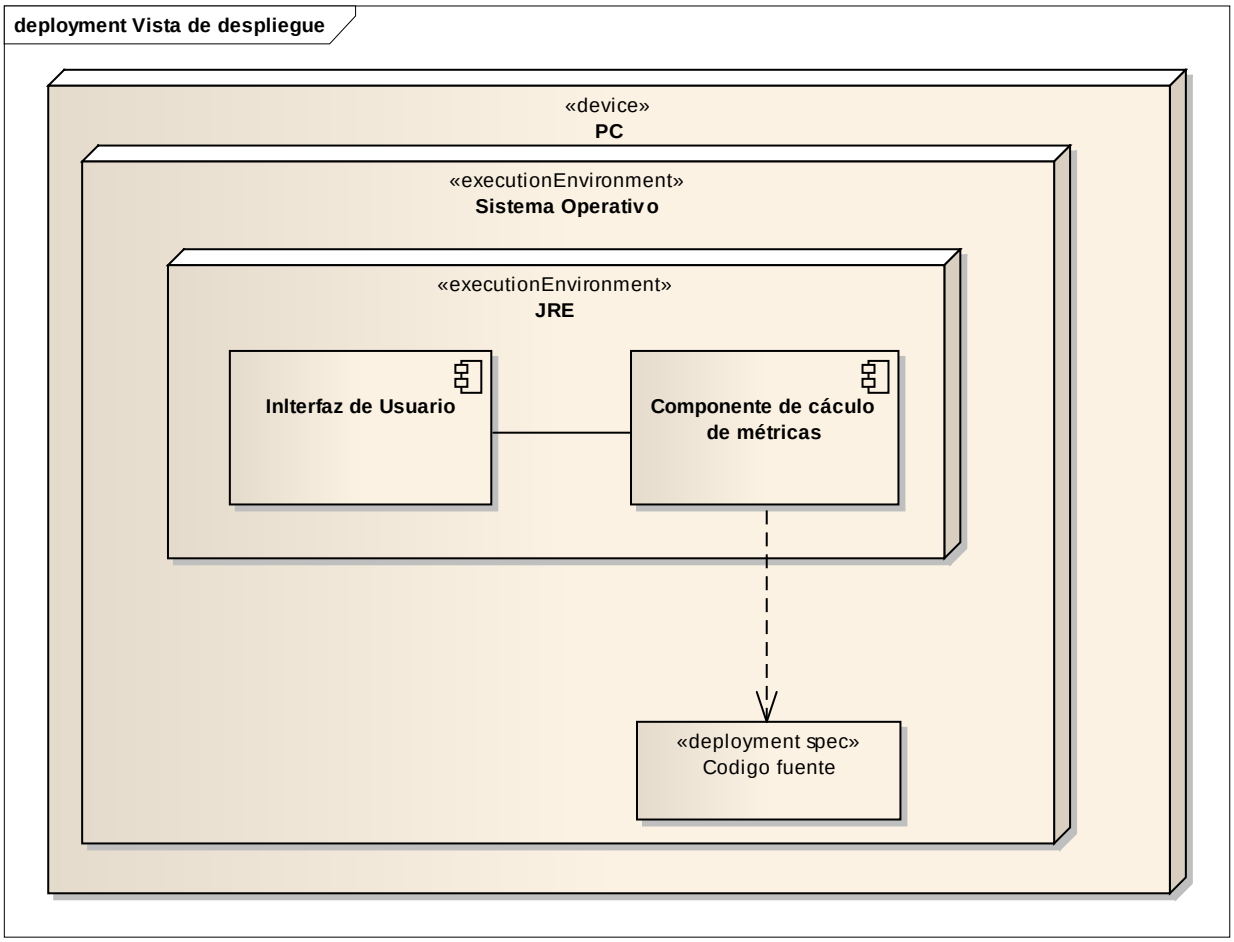


Figura 32. Despliegue del sistema

4.5 Resultados de las Pruebas

En esta sección se presentan los resultados de las pruebas de carga de archivos y de caja negra aplicadas al software. Se inicia con las pruebas de carga de archivos, las cuales se llevan a cabo en el entorno Netbeans IDE 8.2

En este capítulo se especifican los dos tipos de prueba propuestos en el quinto objetivo específico de esta investigación. Se inicia con las pruebas de rendimiento, con las cuales se pretende mostrar los tiempos de respuesta de la aplicación al cargar proyectos de distintos tamaños. En el capítulo análisis de resultados se muestra a través de gráficas, la relación entre el tiempo y el tamaño en megabytes de cuatro proyectos de código fuente.

Seguidamente se describen las pruebas unitarias de caja negra, cuyo objetivo es mostrar que cada funcionalidad cumpla adecuadamente con lo que debe hacer. Se realizan pruebas a las funciones

de carga de archivos java, restricción de archivos distintos al formato java y muestra general de las métricas. Además se hacen pruebas sobre el cálculo de la cantidad de líneas de código, Factor de acoplamiento, Inestabilidad y Falta de cohesión en clases.

Es importante enfatizar que para efectos de este documento se muestran las pruebas realizadas sobre la última versión de la aplicación.

4.5.1 Pruebas de Rendimiento (tiempos de respuesta)

A continuación se detallan los casos de prueba establecidos para el software desarrollado. Se establecieron cuatro (4) casos de prueba y en cada uno de ellos se analizó un proyecto de distinto tamaño (Megabytes) para detectar la variación de los tiempos de respuesta de acuerdo a los mismos. Para cada análisis se registran: el tamaño del proyecto, el tiempo (redondeado) de carga de los archivos .java, el tiempo que tarda JQM3trics en calcular las métricas, el total de líneas de código y el número de clases. Se denomina 'tamaño del proyecto' a la sumatoria de los tamaños de los archivos .java.

Para el tamaño del proyecto únicamente se tienen en cuenta los archivos con extensión .java, cualquier otro tipo de archivo como imágenes, ejecutables jar, etc no son tenidos en cuenta. El tamaño de un proyecto no es calculado por JQM3trics. Por otro lado, el tiempo de Calcular Métricas hace referencia al tiempo que tarda la herramienta en calcular las métricas del proyecto seleccionado y está dado en milisegundos. Este elemento está ubicado en la parte superior de la ventana, tal como se observa en la figura 33. El tiempo de carga de los archivos .java hace referencia al tiempo que tarda la aplicación en cargar el total de archivos .java de un proyecto de código fuente seleccionado, también está ubicado en la parte superior de la Interfaz de Usuario.

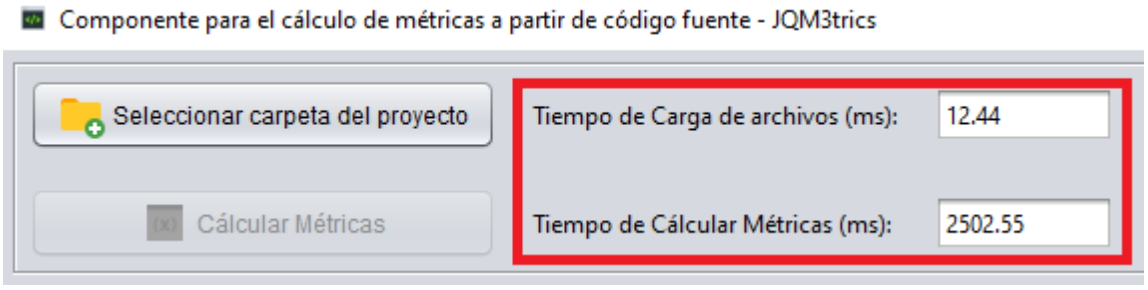


Figura 33. Tiempo de carga de archivos y tiempo del cálculo de métricas

La abreviación ‘ms’ hace referencia a milisegundos. En las siguientes pruebas “Tiempo aproximado de respuesta” hace referencia al “Tiempo de calcular métricas”. Por otro lado, vale la pena recordar que para calcular la cantidad de líneas de código no se tienen en cuenta las líneas comentadas ni los saltos de línea.

4.5.1.1 Caso de prueba 1

Nombre del proyecto: JQM3trics (4374 líneas de código)

Tamaño del proyecto: 246 KB

Tiempo aproximado de carga de archivos (ms): 10.1 = 0.0101 s

Tiempo aproximado de respuesta (ms): 1916.68 = 1.91 s

Total de líneas de código: 4374

Número de archivos JAVA: 18

Problemas en tiempo de ejecución: Ninguno

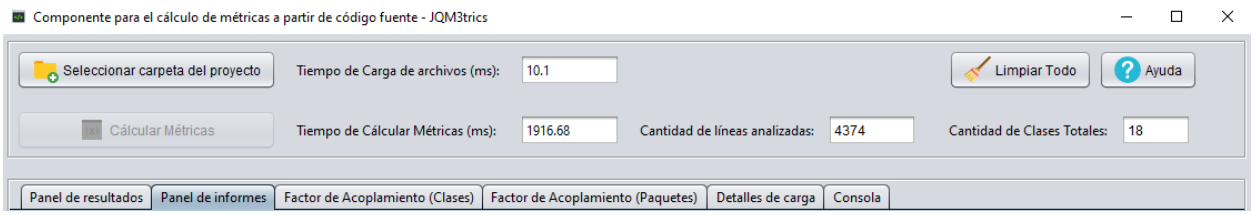


Figura 34. Tiempos de respuesta caso de prueba 1

4.5.1.2 Caso de prueba 2

Nombre del proyecto: RTTool

Tamaño del proyecto: 421 KB

Tiempo aproximado de carga de archivos (ms): 52.66= 0.05266 s

Tiempo aproximado de respuesta: 4476.75= 4.47 s

Total de líneas de código: 8797

Número de archivos .JAVA del proyecto: 49

Problemas en tiempo de ejecución: Ninguno

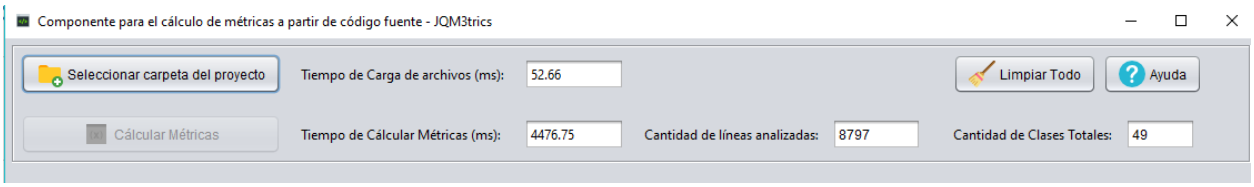


Figura 35. Tiempos de respuesta caso de prueba 2

4.5.1.3 Caso de prueba 3:

Nombre del proyecto: MASU

Tamaño del proyecto (KB): 3430 = 3.43 MB

Número de archivos .JAVA del proyecto: 515

Tiempo aproximado de carga de los archivos (ms): 1784.87= 1.784 s

Tiempo aproximado de respuesta (ms): 17733.66= 17.773 s

Total de líneas de código: 78347

Problemas en tiempo de ejecución: Ninguno

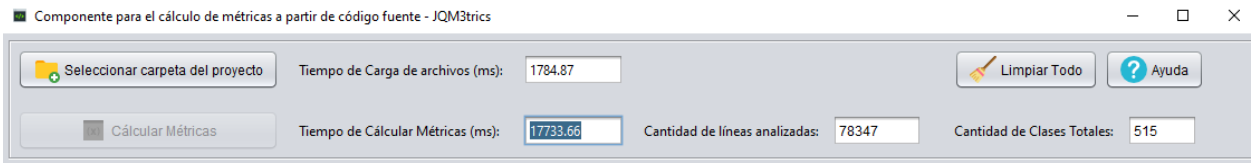


Figura 36. Tiempos de respuesta caso de prueba 3

4.5.1.4 Caso de prueba 4

Nombre del proyecto: JHotDraw

Tamaño del proyecto: 1980 KB= 1.98 MB

Tiempo aproximado de carga de archivos (ms): 552.83= 0.552 s

Tiempo aproximado de respuesta (ms): 7106.47 = 7.106 s

Total de líneas de código: 39435

Número de archivos .JAVA del proyecto: 310

Problemas en tiempo de ejecución: Ninguno

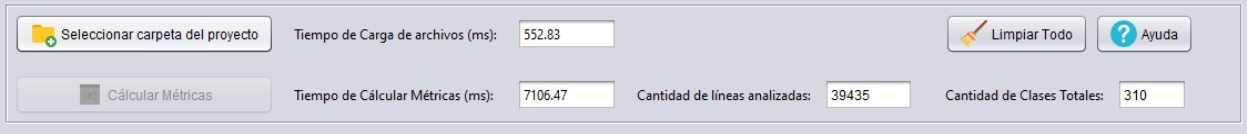


Figura 37. Carga de archivos .JAVA

4.5.2 Pruebas de Caja Negra

En este capítulo se describen dos tipos de pruebas unitarias de caja negra, el primer tipo está enfocado en la carga de archivos y está compuesto por: carga de archivos .java, restricción de archivos que no estén en formato Java y mostrar adecuadamente los resultados de las métricas. El segundo tipo de prueba está orientado al Cálculo de Métricas, y está conformado por pruebas realizadas a seis métricas establecidas en esta investigación, las cuales son: Factor de acoplamiento, acoplamiento eferente, acoplamiento aferente, inestabilidad, falta de cohesión en las clases y cantidad de líneas de código. Cada prueba está descrita en su respectivo formato de casos de prueba.

