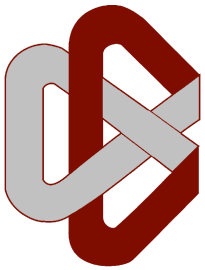


CIMAT

Centro de Investigación en Matemáticas, A.C.

Evaluación de herramientas para medir acoplamiento y cohesión en Java

REPORTE TÉCNICO

Que para obtener el grado de

Maestro en Ingeniería de Software

P r e s e n t a

Víctor Pablo Navarro Belmonte

Director de Reporte Técnico

Jorge Roberto Manjarrez Sánchez

Zacatecas, Zacatecas., 17 de octubre de 2012

Agradecimientos

A CIMAT Unidad Zacatecas y a COZCyT por haberme otorgado la beca de colegiatura y la beca de manutención que me permitieron estudiar esta maestría.

A el personal docente y administrativo de CIMAT Unidad Zacatecas por su amabilidad y dedicación que tuvieron en el trato con mi persona.

A mi familia y seres queridos por su apoyo durante estos años de intenso trabajo.

Resumen

El software debe de ser construido cuidando que sea mantenible y reusable, una forma de saber cuantificar estos aspectos es usando métricas de acoplamiento y cohesión. Medir acoplamiento y cohesión es importante, pero no es fácil hacerlo sin herramientas automatizadas. Este documento explica los conceptos básicos que definen el acoplamiento y cohesión, lista las métricas más importantes para cuantificarlos y evalúa la confiabilidad de las herramientas automatizadas más prominentes para el lenguaje Java al calcular dichas métricas, proporcionando un diagnóstico de su confiabilidad y posibles puntos de mejora.

Abstract

Software should be constructed taking care to keep it maintainable and reusable, a way to quantify these characteristics is to use coupling and cohesion metrics. Measuring coupling and cohesion is important, but it is not easy to do it without automated tools. This document explains the basic concepts that sustain coupling and cohesion, makes a list of the most important metrics and makes an assessment of the most prominent automated tools for the Java language reliability calculating those metrics. After this a reliability diagnosis is provided with probable improvement points.

Tabla de contenidos

Introducción **1**

Conceptos básicos **3**
- **Acoplamiento** **3**
  - The Stable Abstractions Principle (SAP) 3
- **Cohesión** **4**
  - Cohesión a nivel de clase 4
  - Cohesión a nivel librería 4

Métricas **6**
- **Métricas de acoplamiento** **6**
- **Métricas derivadas del acoplamiento** **6**
- **Métricas de cohesión** **9**

Selección de herramientas analizadoras de código **11**
- **Aplicaciones stand alone** **11**
  - JArchitect/ JavaDepend 11
  - JDepend 12
- **Plugins para eclipse** **12**
  - Eclipse Metrics (Continued) 12
  - CodePro Analytix 12
- **Plugins para Intellij Idea** **12**
  - Intellij Idea 13
  - Metrics reloaded 13
- **Criterios de selección de analizadores a probar** **13**
- **Herramientas seleccionadas** **13**

Evaluación de las herramientas **14**
- **Operación de las herramientas analizadoras de código** **14**
  - JArchitect 14
  - Codepro Analytix 16
  - Eclipse Metrics Plugin (Continued) 18
  - Conclusión sobre operación de las herramientas 19
- **Pruebas de sobrecarga** **20**
- **Código utilizado en mediciones.** **20**
- **Evaluación métrica por métrica** **21**
  - Acoplamiento aferente a nivel clase 21
  - Acoplamiento eferente a nivel clase 23
  - Acoplamiento aferente a nivel librería 25
  - Acoplamiento eferente a nivel librería 27
  - Inestabilidad 29
  - Nivel de abstracción 31
  - Distancia de la secuencia principal normalizada 32
  - Falta de cohesión en los métodos 34
  - Falta de cohesión en los métodos Henderson Sellers 36
  - Cohesión Relacional 38
- **Resumen de resultados** **40**
- **Comentarios finales sobre herramientas** **41**
  - JArchitect 41

Codepro Analytix 41
Eclipse metrics continued 42
Lecciones aprendidas **42**
Conclusiones **44**
Trabajo futuro **45**
Bibliografía **47**

Lista de figuras

Figura 3.1 La secuencia principal 8
Figura 5.1 Pantalla inicial de JArchitect 14
Figura 5.2 Creación de archivo de proyecto para JArchitect 15
Figura 5.3 Configuración de proyectos a analizar en JArchitect 15
Figura 5.4 Ventana para hacer consultas CQL en JArchitect 16
Figura 5.5 Opción de menú para mostrar ventana de métricas en Codepro Analytix 17
Figura 5.6 Estado inicial de ventana de métricas en Codepro Analytix 17
Figura 5.7 Selección de proyectos en Codepro Analytix 17
Figura 5.8 Resultado de cálculo de métricas en Codepro Analytix 18
Figura 5.9 Opción de menú para mostrar ventana de métricas en Eclipse Metrics 18
Figura 5.10 Activación de cálculo de métricas para un proyecto en Eclipse Metrics 19
Figura 5.11 Métricas calculadas en Eclipse Metrics 19
Figura 5.12 Tiempo en segundos utilizado por herramientas en analizar proyecto grande 20
Figura 5.13 Acoplamiento aferente a nivel clase en JArchitect 22
Figura 5.14 Acoplamiento aferente de clase BankNote según JArchitect 23
Figura 5.15 Acoplamiento eferente a nivel clase en JArchitect 24
Figura 5.16 Acoplamiento Eferente a nivel clase según JArchitect 25
Figura 5.17 Acoplamiento aferente a nivel librería en JArchitect 26
Figura 5.18 Acoplamiento aferente a nivel librería en Codepro Analytix 26
Figura 5.19 Acoplamiento aferente a nivel librería de Core según JArchitect 27
Figura 5.20 Acoplamiento eferente a nivel librería en JArchitect 28
Figura 5.21 Acoplamiento eferente a nivel librería en Codepro Analytix 28
Figura 5.22 Tipos externos referenciados en Codepro Analytix 29
Figura 5.23 Inestabilidad en JArchitect 30
Figura 5.24 Inestabilidad en Codepro Analytix 31
Figura 5.25 Nivel de abstracción en JArchitect 32
Figura 5.26 Nivel de abstracción en Codepro Analytix 32
Figura 5.27 Distancia de la secuencia principal normalizada en JArchitect 34
Figura 5.28 Valores de distancia de la secuencia principal normalizada en Codepro Analytix 34
Figura 5.29 Falta de cohesión en los métodos en JArchitect 36
Figura 5.30 Falta de cohesión en los métodos Henderson Sellers en JArchitect 37
Figura 5.31 Falta de cohesión en los métodos Henderson sellers en Eclipse Metrics Continued 38
Figura 5.32 Cohesión relacional para ProyectoCohesivo en JArchitect 39

Figura 5.33 Cohesión relacional para ProyectoRelativamenteCohesivo en JArchitect 39
Figura 5.34 Cohesión relacional para ProyectoNoCohesivo en JArchitect....40

Lista de tablas

Tabla 5.1 Desempeño de las herramientas analizando un proyecto grande 20
Tabla 5.2 Comparación de valores de acoplamiento aferente a nivel clase 22
Tabla 5.3 Comparación de valores de acoplamiento eferente a nivel clase 24
Tabla 5.4 Comparación de valores de acoplamiento aferente a nivel librería....27
Tabla 5.5 Comparación de valores de acoplamiento eferente a nivel librería....28
Tabla 5.6 Valores de inestabilidad cálculo manual....30
Tabla 5.7 Valores de inestabilidad en JArchitect 31
Tabla 5.8 Valores de inestabilidad en Codepro Analytix 31
Tabla 5.9 Comparación de valores de inestabilidad 31
Tabla 5.10 Comparación de valores de nivel de abstracción 32
Tabla 5.11 Distancia de la secuencia principal normalizada cálculo manual....33
Tabla 5.12 Valores de distancia de la secuencia principal normalizada en JArchitect 34
Tabla 5.13 Valores de distancia de la secuencia principal normalizada en Codepro Analytix 34
Tabla 5.14 Comparación de valores de distancia de la secuencia principal normalizada 35
Tabla 5.15 Comparación de valores de falta de cohesión en los métodos....36
Tabla 5.16 Comparación de valores de falta de cohesión en los métodos Henderson Sellers....38
Tabla 5.17 Comparación de valores de Cohesión relacional 40
Tabla 5.18 Matriz de comparación de resultados de herramientas....41

Lista de fórmulas

Fórmula 3.1 Inestabilidad (I) 7
Fórmula 3.2 Nivel de abstracción (A)....7
Fórmula 3.3 Distancia de la secuencia principal (D) 9
Fórmula 3.4 Distancia de la secuencia principal normalizada (D') 9
Fórmula 3.5 Falta de cohesión en los métodos (LCOM) 9
Fórmula 3.6 Falta de cohesión en los métodos Henderson Sellers (LCOMHS) 10
Fórmula 3.7 Cohesión Relacional (H) 10
Fórmula 5.1 Falta de cohesión en los métodos (LCOM) según JArchitect 36

1. Introducción

Para poder tener confianza de que el código de un sistema de software sea de calidad, es necesario verificar que tenga ciertas características deseables, como que sea fácil de modificar, que sea fácil de entender y que aporte una buena cantidad de componentes reusables para otros proyectos.

El que un sistema sea fácil de modificar y entender, debe de ser una prioridad de los equipos que desarrollan software, porque estas características hacen que el sistema sea más barato de producir y mantener. Un sistema que es difícil de entender y modificar va a consumir muchas horas de trabajo en implementación de cambios por parte de los ingenieros; un sistema que por otro lado sea fácil de mantener y modificar, va a requerir menos horas de los ingenieros, esto se refleja en un gasto menor para la empresa y por lo tanto un mayor margen de ganancia por la reducción de costos.

Cada vez es más frecuente encontrar empresas que se interesan en contar con sus propias librerías de reúso, por lo cual resulta más importante que los componentes de los sistemas que se desarrollan puedan ser integrados a esas bibliotecas para su reúso en proyectos posteriores. El contar con una buena librería de reúso es benéfico en el aspecto financiero, ya que reduce el tiempo de desarrollo al utilizar código ya generado y probado para ciertas partes de los sistemas de software.

Existe un problema con evaluar qué tan bien cumplen los sistemas con estas características y es que la evaluación es subjetiva por lo tanto no están siendo representadas por datos numéricos verificables, sino por opiniones que pueden variar dependiendo del contexto en el que se emiten o de las personas que las emiten.

Una forma de poder usar valores numéricos objetivos para describir estas características, es medir el acoplamiento y la cohesión de un sistema. Existen métricas definidas por la comunidad científica que pueden medir objetiva y cuantitativamente el acoplamiento y la cohesión a diferentes niveles dentro de una aplicación.

Cuando se tienen los valores numéricos con respecto al acoplamiento y la cohesión, se puede verificar si un sistema tiene un buen diseño. Se dice que un sistema tiene un buen diseño si cuenta con un bajo acoplamiento entre sus componentes pero con una alta cohesión dentro de ellos.

Las métricas de acoplamiento y cohesión pueden ser calculadas de manera manual por un ingeniero que entienda las fórmulas y definiciones que hay en la literatura sobre este tema, pero hay una restricción en este aspecto: el tamaño del sistema. Un

CENTRO DE INVESTIGACIÓN EN MATEMÁTICAS, A.C.

BIBLIOTECA

AUTORIZACION
PUBLICACION EN FORMATO ELECTRONICO DE TESIS

El que suscribe
Autor(s) de la tesis: Víctor Pablo Navarro Belmonte

Título de la tesis: Evaluación de herramientas para medir acoplamiento y cohesión en Java

Institución y Lugar: Cimat Unidad Zacatecas

Grado Académico: Licenciatura ( ) Maestría (x) Doctorado ( ) Otro ( )

Año de presentación: 2012

Área de Especialidad: Ingeniería de software

Director(es) de Tesis: Doctor Jorge Roberto Manjarrez Sánchez

Correo electrónico: pnavarrob@cimat.mx , victorpablo.navarrobelmonte@gmail.com

Domicilio: Mexicapan #1366 col. centro Zacatecas Zacatecas

Palabra(s) Clave(s): Java, acoplamiento, cohesión, evaluación

Por medio del presente documento autorizo en forma gratuita a que la Tesis arriba citada sea divulgada y reproducida para publicarla mediante almacenamiento electrónico que permita acceso al público a leerla y conocerla visualmente, así como a comunicarla públicamente en la Página WEB del CIMAT.

La vigencia de la presente autorización es por un periodo de 3 años a partir de la firma de presente instrumento, quedando en el entendido de que dicho plazo podrá prorrogar automáticamente por periodos iguales, si durante dicho tiempo no se revoca la autorización por escrito con acuse de recibo de parte de alguna autoridad del CIMAT

La única contraprestación que condiciona la presente autorización es la del reconocimiento del nombre del autor en la publicación que se haga de la misma.

Atentamente

Víctor Pablo Navarro Belmonte
Nombre y firma del tesista

CALLE JALISCO S/N MINERAL DE VALENCIANA APDO. POSTAL 402
C.P. 36240 GUANAJUATO, GTO., MÉXICO
TELÉFONOS (473) 732 7155, (473) 735 0800 EXT. 49609 FAX. (473) 732 5749
E-mail: biblioteca@cimat.mx

sistema pequeño con menos de 100 clases puede ser analizado por una persona, pero sistemas de mayor tamaño puede que sean muy complejos para ser analizados en un tiempo aceptable, por lo cual es necesario usar otras formas de obtener estos valores.

Para poder analizar y calcular las métricas para proyectos grandes, han sido creadas varias herramientas automatizadas que analizan el código y aplican las fórmulas y criterios que se han definido para calcular los valores de acoplamiento y cohesión. Estas herramientas ofrecen la ventaja de que al utilizar el poder de procesamiento de una computadora, pueden obtener los valores en un tiempo muy reducido a comparación de un cálculo manual realizado por una persona.

El uso de estas herramientas automatizadas tiene sin embargo ciertos riesgos, como que se desconoce el nivel de exactitud que tienen para el cálculo de estos valores y no se sabe si tienen errores al realizar los cálculos y en caso de que los tengan, se desconoce la magnitud de estos errores.

Es por eso que existe una necesidad de hacer una evaluación del desempeño de estas herramientas para saber en qué grado se puede confiar en sus cálculos y qué se puede hacer para mejorar su exactitud en caso de que sea necesario. Para realizar esta evaluación va a ser necesario, comparar los valores calculados por las herramientas, contra valores calculados manualmente por una persona que siga estrictamente las fórmulas y procedimientos indicados en las definiciones formales de las métricas de acoplamiento y cohesión.

En este documento se explicarán los conceptos básicos referentes a la cohesión, el acoplamiento y las métricas que pueden usarse para cuantificarlos. Después de esto se elegirán los analizadores para el lenguaje Java que se adecúen mejor a las métricas que sean importantes para esta investigación. Estos analizadores serán evaluados en su confiabilidad para calcular las métricas haciendo una comparación entre valores generados por herramientas y valores generados por un cálculo manual confiable. Para concluir, se informará sobre los resultados de la evaluación y las posibles líneas de investigación o trabajo para el futuro.

2. Conceptos básicos

Es importante tener claras las ideas que están detrás de las métricas que van a ser presentadas posteriormente. Es un problema recurrente del software que entre más crece en tamaño y funcionalidad se vuelve más difícil de mantener. La dificultad de hacer un cambio pequeño puede hacer a una tarea muy pesada en costos y tiempo debido a la complejidad del diseño del software.

Para tratar de mantener las aplicaciones en un estado que permitan el mantenimiento sencillo y barato, se han formado dos conceptos que por lo general se relacionan: acoplamiento y cohesión.

2.1.Acoplamiento

Acoplamiento en el software se refiere al grado en el que una parte del software se relaciona con las otras partes que componen al software [1]. Entre más dependencias tenga una parte del software esta se vuelve más propensa a cambiar puesto que los cambios en las partes en las que depende pueden provocar cambios al modificar las entradas o salidas. Por otro lado entre más partes dependan de una determinada parte esta se vuelve más responsable y por lo tanto es más delicado cambiarla puesto que las posibilidades de hacer que una funcionalidad externa se rompa aumentan.

Un sistema con alto acoplamiento entre sus partes, es difícil de probar en términos de pruebas de unidad, difícil de mantener, difícil de entender y por todas estas razones más costoso de mantener y construir.

También el acoplamiento se usa como fundamento para calcular la estabilidad de un componente. El estar consciente de la estabilidad permite al desarrollador seguir el principio de las abstracciones estables [2], que se discute en la siguiente sección.