4.5.2.1 Caso de prueba número 1: Carga de archivos .java

A diferencia de las pruebas de rendimiento descritas en el capítulo 4.5.1, con esta prueba se busca corroborar que la carga de archivos se haga correctamente. A continuación se observa la descripción de la prueba:

Convenciones:

“Id_CP: [numero]”: es el identificador de la prueba, en este caso es la prueba número 1

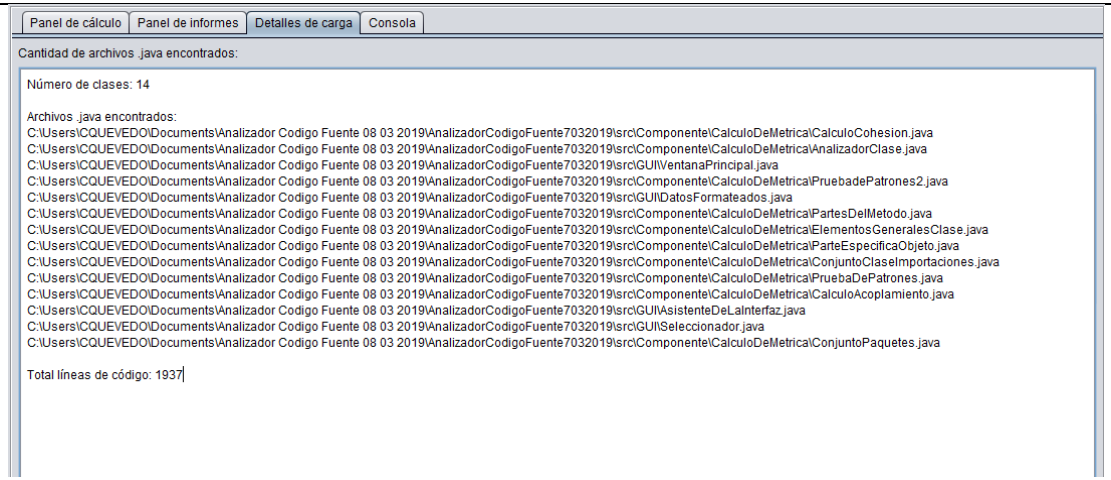
Propósito del caso de prueba: para qué se hace, qué se pretende verificar con el caso de prueba

Pasos o secuencia lógica: secuencia de pasos que se deben seguir para realizar la prueba

Estado de la prueba: la prueba es pasada si los resultados obtenidos concuerdan con los resultados esperados.

Tabla 8. Caso de prueba número 1

Id_CP:1	**Funcionalidad a probar: Carga de archivos .java**	Fecha: 21/04/2019 D M A
Nombre del proyecto: JQM3trics		**Plataforma**: aplicación JAVA
Propósito del caso de prueba:	Verificar que la aplicación cargue archivos .java correctamente.	
Pre Requisitos	La aplicación debe estar lanzada o en ejecución.	
Pasos o secuencia lógica	• Presionar el botón “Seleccionar carpeta del proyecto” • El sistema despliega un cuadro de diálogo que muestra los directorios locales de la computadora. • Seleccionar un directorio que contenga la carpeta de un proyecto de código fuente java. • Presionar el botón “Abrir”	
Resultados esperados	En la pestaña “detalles de carga” de la interfaz gráfica, el sistema muestra las rutas de los archivos .java contenidos en el proyecto seleccionado.	
Resultados obtenidos	En la pestaña “detalles de carga” de la interfaz gráfica, el sistema muestra las rutas de los archivos .java contenidos en el proyecto seleccionado.	

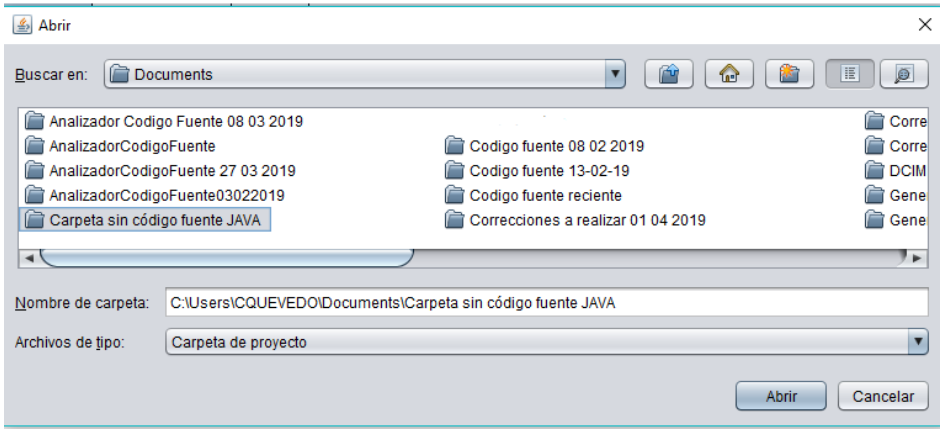
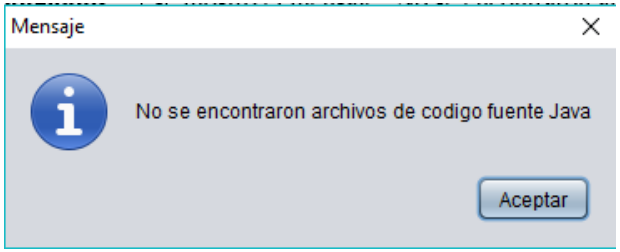
	
Estado de la prueba:	☑ Pasada ☐ Fallida

4.5.2.2 Caso de prueba número 2: Restricción de archivos distintos a formato .java

En la siguiente tabla se describe el caso de prueba cuyo resultado esperado es solo permitir el análisis de archivos código fuente en formato .java

Tabla 9. Caso de prueba número 2

Id_CP:2	**Funcionalidad a probar: No aceptación de archivos distintos a formato .java**	Fecha: 21/04/2019 D M A
Nombre del proyecto: JQM3trics		**Plataforma**: aplicación JAVA
Propósito del caso de prueba:	Verificar que la aplicación **NO** acepte archivos distintos al formato .java	
Pre Requisitos	La aplicación debe estar lanzada o en ejecución. Debe seleccionarse una carpeta local en la máquina donde se ejecuta la aplicación	
Pasos o secuencia lógica	• Presionar el botón "Seleccionar carpeta del proyecto" • El sistema despliega un cuadro de diálogo que muestra los directorios locales de la computadora. • Seleccionar un directorio que contenga la carpeta de un proyecto de código fuente java. • Presionar el botón "Abrir"	
Resultados esperados	Aparece un mensaje de estado indicando lo siguiente: "No se encontraron archivos de código fuente java"	

Resultados obtenidos	Se probó con archivos cuya extensión es .class y html. En ambos casos se mostró el mensaje “No se encontraron archivos de código fuente java”  
Estado de la prueba:	■ Pasada ☐ Fallida

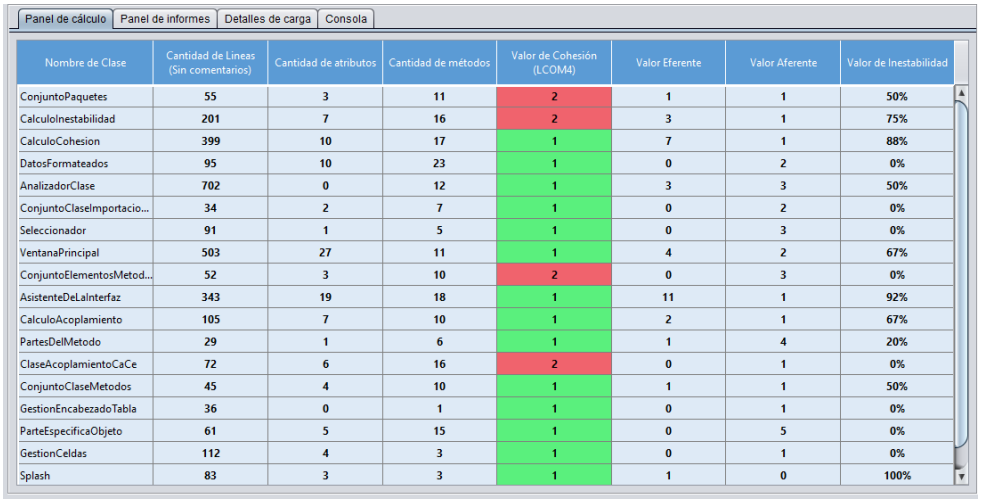
4.5.2.3 Caso de prueba número 3: mostrar resultados de las métricas

En la siguiente tabla se describe el caso de prueba cuyo resultado esperado es calcular y mostrar el valor de las métricas adecuadamente. Las métricas son: cantidad de líneas de código por clase, cantidad de atributos por clase, cantidad de métodos por clase, falta de cohesión en métodos, acoplamiento eferente, acoplamiento aferente e inestabilidad.

Tabla 10. Caso de prueba número 3

Id_CP:3	**Funcionalidad a probar: Cálculo de métricas**	Fecha: 21/04/2019 D M A
Nombre del proyecto: JQM3trics		**Plataforma**: aplicación JAVA
Propósito del caso de prueba:	Verificar que la aplicación calcule el valor de las métricas adecuadamente	
Pre Requisitos	• La aplicación debe estar lanzada o en ejecución. • Debe haber por lo menos un archivo de código fuente java cargado en la aplicación. En este caso se cargó una versión antigua (08/02/2019) del código fuente de JQM3trics.	
Pasos o secuencia lógica	• Presionar el botón “Calcular métricas” • El sistema despliega un mensaje de estado con el interrogante “¿Son los archivos que desea analizar?” y se presiona “Sí”. • Se muestra un mensaje de estado con el mensaje “El análisis ha finalizado” • Se muestran los resultados de las métricas en la pestaña “panel de cálculo”.	
Resultados esperados	• Mostrar la cantidad de líneas de código de cada una de las clases.	

	• Mostrar la cantidad de clases. • Mostrar la cantidad de atributos de cada una de las clases. • Mostrar la cantidad de métodos de cada una de las clases. • Mostrar el valor de la falta de cohesión en métodos de cada una de las clases. • Mostrar el grado de acoplamiento eferente. • Mostrar el grado de acoplamiento aferente. • Mostrar el valor de la inestabilidad del proyecto analizado.
Resultados obtenidos	• Se mostró la cantidad de líneas de código (sin comentarios) de cada una de las clases. En total son 1764 líneas de código. • Se mostraron en total trece clases. • Se calcularon los atributos correctamente, se corroboró manualmente. • Se calcularon la cantidad de métodos incluyendo setters y getters. • El cálculo de la cohesión fue basado en LCOM4 y se calculó adecuadamente. • El grado de acoplamiento aferente y eferente está dado por el número de relaciones que cumplan una de las siguientes condiciones: Si hay extensión, si hay implementación, si dentro de los atributos hay una instancia de, si dentro de los métodos se recibe parámetros o devuelve instancias; o si dentro de los métodos se crean instancias de. • El valor de la inestabilidad se mostró correctamente. Se obtiene de dividir el acoplamiento eferente / (acoplamiento eferente + acoplamiento aferente). Para los casos en que el denominador fue igual a cero la inestabilidad dio cero, si el denominador fue mayor que cero, se calculó la inestabilidad con la fórmula mencionada

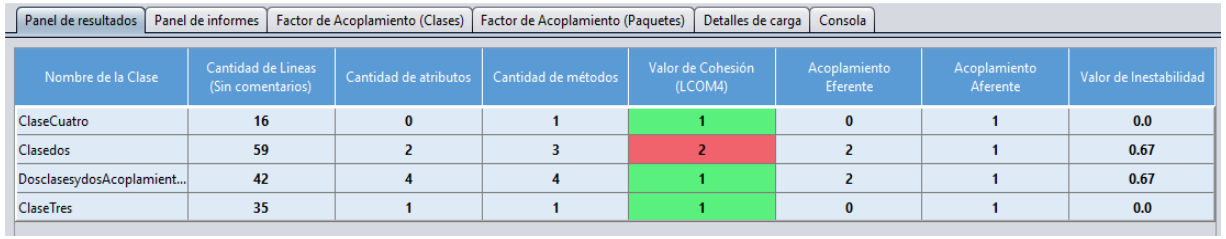
	anteriormente.  Panel de cálculo | Panel de informes | Detalles de carga | Consola Nombre de Clase | Cantidad de Lineas (Sin comentarios) | Cantidad de atributos | Cantidad de métodos | Valor de Cohesión (LCOM4) | Valor Eferente | Valor Aferente | Valor de Inestabilidad ConjuntoPaquetes 55 3 11 2 1 1 50% CalculoInestabilidad 201 7 16 2 3 1 75% CalculoCohesion 399 10 17 1 7 1 88% DatosFormateados 95 10 23 1 0 2 0% AnalizadorClase 702 0 12 1 3 3 50% ConjuntoClaseImportacio... 34 2 7 1 0 2 0% Seleccionador 91 1 5 1 0 3 0% VentanaPrincipal 503 27 11 1 4 2 67% ConjuntoElementosMetod... 52 3 10 2 0 3 0% AsistenteDeLaInterfaz 343 19 18 1 11 1 92% CalculoAcoplamiento 105 7 10 1 2 1 67% PartesDelMetodo 29 1 6 1 1 4 20% ClaseAcoplamientoCaCe 72 6 16 2 0 1 0% ConjuntoClaseMetodos 45 4 10 1 1 1 50% GestionEncabezadoTabla 36 0 1 1 0 1 0% ParteEspecificaObjeto 61 5 15 1 0 5 0%
Estado de la prueba:	☐ Pasada ☐ Fallida

4.5.2.4 Caso de prueba número 4: cálculo de LCOM4

En la siguiente tabla se describe el caso de prueba cuyo resultado esperado es que el valor de LCOM4 para una determinada clase del código fuente PruebaFactorAcoplamiento sea igual a dos.

Tabla 11. Caso de prueba número 4

Id_CP:3	**Métrica a probar: cálculo de la falta de cohesión en clases (LCOM4)**	Fecha: 26/05/2019 D M A
Nombre del proyecto: JQM3trics		**Plataforma**: aplicación JAVA
Propósito del caso de prueba:	Verificar que la aplicación calcule el valor de la falta de cohesión de una clase adecuadamente	
Pre Requisitos	• La aplicación debe estar lanzada o en ejecución. • Debe estar cargado el código fuente llamado “PruebaFactorAcoplamiento”.	
Pasos o secuencia lógica	• Presionar el botón “Calcular métricas” • El sistema despliega un mensaje de estado con el interrogante “¿Son los archivos que desea analizar?” y se presiona “Sí”. • Se muestra un mensaje de estado con el mensaje “El análisis ha finalizado” • Se muestra la métrica LCOM4 en la pestaña “panel de resultados”.	
Resultados	• Mostrar el valor de la falta de cohesión en métodos de la clase “Clasedos” con un valor igual a 2	

esperados	
Resultados obtenidos	. • El cálculo de la cohesión fue basado en LCOM4 y se calculó adecuadamente. 
Estado de la prueba:	☐ Pasada ☐ Fallida

Panel de resultados | Panel de informes | Factor de Acoplamiento (Clases) | Factor de Acoplamiento (Paquetes) | Detalles de carga | Consola

Nombre de la Clase	Cantidad de Lineas (Sin comentarios)	Cantidad de atributos	Cantidad de métodos	Valor de Cohesión (LCOM4)	Acoplamiento Eferente	Acoplamiento Aferente	Valor de Inestabilidad
ClaseCuatro	16	0	1	1	0	1	0.0
Clasedos	59	2	3	2	2	1	0.67
DosclasesydosAcoplament...	42	4	4	1	2	1	0.67
ClaseTres	35	1	1	1	0	1	0.0

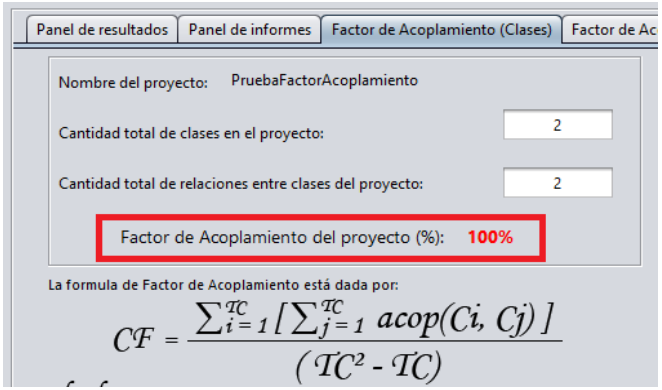
4.5.2.5 Caso de prueba número 5: Factor de acoplamiento a partir de clases

En la siguiente tabla se describen los resultados de una prueba que cuenta con cuatro resultados esperados de acuerdo a cuatro entradas específicas que tienen las siguientes características:

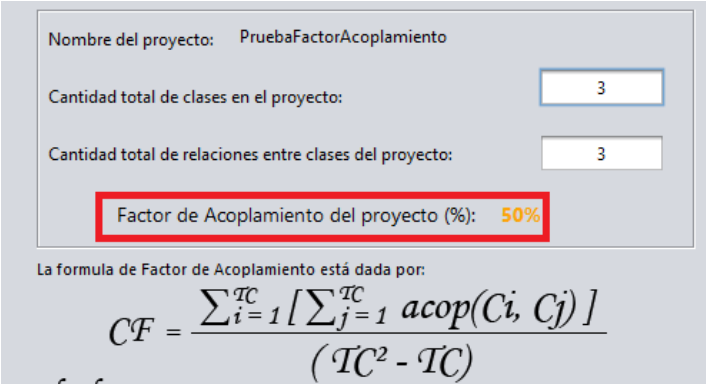
- Código fuente con dos clases y dos acoplamientos
- Código fuente con tres clases y tres acoplamientos
- Código fuente con tres clases y cuatro acoplamientos
- Código fuente con cuatro clases y tres acoplamientos

Tabla 12. Caso de prueba número 5

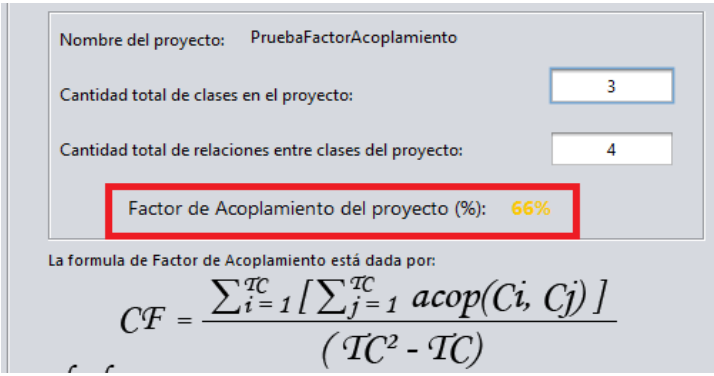
Id_CP:3	**Métrica a probar: Factor de acoplamiento a partir de clases**	Fecha: 26/05/2019 D M A
Nombre del proyecto: JQM3trics versión 26/05/19		**Plataforma**: aplicación JAVA
Propósito del caso de prueba:	Corroborar que la aplicación calcule adecuadamente el valor de la métrica de acuerdo a las entradas que reciba la herramienta.	
Pre Requisitos	• La aplicación debe estar lanzada o en ejecución. • Como la prueba consta de cuatro momentos, se debe seleccionar inicialmente un proyecto con dos clases y dos acoplamientos, luego un proyecto con tres clases y tres acoplamientos, seguidamente un proyecto con tres clases y cuatro acoplamientos; y finalmente un proyecto con cuatro clases y tres acoplamientos. Para esto se ha definido el proyecto "PruebaFactorAcoplamiento" el cual será modificado acorde a los distintos momentos de la prueba.	
Pasos o secuencia lógica	• Presionar el botón "Calcular métricas" • El sistema despliega un mensaje de estado con el interrogante "¿Son los archivos que desea analizar?" y se presiona "Sí". • Se muestra un mensaje de estado con el mensaje "El análisis ha finalizado" • Observar el resultado de la métrica en la pestaña "Factor de acoplamiento (clases)".	
Resultados	De acuerdo a la fórmula de Factor de Acoplamiento, los resultados deben ser los siguientes:	

esperados	• Para el código fuente con dos clases y dos acoplamientos el resultado de Factor de acoplamiento (CF) debe ser igual a 100% • Para el código fuente con tres clases y tres acoplamientos el resultado de CF debe ser igual a 50% • Para el código fuente con tres clases y cuatro acoplamientos el resultado de CF debe ser de 66,6% • Para el código fuente con cuatro clases y tres acoplamientos el resultado de CF debe ser de 25%
Resultados obtenidos	A continuación se observan los resultados de las cuatro partes de esta prueba. • **Código fuente con dos clases y dos acoplamientos**: JQM3trics arrojó un Factor de acoplamiento de 100% (ver imagen abajo), es decir el resultado esperado coincide con el resultado obtenido, por tanto pasó la primera parte de esta prueba. Para llevarla a cabo se construyó un proyecto llamado "Prueba Factor de Acoplamiento" que consta de dos clases y dos acoplamientos, lo que significa que ambas clases están mutuamente acopladas debido a que el número máximo de dependencias para un proyecto con dos clases es dos.  • **Código fuente con tres clases y tres acoplamientos**: JQM3trics arrojó un Factor de acoplamiento de 50% (ver imagen abajo), es decir el resultado esperado coincide con el resultado obtenido, por tanto pasó la

segunda parte de esta prueba. Para llevarla a cabo se modificó el proyecto llamado “Prueba Factor de Acoplamiento” que ahora consta de tres clases y tres acoplamientos. Por supuesto al reemplazar estos valores en la formula y multiplicar el Factor de Acoplamiento por 100, el resultado es 50%

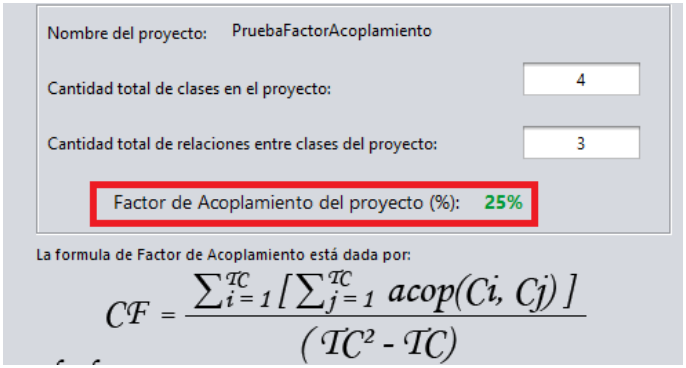


- **Código fuente con tres clases y cuatro acoplamientos**: JQM3trics arrojó un Factor de acoplamiento de 66% redondeando (ver imagen abajo), es decir el resultado esperado coincide con el resultado obtenido, por tanto pasó la tercera parte de esta prueba. Para llevarla a cabo se modificó el proyecto llamado “Prueba Factor de Acoplamiento” que ahora consta de tres clases y cuatro acoplamientos. Por supuesto al reemplazar estos valores en la formula y multiplicar el Factor de Acoplamiento por 100, el resultado es 66%.