2.1.1.The Stable Abstractions Principle (SAP)

Establece que un componente debe de ser tan abstracto como es estable, esto se traduce en que los componentes idealmente deben de estar en uno de los dos estados ideales:

- Ser totalmente estables y abstractos para evitar que el código concreto limite la extensión de un diseño.
- Ser totalmente inestables y concretos para que la misma inestabilidad permita realizar cambios al código concreto mas fácilmente.

Los componentes de un sistema deben de apegarse a estas dos categorías ideales o bien lograr un equilibrio entre estabilidad y nivel de abstracción. En la sección de métricas se presentan formas de medir estos atributos cuantitativamente.

2.2.Cohesión

Hay dos tipos de cohesión a nivel clase y a nivel librería.

2.2.1.Cohesión a nivel de clase

Se refiere a cómo los elementos de una clase están correctamente agrupados juntos, esto se refiere a que la funcionalidad que implementa está relacionada y la clase solo tiene una responsabilidad como lo dice el principio de diseño de una sola responsabilidad, que se revisa a continuación.

The Single Responsibility Principle

Indica que para tener clases robustas, fáciles de entender y modificar es necesario que tengan una sola responsabilidad [3]. Este principio define la responsabilidad de una clase como una razón para cambiar. Si se puede pensar en varias razones que podrían causar la necesidad de un cambio, entonces la clase tiene varias responsabilidades.

Una clase con más de una responsabilidad, puede tener varios efectos no deseados sobre nuestra aplicación. Uno de ellos es que las clases van a ser más difíciles de entender ya que la agrupación que tenemos de clases mezcla varias responsabilidades pudiendo causar confusión e ideas erróneas sobre cómo funcionan internamente. Otro posible problema es que las clases con más de una responsabilidad son más propensas a la inserción de errores durante el mantenimiento. Al tener mezcladas varias responsabilidades es posible que un cambio que se hace para modificar el comportamiento de una sola responsabilidad afecte a las demás responsabilidades inadvertidamente.

2.2.2.Cohesión a nivel librería

Es importante que en nuestro diseño cada clase sea una unidad cohesiva, pero de igual manera es importante que la distribución de nuestras clases en librerías sea de una manera cohesiva para garantizar que nuestras librerías van a poder ser reusadas y distribuidas confiando en que no causen problemas en nuevos sistemas.

Para verificar esto se propone el uso de 3 principios para tener en mente al partir una aplicación en librerías [2].

The Reuse/Release Equivalence Principle

Debemos de hacer una librería un conjunto de clases que agrupándolas consideremos que pueden ser útiles para ser reusadas en otro proyecto. No tiene caso hacer una librería con una clase y decir que es reusable si esa librería requiere de muchas otras librerías para poder hacer algo útil. De igual manera debemos de considerar solo incluir clases que son útiles en el contexto de reúso o solo incluir clases que no son útiles para el reúso ya que es importante mantener las librerías de reusó lo más limpias y concisas que sea posible, mezclar clases reusables con no reusables en una misma librería solo hace su manejo mas complicado.

The Common Reuse Principle

Este principio consiste en que si se usa una clase de una librería, se deben de reusar todas las clases dentro de esa librería. Esto lo podemos interpretar como que una librería de reúso debe de contener solo las clases que son más cohesivas entre sí. Si se están incluyendo clases con poca relación con las demás, es posible que se estén insertando dependencias que no son importantes para el reúso y solo van a complicar el manejo de la librería.

The Common Closure Principle

Las clases en una librería deben de estar cerradas a los mismos tipos de cambios. Un cambio que afecta a la librería afecta a todas las clases dentro de la misma, pero no afecta a ninguna otra librería. Este es el equivalente del Single Responsibility Principle aplicado a librerías y básicamente busca mejorar la mantenibilidad de la librería y del sistema que las usa, al conservar los cambios en una sola librería, reduciendo de esta manera el tiempo en validar, recompilar e instalar solo una librería en lugar de un conjunto de ellas.

El mantener estos principios en mente cuando se diseña un sistema ayuda a mantener una estructura de componentes más cohesiva que va a beneficiar un mayor reúso y mantenibilidad.

El tener un diseño con una alta cohesión por lo general tiene como consecuencia bajo acoplamiento y de la misma forma, el tener un diseño donde se cuida tener un bajo acoplamiento generalmente resulta en tener una alta cohesión. El tener alta cohesión y bajo acoplamiento es un atributo deseable en el software pero para poder saber en qué grado se cumple con estos atributos, es necesario tener una forma de medir de manera objetiva y cuantitativa.

3. Métricas

Con el propósito de evaluar el nivel de acoplamiento y cohesión de el diseño de una aplicación se han creado métricas. A continuación se listan las que se consideran más importantes para medir estos aspectos del software.

3.1.Métricas de acoplamiento

Acoplamiento eferente a nivel clase (Cec) : Es el número de clases a las que hace referencia directamente esta clase. Una clase con un acoplamiento eferente mayor a 50 debe de ser revisada cuidadosamente ya que es muy probable que tenga más de una responsabilidad y sea un buen candidato para ser refactorizado [4].

Acoplamiento aferente a nivel clase (Cac): Es el número de clases que referencian directamente a esta clase [4].

Acoplamiento eferente a nivel librería (Ce): Es el número de clases externas a las que hacen referencia las clases dentro de esta librería. Esta métrica se puede interpretar como una medida de dependencia. Un alto nivel de acoplamiento eferente indica un grado de dependencia alto, esto se puede interpretar como una mayor propensión al cambio ya que cambios en las dependencias externas pueden inducir cambios este componente [2].

Acoplamiento aferente a nivel librería(Ca): Es el número de clases externas a este componente que hacen referencia directamente a esta clases dentro de esta librería. Esta métrica se puede interpretar como un nivel de responsabilidad, un acoplamiento aferente alto representa un grado de responsabilidad alto, esto se relaciona con un componente que tiene una alta resistencia a cambiar [2].

Al respecto de estas métricas de acoplamiento se puede decir que un alto nivel de acoplamiento no es absolutamente malo, aquí se trata de que el diseñador utilice su criterio y vea al acoplamiento eferente alto como una característica que hace a un elemento inestable y propenso al cambio y de la misma manera que vea al acoplamiento aferente alto como una característica que implica que un elemento es altamente responsable. Una combinación de alta responsabilidad y alta inestabilidad implica elementos propensos a ser cuellos de botella en el desarrollo o mantenimiento [5].

3.2.Métricas derivadas del acoplamiento

Inestabilidad(I): Representa la resistencia de una librería a cambiar. Se calcula la relación entre acoplamiento eferente y el acoplamiento total de la librería [6]. Su valor se puede establecer como una relación entre el nivel de dependencia y el nivel de responsabilidad.

$$I = Ce/(Ce+Ca)$$

Fórmula 3.1 Inestabilidad (I)

Su rango de valores va desde 0 que es considerada una librería totalmente estable hasta 1 donde se considera que la librería es totalmente inestable.

Nivel de abstracción (A): Representa qué tan abstracto es un componente. Se obtiene al dividir el número de clase abstractas entre el número total de clases [2].

$$A= TCA / TC$$

Fórmula 3.2 Nivel de abstracción (A)

Donde:
TCA = número total de clases abstractas (las interfaces son tomadas como clases abstractas).
TC = número total de clases.

El rango de valores de esta métrica va de 0 a 1, donde cero indica que el componente no tiene clases abstractas y uno representa a un componente que solo tiene clases abstractas.

La secuencia principal

Con un valor calculado para las métricas inestabilidad (I) y el nivel de abstracción (A) se puede establecer una relación entre ellas. Se considera que si se hace una gráfica con el nivel abstracción (A) como el eje vertical y usamos el valor de la inestabilidad (I) como el eje horizontal, se pueden representar dos tipos de componentes ideales los que son máximamente abstractos y estables y por otro lado, los que son máximamente concretos e inestables [3]. Esto de acuerdo al Stable Abstractions Principle es muy difícil de lograr por lo cual se establece que los componentes idealmente deben de ser tan abstractos como estables; esta relación entre estas dos características puede ser expresada en la gráfica como una línea recta que va del punto (1,0) al (0,1), esta línea es llamada la secuencia principal. Este concepto es ilustrado en la Figura 3.1.

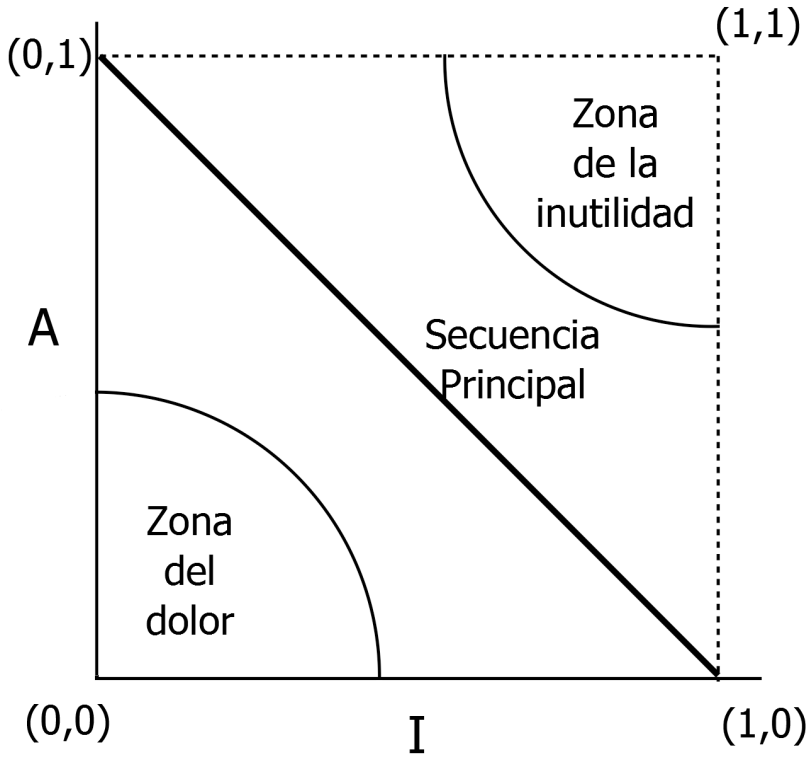


Figura 3.1 La secuencia principal

El contar con esta secuencia principal permite tener una referencia clara de cómo deben de ubicarse los componentes en esta gráfica, lo deseable es que todos los componentes estén cerca de la secuencia principal, es decir, que estén balanceados en cuanto a su nivel de abstracción y estabilidad, de la misma manera se encuentran lejos de las dos zonas de exclusión donde los componentes que se encuentran dentro de esas zonas pueden presentar diversos problemas, estas dos zonas son detalladas a continuación [2]:

Zona del dolor: cuando un componente se encuentra cerca del punto (0,0) es un componente concreto y estable, esto es indeseable porque el componente es rígido, es difícil de extender debido a que es concreto y difícil de cambiar porque es muy estable, aunque hay algunos componentes que se pueden acercar a este punto de manera inevitable. Un ejemplo recurrente de esto son los componentes de acceso a base de datos.

Zona de la inutilidad: un componente que se acerca al punto (1,1) indica un componente no deseable porque por un lado es muy abstracto, incapaz de hacer nada por sí mismo y por otro lado, como indica su alta inestabilidad no tiene responsabilidades, no es referenciado por los demás componentes. Un componente en esta zona debe de tener una utilidad baja para el sistema y puede ser considerado para refactorización.

Distancia de la secuencia principal (D): Para tener un dato numérico que indique qué tan cerca se encuentra un componente de la secuencia principal existe una fórmula [2]:

$$D= (A+I+1)/ \sqrt{2}$$

Fórmula 3.3 Distancia de la secuencia principal (D)

El rango de valores para D es [0, ~0.707] donde 0 indica que el componente se encuentra directamente sobre la secuencia principal y 0.707 indica que el valor está lo más lejos posible de la secuencia principal.

Distancia de la secuencia principal normalizada(D'): Es una métrica que de igual manera indica qué tan cerca se encuentra un componente de la secuencia principal, pero es más conveniente porque tiene un rango de [0,1] donde 0 indica un componente que está en la secuencia principal y 1 un componente que está lo más lejos posible de la secuencia principal [2], por esto es la métrica que se va usar en las mediciones de esta evaluación. La fórmula es la siguiente:

$$D'= A+I-1$$

Fórmula 3.4 Distancia de la secuencia principal normalizada (D')

3.3.Métricas de cohesión

Falta de cohesión en los métodos (LCOM): esta métrica trata de capturar la cohesión de los métodos de una clase, al considerar por un lado pares de métodos similares (métodos que comparten una o más variables de clase) y de la misma manera pares de métodos no similares (métodos que no comparten variables). La fórmula para calcularla es [7] [8]:

$$\begin{gathered} SI\ NPS> NPNS \\ LCOM = NPNS - NPS \\ SI\ NPS<=NPNS \\ LCOM=0 \end{gathered}$$

Fórmula 3.5 Falta de cohesión en los métodos (LCOM)

Donde:
NPNS = Número de pares no similares.
NPS = Número de pares similares.

Esta fórmula se enfoca en ver las similitudes entre métodos como una variable booleana (cierto o falso), por eso no es tan conveniente porque no mide la intensidad de las relaciones, cosa que puede ser un factor importante para tomar en cuenta.

Falta de cohesión en los métodos Henderson Sellers (LCOMHS): esta métrica trata de capturar la cohesión de los métodos de una clase. Al considerar las similitudes de los métodos estas se cuentan como las variables de instancia que acceden en común los métodos [9].

$$LCOMHS = (Sum(MF)/F - M)/(1-M)$$

Fórmula 3.6 Falta de cohesión en los métodos Henderson Sellers (LCOMHS)

Donde :
M es el número de métodos en una clase.
F es el número de variables de instancia en la clase.
MF es el número de métodos en una clase que usan una variable de instancia en particular.
Sum(MF) es la sumatoria de todos los valores MF para todas las variables de instancia en la clase.

Esta fórmula fue pensada en remediar las deficiencias de la fórmula de LCOM, puesto que mide las relaciones de una manera gradual donde puede haber relaciones fuertes y débiles.

El rango que puede tener LCOM HS es de 0 a 1. Un valor de 0 representa una clase con cohesión total o perfecta en el otro extremo, un valor de 1 representa una clase sin cohesión donde cada método accede a sus propias variables de instancia[7], entre más se acerque un valor al 1 es posible que la clase tenga más de una responsabilidad y sea candidata para una refactorización en clases más cohesivas.

Cohesión relacional (H) : esta métrica captura la cohesión de las clases dentro de una librería al medir la cantidad de dependencias que tienen las clases de la librería.

$$H = (R+1)/N$$

Fórmula 3.7 Cohesión Relacional (H)

Donde:
R es el número de relaciones entre las clases dentro de la librería
N es el número de clases dentro de la librería.
En esta fórmula las dependencias mutuas se cuentan dos veces, una vez por cada dirección [2].

Las clases dentro de una librería deben de estar íntimamente relacionadas, por lo cual un valor deseable para H sería > 1.5 y <4 ; esto porque un valor muy alto puede indicar problemas de sobre acoplamiento y tal vez requiera que algunas clases sean fundidas en unidades más cohesivas [4].

4. Selección de herramientas analizadoras de código

El foco de este estudio será sobre el lenguaje de programación Java, porque en la actualidad cuenta con una gran aceptación por parte de la industria del desarrollo, por lo cual también hay disponible una buena cantidad de herramientas analizadoras de código. A continuación se presenta una lista con las más destacadas. Para propósitos de organización han sido separados en tres categorías:

- Aplicaciones stand alone
- Plugins de Eclipse
- Plugins de Intellij Idea

4.1.Aplicaciones stand alone

4.1.1.JArchitect/ JavaDepend

Aplicación comercial con licencia gratis para propósitos académicos o para uso en proyectos open source que puede analizar código fuente organizado en proyectos de varios entornos de desarrollo de Java y calcula varias métricas. Las que son relevantes para este trabajo son:

- Acoplamiento aferente a nivel clase y librería.
- Acoplamiento eferente a nivel clase y librería.
- Inestabilidad
- Nivel de abstracción
- Distancia de la secuencia principal normalizada
- Cohesión relacional
- Falta de cohesión en los métodos
- Falta de cohesión en los métodos Henderson Sellers

Puede analizar :

- Archivos jar
- Archivos war
- Proyectos de Eclipse.
- Proyectos de Maven.
- Proyectos de NetBeans.
- Proyecos de IntelliJ Idea.

Cuenta con su propio lenguaje CQL (Code Query Language) que es un lenguaje con bastantes similitudes a SQL(Structured Query Language) de bases de datos que

permite realizar consultas en base a las métricas obtenidas al hacer un análisis del código previo.