<table>
<tr>
<td></td>
<td>

Código fuente con cuatro clases y tres acoplamientos: JQM3trics arrojó un Factor de acoplamiento de 25% (ver imagen abajo), es decir el resultado esperado coincide con el resultado obtenido, por tanto pasó la cuarta y última parte de esta prueba. Para llevarla a cabo se modificó el proyecto llamado “Prueba Factor de Acoplamiento” que ahora está compuesta de cuatro clases y tres acoplamientos. Por supuesto al reemplazar estos valores en la formula y multiplicar el Factor de Acoplamiento por 100, el resultado es 25%.

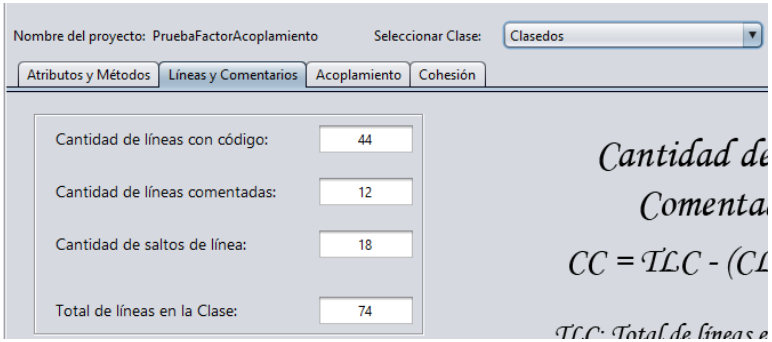
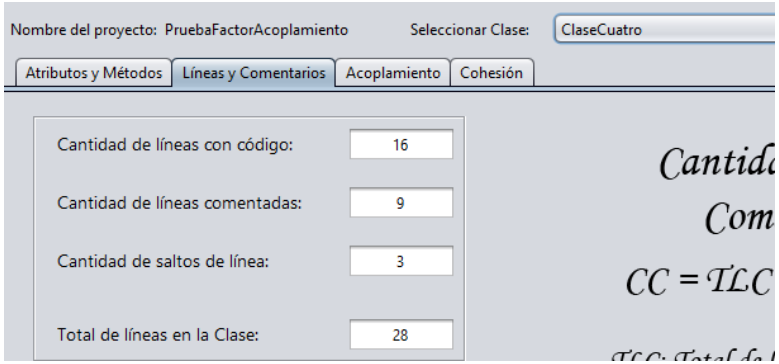


Puesto que JQM3trics pasó los cuatro ítems, esta prueba se cataloga como “Pasada”.
</td>
</tr>
<tr>
<td>Estado de la prueba:</td>
<td>☑ Pasada ☐ Fallida</td>
</tr>
</table>

4.5.2.6 Caso de prueba número 6: Cantidad de líneas de código

Tabla 13. Caso de prueba número 6

Id_CP:3	**Métrica a probar: Cantidad de líneas de código**	Fecha: 26/05/2019 D M A
Nombre del proyecto: JQM3trics		**Plataforma**: aplicación JAVA
Propósito del caso de prueba:	Corroborar que JQM3trics calcule la cantidad de líneas de código de una clase correctamente	
Pre Requisitos	• La aplicación debe estar lanzada o en ejecución. • Debe estar cargado el código fuente llamado “PruebaFactorAcoplamiento”.	
Pasos o secuencia lógica	• Presionar el botón “Calcular métricas” • El sistema despliega un mensaje de estado con el interrogante “¿Son los archivos que desea analizar?” y se presiona “Sí”. • Se muestra un mensaje de estado con el mensaje “El análisis ha finalizado” • Se muestra la métrica LCOM4 en la pestaña “panel de resultados”.	
Resultados esperados	• Mostrar para el archivo Clasedos.java: cantidad total de líneas de la clase= 75, cantidad de líneas de código=44 y líneas de comentario=12. • Mostrar para el archivo ClaseCuatro.java: cantidad total de líneas de la clase= 29, cantidad de líneas de código=16 y líneas de comentario=9.	

Resultados obtenidos	Los resultados obtenidos de las líneas de código y líneas de comentario coinciden con los resultados esperados, sin embargo hay una línea menos en la cantidad total de líneas para ambos casos: • Para el archivo Clasedos.java: cantidad total de líneas de la clase= 74, cantidad de líneas de código=44 y líneas de comentario=12.  • Para el archivo ClaseCuatro.java: cantidad total de líneas de la clase= 28, cantidad de líneas de código=16 y líneas de comentario=9. 
Estado de la prueba:	■ Pasada ☐ Fallida

4.6 Análisis de los resultados

La presente sección inicia con la descripción de los resultados obtenidos de acuerdo a cada objetivo específico establecido. Se finaliza con el análisis de los resultados de las pruebas de rendimiento y caja negra aplicadas a la herramienta. A continuación se observa un resumen de los objetivos específicos cumplidos.

Resultado general (Objetivo general): Aplicando la metodología RUP se desarrolló un componente para el cálculo de métricas a partir de código fuente, con el fin de apoyar el análisis de productos software.

Resultados específicos:

- Se construyó un modelo de negocio que permite tener una visión holística del problema en cuestión.
- Se identificaron los requisitos del componente y con base en ellos se establecieron las funcionalidades y características del mismo.
- Se construyeron los artefactos de diseño adecuados para el componente desarrollado, mediante patrones arquitectónicos basados en los requerimientos funcionales.
- Se implementó el componente software de cálculo de métricas mediante el lenguaje de programación JAVA con el paradigma orientado a objetos.
- Se realizaron pruebas de carga de archivos, de rendimiento y de caja negra. Además se verificó el correcto análisis de productos software.

4.6.1 Modelo de negocio

Realizando un análisis de este ítem, se puede decir que previamente se llevó a cabo una revisión rigurosa de la literatura con el fin de indagar el estado actual de la temática referente a cálculo de métricas. Comparando los resultados obtenidos en esta investigación con la literatura previa, se puede afirmar que coinciden en gran parte pues la característica común es el análisis sobre el código fuente, lo cual se logró en este proyecto. Sin embargo, existen ciertas diferencias marcadas respecto a las herramientas identificadas en la literatura y la principal es que el componente implementado en este proyecto muestra una explicación de las fórmulas utilizadas para calcular las métricas establecidas.

4.6.2 Requisitos

Teniendo claro el estado actual de la literatura en materia de cálculo de métricas sobre código fuente y habiendo revisado herramientas de análisis propuestas en determinados estudios, se trazó un panorama claro que permitió la identificación del requisito primordial de la herramienta desarrollada: calcular métricas sobre el código fuente JAVA. Este requisito coincide con la funcionalidad principal de las herramientas consultadas en la literatura. Además de la muestra de resultados del cálculo, se agregó como factor diferencial la capacidad de explicar qué formulas se emplean para el cálculo de las métricas de modo que el Ingeniero de software pueda conocer cuáles son los parámetros utilizados para la obtención de los resultados de las mismas.

4.6.3 Modelo de Diseño

La identificación de los requisitos permitió continuar con el siguiente paso: definir la arquitectura de la herramienta. Una vez hecho esto se procedió con la construcción de los distintos artefactos de diseño como la Vista de Escenarios, Vista Lógica y Vista de procesos. La Vista de escenarios fue orientada al mundo real y a nivel de software ya que habrá casos en que el Ingeniero de Software no cuente con una herramienta para apoyar análisis de código fuente y debe hacerlo manualmente.

4.6.4 Implementación de la herramienta

Se puede decir que se cumplió a cabalidad con la implementación de la herramienta propuesta, esto se evidencia en la capacidad de JQM3trics de calcular todas las métricas que se propusieron. La coherencia entre los resultados obtenidos y esperados de las pruebas también permite corroborar el cumplimiento de la implementación.

A continuación se puede observar una tabla que relaciona las métricas propuestas, métricas logradas y el ámbito de aplicación de cada métrica. El ámbito es el nivel de aplicación de la métrica, es decir si es una métrica que se calcula para el sistema entero o por cada clase. No se debe confundir ámbito con módulo de medida. Por ejemplo, para calcular el factor de acoplamiento del sistema (ámbito), JQM3trics usa como módulo de medida a las clases. En ese orden de ideas es correcto decir que el factor de acoplamiento del sistema se calcula a partir de las clases del mismo.

Tabla 14. Equivalencia entre métricas propuestas y métricas implementadas

Métrica propuesta	**Métrica implementada**	**Ámbito de la métrica**
Acoplamiento entre clases	Inestabilidad	Se calculan para cada clase
	Acoplamiento Eferente	
	Acoplamiento Aferente	
	Factor de acoplamiento	Se calcula para el sistema
Falta de cohesión en los métodos	Falta de cohesión en métodos (LCOM4)	Se calcula para cada clase
Número de líneas de código	Cantidad total de líneas (líneas de código + saltos de línea+ comentarios que no compartan línea con líneas de código)	Se calcula para el sistema y para cada clase
	Cantidad real de líneas de código[11]	Se calcula para el sistema y cada clase
	Cantidad de líneas de solo comentarios	Se calcula para el sistema y para cada clase
	Cantidad de saltos de línea	Para el sistema y para cada clase
Número de clases	Cantidad total de clases	Se calcula para el sistema
Número de métodos por clase	Cantidad de métodos	Se calcula para cada clase
(No propuesta)	Cantidad de atributos	Se calcula para cada clase
Total propuestas: 5	**Total implementadas: 12**	

[11] Cantidad de líneas de código sin incluir líneas de comentario puras, saltos de línea y líneas en blanco.

4.6.5 Pruebas

En este capítulo se realiza el análisis de las pruebas de carga y respuesta, seguidamente se muestra el análisis de resultados para las pruebas de caja negra aplicadas a las métricas calculadas por JQM3trics.

4.6.5.1 Análisis de los resultados de las pruebas realizadas en el capítulo 4.5

Los resultados arrojados por las pruebas de carga de archivos .java y de respuesta permiten realizar varias aseveraciones, pero antes de ello es pertinente observar en la siguiente tabla un resumen de las pruebas de tiempos de respuesta en la carga de archivos y el cálculo de las métricas:

Tabla 15. Resumen de las pruebas

Proyecto / **Items**	**JQM3trics**	**RTTOOL**	**jHotDraw**	**MASU**
Tamaño en MB	0.246	0.421	1.98	3.43
Tiempo de carga (s)	0.0101	0.05266	0.552	1.784
Tiempo de respuesta (s)	1.91	4.47	7.106	17.773
Número de archivos JAVA	18	49	310	515
Número de líneas de código (Sin Comentarios ni saltos de línea)	4374	8797	39435	78347
Factor de acoplamiento (%)	13	5	1	2

De acuerdo a la tabla 16 se puede decir que entre mayor sea el tamaño de almacenamiento del proyecto, más tiempo le va a tomar a la herramienta cargar el proyecto y más tiempo le va a tomar realizar el análisis al código fuente.

El tamaño de almacenamiento (definido en la tabla 16 como Tamaño en MB) va a estar determinado en primer lugar por la cantidad de líneas de código. En segundo lugar, el número de archivos JAVA tiene cierta incidencia pero no es determinante puesto que, por ejemplo, puede haber 50 archivos JAVA que contengan una sola línea de código cada uno, en ese caso el tamaño en MB no sería muy alto. Por tal motivo se concluye que el tamaño en MB de un proyecto que

contiene solo archivos .java va a estar determinado por la cantidad total de líneas de código que tenga dicho proyecto.

En ese sentido, puesto que el tamaño en MB está determinado por la cantidad de líneas de código, se puede decir que la cantidad de líneas de código influye directamente en los tiempos de carga y de calcular métricas.