Es importante mencionar que esta herramienta analiza los archivos compilados y el código, por lo cual el análisis puede incluir datos de elementos generados en tiempo de compilación sin un vínculo claro con el código.

4.1.2.JDepend

Aplicación de código abierto sin actualizaciones desde el 2010, que realiza un análisis de código y calcula varias métricas. Las que más se acercan a ser relevantes para este trabajo son :

- Acoplamiento aferente a nivel paquete.
- Acoplamiento eferente a nivel paquete.

No soporta propiamente el manejo de alguna estructura de proyecto de Java, sino que se basa en analizar directorios con código fuente.

4.2.Plugins para eclipse

4.2.1.Eclipse Metrics (Continued)

Plugin de eclipse de código abierto que realiza un análisis de código y calcula varias métricas. La que es relevante para este trabajo es:

- Falta de cohesión en los métodos

4.2.2.CodePro Analytix

Plugin de eclipse de código abierto que realiza un análisis de código y calcula varias métricas. Las que son relevantes para este trabajo son :

- Acoplamiento aferente a nivel librería.
- Acoplamiento eferente a nivel librería.
- Inestabilidad
- Nivel de abstracción
- Distancia de la secuencia principal normalizada

4.3.Plugins para Intellij Idea

4.3.1.Intellij Idea

Aunque no es un plugin en sí, una instalación de la versión 11 de Intellij Idea cuenta con un analizador de código. Las métricas que ofrece no se apegan a las que consideramos importantes en este trabajo. Las que más se aproximan a lo que busca este estudio son:

- Clase demasiado acoplada
- Método demasiado acoplado

4.3.2.Metrics reloaded

Plugin de Intellij Idea de código abierto con muy poca documentación que parece abandonado. Se intentó probar pero no funciona correctamente. Realiza un análisis de código y calcula varias métricas. Las que son relevantes para este trabajo son:

- Acoplamiento aferente a nivel clase y librería.
- Acoplamiento eferente a nivel clase y librería.

4.4.Criterios de selección de analizadores a probar

Es la intención de esta investigación realizar pruebas sobre las capacidades de medición de acoplamiento y cohesión de las herramientas de análisis de código disponibles en la actualidad, se han tomado tres criterios para decidir cuáles son las herramientas más viables para probar:

1. Cantidad de métricas obtenidas
2. Documentación
3. Mantenimiento reciente del proyecto

4.5.Herramientas seleccionadas

Tomando en cuenta los criterios previamente mencionados, las herramientas seleccionadas para probar son las siguientes (se incluye versión usada la cual es la mas reciente en el momento de la elaboración de este trabajo):

- JArchitect 2.0.1 (10 métricas calculadas)
- CodePro Analytix 1.3.8 (5 métricas calculadas)
- Eclipse Metrics plugin (Continued) 7.1.0 (1 métrica calculada)

Las demás contienen pocas métricas que son de relevancia para este trabajo o tienen un nivel deficiente en mantenimiento y documentación. Para Eclipse Metrics se hizo una excepción puesto que la única métrica que calcula relevante para este estudio solo es calculada por otra herramienta

5. Evaluación de las herramientas

Con un conjunto de herramientas seleccionadas el siguiente paso es evaluarlas. Se busca evaluar por un lado su facilidad de uso y desempeño con proyectos de gran magnitud y por otro lado se busca ver su precisión al calcular las métricas. En este capítulo se explica cómo se evaluaron y cuáles fueron los resultados obtenidos.

5.1.Operación de las herramientas analizadoras de código

En esta sección se presenta una pequeña reseña de qué hace falta hacer desde una perspectiva del usuario para analizar las métricas de un proyecto de Java.

5.1.1.JArchitect

Esta herramienta está disponible en Windows y Linux. Por conveniencia se eligió la versión de Windows, ya que su instalación es la estándar de las aplicaciones de Windows y no requiere ninguna instrucción especial.

Al iniciar la aplicación se presenta una pantalla de opciones para abrir o crear proyectos propios de JArchitect. Inicialmente se tiene que crear un proyecto usando el botón que se resalta en la Figura 5.1.

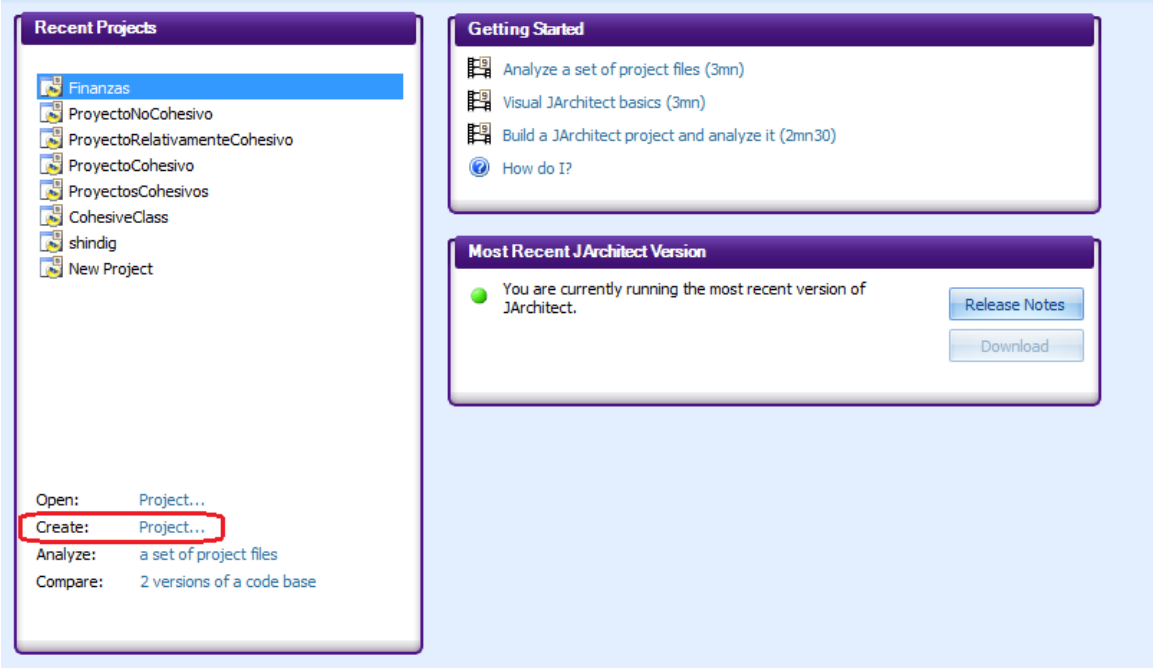


Figura 5.1 Pantalla inicial de JArchitect

Después de esto, se presenta la ventana de creación del archivo de proyecto de JArchitect; aquí el usuario debe decidir el nombre del proyecto, directorio donde se va a guardar el archivo y nombre del archivo, como se ilustra en la Figura 5.2.

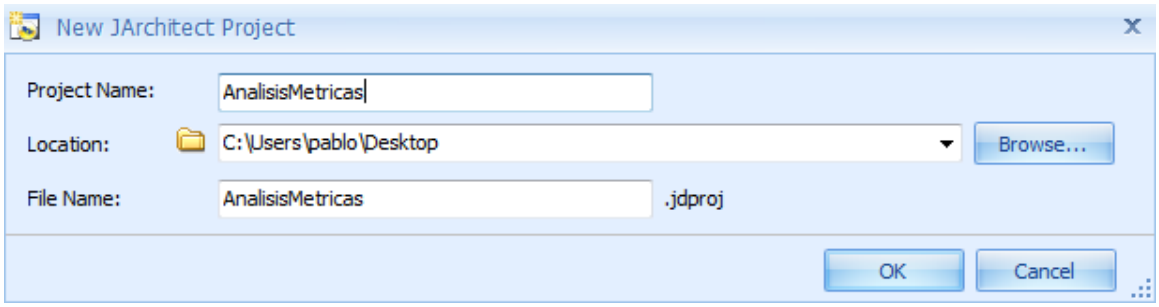


Figura 5.2 Creación de archivo de proyecto para JArchitect

Una vez que se tiene el archivo creado, se presenta una pantalla donde se pueden agregar proyectos de desarrollo de Java para analizarlos, es importante que estos proyectos incluyan sus archivos binarios generados a partir del código. Estos proyectos pueden ser agregados utilizando el botón "Browse" o haciendo "drag and drop" de los archivos en la ventana. Una vez que se tienen cargados todos los proyectos deseados se da clic en el botón de "Play" para iniciar el análisis del código y la obtención de métricas. Esto se ilustra en la Figura 5.3.

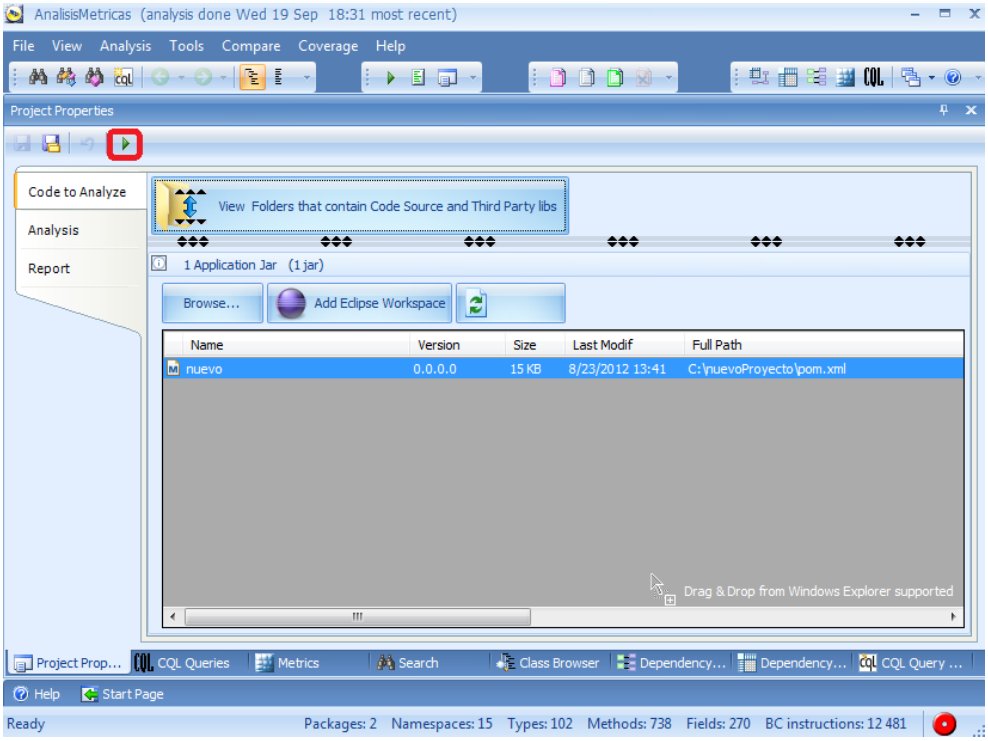


Figura 5.3 Configuración de proyectos a analizar en JArchitect

Después de lo descrito anteriormente, la herramienta está lista para proporcionar los datos que le interesen al usuario. Para esto se cuenta con varias pestañas que presentan la información en diferentes formatos, pero la más útil y flexible es la llamada "CQL Query Edit", la cual usa un lenguaje parecido a SQL, muy flexible para proporcionar información de métricas. Esta ventana se muestra en la Figura 5.4.

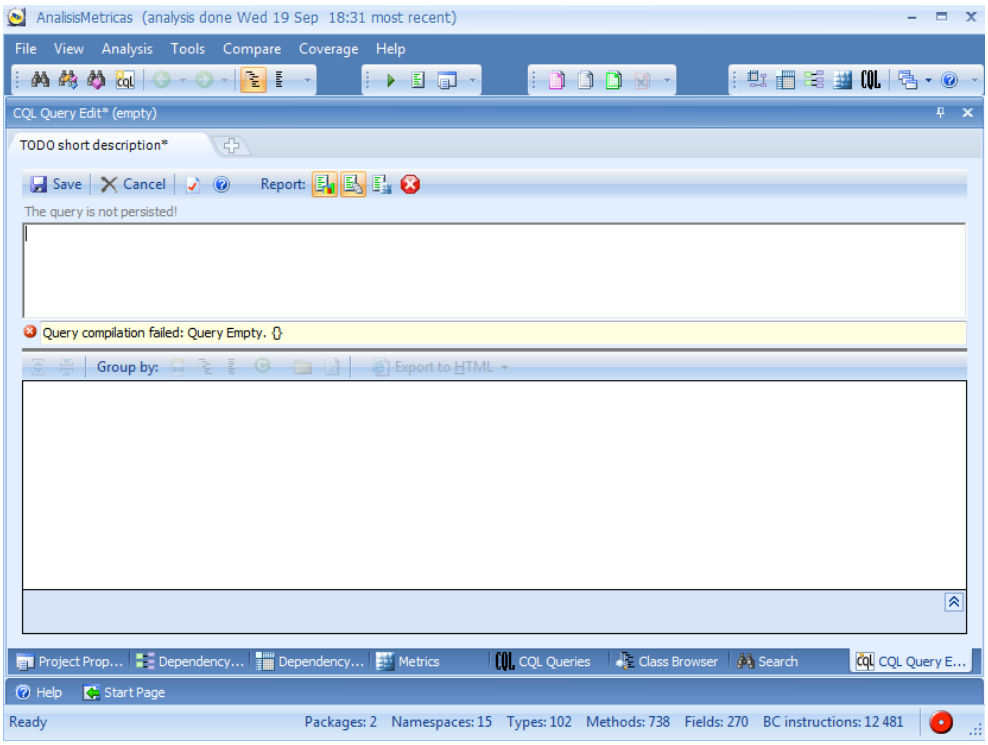


Figura 5.4 Ventana para hacer consultas CQL en JArchitect

5.1.2.Codepro Analytix

Esta herramienta corre como un plugin de eclipse, por lo cual está disponible en cualquier plataforma en la que eclipse pueda correr. Su instalación es muy sencilla y requiere un proceso estándar para instalar un plugin de eclipse.

El utilizarlo requiere tener agregados al workspace los proyectos que se quieren analizar. Después de esto, se necesita mostrar la ventana de métricas, que puede ser encontrada en el menú en Window → Show View → Metrics; esto se ilustra en la Figura 5.5.

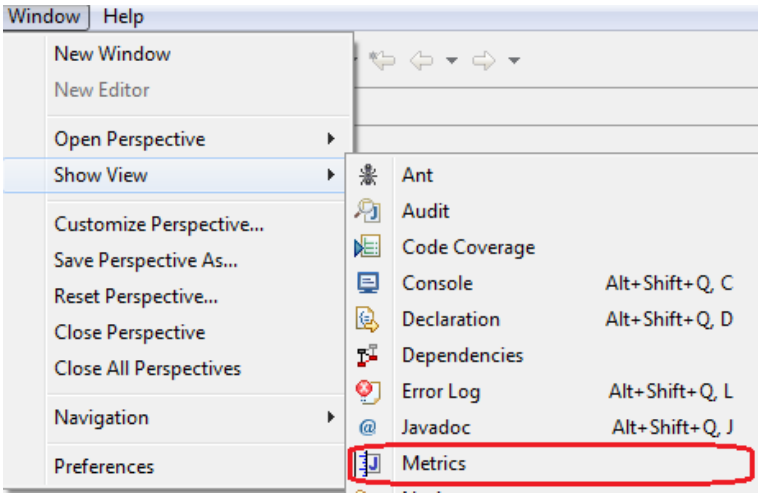


Figura 5.5 Opción de menú para mostrar ventana de métricas en Codepro Analytix

Como resultado de seleccionar esa opción de menú, se muestra la pantalla de métricas, donde se presenta la opción de seleccionar proyectos para calcular sus métricas, como se muestra en la Figura 5.6.

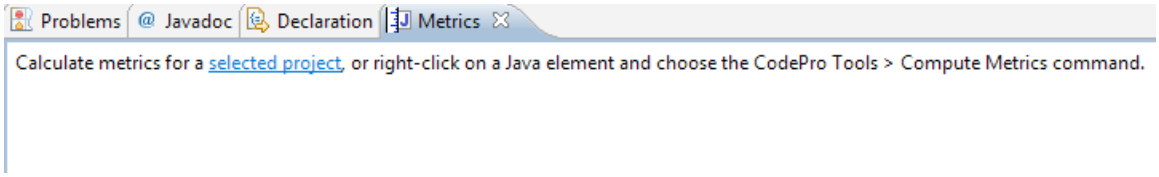


Figura 5.6 Estado inicial de ventana de métricas en Codepro Analytix

Una vez que se elige la opción de "seleccionar proyectos", se muestra una ventana donde el usuario puede elegir cuáles proyectos quiere analizar. Esto se ilustra en la Figura 5.7.

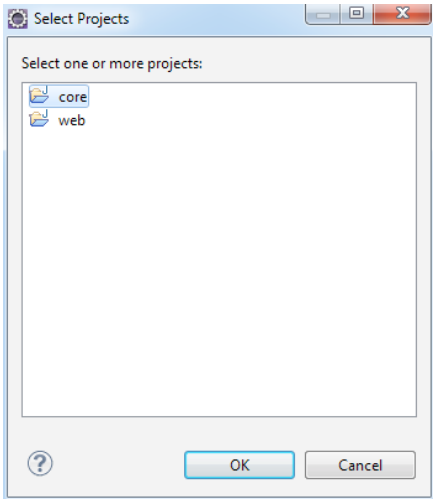


Figura 5.7 Selección de proyectos en Codepro Analytix

Después de esta acción, Codepro Analytix procede a analizar el código y muestra los resultados en la misma ventana de métricas donde pueden ser consultadas de manera fácil. Esto es ilustrado en la Figura 5.8.

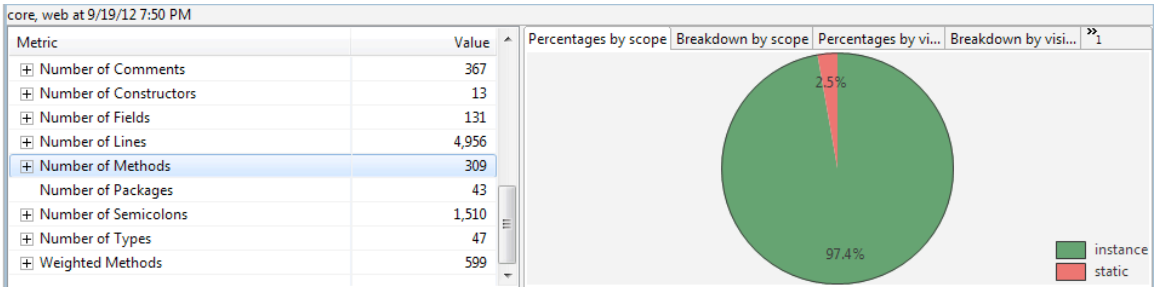


Figura 5.8 Resultado de cálculo de métricas en Codepro Analytix

5.1.3.Eclipse Metrics Plugin (Continued)

Esta herramienta es un plugin de eclipse, por lo cual puede correr en cualquier plataforma en la que eclipse pueda correr. Su instalación es muy sencilla, ya que se encuentra en los repositorios de plugins de eclipse que vienen por default en cualquier instalación de eclipse .

Para hacerla funcionar dentro de eclipse, hace falta hacer algunas configuraciones que se detallan a continuación. El primer paso es hacer que se muestre la ventana de las métricas dentro de eclipse; esto se logra en el menú de eclipse, seleccionando Window → Show View → Metrics View, como se muestra en la Figura 5.9.

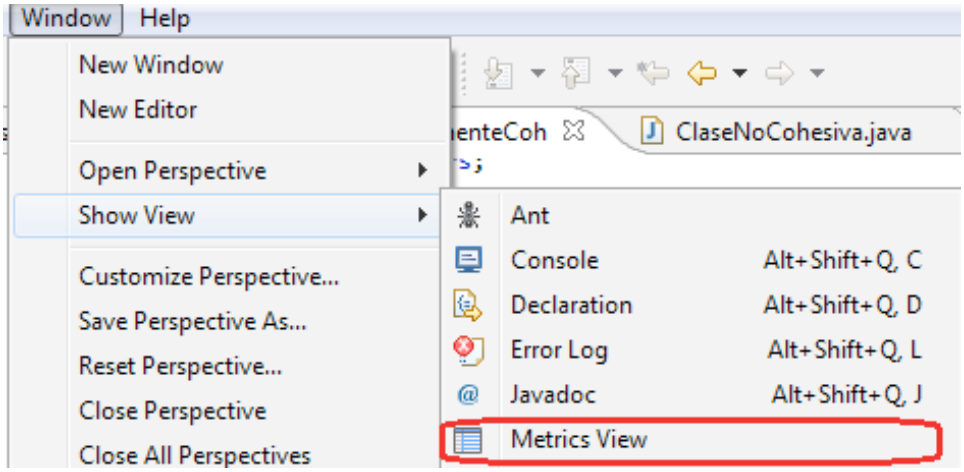


Figura 5.9 Opción de menú para mostrar ventana de métricas en Eclipse Metrics

Después de esto, ya se tiene visible la ventana de métricas en eclipse, pero no mostrará ninguna información si los proyectos que se tienen dentro del workspace no tienen activada una opción que se activa dentro del menú de propiedades del proyecto. Esto se ilustra en la Figura 5.10.

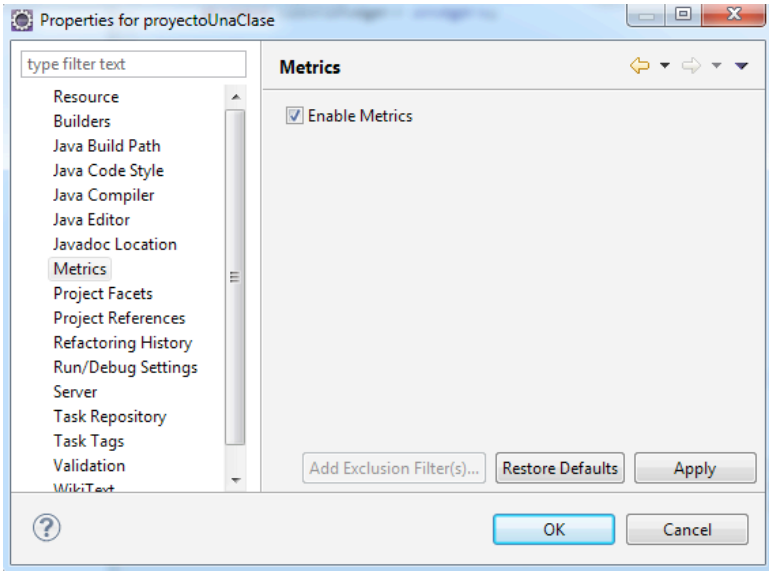


Figura 5.10 Activación de cálculo de métricas para un proyecto en Eclipse Metrics

Con esto activado, la ventana de métricas muestra resultados de sus cálculos cada vez que el usuario hace un build de los proyectos que tengan activado el cálculo de métricas, como se ilustra en la Figura 5.11.

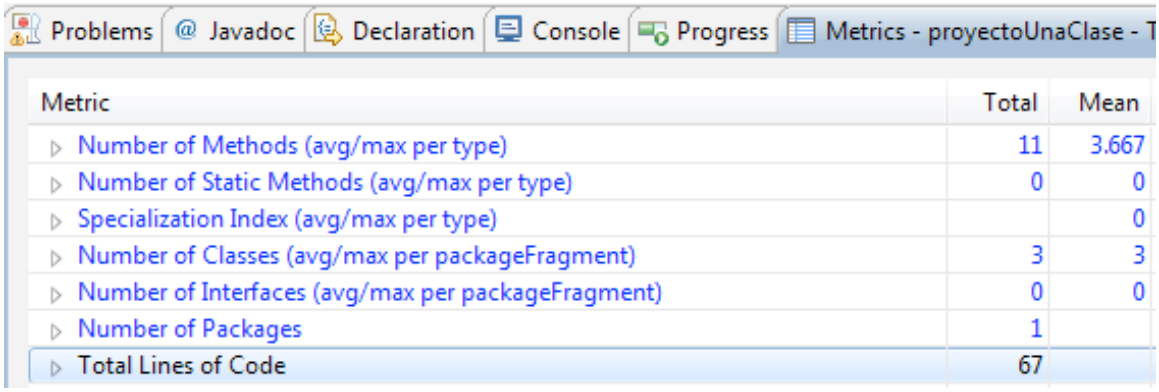


Figura 5.11 Métricas calculadas en Eclipse Metrics

5.1.4.Conclusión sobre operación de las herramientas

Las herramientas son en general fáciles de operar y un usuario que tenga experiencia usando herramientas de desarrollo de Java no tendrá problemas al utilizarlas, aunque en este rubro destaca JArchitect, porque permite analizar proyectos sin tener dependencia con ningún entorno de desarrollo específico y también porque cuenta con CQL, que permite realizar consultas poderosas.

Es importante mencionar que todas las herramientas funcionan bajo el mismo esquema de analizar todo el código del proyecto seleccionado y calcular todas las métricas que se tienen disponibles, es decir, no es posible seleccionar solo un subconjunto de métricas a ser calculadas.

5.2.Pruebas de sobrecarga

Para probar que las herramientas analizadoras de código puedan funcionar aún al ser utilizadas para procesar proyectos con una cantidad muy elevada de clases, se utilizaron las herramientas para analizar un proyecto de una complejidad considerada como alta. El propósito de esta prueba es identificar el tiempo que consumen las herramientas para analizar proyectos de magnitud considerable y en caso de que se presenten, errores como terminación abrupta del proceso de análisis o dejar de responder al quedarse corriendo ciclos infinitos.

El proyecto en cuestión cuenta con 4,876 clases, todas las herramientas fueron probadas bajo las mismas condiciones: usando el mismo código, eran el único programa ejecutándose y usaron la misma configuración de hardware. Los resultados se muestran en la Tabla 5.1.

Herramienta	Análisis terminado	Tiempo utilizado (segundos)
JArchitect	Cierto	121
Eclipse Metrics (Continued)	Cierto	1085
Codepro Analytix	Cierto	317

Tabla 5.1 Desempeño de las herramientas analizando un proyecto grande

Para poder comparar estos datos de manera más sencilla podemos analizar la Figura 5.12.

Figura 5.12 Tiempo en segundos utilizado por herramientas en analizar proyecto grande

Se puede observar que hay diferencias importantes en el tiempo que utilizan las herramientas, donde JArchitect es la más rápida y Eclipse Metrics (continued) es la más lenta.

Esto se puede explicar al notar que Eclipse Metrics realiza la recolección de los datos al momento de hacer el build, por lo cual añade más procesamiento a una acción que por sí misma ya es muy demandante en cuestión de procesador. JArchitect por su parte no se ocupa del proceso de build y deja esa parte como una tarea para el usuario. Codepro Analytix por su parte, no requiere de ningún build y solo necesita código.
Al ver estos resultados se puede decir que Eclipse Metrics no es muy recomendable para ser usado en proyectos grandes o que por lo menos hace falta tomar en cuenta estos resultados para tener algún plan de mitigación como hardware más poderoso o cálculo de las métricas en horarios de poca carga para los equipos.

5.3.Código utilizado en mediciones.

En la evaluación de estas herramientas, se utilizó para su análisis y medición un proyecto de Java que consiste en una interfaz web para controlar recursos financieros. El proyecto estaba compuesto originalmente por dos librerías:

- Core: concentra el acceso a la base de datos y la lógica de negocio, contiene 32 clases.
- Web: contiene todo lo referente a interacción con el usuario a través de páginas web, contiene 70 clases.

Este proyecto fue levemente modificado para incluir nuevas librerías, con código preparado especialmente para probar ciertas métricas. Estas librerías son:

- ProyectoCohesivo: Un proyecto hecho para tener una cohesión relacional alta.
- ProyectoRelativamenteCohesivo: Un proyecto hecho para tener una cohesión relacional media.
- ProyectoNoCohesivo: Un proyecto hecho para tener una cohesión relacional baja.
- ClasesCohesivas: Un proyecto que contiene clases con distintos niveles de falta de cohesión de los métodos.

5.4.Evaluación métrica por métrica

A continuación se presenta una comparación del desempeño del cálculo de métricas. La metodología empleada consiste en hacer un cálculo manual de la métrica de algún segmento de código. El resultado de este cálculo es tomado como la línea base con la que se compararán los resultados que generen las herramientas analizadoras de código. A partir de esta comparación se establece si la herramienta realiza correctamente el cálculo y en caso de que no, se busca una explicación para esta falta de precisión.

Para representar con un valor numérico la diferencia de valores entre la línea base y los resultados obtenidos por las herramientas en cada caso, se calcula el porcentaje de diferencia [10].

5.4.1.Acoplamiento aferente a nivel clase

Para evaluar el desempeño de esta métrica se parte primero de un cálculo manual del acoplamiento aferente de algunas clases de el proyecto, después se utiliza a la única herramienta que calcula esta métrica JArchitect.

En la Figura 5.13 se muestra la pantalla que muestra JArchitect con los resultados para acoplamiento aferente en las clases que se seleccionaron para probar.

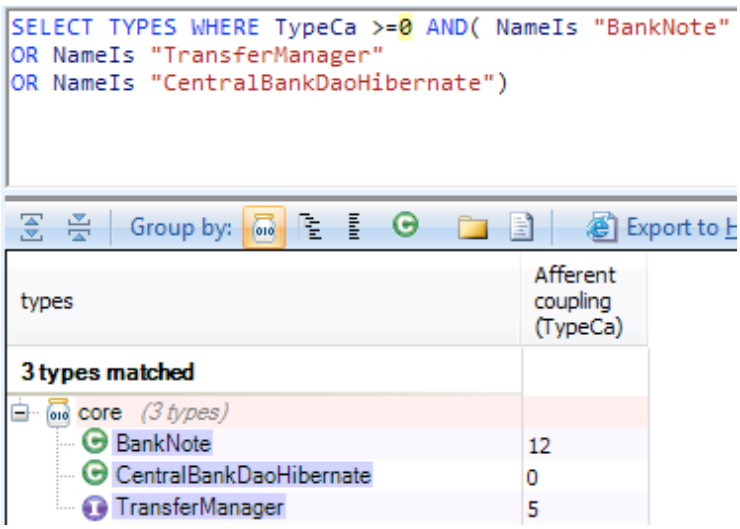


Figura 5.13 Acoplamiento aferente a nivel clase en JArchitect

Se comparan los resultados en la Tabla 5.2.

Clase analizada	Cálculo Manual	JArchitect	Porcentaje de error
CentralBankDaoHibernate	0	0	0%
TransferManager	5	5	0%
BankNote	11	12	9.09%

Tabla 5.2 Comparación de valores de acoplamiento aferente a nivel clase

Estos datos no coinciden en su totalidad debido a que JArchitect contó mal el número de referencias para la clase BankNote. Para revisar esto a mayor detalle se introduce un query CQL que es mostrado junto con sus resultados en la Figura 5.14.

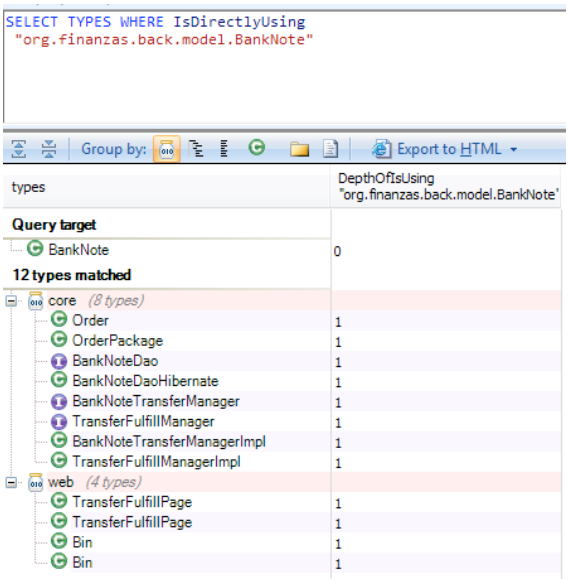


Figura 5.14 Acoplamiento aferente de clase BankNote según JArchitect

En la Figura 5.14 se puede apreciar que la sumatoria tiene dos referencias repetidas TransferFulfillPage y Bin. También en la misma librería web falta una referencia que sí fue encontrada en el cálculo manual, la cual tiene el nombre TransferRequestPage. El porqué de estas referencias duplicadas y el ignorar una que sí existe no está claro, pero se puede asumir que hay errores en los métodos de conteo y también es probable que el uso de archivos de bytecode de Java cause que las referencias sean contadas de manera incorrecta.

Por lo anterior se puede declarar esta métrica como no confiable, ya que no pudo calcular correctamente todos los valores.

5.4.2.Acoplamiento eferente a nivel clase

Para evaluar el desempeño del cálculo de esta métrica, se parte primero de un cálculo manual del acoplamiento eferente de algunas clases del proyecto; después se utiliza a la única herramienta que calcula esta métrica :JArchitect.
En la Figura 5.15 se muestra la pantalla que muestra JArchitect con los resultados para acoplamiento eferente en las clases que se seleccionaron para probar.