Lo anterior también se puede definir de la siguiente forma: existe una relación directamente proporcional lineal entre la cantidad de líneas de código y el tamaño en MB de un proyecto de únicamente código fuente JAVA. Existe una relación directamente proporcional lineal entre el tamaño en MB de un determinado proyecto y los tiempos de carga y de calcular métricas del mismo. La relación explicada se puede observar en las gráficas de las figuras 38 y 39:

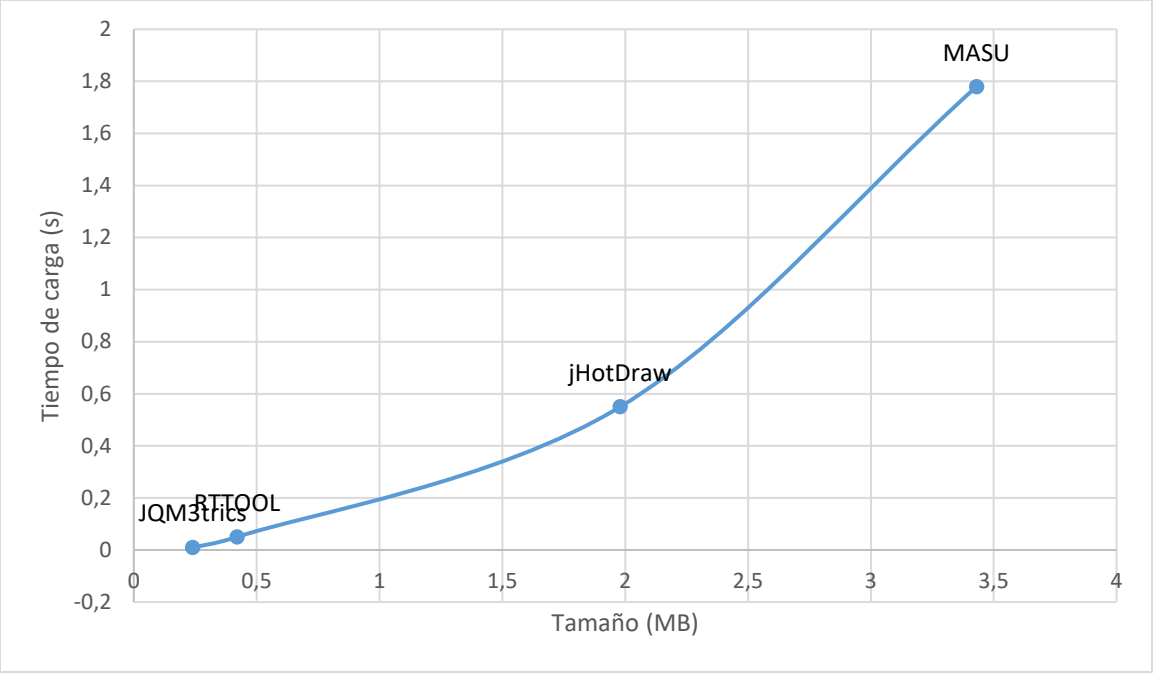


Figura 38. Relación directamente proporcional entre tamaño en MB y tiempo de carga (s) de los proyectos de la tabla 9.

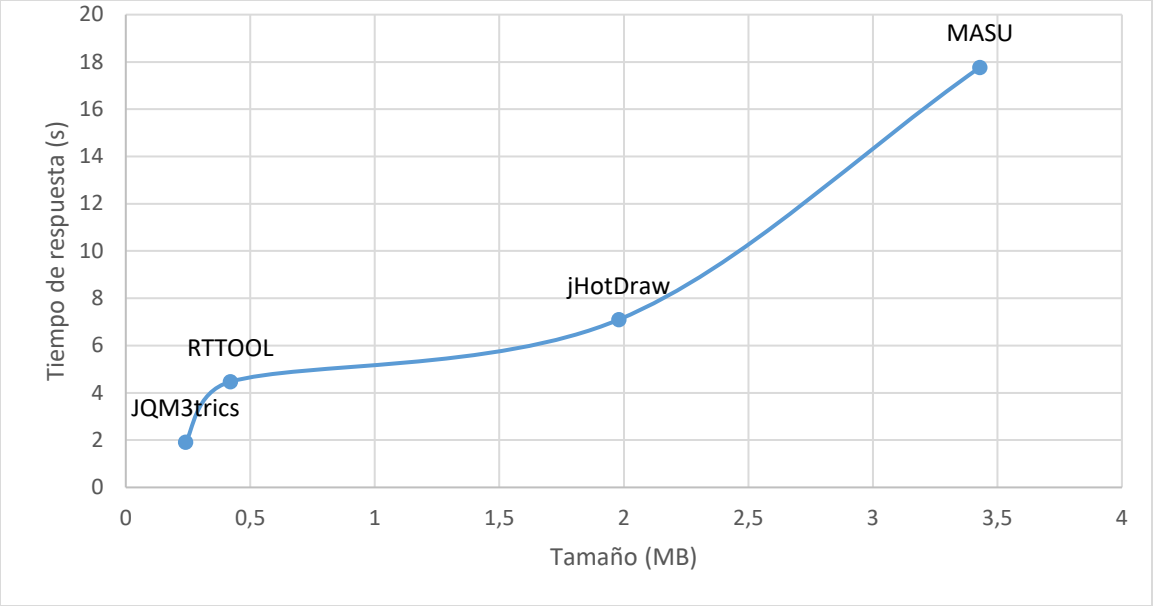


Figura 39. Relación directamente proporcional entre tamaño en MB y tiempo de respuesta (s) de los proyectos de la tabla 9.

4.6.5.2 Análisis de prueba realizada a la Cantidad de líneas de código

A continuación se observa el análisis de la prueba aplicada en el capítulo 4.5.2.6 Caso de prueba número 6: Cantidad de líneas de código.

Inicialmente es pertinente saber que, para JQM3trics, la cantidad total de líneas de un archivo java está conformada por:

- Líneas de código
- Líneas de comentario
- Líneas con saltos de línea o espacios en blanco

En esta prueba se calcularon las líneas totales, las líneas de código y las líneas con comentario para los archivos ClaseCuatro.java y Clasedos.java

En los resultados obtenidos de las líneas totales se obtuvo una línea de código menos, esto ocurre debido a que hay archivos java cuya última línea es null, por tanto dicha línea no es tenida en cuenta en el cálculo.

Este comportamiento inesperado no ocurre siempre, aquellos archivos java que no terminen en null tendrán el cálculo de la totalidad de sus líneas. Por ejemplo, se realizó la misma prueba a un archivo java (del mismo proyecto) llamado “ClaseTres.java” y se obtuvo que el resultado obtenido coincide con el esperado, las líneas totales son 56. Obsérvese la figura 40 y la figura 41 seguidamente.

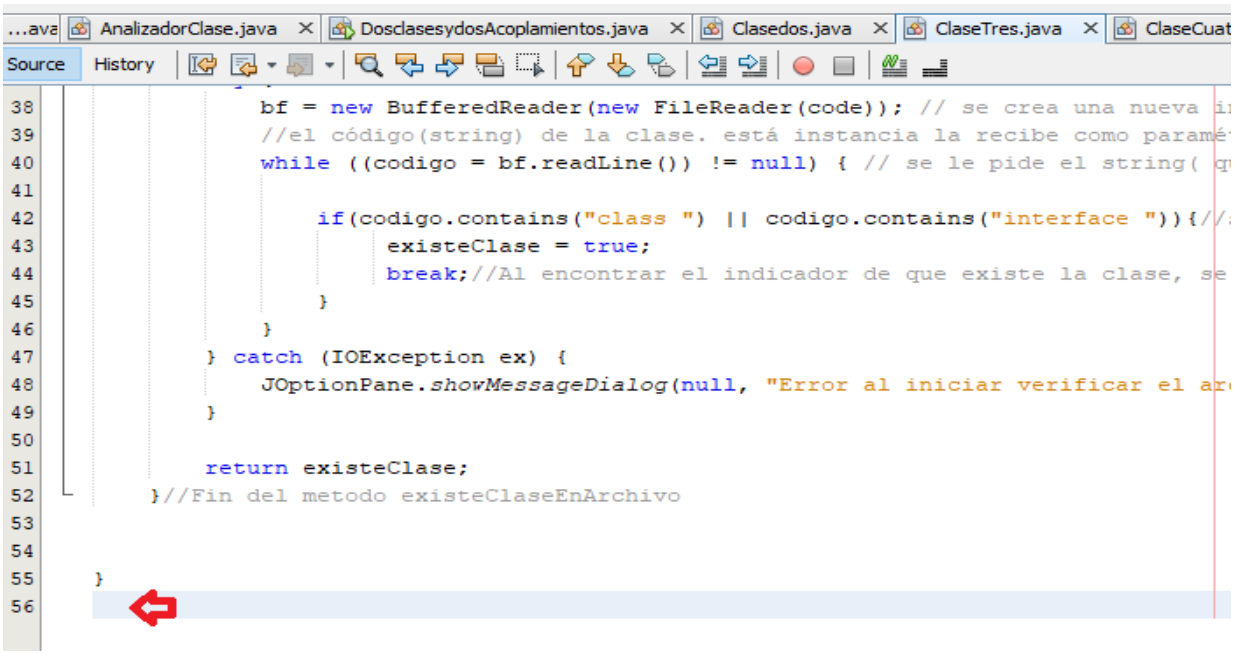


Figura 40. Archivo java con 56 líneas visto desde Netbeans IDE 8.2

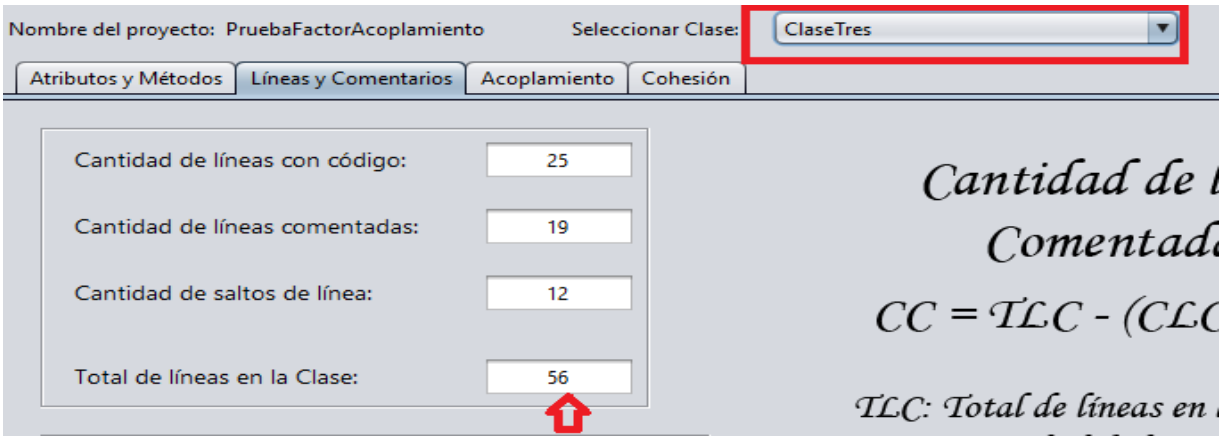


Figura 41. Resultado arrojado por JQMetrics, 56 líneas

Esta última prueba permite corroborar que NO en todos los casos JQM3trics arroja una línea de menos. Esto ocurre, como ya se mencionó, en aquellos archivos cuya última línea sea null. Por tratarse de un resultado inesperado, se ha optado por establecer que el cálculo de las líneas totales de una clase puede tener un margen de error de ±1 línea.

4.6.5.3 Análisis de la prueba de Factor de acoplamiento (clases)

A continuación se aprecia el análisis del caso de prueba número 5 aplicado en el capítulo 4.5.2.5 Caso de prueba número 5: Factor de acoplamiento a partir de clases

Como bien se sabe, esta prueba se dividió en cuatro partes:

- Para el código fuente con dos clases y dos acoplamientos el resultado de Factor de Acoplamiento (CF) fue de 100%
- Para el código fuente con tres clases y tres acoplamientos el resultado de CF fue de 50%
- Para el código fuente con tres clases y cuatro acoplamientos el resultado de CF fue de 66%
- Para el código fuente con cuatro clases y tres acoplamientos el resultado de CF fue de 25%

De los resultados anteriores se puede afirmar lo siguiente:

i) Un proyecto tiene su valor de Factor de Acoplamiento en 100% cuando tiene el número máximo de acoplamientos posibles (ver Figura 42) de acuerdo a la cantidad de clases del mismo, en este caso son dos clases. Por supuesto este valor indica un problema grave de acoplamiento y hacer un cambio mínimo puede impactar en todo el sistema.

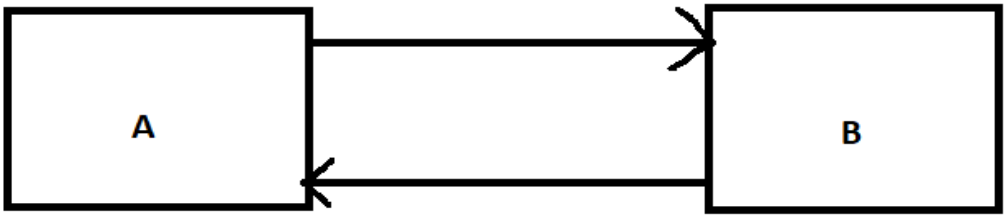


Figura 42. Ejemplo de un sistema que consta de dos paquetes A y B mutuamente dependientes

ii) A medida que aumente la cantidad de acoplamientos con otras clases y la cantidad de clases permanezca igual, el valor de Factor de Acoplamiento aumentará.

iii) Mientras aumente la cantidad de clases y la cantidad de acoplamientos permanezca igual, el Factor de Acoplamiento disminuirá. Es decir, entre más amplio es un sistema, el Factor de Acoplamiento tenderá a ser más pequeño.

De forma general se puede concluir que el factor de acoplamiento tiene una correlación positiva muy alta con todas las medidas de calidad. Por lo tanto, a medida que aumenta el acoplamiento también se espera que aumente la densidad de defectos y el esfuerzo necesario para realizar un cambio. Lo anterior muestra que el acoplamiento en los sistemas de software tiene un fuerte impacto negativo en la calidad del software y, por lo tanto, debe mantenerse al mínimo requerido durante la fase de diseño. Es deseable que las clases tengan la menor cantidad de relaciones posibles con otras clases porque las relaciones de acoplamiento aumentan la complejidad, reducen la encapsulación y la posible reutilización; limitan la comprensibilidad y la capacidad de mantenimiento.

Comparación entre Factor de acoplamiento (clases) y Factor de acoplamiento (paquetes)

Primeramente es importante recordar que JQM3trics calcula esta métrica a partir de clases y a partir de paquetes, sin embargo el Factor de acoplamiento (CF a partir de ahora) a partir de clases es la métrica principal ya que por utilizar un módulo de medida más pequeño el valor de la métrica es mucho más preciso que el CF a partir de paquetes.

Antes de profundizar es pertinente saber las siguientes consideraciones para el cálculo del CF (paquetes):

- Se tienen en cuenta las importaciones para el calculo
- Se asume que todo paquete importado es usado por la clase que lo importa.

Ahora conviene colocarse en contexto con lo que ocurre entre el CF a partir de paquetes y el CF a partir de clases. Para ello, se llevó a cabo una prueba analizando un proyecto llamado “PruebaFactorAcoplamiento” que consta de tres paquetes, cuatro clases y tres relaciones **entre paquetes.**

En la figura 43 se puede observar cómo están distribuidas las clases en los paquetes, nótese que el paquete 1 contiene dos clases.

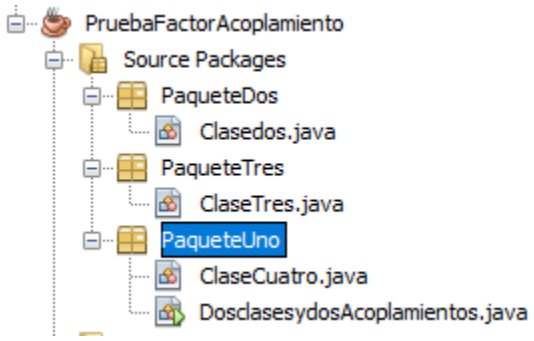


Figura 43. Visualización de un proyecto de prueba

El valor de CF a partir de paquetes arrojado para este proyecto fue de 50% (ver Figura 44), es decir un nivel medio.

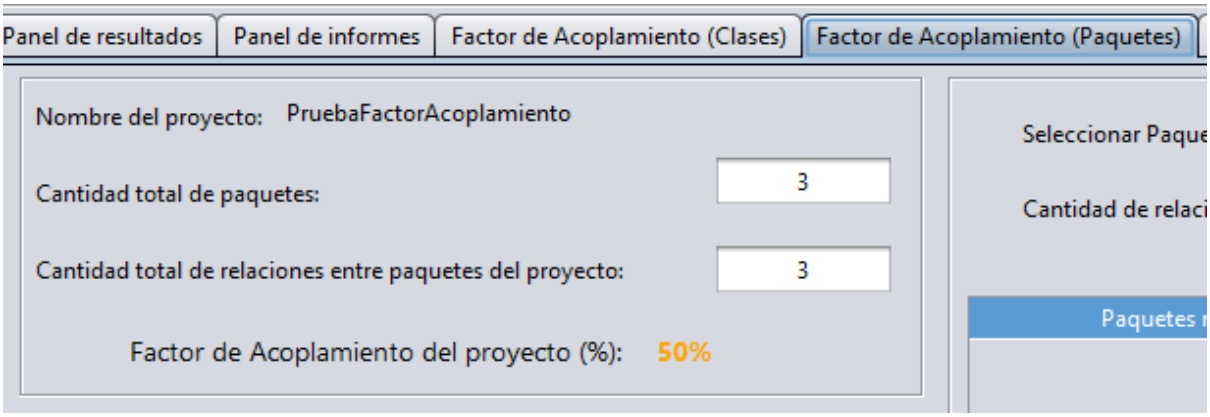


Figura 44. Factor de acoplamiento (paquetes) del proyecto PruebaFactorAcoplamiento

En cambio, el valor arrojado de CF a partir de clases fue de 33% (ver Figura 45).

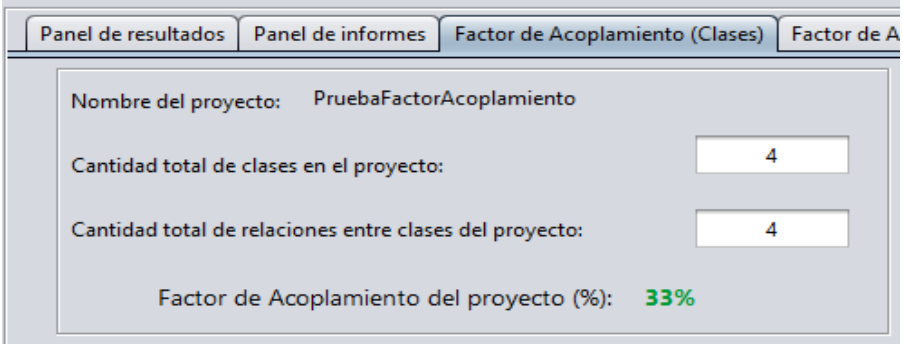


Figura 45. Factor de acoplamiento (Clases) del proyecto PruebaFactorAcoplamiento

El resultado de CF a partir de clases es más preciso que el CF paquetes por el hecho de que en el primero se tiene en cuenta los acoplamientos individuales que cada clase tiene con el resto sin importar si pertenecen o no al mismo paquete.

En cuanto al CF a partir de paquetes, por ejemplo, si la clase "Clasedos" (perteneciente al paquete "PaqueteDos") tiene acoplamientos con las clases "ClaseCuatro" y "DosclasesydosAcoplamientos" pertenecientes al paquete "PaqueteUno" solamente se tendrá en cuenta una relación de acoplamiento en sentido PaqueteDos–PaqueteUno ya que el módulo de referencia o medida es el paquete, no la clase.

Por lo anterior se concluye que los valores de CF a partir de clases y CF a partir de paquetes no necesariamente deben coincidir ya que tienen módulos de medida distintos.

4.6.5.4 Análisis de la prueba realizada a la Falta de cohesión (LCOM4)

A continuación se muestra el análisis del caso de prueba número 4 aplicado en el capítulo 4.5.2.4 Caso de prueba número 4: cálculo de LCOM4

Se sabe que valores de LCOM>=2 sugieren que la clase analizada no es cohesiva, es decir tiene problemas ya que lo ideal es que haya un único conjunto en cada clase. Hay un solo conjunto cuando los métodos y atributos de una clase están todos relacionados directa o indirectamente.

La base teórica encontrada en la revisión de la literatura sugiere que clases con valores de LCOM4 mayores o iguales a 2 deben ser divididas en clases más pequeñas. Obsérvese la clase Clasedos.java cuyo valor de LCOM4 es 2 (ver Figura 46), es decir tiene dos conjuntos, lo cual significa que no están relacionados. Aplicando lo sugerido por (Hitz & Montazeri, 1995), esta

clase debe ser dividida en dos clases de modo que cada una tenga un único conjunto y por ende sus valores de LCOM4 sean igual a 1.

Nombre de la Clase	Cantidad de Lineas (Sin comentarios)	Cantidad de atributos	Cantidad de métodos	Valor de Cohesión (LCOM4)	Acoplamiento Eferente	Acoplamiento Aferente	Valor de Inestabilidad
ClaseCuatro	16	0	1	1	0	1	0.0
Clasedos	59	2	3	2	2	1	0.67
DosclasesydosAcoplament...	42	4	4	1	2	1	0.67

Figura 46. Valor de LCOM4 (en rojo) para la clase "Clasedos" del proyecto PruebaFactorAcoplamiento

Observando la lógica de la clase (ver Figura 47), es fácil notar los dos conjuntos diferentes que no tienen atributos en común. Cada uno accede a uno de los atributos pero no acceden a un mismo atributo ya sea directamente o mediante un método:

```
20  public class Clasedos {
21
22      public DosclasesydosAcoplamientos dos;
23
24
25      public ClaseCuatro cuatro;
26
27      private int metodoDos (){
28      int cantidad;
29
30        cantidad=  dos.contador("", "");
31
32        return cantidad;
33      }
34
35        private int metodoCuatro (){
36      int conteo;
37
38        conteo=  cuatro.cuenta("", "");
39
40        return conteo;
41      }
```

Figura 47. Componentes separados en una clase

Luego, se modificó la clase Clasedos con el fin de disminuir el valor de cohesión. La sección de código resultante se observa en la figura 48.

```
20  public class Clasedos {
21
22      public DosclasesydosAcoplamientos dos;
23
24
25      public ClaseCuatro cuatro;
26
27      private int metodoDos (){
28      int cantidad;
29
30        cantidad=  dos.contador("", "");
31
32        return cantidad;
33      }
34
35       private int metodoCuatro (){
36      int conteo;
37
38        conteo=  cuatro.cuenta("", "") + dos.contador("Hola", "Prueba");
39
40        return conteo;
41      }
```

Figura 48. Clasedos modificada con el fin de que haya un unico componente y disminuir LCOM4

Como se puede apreciar, ahora ambos componentes (métodos) están conectados por medio del atributo “dos”.

Luego de modificar la clase se analizó nuevamente y LCOM4 arrojó un valor de 1 (ver Figura 49). A partir de este hallazgo se puede concluir que no necesariamente se debe dividir una clase cuando tenga varios conjuntos aislados, sino que inicialmente se debe considerar modificar la estructura de los métodos procurando que no se afecte el flujo lógico de la clase. Si se llega a ver comprometida la funcionalidad lógica de la clase es recomendable no modificarla y proceder con la división de la misma.

Panel de resultados | Panel de informes | Factor de Acoplamiento (Clases) | Factor de Acoplamiento (Paquetes) | Detalles de carga | Consola

Nombre de la Clase	Cantidad de Lineas (Sin comentarios)	Cantidad de atributos	Cantidad de métodos	Valor de Cohesión (LCOM4)	Acoplamiento Eferente	Acoplamiento Aferente	Valor de Inestabilidad
ClaseCuatro	17	0	1	1	0	1	0.0
Clasedos	61	2	3	1	2	1	0.67
DosclasesydosAcoplamient...	42	4	4	1	2	1	0.67

Figura 49. Valor de LCOM4 luego de modificar a nivel de código la clase Clasedos

4.6.5.5 Comparación entre resultados obtenidos e investigaciones presentadas en el estado del arte

A continuación se puede observar un cuadro comparativo que destaca las similitudes y diferencias entre los resultados obtenidos en esta investigación y las investigaciones presentadas en el estado del arte. La diferencia más marcada es que en esta investigación se muestra, a través de la IU desarrollada para invocar el componente, la explicación de las fórmulas utilizadas para calcular cada métrica. Las investigaciones presentadas en el estado del arte no tienen la característica mencionada.

Tabla 16. Similitudes y diferencias entre investigaciones presentadas en estado del arte e investigación descrita en este documento.