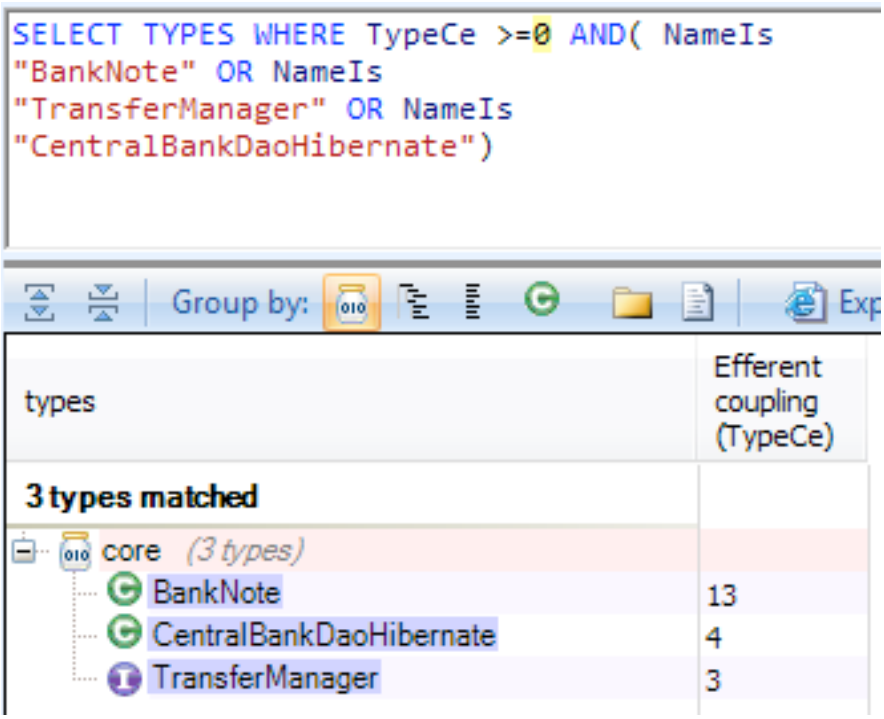


Figura 5.15 Acoplamiento eferente a nivel clase en JArchitect

Se comparan los resultados en la Tabla 5.3.

Clase analizada	Cálculo Manual	JArchitect	Porcentaje de error

CentralBankDaoHibernate	4	4	0%
TransferManager	3	3	0%
BankNote	14	13	7.14%

Tabla 5.3 Comparación de valores de acoplamiento eferente a nivel clase

Estos datos muestran que JArchitect no realizó el conteo de relaciones eferentes correctamente, ya que para la clase BankNote le faltó una referencia eferente. Para saber cuáles son las referencias que está tomando usamos otro query CQL como se muestra en la Figura 5.16.

```
SELECT TYPES WHERE IsDirectlyUsedBy
  "org.finanzas.back.model.BankNote"
```

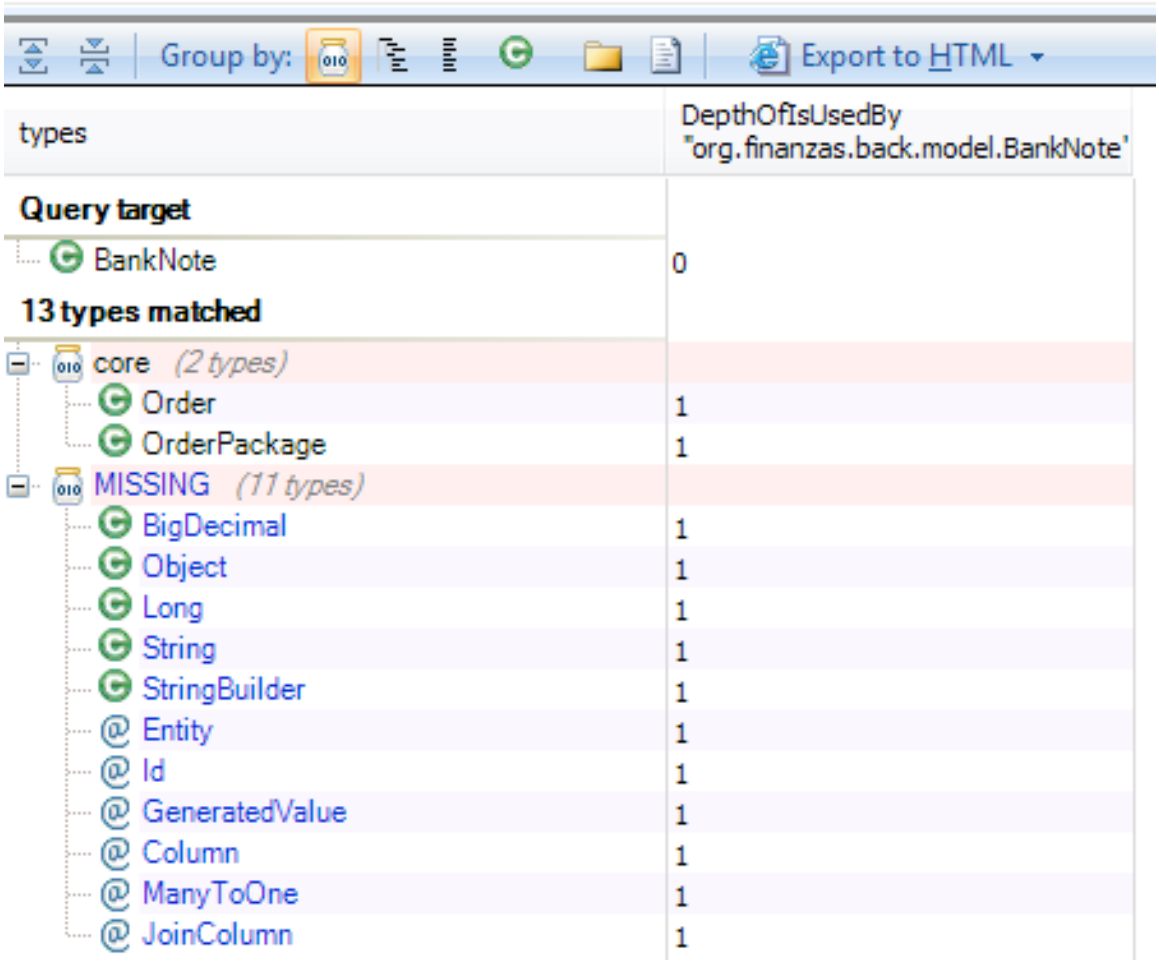


Figura 5.16 Acoplamiento Eferente a nivel clase según JArchitect

Al cotejar esta lista de referencias con las que se encontraron manualmente, se puede notar que la referencia faltante se encuentra en esta línea de código:

@GeneratedValue(strategy = GenerationType.AUTO)

En esta línea de código el tipo enum GenerationType no es tomado en cuenta por JArchitect, esto demuestra que esta herramienta no cuenta a los tipos enum como parte de las referencias eferentes, lo cual es un error puesto que un enum es una clase de Java que hereda de Object, por lo cual debe de ser contada en las referencias eferentes. Sin embargo los tipos enum en la mayoría de los casos puede que no sean tan importantes para el cálculo de esta métrica, puesto que solo contienen valores constantes, no tienen funcionalidad compleja y son poco propensos a cambios. Esto hace que los cálculos de JArchitect para acoplamiento eferente sean considerados confiables con un pequeño inconveniente. Esta interpretación puede variar para distintos proyectos y se considera válida para el común denominador de los proyectos de Java pero puede haber proyectos que sí necesiten incluir a los tipos enum en su cálculo de métricas de acoplamiento.

5.4.3.Acoplamiento aferente a nivel librería

Para evaluar el desempeño del cálculo de esta métrica se parte primero de un cálculo manual del acoplamiento aferente de dos librerías que componen un proyecto web, después de esto, se utilizaron las herramientas JArchitect y Codepro Analytix para calcular estas métricas. Los resultados de JArchitect pueden ser vistos en la Figura 5.17 y los resultados de Codepro Analytix en la Figura 5.18.

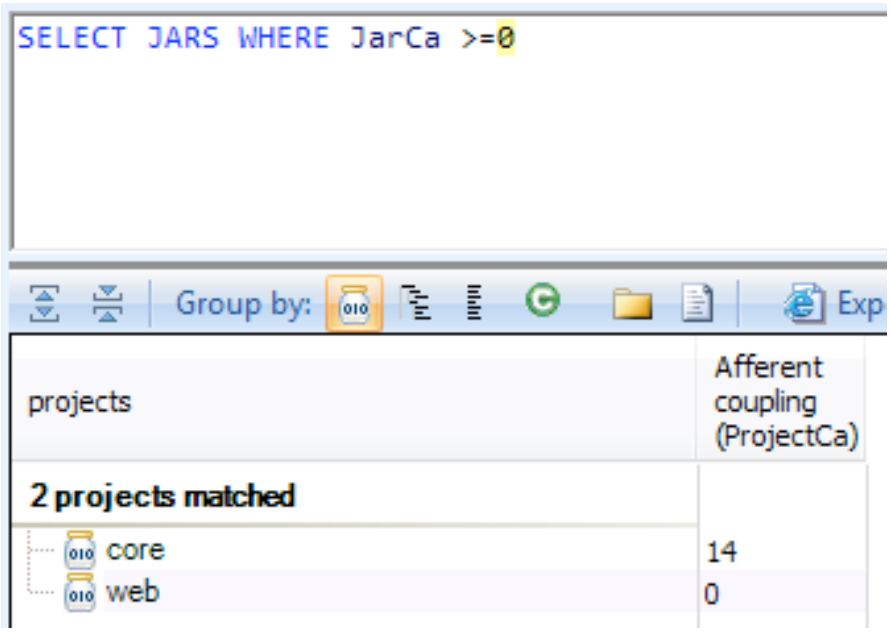


Figura 5.17 Acoplamiento aferente a nivel librería en JArchitect

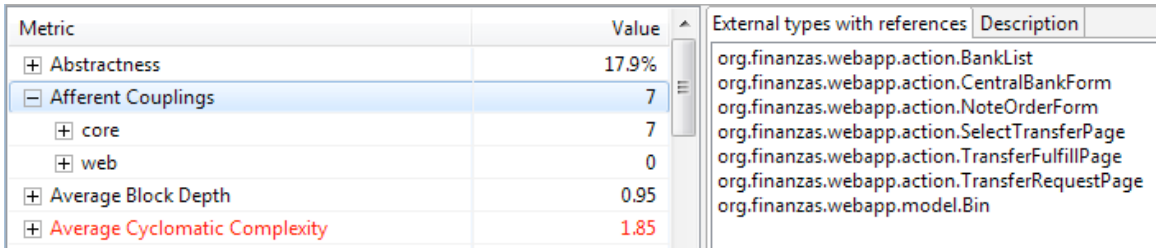


Figura 5.18 Acoplamiento aferente a nivel librería en Codepro Analytix

Se comparan los resultados en la Tabla 5.4.

Librería analizada	Cálculo Manual	JArchitect	Porcentaje de error	Codepro Analytix	Porcentaje de error
Core	7	14	100%	7	0%
Web	0	0	0%	0	0%

Tabla 5.4 Comparación de valores de acoplamiento aferente a nivel librería

Se puede observar en estos datos que los de JArchitect son incorrectos para la librería Core, esto parece obedecer a que realizó un conteo doble de las clases que hacen referencia y se sustenta al ver los resultados de un query que busca las referencias de la librería Web a la librería Core como se ilustra en la Figura 5.19.

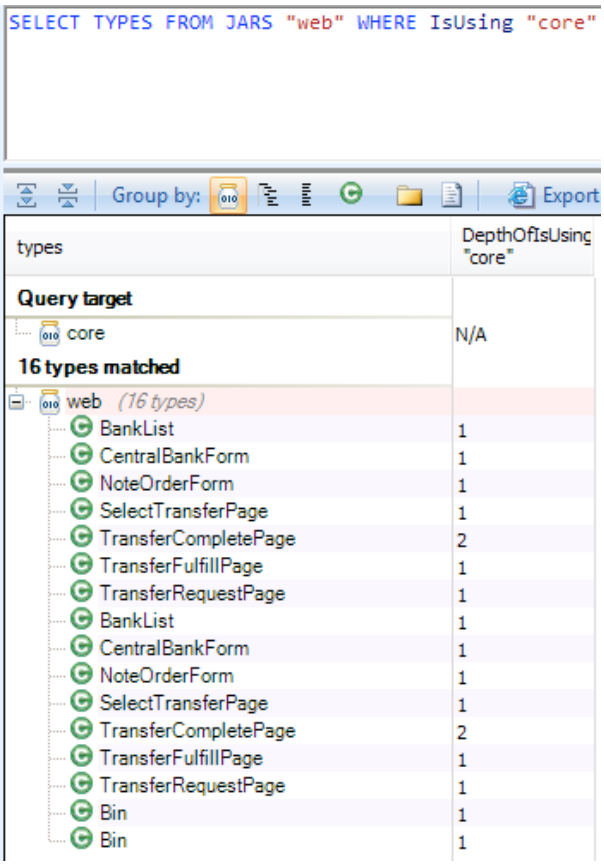


Figura 5.19 Acoplamiento aferente a nivel librería de Core según JArchitect

El resultado de este query muestra una lista de clases con elementos duplicados, además de TransferCompletePage que realmente no hace referencia a Core. Esto indica que existe un bug que provoca que el conteo de referencias se doble bajo ciertas circunstancias. Al tomar esto en cuenta se puede decir que los cálculos de acoplamiento aferente a nivel librería no son confiables en JArchitect y en Codepro Analytix sí lo son.

5.4.4.Acoplamiento eferente a nivel librería

Para evaluar el desempeño del cálculo de esta métrica, se parte primero de un cálculo manual del acoplamiento eferente de dos librerías que componen el proyecto, después de esto se utilizaron las herramientas JArchitect y Codepro Analytix para calcular estas métricas. Los resultados de JArchitect pueden ser vistos en la Figura 5.20 y los resultados de Codepro Analytix en la Figura 5.21.

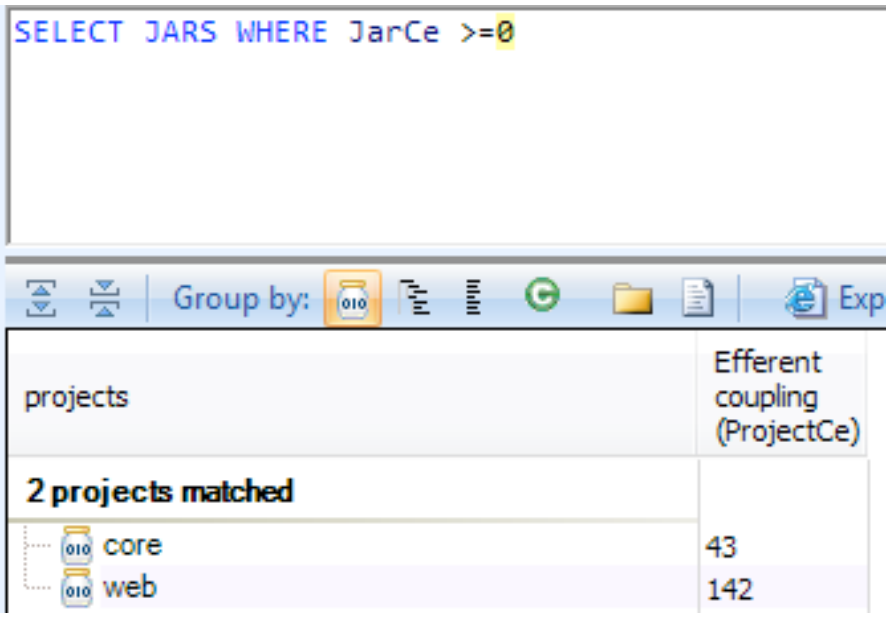


Figura 5.20 Acoplamiento eferente a nivel librería en JArchitect

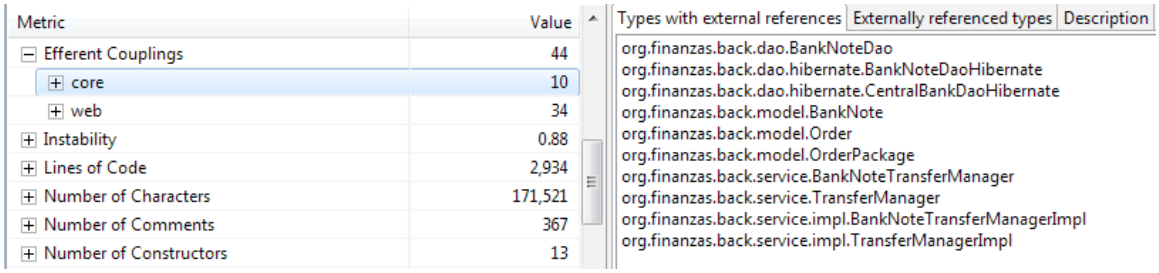


Figura 5.21 Acoplamiento eferente a nivel librería en Codepro Analytix

Librería analizada	Cálculo Manual	JArchitect	Porcentaje de error	Codepro Analytix	Porcentaje de error
Core	46	43	-6.52%	30	-34.78%
Web	142	142	0%	34	-76.05%

Tabla 5.5 Comparación de valores de acoplamiento eferente a nivel librería

Al analizar los datos de la Tabla 5.5 se muestra que ninguna de las dos herramientas pudieron calcular correctamente el acoplamiento eferente a nivel librería, cada herramienta tiene una razón en particular para esta falla:

JArchitect: estuvo cerca de los números por que su única falla fue el no contar las referencias a tipos enum, esta puede ser interpretada como una falla no tan importante porque los tipos enum aunque sean clases, carecen de funcionalidad compleja dentro de sí y son poco propensos a cambios.

Codepro Analytix: Muestra diferencias bastante amplias al comparar sus resultados con los números del cálculo manual, esto se debe a que tiene una forma incorrecta

de calcular el acoplamiento eferente a nivel clase, lo calcula contando todas las clases que tienen referencias a librerías externas. Esto es incorrecto ya que lo correcto debería ser contar el número de clases externas a las que se hace referencia dentro de la librería.

El número que proporciona es muy poco útil, ya que tiene el mismo valor para esta fórmula una clase que hace referencia a 100 clases fuera de esta librería que una clase que hace una sola referencia. Por otro lado encontrar una clase sin referencias externas es algo casi imposible puesto que una clase con cero referencias externas es posible que realmente no haga nada puesto que no hace referencia los tipos básicos de java ni a librerías externas.

Sin embargo es importante mencionar que parece tener los elementos para hacer bien el cálculo con un cambio ligero, ya que en una pestaña llamada "Externally referenced types" muestra un listado que tiene las referencias externas que deberían de sumarse para tener el acoplamiento eferente, esto es mostrado en la Figura 5.22. Esto indica que los datos necesarios para calcular la métrica están presentes en esta herramienta solo no están siendo utilizados correctamente.

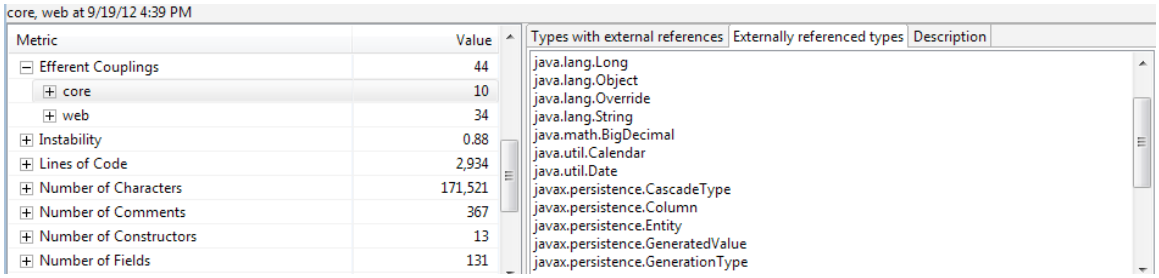


Figura 5.22 Tipos externos referenciados en Codepro Analytix

Al revisar estos resultados se puede concluir que los cálculos de acoplamiento eferente a nivel librería de JArchitect no son totalmente correctos porque no incluyen a los tipos enum, pero pueden ser utilizados si es que esa omisión es considerada despreciable, se considera que para la mayoría de los casos esta omisión es despreciable. Por otro lado los cálculos de Codepro Analytix están muy alejados de la definición formal de la métrica, por lo cual deben de tomarse como completamente erróneos.

5.4.5.Inestabilidad

Para evaluar el desempeño del cálculo de esta métrica se parte primero de un cálculo manual de la inestabilidad de dos librerías que componen el proyecto, después de esto se utilizaron las herramientas JArchitect y Codepro Analytix para calcular estas métricas. Como se demostró previamente estas dos herramientas no obtuvieron los valores correctos para acoplamiento eferente y aferente a nivel

librería por lo que para evaluar únicamente el cálculo de esta métrica se verificará que se utilicen esos valores de manera correcta en la fórmula de la inestabilidad.

Cálculo manual

En la Tabla 5.6 Valores de inestabilidad cálculo manual se muestra el valor de la inestabilidad de las librerías usando los valores de acoplamiento aferente y acoplamiento eferente calculados manualmente en la Fórmula 3.1 que se definió previamente.

Librería analizada	Cálculo manual
Core	0.867
Web	1

Tabla 5.6 Valores de inestabilidad cálculo manual

JArchitect

A continuación se muestra en la Figura 5.23 los valores obtenidos por JArchitect para la métrica de inestabilidad.

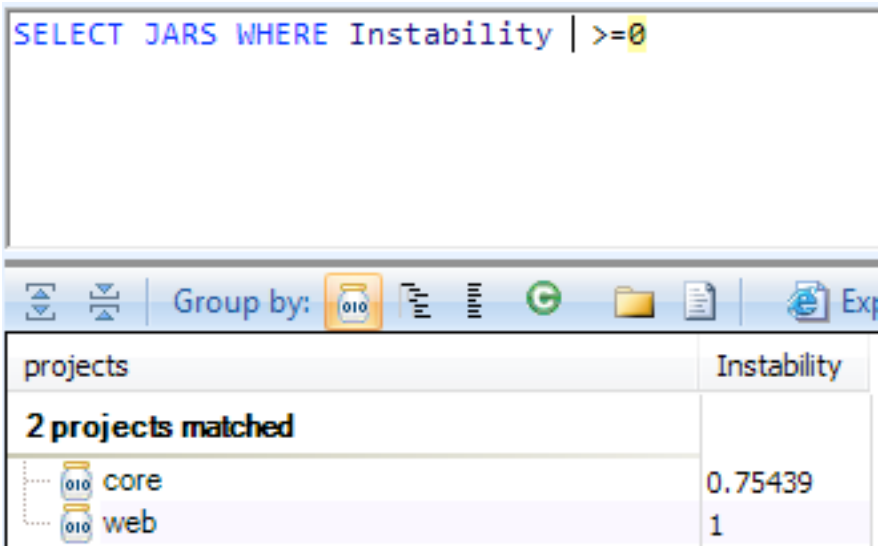


Figura 5.23 Inestabilidad en JArchitect

A continuación se presenta la Tabla 5.7 donde se comparan los valores obtenidos por JArchitect y un cálculo manual usando la Fórmula 3.1 con los valores de acoplamiento aferente y acoplamiento eferente que obtuvo JArchitect, los resultados coinciden.

Librería analizada	Cálculo manual	JArchitect
Core	0.754	0.754
Web	1	1

Tabla 5.7 Valores de inestabilidad en JArchitect

Codepro Analytix

A continuación se muestra en la Figura 5.24 los valores obtenidos por Codepro Analytix para la métrica de inestabilidad.

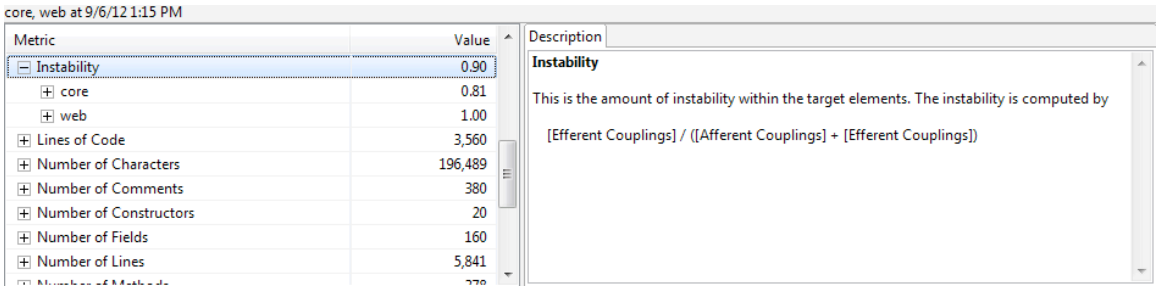


Figura 5.24 Inestabilidad en Codepro Analytix

En la Tabla 5.8 se presenta la comparación de los valores obtenidos por Codepro Analytix y un cálculo manual usando la Fórmula 3.1 con los valores de acoplamiento aferente y acoplamiento eferente que obtuvo Codepro Analytix, los resultados coinciden.

Librería analizada	Cálculo manual	Codepro Analytix
Core	0.81	0.81
Web	1	1

Tabla 5.8 Valores de inestabilidad en Codepro Analytix

Se puede constatar que las dos herramientas lograron calcular correctamente el valor de la inestabilidad usando como parámetros sus valores de acoplamiento aferente y acoplamiento eferente. Aunque los cálculos sean correctos desde su perspectiva, no pueden tomarse como válidos pues son calculados en base a valores de acoplamiento aferente y eferente poco confiables, la diferencia de resultados se muestra en la Tabla 5.9.

Librería analizada	Cálculo Manual	JArchitect	Porcentaje de error	Codepro Analytix	Porcentaje de error
Core	0.867	0.754	-13.03%	0.81	-6.57%
Web	1	1	0%	1	0%

Tabla 5.9 Comparación de valores de inestabilidad

5.4.6.Nivel de abstracción

Para evaluar el desempeño del cálculo de esta métrica se parte primero de un cálculo manual de el nivel de abstracción de dos librerías que componen el proyecto usando la Fórmula 3.2, después de esto se utilizaron las herramientas JArchitect y

Codepro Analytix para calcular estas métricas. Los resultados de JArchitect pueden ser vistos en la Figura 5.25 y los resultados de Codepro Analytix en la Figura 5.26.

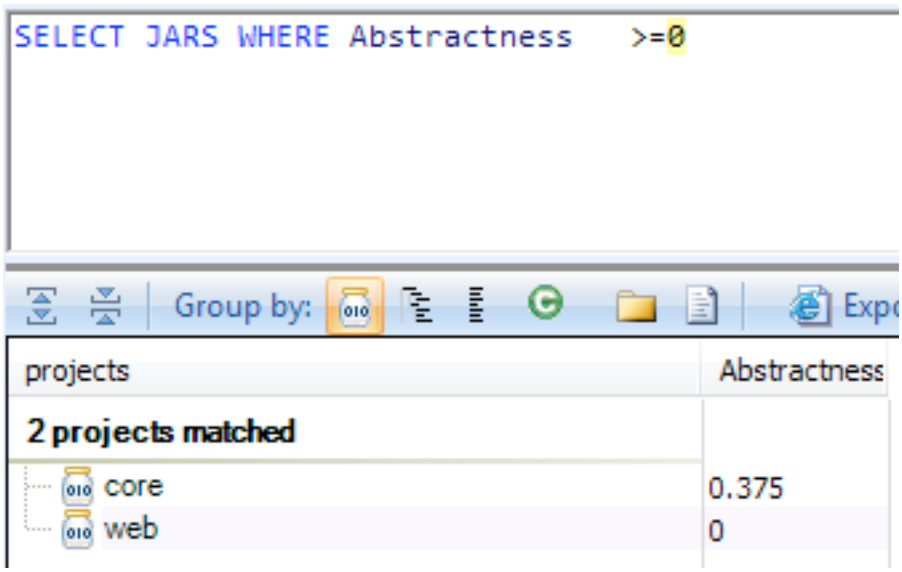


Figura 5.25 Nivel de abstracción en JArchitect

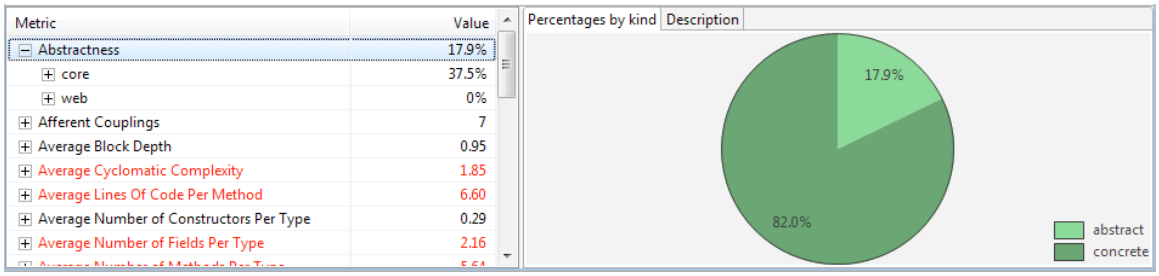


Figura 5.26 Nivel de abstracción en Codepro Analytix

Librería analizada	Cálculo Manual	JArchitect	Porcentaje de error	Codepro Analytix	Porcentaje de error
Core	0.375	0.375	0%	0.375	0%
Web	0	0	0%	0	0%

Tabla 5.10 Comparación de valores de nivel de abstracción

En la Tabla 5.10 se puede observar que los resultados coinciden totalmente, esto no es raro puesto que esta métrica requiere un análisis relativamente sencillo. El cálculo de esta métrica en las dos herramientas puede ser considerado como confiable.

5.4.7.Distancia de la secuencia principal normalizada

Para evaluar el desempeño del cálculo de esta métrica se parte primero de un cálculo manual de la secuencia principal normalizada de dos librerías que

componen el proyecto usando la Fórmula 3.4, después de esto se utilizaron las herramientas JArchitect y Codepro Analytix para calcular estas métricas. Como se demostró previamente estas dos herramientas no obtuvieron los valores correctos para acoplamiento eferente y aferente, por ende tampoco tienen un valor correcto para la inestabilidad de las librerías, por lo que para evaluar únicamente el cálculo de esta métrica se verificará que se utilicen esos valores de manera correcta en la fórmula de la distancia de la secuencia principal normalizada.

Cálculo manual

En la Tabla 5.11 se muestra el valor de la distancia de la secuencia principal normalizada de las librerías usando los valores de inestabilidad y nivel de abstracción calculados manualmente.

Librería analizada	Cálculo manual
Core	0.242
Web	0

Tabla 5.11 Distancia de la secuencia principal normalizada cálculo manual

JArchitect

A continuación se muestra en la Figura 5.27 los valores obtenidos por JArchitect para la métrica de distancia de la secuencia principal normalizada.

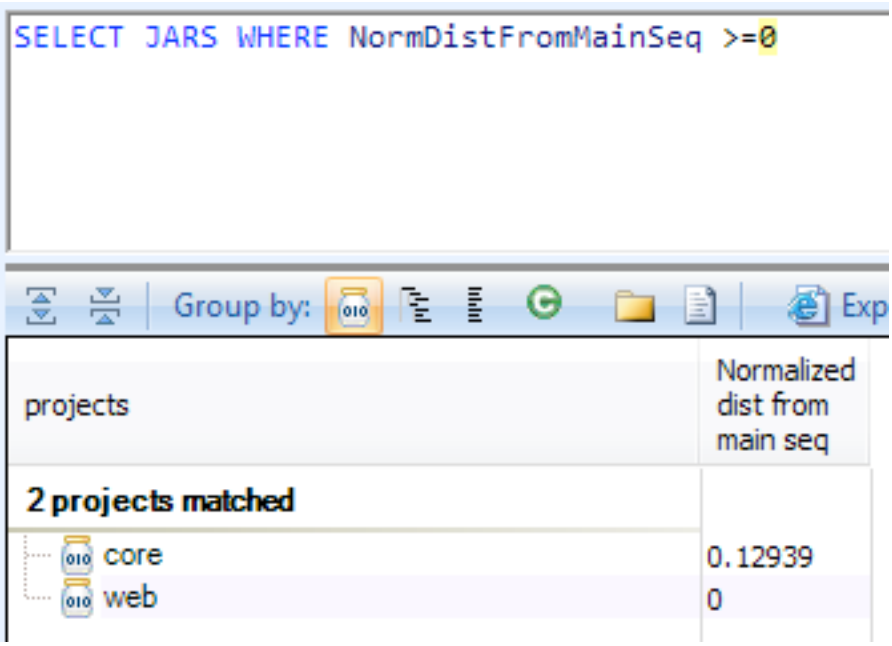


Figura 5.27 Distancia de la secuencia principal normalizada en JArchitect

Librería analizada	Cálculo manual	JArchitect

Core	0.129	0.129
Web	0	0

Tabla 5.12 Valores de distancia de la secuencia principal normalizada en JArchitect

Codepro Analytix

A continuación se muestra en la Figura 5.28 los valores obtenidos por Codepro Analytix para la métrica de distancia de la secuencia principal normalizada.