Investigación	Diferencias	Similitudes
Boehm, B. W., Brown, J. R., & Lipow, M. (1976). Quantitative evaluation of software quality	• En su investigación no se menciona alguna herramienta software para validar la propuesta. • La investigación es meramente conceptual y analítica. La investigación descrita en este documento es aplicada y basada en un marco teórico, es decir no se proponen métricas sino que se aplican las ya propuestas en estudios previos.	• Se muestra un marco conceptual sobre aspectos clave en el análisis de las características de la calidad del software. • Se llega a la conclusión de que una mayor atención a las características de la calidad del software puede suponer ahorro de recursos durante el ciclo de vida del software. • En ambas investigaciones se definen métricas de evaluación de calidad de software. • Ambas investigaciones se soportan en estudios previos. Esto se valida al ver las referencias.
Coupal, D., & Robillard, P. N. (1990). Factor analysis of	• Investigación no disponible a la comunidad. En cambio la investigación descrita en este	Puesto que la investigación no está disponible a la comunidad en general, no

source code metrics	documento está disponible a la comunidad hispanohablante que tenga conocimientos en ingeniería de software, específicamente en el área de cálculo de métricas	se obtuvieron detalles de la misma con el fin de encontrar similitudes.
Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design	• En su investigación, Chidamber & Kemerer, desarrollaron las métricas y las evaluaron. En la investigación tratada en este documento se implementaron métricas propuestas en estudios previos. • En su investigación no se menciona que la herramienta que propusieron muestre explicación del cálculo de cada métrica. En la herramienta implementada en esta investigación se muestra un informe de acuerdo a los valores obtenidos de cada métrica • La herramienta implementada en su investigación está orientada a sugerir, a los gerentes de organizaciones, formas de emplear las métricas para la mejora de los procesos de desarrollo.	• Ambas investigaciones son de tipo analítica y aplicada. • En ambas investigaciones se desarrollaron y se implementaron herramientas para validar lo propuesto. • Las investigaciones están enfocadas en métricas orientadas a objetos como Lack of Cohesion in Methods y Coupling Between Objects • Ambas investigaciones se soportan en estudios previos. Esto se valida en las referencias.
Fenton, N. E., & Neil, M. (Mayo de 2000). Software	• No implementaron herramienta software.	• Ambas investigaciones están

metrics: roadmap	• Su investigación concluye con una propuesta. La investigación descrita en este documento concluye con resultados de la implementación y pruebas de la herramienta software. • Su investigación es de tipo analítica; la investigación de este documento, además de analítica, es aplicada.	enfocadas en métricas de software • Ambas investigaciones son de tipo analítico. • Ambas investigaciones se soportan en estudios previos. Esto se valida en las referencias.
Raemaekers, S., van Deursen, A., & Visser, J. (2012). Measuring software library stability through historical version analysis	• En su investigación proponen métricas, en la investigación de este documento no, puesto que no es el enfoque de la misma. Esta investigación se basa en métricas propuestas en estudios previos. • En su investigación no se menciona que la herramienta que propusieron muestre explicación del cálculo de las métricas que propusieron. En la herramienta implementada en esta investigación se muestra explicación de los valores obtenidos de cada métrica.	• Ambas investigaciones son de tipo analítica y aplicada. • Ambas investigaciones implementan herramientas software para corroborar supuestos (propios o basados en investigaciones previas). • Ambas investigaciones se soportan en estudios previos. Esto se valida en las referencias.
Higo, Y., Saitoh, A., Yamada, G., Miyake, T., Kusumoto, S., & Inoue, K. (Noviembre de 2011). A pluggable tool for	• Desarrollaron una herramienta que permite agregar nuevas métricas orientadas a objetos según el requerimiento del usuario. En esta investigación se desarrolló una herramienta que	• Ambas investigaciones son de tipo analítica y aplicada. • Ambas investigaciones implementan herramientas

measuring software metrics from source code.	calcula las métricas y que muestra la interpretación de los resultados.	software para corroborar supuestos (propios o basados en investigaciones previas). • Ambas investigaciones se soportan en estudios previos. Esto se valida en las referencias • Ambas investigaciones utilizan métricas orientadas a objetos
Oliveira, P., Valente, M. T., & Lima, F. P. (Febrero de 2014). Extracting relative thresholds for source code metrics.	• En su investigación no desarrollaron una herramienta, la propusieron. En esta investigación se implementó una herramienta.	• Ambas investigaciones son de tipo analítica. • Ambas investigaciones se soportan en estudios previos. Esto se valida en las referencias.
Oliveira, P., Lima, F. P., Valente, M. T., & Serebrenik, A. (Septiembre de 2014). RTTool: A tool for extracting relative thresholds for source code metrics.	• La herramienta propuesta por ellos no calcula métricas sino que a partir de métricas ya calculadas, extrae umbrales. La investigación descrita en este documento calcula métricas orientadas a objetos a partir de código fuente.	• Ambas investigaciones son de tipo analítica y aplicada. • Ambas investigaciones implementan herramientas software para corroborar supuestos (propios o basados en investigaciones previas). • Ambas investigaciones se

		soportan en estudios previos. Esto se valida en las referencias • Ambas investigaciones utilizan métricas orientadas a objetos. • Ambas investigaciones clasifican sus resultados de acuerdo a umbrales o límites.
Metaute, P., & Serna, A. (2016). Diagnóstico sobre la apropiación de las métricas en medianas empresas de software de la ciudad de Medellín.	• El estudio se trata de un diagnóstico de la apropiación y utilización de las métricas en medianas empresas. • No se propone una herramienta. • El estudio se enfoca en verificar qué tanto las empresas usan métricas para apoyar sus procesos: por ejemplo, programación.	• Ambos estudios tienen como parte fundamental la contribución a la academia.

Los resultados obtenidos en la métrica Factor de Acoplamiento coinciden con el estudio titulado “Evaluating the Impact of Object-Oriented Design on Software Quality” de Abreu & Melo ya que la fórmula que se extrajo de dicho artículo y fue implementada en JQMetrics, arroja los resultados bajo las consideraciones establecidas en la base teórica. Además, en dicho estudio también desarrollan una herramienta para calcular métricas orientadas a objetos a distintos proyectos y los resultados los muestran en una tabla. Entre las métricas mostradas está el Factor de Acoplamiento. Similarmente en este documento también se calcularon métricas orientadas a objetos a cuatro proyectos distintos cuyos resultados fueron relacionados en una tabla.

5. CONCLUSIONES Y RECOMENDACIONES

Habiendo desarrollado un componente de análisis estático para calcular métricas sobre código fuente, se identificó lo siguiente: el grado de acoplamiento, la inestabilidad y la falta de cohesión son métricas orientadas a objetos, es decir se recomienda calcular las métricas sobre proyectos de código fuente que hayan sido codificados bajo el paradigma orientado a objetos. En la literatura previa consultada, las métricas propuestas e implementadas son en su totalidad orientadas a objetos.

La metodología usada para calcular las métricas implementadas en esta investigación está basada en fórmulas y métodos propuestos en los estudios existentes sobre métricas de código fuente. Para el cálculo de la inestabilidad y los tipos de acoplamiento se tomó como referencia la teoría de métricas a nivel de componentes explicada por Robert C. Martín en su libro *Agile Principles, Patterns, and Practices in C#.* Luego de realizar varias pruebas (tanto manuales como con la herramienta) y observar los resultados de las métricas por cada clase, se puede decir que los valores son coherentes con las pruebas manuales, en ese orden de ideas los resultados coinciden con el procedimiento explicado en el libro.

Por otra parte, el siguiente párrafo describe la respuesta a la pregunta: ¿Cómo realizar el cálculo de métricas sobre código fuente para apoyar el análisis de productos software, de modo que se brinde información detallada sobre sus atributos?

En términos generales, la pregunta fue respondida así: se realizó una revisión exhaustiva de la literatura disponible, a partir del análisis de la misma se construyó el Modelo de Dominio el cual permitió identificar el requisito principal (Calcular Métricas) y las restricciones que debía tener el sistema. Teniendo lo anterior definido se pudo construir el Modelo de Diseño, a partir del cual se inició la Implementación de la herramienta. Una vez finalizada la codificación, se efectuaron sobre la herramienta ciertas pruebas cuyos resultados fueron los esperados. Finalmente se agregó la interpretación de los resultados del cálculo de métricas sobre un determinado proyecto de código fuente para dar información detallada sobre el estado de algunos atributos del producto como son:

Cantidad de líneas de código, numero de métodos por clase, número de atributos por clase, cantidad de clases, acoplamiento entre clases (Aferente y Eferente), factor de acoplamiento, inestabilidad de las clases y falta de cohesión en las clases.

El procedimiento para llegar a responder la pregunta de investigación se explica con más detalle a continuación:

Se construyó el modelo de negocio basado en los hallazgos encontrados durante la revisión de la literatura existente. La revisión minuciosa del estado del arte permitió describir el estado de la literatura actual y construir el modelo de dominio que refleja las entidades identificadas a nivel del mundo real. La revisión del estado actual de la literatura y el Modelo de Dominio permitieron identificar el requisito fundamental que es Calcular Métricas de código fuente JAVA, establecer las restricciones de la aplicación y construir el documento SRS (Software Requirements Specifications) Especificación de Requerimientos del Software.

A través de este documento también se evidencia la coherencia entre los requisitos y la construcción del Modelo de Diseño. Inicialmente a partir de las restricciones se definió la arquitectura del Componente y el comportamiento de la Interfaz de Usuario. Del mismo modo la arquitectura se ve reflejada en la Vista Lógica (Diagrama de Componentes y de Clases). Por otra parte, se describió el modelo de implementación, el cual muestra la interacción de la aplicación con el entorno de ejecución y el Sistema Operativo donde se aloja la misma.

Ya teniendo el Modelo de Diseño completo se inició con la codificación de la herramienta utilizando el entorno de trabajo IDE Netbeans 8.0 y 8.2. Se implementó el cálculo de la cantidad de líneas de código de cada archivo .java manipulando su contenido, para ello se utilizaron autómatas (expresiones regulares) de modo que se pudiera diferenciar las líneas de código de las líneas de comentario. Las líneas de código se diferencian una de otra por medio de un autómata cuyo criterio de diferencia es el carácter “;” (punto y coma). Para el cálculo de la cantidad de clases, métodos, atributos, Acoplamiento entre clases, Falta de Cohesión y Factor de Acoplamiento también se utilizaron autómatas a nivel de código.

Finalmente se efectuaron pruebas de rendimiento y pruebas unitarias de caja negra. Las primeras permitieron corroborar que la aplicación responde acertadamente a proyectos con una cantidad considerable de líneas de código, por ejemplo se analizó un proyecto llamado JHotDraw que

consta de 39435 líneas de código y el tiempo de respuesta fue de nueve segundos aproximadamente. También se realizaron pruebas unitarias de caja negra para verificar que, de acuerdo a una entrada establecida, el sistema respondiese con una salida esperada. Se probaron tres casos (o módulos) principales: carga de archivos de código fuente .java, restricción de archivos diferentes al formato .java y cálculo de métricas. La aplicación pasó los tres casos de prueba satisfactoriamente.

Respecto a la **Falta de Cohesión** es bien sabido que una clase con valor de LCOM>=2 indica que la clase tiene un problema, pudiera ser una clase mal diseñada. Sin embargo no necesariamente sugiere que la clase deba dividirse en otras más pequeñas puesto que existe la posibilidad de modificar y/o adecuar la estructura de los métodos de modo que se disminuya la cohesión sin afectar el flujo lógico de la clase. En este caso se deja a consideración del desarrollador o analista de pruebas si desea dividir la clase o modificarla. Es importante tener especial cuidado con las modificaciones que se realicen a una determinada clase, si estas llegan a afectar la lógica de la misma lo recomendable es dividirla en otras clases.

De acuerdo a los resultados de **Factor de Acoplamiento** (CF) se puede concluir que esta métrica tiene una correlación positiva muy alta con las métricas de Falta de cohesión e inestabilidad. Por lo tanto, a medida que aumenta el acoplamiento entre clases, también se espera que aumente la densidad de defectos y el esfuerzo requerido para realizar un cambio. En ese orden de ideas, un acoplamiento alto en los sistemas tiene un fuerte impacto negativo en la calidad del software y, por lo tanto, debe mantenerse al mínimo requerido durante la etapa de diseño. Es deseable que las clases tengan la menor cantidad de relaciones posibles con otras clases porque las relaciones de acoplamiento aumentan la complejidad, reducen la encapsulación y la posible reutilización; limitan la comprensibilidad y la capacidad de mantenimiento.