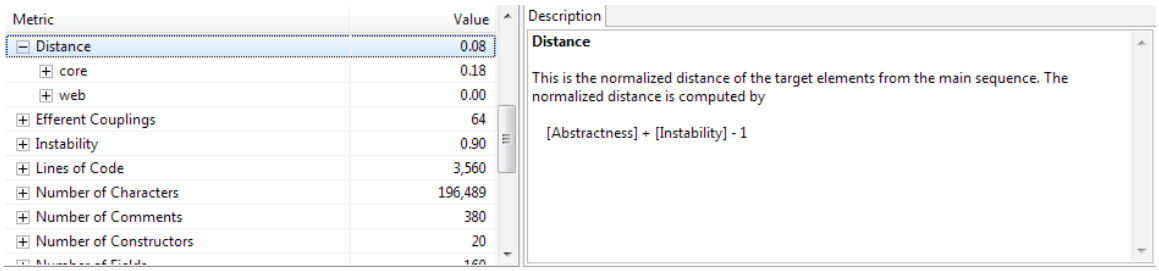


Figura 5.28 Valores de distancia de la secuencia principal normalizada en Codepro Analytix

Librería analizada	Cálculo manual	Codepro Analytix
Core	0.18	0.18
Web	0	0

Tabla 5.13 Valores de distancia de la secuencia principal normalizada en Codepro Analytix

Se puede apreciar en la Tabla 5.12 y en la Tabla 5.13 que todas las herramientas que lograron calcular correctamente la distancia de la secuencia principal normalizada, aplicaron correctamente la Fórmula 3.4, aunque el uso de los valores incorrectos que fueron calculados en acoplamiento aferente y eferente no permite que estas métricas sean consideradas confiables, la diferencia de resultados se muestra en la Tabla 5.14.

Librería analizada	Cálculo Manual	JArchitect	Porcentaje de error	Codepro Analytix	Porcentaje de error
Core	0.242	0.129	-46.69%	0.18	-25.61%
Web	0	0	0%	0	0%

Tabla 5.14 Comparación de valores de distancia de la secuencia principal normalizada

5.4.8.Falta de cohesión en los métodos

Para evaluar el desempeño del cálculo de esta métrica se parte primero de un cálculo manual de la falta de cohesión de los métodos sobre algunas clases de muestra que se prepararon con este propósito. Para esto se usa la Fórmula 3.5, después se calcula la misma métrica pero con la herramienta JArchitect.

Las clases se prepararon para probar tres situaciones totalmente distintas, a continuación una descripción de ellas:
ClaseCohesiva: es totalmente cohesiva ya que cuenta con una sola variable de clase que es accedida por todos los métodos.
ClaseNoCohesiva: Es totalmente opuesta a la anterior, puesto que tiene cinco variables y cinco métodos que acceden cada uno a una sola variable sin repetir variable, por lo cual es totalmente no cohesiva.
ClaseParcialmenteCohesiva: Una clase que es un punto medio entre las dos anteriores, tiene una variable de clase que es accedida por 2 métodos y no es accedida por otros dos métodos.
En la Figura 5.29 se muestran los resultados que obtiene JArchitect para estas clases.

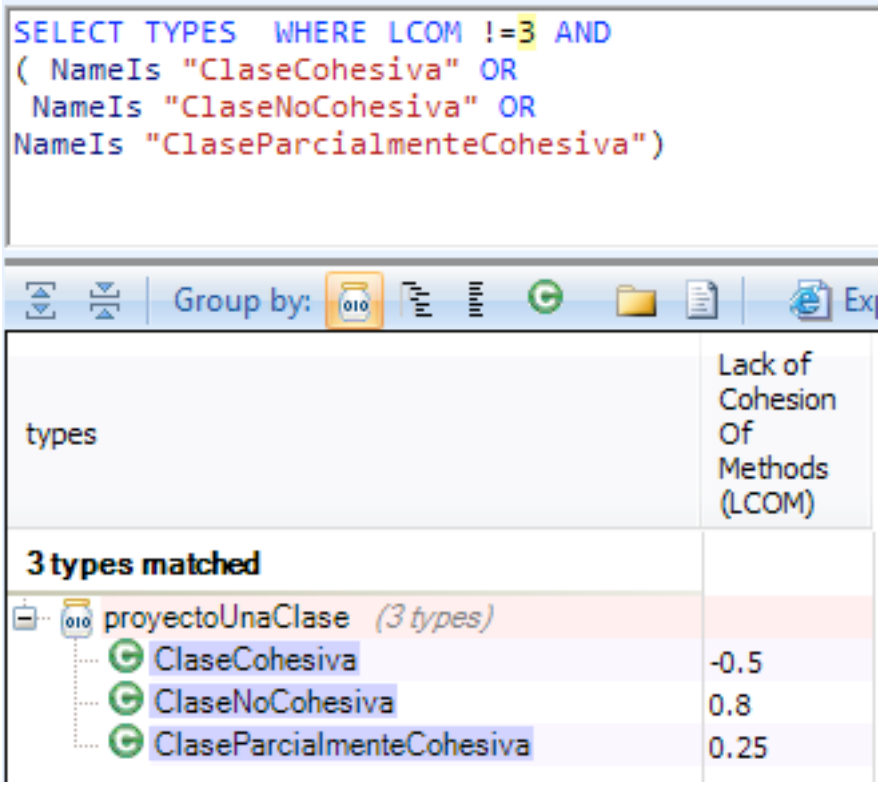


Figura 5.29 Falta de cohesión en los métodos en JArchitect

Clase analizada	Cálculo Manual	JArchitect	Porcentaje de error
ClaseCohesiva	0	-0.5	Indefinido
ClaseNoCohesiva	10	0.8	-92%
ClaseParcialmenteCohesiva	4	0.25	-93.75%

Tabla 5.15 Comparación de valores de falta de cohesión en los métodos

Al revisar los datos de la Tabla 5.15 se puede notar que los resultados son totalmente diferentes en todos los casos, esto se debe a que JArchitect utiliza una fórmula distinta de la que fue definida por Chidamber y Kemerer, esto explica

porqué valores tan distintos. También se nota que hay un valor donde el porcentaje de error es indefinido esto se debe a que al aplicar la fórmula resulta una división entre cero. La fórmula utilizada por JArchitect es definida de la siguiente manera [4]:

$$LCOM = 1 - (sum(MF) / M*F)$$

Fórmula 5.1 Falta de cohesión en los métodos (LCOM) según JArchitect

Donde :
M es el número de métodos en una clase.
F es el número de variables de instancia en la clase.
MF es el número de métodos en una clase que usan una variable de instancia en particular.
Sum(MF) es la sumatoria de todos los valores MF para todas las variables de instancia en la clase.

Su rango de valores es [0-1] [4].

Al ver que la fórmula es totalmente distinta a la que se tomó como la fórmula correcta por este documento se puede decir que no es posible evaluar el desempeño de esta fórmula, sin embargo es pertinente mencionar que uno de los valores calculados está fuera del rango que indica su misma documentación, por lo cual se puede inferir que hay un error en la implementación de la fórmula que usa JArchitect.

5.4.9.Falta de cohesión en los métodos Henderson Sellers

Para evaluar el desempeño del cálculo de esta métrica se parte primero de un cálculo manual de la falta de cohesión de los métodos Henderson Sellers sobre las mismas clases que fueron utilizadas para calcular la falta de cohesión en los métodos usando la Fórmula 3.6, después se calcula la misma métrica con las herramienta JArchitect y Eclipse Metrics continued , los resultados que obtiene estas herramienta pueden ser vistos en la Figura 5.30 y la Figura 5.31.

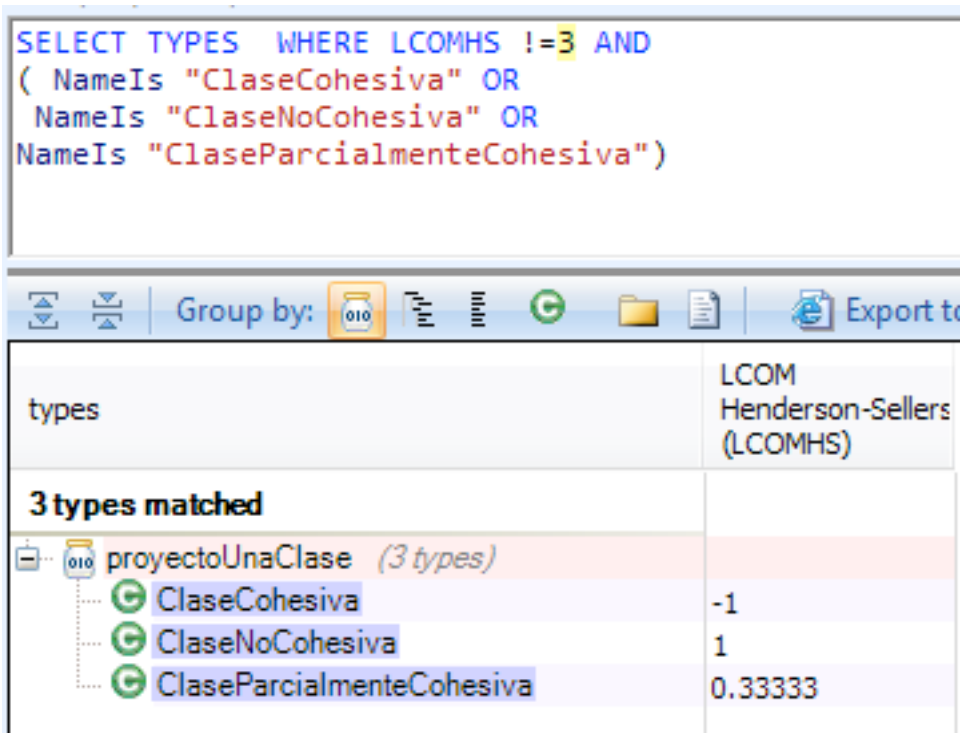
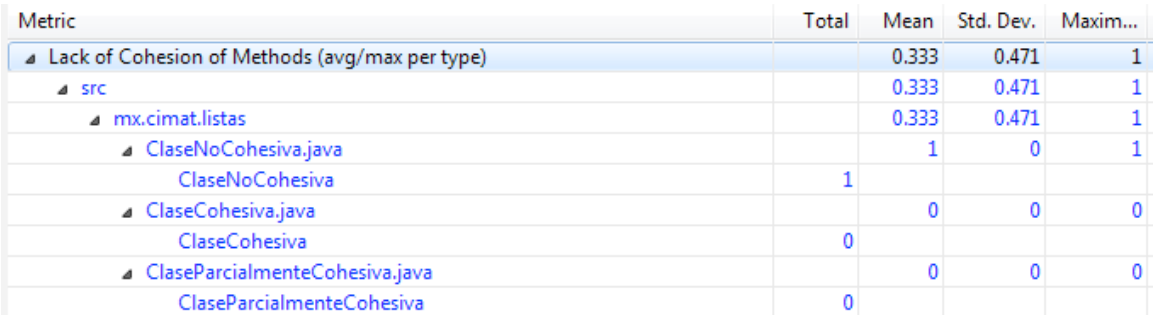


Figura 5.30 Falta de cohesión en los métodos Henderson Sellers en JArchitect



Metric	Total	Mean	Std. Dev.	Maxim...
Lack of Cohesion of Methods (avg/max per type)		0.333	0.471	1
src		0.333	0.471	1
mx.cimat.listas		0.333	0.471	1
ClaseNoCohesiva.java		1	0	1
ClaseNoCohesiva	1			
ClaseCohesiva.java		0	0	0
ClaseCohesiva	0			
ClaseParcialmenteCohesiva.java		0	0	0
ClaseParcialmenteCohesiva	0			

Figura 5.31 Falta de cohesión en los métodos Henderson sellers en Eclipse Metrics Continued

Clase analizada	Cálculo Manual	JArchitect	Porcentaje de error	Eclipse Metrics	Porcentaje de error
ClaseCohesiva	0	-1	Indefinido	0	0%
ClaseNoCohesiva	1	1	0%	1	0%
ClaseParcialmenteCohesiva	0.66	0.33	-50%	0	-100%

Tabla 5.16 Comparación de valores de falta de cohesión en los métodos Henderson Sellers

Al revisar los datos obtenidos en la Tabla 5.16 se puede notar que solo hay un resultado correcto en JArchitect y dos en Eclipse Metrics Continued, las dos herramientas fallaron en calcular correctamente esta métrica. También se nota que hay un valor donde el porcentaje de error es indefinido esto se debe a que al aplicar la fórmula resulta una división entre cero. Se puede inferir que tienen errores en sus fórmulas para calcular esta métrica y en el caso de JArchitect, puede tener

problemas ocasionados por métodos o variables agregados en tiempo de compilación. Las dos herramientas fallaron de manera importante ya que JArchitect regresó un valor fuera del rango de la fórmula y por su parte Eclipse Metrics continued no pudo diferenciar entre una clase totalmente cohesiva y otra parcialmente cohesiva. Las dos herramientas se pueden considerar como no confiables en el cálculo de esta métrica.

5.4.10.Cohesión Relacional

Para evaluar el desempeño de esta métrica se parte primero de un cálculo manual de la cohesión relacional de algunos proyectos, después se utiliza a la única herramienta que calcula esta métrica, JArchitect. Los resultados obtenidos por esta herramienta pueden ser vistos en la Figura 5.32, la Figura 5.33 y la Figura 5.34 .

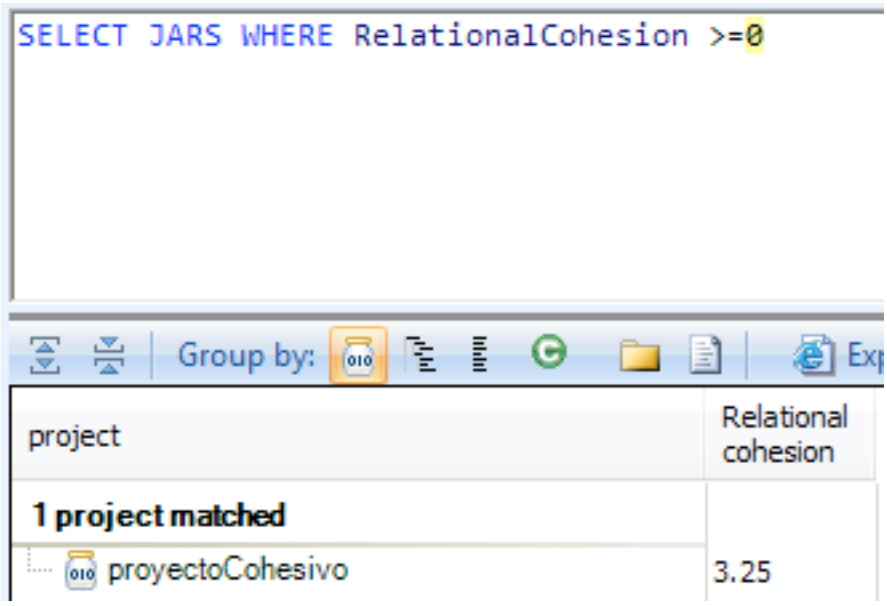


Figura 5.32 Cohesión relacional para ProyectoCohesivo en JArchitect

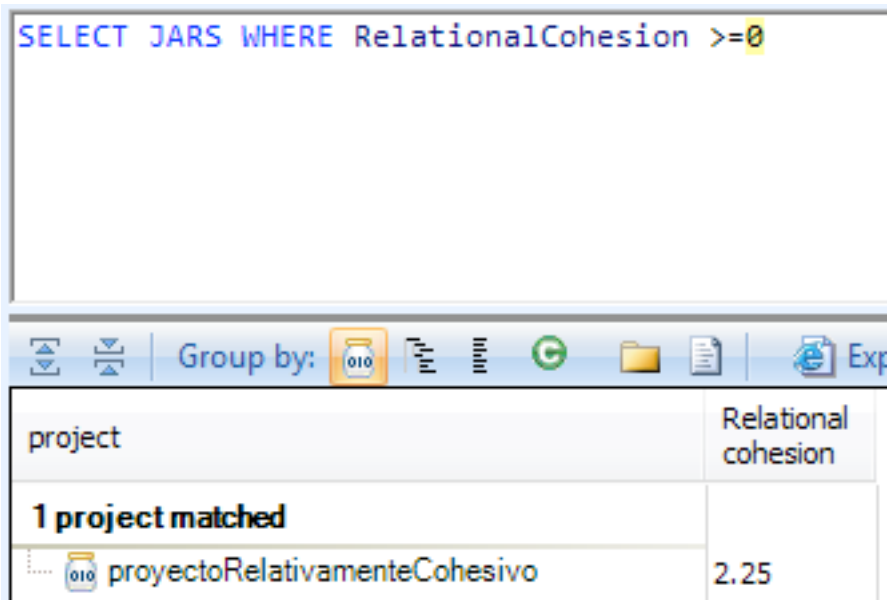


Figura 5.33 Cohesión relacional para ProyectoRelativamenteCohesivo en JArchitect

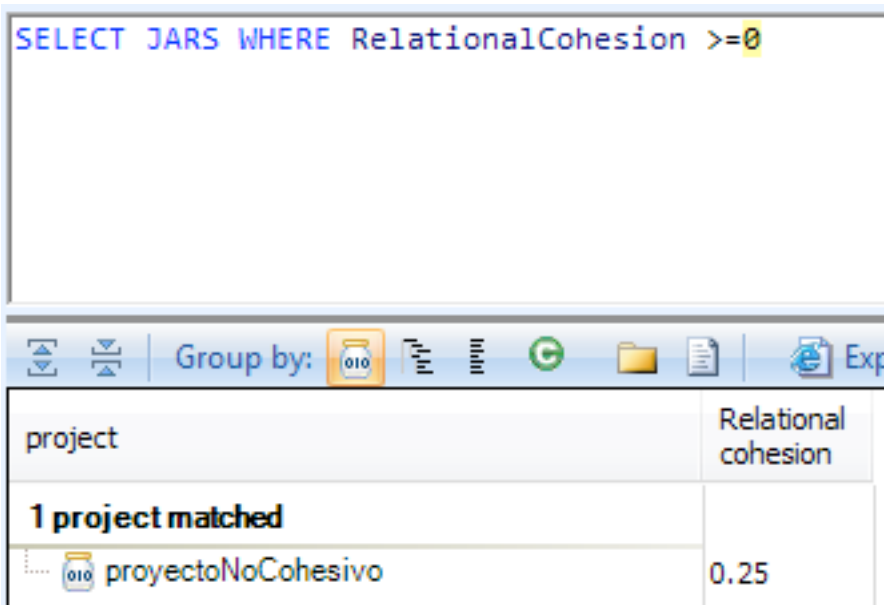


Figura 5.34 Cohesión relacional para ProyectoNoCohesivo en JArchitect

Proyecto analizado	Cálculo manual	JArchitect	Porcentaje de error
ProyectoCohesivo	3.25	3.25	0%
ProyectoRelativamenteCohesivo	2.25	2.25	0%
ProyectoNoCohesivo	0.25	0.25	0%

Tabla 5.17 Comparación de valores de Cohesión relacional

Al analizar los datos en la Tabla 5.17 se puede observar que JArchitect realiza los cálculos correctamente, por lo cual se puede considerar que el usar esta herramienta para calcular la cohesión relacional de una librería es confiable.