JQM3trics calcula el Factor de Acoplamiento (CF) de un proyecto de código fuente Java de dos formas: a partir de paquetes y a partir de clases. El valor de CF (paquetes) se obtiene a partir de las importaciones que haya en cada clase, es decir se asume que las importaciones son utilizadas. Esto significa que si hay una línea como la siguiente “Import *paquete_del_proyecto”*, JQM3trics la considera un acoplamiento independientemente si la importación es usada o no. En cambio, el CF (clases) realiza un análisis mucho más exhaustivo del código fuente de cada archivo java para

determinar exactamente la cantidad de acoplamientos con otras clases. Por lo anterior se concluye que:

i. Es normal que los valores de CF (paquetes) y CF (clases) difieran.
ii. El valor de CF (clases) es mucho más exacto pues el análisis se realiza sobre la unidad modular más pequeña para la JQM3trics: un archivo java.
iii. Respecto a CF (paquetes), los paquetes importados y no usados pueden causar que se muestre un valor de CF diferente al real. Por ello se recomienda al usuario eliminar las importaciones que no se usen ya que esto representa una mala práctica de codificación.

Es pertinente aclarar que las clases internas (archivos java con más de una clase) no son tenidas en cuenta a la hora de calcular la cantidad de clases y tampoco se calculan métricas específicamente para ese tipo de clases. Esto debido a que considerarlas afectaría el cálculo de la métrica Falta de Cohesión en Métodos (métrica a nivel de clase) ocasionando inconsistencias en el resultado final. Un ejemplo de archivos .java con clases internas es el proyecto de código fuente MASU (ver capítulo 4.5.1.3 Caso de prueba 3). Para realizar calcular la falta de cohesión en métodos e inestabilidad de ficheros .java que contengan clases internas, JQM3trics solo tiene en cuenta la clase principal.

Similarmente, para el cálculo de la falta de cohesión en métodos no se tienen en cuenta procedimientos (métodos) vacíos. Los métodos vacíos tienden a aumentar el valor de LCOM4 ya que no acceden a ninguna variable u otros métodos. En ese orden de ideas, una clase cohesiva con procedimientos vacíos tendría un alto LCOM4. También se ignoran los constructores y destructores. Los constructores y destructores frecuentemente establecen y borran todas las variables en la clase, haciendo que todos los métodos se conecten a través de estas variables, lo que aumenta la cohesión artificialmente (Aivosto Oy, s.f.).

Los resultados obtenidos en esta investigación coinciden con la base teórica establecida en la literatura previa. Un indicador para verificar lo mencionado anteriormente es revisar si las pruebas fueron satisfactorias, evidentemente las pruebas realizadas a las distintas métricas que calcula JQM3trics fueron exitosas. Lo cual quiere decir que se cumple a cabalidad con la implementación de métricas como Factor de Acoplamiento cuya base teórica fue extraída del estudio “Evaluating the Impact of Object-Oriented Design on Software Quality” de Abreu & Melo; y complementada por la base teorica establecida en los artículos: “An Overview of Object-

Oriented Design Metrics", "A Quality Based Novel Subclass Coupling Factor Metric for Evaluating Software" y "Evaluation and Metrication of Object Oriented System".

Por otra parte, en lo concerniente a la falta de cohesión (LCOM4), los resultados obtenidos luego de probar esta métrica en JQM3trics coinciden con la métrica presentada por Hitz & Montazeri en su artículo titulado "Measuring Coupling and Cohesion in Object-Oriented Systems. LCOM4 es presentada en dicho artículo como una versión mejorada de la métrica de cohesión.

El presente estudio es importante puesto que la herramienta desarrollada se puede utilizar para apoyar procesos de enseñanza y aprendizaje en ambientes académicos, lo que supondría un beneficio para el instructor y el estudiante. Específicamente, la explicación teórica de cómo se calcula cada métrica y la interpretación de los resultados obtenidos para un determinado producto de código fuente pueden servir para guiar la enseñanza y aprendizaje en el campo de la Ingeniería de Software. Es pertinente reiterar que lo mencionado anteriormente es un factor diferencial respecto a los estudios encontrados en la revisión de la literatura disponible.

Resultados inesperados

El resultado inesperado que se obtuvo durante la implementación y aplicación de pruebas sobre la herramienta fue el hecho de que JQM3trics no detecta la última línea de algunas clases, se explicó en el capítulo 4.5.2.6 Caso de prueba número 6: Cantidad de líneas de código que esto ocurre cuando la última línea es null y por consiguiente JQM3trics no la detecta. Hasta ahora la única forma de saber si una clase termina o no en null es analizándola con la herramienta, si arroja el total de líneas de código con una línea menos, quiere decir que tiene null en la última línea, si por el contrario el resultado es la cantidad real de líneas se puede decir que el archivo java no tiene null en su última línea.

Recomendaciones

En esta investigación se propuso y se desarrolló un componente de escritorio, o para ser usado en aplicaciones de escritorio. En ese orden de ideas, a futuro se recomienda construir una versión Web en la que el usuario suba sus archivos de código fuente, la desventaja en este caso es que no todos los usuarios están dispuestos a subir sus archivos a la Internet por razones de seguridad y ahí es donde entra a jugar un papel fundamental la herramienta que fue desarrollada en esta investigación por ser de tipo escritorio. Desde esa perspectiva, esta aparente limitante a la vez

supone una ventaja para aquellos usuarios que prefieran mantener confidencialidad de sus proyectos de código fuente.

La aplicación hasta el momento ha sido probada en sistemas operativos Windows y Ubuntu 14.04 TLS. En ambos la aplicación es ejecutada, sin embargo en Ubuntu presenta problemas al calcular las métricas, se muestra el mensaje “Ups! Algo pasó”. Para el trabajo futuro se recomienda hacer pruebas en sistemas operativos tales como Linux y MAC OS en aras de fortalecer la interoperabilidad de la herramienta de modo que solo sea necesario tener instalado el Entorno de Ejecución de Java (JRE) en la máquina donde se ejecute la aplicación.

En cuanto al cálculo de la falta de cohesión, se recomienda establecer una clasificación específica para los valores de LCOM4=1. Esto con el fin de encontrar una medida más precisa para determinar qué tan cohesiva es una clase.

Finalmente, como trabajo futuro también se recomienda que JQM3trics pueda mostrar un diagrama de dependencias, en el cual se visualicen gráficamente las clases del proyecto analizado y sus relaciones entre sí.

REFERENCIAS

Abreu, F. B., & Melo, W. (1996). Evaluating the Impact of Object-Oriented Design on Software Quality. *Software Metrics Symposium (METRICS'96) in Proceedings of the 3rd International*, 4. Recuperado el 22 de mayo de 2019

Aivosto Oy. (s.f.). Obtenido de https://www.aivosto.com/project/help/pm-oo-cohesion.html

Boehm, B. W., Brown, J. R., & Lipow, M. (1976). Quantitative evaluation of software quality. *In Proceedings of the 2nd international conference on Software engineering*, (pp. 592-605). Recuperado el 11 de Marzo de 2017

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on software engineering, 20(6)*, 476-493. Recuperado el 16 de Febrero de 2017

Constanzo, M. A. (Abril de 2014). Comparación de modelos de Calidad, Factores y Metricas en el ambito de la Ingeniería de Software. *Universidad Nacional de la Patagonia Austral - Unidad Académica Río Gallegos, Dpto. Ciencias Exactas y Naturales*.

Coupal, D., & Robillard, P. N. (1990). Factor analysis of source code metrics. *Journal of systems and software*, 263-269. Recuperado el 10 de Marzo de 2017

Fenton, N. E., & Neil, M. (Mayo de 2000). Software metrics: roadmap. *In Proceedings of the Conference on the Future of Software Engineering*, 357-370. Recuperado el 24 de Febrero de 2017

Fuggetta, A., & Di Nitto, E. (2014). Software process. *In Proceedings of the on Future of Software Engineering*, 1-12. Recuperado el 16 de Febrero de 2017

Higo, Y., Saitoh, A., Yamada, G., Miyake, T., Kusumoto, S., & Inoue, K. (Noviembre de 2011). A pluggable tool for measuring software metrics from source code. *In Software Measurement, 2011 Joint Conference of the 21st Int'l Workshop on and 6th Int'l Conference on Software Process and Product Measurement*, 3-12. Recuperado el 9 de Marzo de 2017

Hitz, M., & Montazeri, B. (1995). Measuring Coupling and Cohesion. *Conference: Proc. Int. Symposium on Applied Corporate Computing,*, P3.

IEEE. (1993). IEEE Standards Collection: Software Engineering. *IEEE Standard 610.12-1990*.

Lincke, R., Lundberg, J., & Löwe, W. (Julio de 2008). Comparing software metrics tools. *In Proceedings of the 2008 international symposium on Software testing and analysis*, (pp. 131-142). Recuperado el 24 de Febrero de 2017

Martin C. Robert, M. M. (2006). *Agile Principles, Patterns, and Practices in C#.* Prentice Hall. Recuperado el 09 de Abril de 2019

Martínez, A. (21 de Noviembre de 2017). *Medium*. Recuperado el 08 de Abril de 2019, de https://medium.com/@alanmartinez/10-principios-de-usabilidad-para-dise%C3%B1o-de-interfaces-de-usuario-f35d9d01643f

McCall, J. A., Richards, P. K., & Walters, G. F. (1977). *Factors in Software Quality. Volume I. Concepts and Definitions of Software Quality.* GENERAL ELECTRIC CO SUNNYVALE CA. Recuperado el 12 de Marzo de 2017

Metaute, P., & Serna, A. (2016). Diagnóstico sobre la apropiación de las métricas en medianas empresas de software de la ciudad de Medellpin. *Revista Antioqueña de Las Ciencias Computacionales y la Ingeniería de Software*, 6-14. Recuperado el 7 de Marzo de 2017

Monroy, M. (2016). *Marco de referencia para la recuperación y análisis de vistas arquitectónicas de comportamiento.* Tesis doctoral, Universidad del Cauca.

Monroy, M. E., Arciniegas, J. L., & Rodríguez, J. C. (2013). Monroy, M. E., Arciniegas, J. L., & Rodríguez, J. C. (2013). Propuesta Metodológica para Caracterizar y Seleccionar Métodos de Ingeniería Inversa. Información tecnológica, 24(5), 23-30. *Información tecnológica, 24(5)*, 26-27. Recuperado el 16 de Febrero de 2017

Oliveira, P., Lima, F. P., Valente, M. T., & Serebrenik, A. (Septiembre de 2014). RTTool: A tool for extracting relative thresholds for source code metrics. *In Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on (pp. 629-632). IEEE.* Recuperado el 9 de Marzo de 2017

Oliveira, P., Valente, M. T., & Lima, F. P. (Febrero de 2014). Extracting relative thresholds for source code metrics. *In Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on* , 254-263. Recuperado el 9 de Marzo de 2017

Pressman, R. S. (2010). Calidad de Software. En R. S. Pressman, *Ingeniería del Software - Un enfoque práctico* (págs. 340-344). Mexico: Mc Graw Hill.

Pressman, R. S. (2010). Factores de la calidad de McCall. En R. S. Pressman, *Ingeniería del Software - Un enfoque práctico* (págs. 342-343). Mexico: Mc Graw Hill.

Pressman, R. S. (2010). Ingeniería de Software. En R. S. Pressman, *Ingeniería de Software - Un enfoque práctico* (págs. 11 - 12). México: McGraw - Hill.

Raemaekers, S., van Deursen, A., & Visser, J. (2012). Measuring software library stability through historical version analysis. *In Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, 378-387. Recuperado el 10 de Marzo de 2017

Robert C. Martin, M. M. (2006). *Agile Principles, Patterns, and Practices in C#.* Prentice Hall. Recuperado el 09 de Abril de 2019

Rodríguez, D., & Harrison, R. (2001). An Overview of Objetc-Oriented Design Metrics. 8. Recuperado el 22 de 05 de 2019

Rodríguez, D., & Harrison, R. (2007). Medición en la Orientación a Objetos. *School of Computer Science, Cybernetics & Electronic Engineering University of Reading, UK*. Recuperado el 19 de Febrero de 2017