5.5.Resumen de resultados

A continuación en la Tabla 5.18 se resumen los resultados de las pruebas de las herramientas obteniendo las métricas seleccionadas, cuando la métrica no presentó solo resultados correctos se indica cuál es la media cuadrática [11] del porcentaje de error para los cálculos:

Métrica	JArchitect	Codepro Analytix	Eclipse Metrics C.
Acoplamiento aferente a nivel clase(Cac)	Incorrecta, con una media cuadrática del porcentaje de error de 5.24%.		
Acoplamiento eferente a nivel clase(Cec)	Correcta.		
Acoplamiento aferente a nivel librería (Ca)	Incorrecta, con una media cuadrática del porcentaje de error de 70.71%.	Correcta.	
Acoplamiento eferente a nivel librería (Ce)	Correcta.	Incorrecta, con una media cuadrática del porcentaje de error de 59.13%.	
Inestabilidad (I)	Aplicación correcta de la fórmula, pero uso de datos incorrectos provoca resultados incorrectos con una media cuadrática del porcentaje de error de 9.21%.	Aplicación correcta de la fórmula, pero uso de datos incorrectos provoca resultados incorrectos con una media cuadrática del porcentaje de error de 4.64%.	
Nivel de abstracción (A)	Correcta.	Correcta.	
Distancia de la secuencia principal normalizada (D')	Aplicación correcta de la fórmula, pero uso de datos incorrectos provoca resultados incorrectos con una media cuadrática del porcentaje de error de 32.75%.	Aplicación correcta de la fórmula, pero uso de datos incorrectos provoca resultados incorrectos con una media cuadrática del porcentaje de error de 18.10%.	

Métrica	JArchitect	Codepro Analytix	Eclipse Metrics C.
Falta de cohesión en los métodos (LCOM)	Incorrecta, con una media cuadrática del porcentaje de error de 92.87%.		
Falta de cohesión en los métodos Henderson Sellers (LCOM HS)	Incorrecta, con una media cuadrática del porcentaje de error de 35.35%.		Incorrecta, con una media cuadrática del porcentaje de error de 57.73%.
Cohesión Relacional (H)	Correcta.		

Tabla 5.18 Matriz de comparación de resultados de herramientas

5.6.Comentarios finales sobre herramientas

Para complementar la información mostrada a través de este capítulo se pueden agregar comentarios adicionales sobre los resultados que obtuvieron las herramientas y las posibilidades de mejorarlos en un futuro.

5.6.1.JArchitect

Esta herramienta calcula todas las métricas que son consideradas de interés para esta evaluación, pero tuvo 6 de 11 métricas calculadas incorrectamente. Este es un número muy elevado y la razón de estos errores parece ser que además de analizar el código, analiza los archivos compilados generados a partir del código. El proceso de compilación puede introducir datos que no se pueden explicar con solo ver al código. Esta parece ser su principal debilidad, pero al ser este un producto comercial, es posible que estén enterados de estos errores y tengan un plan para arreglarlos.

5.6.2.Codepro Analytix

Esta herramienta calcula 5 de las 11 métricas que son consideradas de interés para esta evaluación y solo 2 métricas fueron consideradas como correctas. Las 3 erróneas se derivan de un cálculo erróneo que parece fácilmente corregible como ya se demostró en una sección previa. El futuro de este proyecto es algo incierto puesto que fue anunciado que iba a ser donado por parte de Google a la fundación Eclipse para convertirlo en open source en diciembre del 2011, pero hasta septiembre del 2012 esta transición aún no ha ocurrido y su código fuente aún no está disponible como código abierto.

5.6.3.Eclipse metrics continued

Esta herramienta calcula 1 de las 11 métricas que son consideradas de interés para esta evaluación. La única métrica que calcula fue incorrecta, por lo cual, su desempeño en términos de esta evaluación fue muy pobre; por otro lado este plugin sí es totalmente open source y podría ser arreglado por alguien interesado en tener métricas correctas y se ve que tiene en principio las bases para obtener más métricas relativas al acoplamiento como: acoplamiento aferente a nivel librería, acoplamiento eferente a nivel librería, nivel de abstracción, inestabilidad y distancia de la secuencia principal. Esto porque ya calcula acoplamiento aferente y eferente pero a nivel paquete y es probable que solo requiera modificaciones de complejidad media para poder calcular el acoplamiento aferente y eferente a nivel librería.

5.7.Lecciones aprendidas

Las fallas que tuvieron estas herramientas se pueden categorizar en dos clasificaciones:

- Error en el conteo de elementos del código: Es el error más común y se puede ver como un análisis de código incorrecto o incompleto, también es el más difícil de corregir. Puede implicar conceptos erróneos en el conteo o métodos erróneos de conteo.
- Error en la aplicación de la fórmula de la métrica: Es el error menos común, pero puede arruinar un proceso de análisis y conteo bien hecho al no realizar el último cálculo correctamente.

Si se pretende construir una nueva herramienta o mejorar una de las ya existentes, es necesario tomar en cuenta esos dos tipos de errores y crear procesos que busquen eliminarlos. Se proponen estos puntos para tomar en cuenta en un proceso de construcción o mejora de una herramienta:

- Definir claramente cómo se cuentan las relaciones entre clases y tener un documento que explique esto sin dejar libertad de interpretación.

- Verificar que la implementación que se hace de las fórmulas no permita que se rebasen los límites de valores definidos para las fórmulas.

- Si se analizan archivos con bytecode, buscar la forma de diferenciar los elementos que fueron definidos en el código, de los que se definieron en tiempo de compilación.

- Verificar que la terminología de la literatura que se utilice para documentarse sobre las métricas sea entendida completamente. Esto es importante sobre todo al usar literatura de distintos lenguajes de programación, donde una misma palabra puede tener distintos significados.

Como un ejemplo, un Java Package y un C Package son cosas muy distintas y al documentarse es necesario estar consciente de esto.

El uso y desarrollo de estas ideas puede ayudar a los futuros desarrolladores de herramientas de este tipo, para no caer en los errores que fueron encontrados en las herramientas analizadas en este trabajo.

6. Conclusiones

Al verificar la confiabilidad de las herramientas para medir el acoplamiento y cohesión se ha podido observar que son pocas las métricas que son calculadas correctamente, por lo cual no pueden ser usadas en su totalidad. Es necesario tomar en cuenta si es posible obtener correctamente los datos que son importantes para el desarrollo por medio del uso de alguna herramienta y usarla en caso de que sea posible. Es probable que las necesidades de un proyecto no abarquen todas las métricas que se mencionaron en este trabajo y por lo cual pueda utilizarse alguna herramienta con el propósito de tener mediciones de acoplamiento y cohesión.

Aunque todas las herramientas tuvieron fallas en algunas métricas, esto no significa que vayan a tener estos problemas para siempre. JArchitect, al ser un proyecto comercial es muy probable que si se adquiere licencia para su uso, se pueda recibir soporte y se puedan resolver los problemas que tiene para calcular algunas métricas. Por otro lado, los dos proyectos de código abierto pueden ser parchados por un desarrollador con experiencia para corregir los errores en cálculos que fueron detectados en este trabajo.

El uso de herramientas de este tipo puede ser beneficioso para muchos proyectos en Java y puede ser crucial para algunos, la recomendación que se puede hacer a los desarrolladores es usarlas, pero estar conscientes de sus limitaciones y en la medida de sus posibilidades tratar de superarlas, ya sea con cálculos manuales o herramientas propias.

7. Trabajo futuro

Esta investigación al final deja varias líneas abiertas para trabajo futuro. Es obvio que las herramientas para calcular métricas de acoplamiento y cohesión en Java actualmente son mejorables. Se puede intentar crear una nueva herramienta que logre calcular satisfactoriamente estas métricas o por otro lado, se puede tomar el trabajo que ya hay en las herramientas de código libre y contribuir con mejoras para hacerlas más confiables. En este trabajo se han aportado algunas ideas para tomar en cuenta en desarrollos que se encaminen a la creación o mejora de herramientas de este tipo.

Por otro lado, se puede profundizar en los aspectos propios de la codificación en Java, ya que este lenguaje cuenta con muchos frameworks y estándares que guían el desarrollo de muchas aplicaciones. El uso de estos elementos puede crear la necesidad de modificar los procedimientos que se tienen para calcular las métricas. Se puede hacer un estudio acerca de los efectos que tienen sobre el acoplamiento y la cohesión el uso de estos frameworks y estándares. Hacer esto puede requerir un esfuerzo considerable y posiblemente la adecuación de las formas de conteo y medición que existen actualmente. Ya hay algunos avances en esta dirección por parte de la comunidad científica, como el articulo Improving Object-Oriented Lack-of-Cohesion Metric by Excluding Special Methods de Jehad Al Dallal [12] , donde se explora la posibilidad de mejorar las mediciones de cohesión en las clases al excluir de las mediciones los métodos especiales como los get y set.

También existen otras alternativas para continuar esta investigación enfocándola a cómo se puede aplicar el uso de estas herramientas en las empresas de desarrollo de software. Asimismo, se puede investigar cómo se puede compaginar el uso de estas herramientas que calculan valores importantes para la calidad de un sistema de software con las métricas de calidad que ya tienen las empresas. El análisis del acoplamiento y cohesión en una empresa también puede enfocarse en encontrar los mejores proyectos, los mejores equipos y las mejores prácticas. Una investigación en este sentido puede llevar a conocer cuáles son las mejores condiciones para tener desarrollos donde se cuide el acoplamiento y cohesión y de esta manera tratar de replicar estas condiciones para tener desarrollos con mayor calidad.

Una línea de investigación importante, es identificar qué magnitud de beneficios financieros se puede esperar por parte de proyectos que usan herramientas automatizadas para monitorear y controlar los valores de acoplamiento y cohesión. Para profundizar en este tema, se podrían tener algunos proyectos piloto donde se aplique un control del acoplamiento y cohesión con herramientas automatizadas; una vez que los proyectos sean entregados o liberados, se podría cuantificar los beneficios del control de acoplamiento y cohesión en términos financieros, comparándolos con aquellos proyectos que carecieron de este control.

Existe también otra línea de investigación que puede aportar datos muy útiles, se puede comprobar el nivel de aceptación y uso de estas métricas de acoplamiento y cohesión en la industria del desarrollo en general. En base a los resultados obtenidos se puede investigar cuáles son las razones de que estas métricas sean aceptadas o no y qué se podría hacer para mejorar su aceptación y utilidad para la industria en general.

Bibliografía

1 Rajaraman, Chandrashekar. *Reliability and Maintainability Related Software Coupling Metrics in C++ Programs*. Dept. of Comput. Sci., Iowa Univ., Iowa City, 1992.

2 Martin, Robert and Martin, Micah. *Agile Principles, Patterns, and Practices in C#*. Prentice Hall PTR, Upper Saddle River, 2006.

3 Martin, Robert. *The Single Responsibility Principle*. 2002.

4 JARCHITECT. *Java Metrics http://www.javadepend.com/Metrics.aspx*.

5 Cain, James Westland and McCrindle, Rachel Jane. An Investigation into the Effects of Code Coupling on Team Dynamics and Productivity. In *Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment* (Washington 2002), IEEE Computer Society.

6 Reißing, Ralf. Towards a model for object-oriented design measurement. In *5th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering* (2001).

7 Chidamber, Shyam R. and Kemerer, Chris F. Towards a metrics suite for object oriented design. In *Conference proceedings on Object-oriented programming systems, languages, and applications* (Phoenix 1991), ACM.

8 Hitz, Martin and Montazeri, Behzad. Measuring coupling and cohesion in object-oriented systems. In *Proc. Intl. Sym. on Applied Corporate Computing* (1995).

9 Henderson-Sellers, Brian. *Object-oriented metrics: measures of complexity*. Prentice-Hall, Inc., 1995.

10 Weisstein, Eric W. *Percentaje Error - http://mathworld.wolfram.com/ PercentageError.html*. MathWorld--A Wolfram Web Resource.

11 Weisstein, Eric W. *Root-Mean-Square -- http://mathworld.wolfram.com/Root-Mean-Square.html*. MathWorld--A Wolfram Web Resource.

12 Al Dallal, Jehad. Improving object-oriented lack-of-cohesion metric by excluding special methods. In *Proceedings of the 10th WSEAS international conference on Software engineering, parallel and distributed systems* (Cambridge, UK 2011), World Scientific and Engineering Academy and Society (WSEAS).

CENTRO DE INVESTIGACIÓN EN MATEMÁTICAS, A.C.

BIBLIOTECA

AUTORIZACION
PUBLICACION EN FORMATO ELECTRONICO DE TESIS

El que suscribe

Autor(s) de la tesis: Víctor Pablo Navarro Belmonte

Título de la tesis: Evaluación de herramientas para medir acoplamiento y cohesión en Java

Institución y Lugar: Cimat Unidad Zacatecas

Grado Académico: Licenciatura ( ) Maestría (x) Doctorado ( ) Otro ( )

Año de presentación: 2012

Área de Especialidad: Ingeniería de software

Director(es) de Tesis: Doctor Jorge Roberto Manjarrez Sánchez

Correo electrónico: pnavarrob@cimat.mx , victorpablo.navarrobelmonte@gmail.com

Domicilio: Mexicapan #1366 col. centro Zacatecas Zacatecas

Palabra(s) Clave(s): Java, acoplamiento, cohesión, evaluación, herramientas, métricas

Por medio del presente documento autorizo en forma gratuita a que la Tesis arriba citada sea divulgada y reproducida para publicarla mediante almacenamiento electrónico que permita acceso al público a leerla y conocerla visualmente, así como a comunicarla públicamente en la Página WEB del CIMAT.

La vigencia de la presente autorización es por un periodo de 3 años a partir de la firma de presente instrumento, quedando en el entendido de que dicho plazo podrá prorrogar automáticamente por periodos iguales, si durante dicho tiempo no se revoca la autorización por escrito con acuse de recibo de parte de alguna autoridad del CIMAT

La única contraprestación que condiciona la presente autorización es la del reconocimiento del nombre del autor en la publicación que se haga de la misma.

Atentamente

Víctor Pablo Navarro Belmonte
Nombre y firma del tesista

CALLE JALISCO S/N MINERAL DE VALENCIANA APDO. POSTAL 402
C.P. 36240 GUANAJUATO, GTO., MÉXICO
TELÉFONOS (473) 732 7155, (473) 735 0800 EXT. 49609 FAX. (473) 732 5749
E-mail: biblioteca@cimat.mx