Editorial de la
Universidad Tecnológica Nacional

La Universidad Tecnológica Nacional - U.T.N. - en el Nordeste Argentino – N.E.A.

Investigación y Desarrollo en la Facultad Regional Resistencia

Compiladoras:

Carola Sosa
Nidia Dalfaro

CALIDAD SOFTWARE

Medición de atributos de calidad en aplicaciones orientadas a objeto

Autores: Demchum, Greiner, Dapozo, Cuenca Pletsch y Estayno.

Editorial de la Universidad Tecnológica Nacional – edUTecNe
http://www.edutecne.utn.edu.ar
edutecne@utn.edu.ar

MEDICIÓN DE ATRIBUTOS DE CALIDAD EN APLICACIONES ORIENTADAS A OBJETO

Autores: *DEMCHUM, D.[1], GREINER, C.[1], DAPOZO G.[1]; CUENCA PLETSCH L.[2]*, ESTAYNO M.[3]*
Afiliación de los autores:
[1] Departamento de Informática. Facultad de Ciencias Exactas y Naturales y Agrimensura - Universidad Nacional del Nordeste, Av.Libertad 5450, 3400, Corrientes, Argentina
[2] Departamento de Ingeniería en Sistemas de Información. Facultad Regional Resistencia - Universidad Tecnológica Nacional – e-mail: cplr@frre.utn.edu.ar - Tel-Fax: 03722-432683 / 432928
[3] Departamento de Informática. Facultad de Ingeniería. Universidad Nacional de Lomas de Zamora, Ruta 4 Km 2, 1832 Lomas de Zamora, Buenos Aires, Argentina

Resumen. La medida de la calidad del software es una necesidad para las empresas desarrolladoras de software dado que representa una ventaja estratégica al proporcionar conocimiento de los procesos productivos y permitir mejoras en las tareas menos eficientes. En este trabajo se describe una metodología de evaluación de atributos de calidad de una aplicación Orientada a Objetos (OO), escrita en Java, cuyo objetivo es brindar información a los desarrolladores para detectar situaciones que pudieran afectar la comprensión y mantenibilidad del software. Los valores de las métricas permitirán mantener un registro histórico que contribuirá a los procesos de estimación de software en el marco de la gestión de proyectos de software.

Palabras clave: Calidad del software, Calidad del producto, Métricas Orientadas a Objeto. Gestión de Proyectos de Software.

1. INTRODUCCIÓN

La calidad del software está estrechamente vinculada con la medición del mismo. Los modelos de evaluación y mejora de procesos, tales como: CMM (Capability Maturity Model), CMMI (Capability Maturity Model Integration) propuestos por el SEI (Software Engineering Institute) [1], por una parte, y la norma ISO 15504 SPICE propuestas por el ISO (Internacional Organization for Standardization) [2], incorporan en sus primeros niveles, como parte de las buenas prácticas recomendadas, técnicas y procesos para el aseguramiento de la calidad que se corresponden con la medición de software, los procesos de revisión y auditoría y las pruebas de software [3].
La medida de la calidad del software es una necesidad para las empresas de software y servicios informáticos (SSI), representa una ventaja estratégica al proporcionar el conocimiento de los procesos productivos y permitir mejorar las tareas menos eficientes. Medir es conocer, y este conocimiento permite controlar aquellos factores que aportan una mayor eficacia en el proceso productivo, obteniendo productos con un nivel de calidad mayor haciendo a las organizaciones más eficientes y permitiendo una ventaja estructural frente a sus competidores [4].
Para obtener software de calidad es preciso medir el proceso de desarrollo, cuantificar lo que se ha hecho y lo que falta por hacer, estimar el tamaño del programa, costos, tiempo de desarrollo y otros parámetros. La medición del producto se realiza mediante las métricas, para caracterizar numéricamente los distintos aspectos de desarrollo del software.
Las métricas de software tienen un papel decisivo en la obtención de un producto de alta calidad porque determinan, mediante estadísticas basadas en la experiencia, el avance del software y el cumplimiento de parámetros requeridos. Siempre habrá elementos cualitativos para la creación de software. El problema estriba en que la valoración cualitativa puede no ser suficiente. Un ingeniero del software necesita criterios objetivos para guiarse en el diseño de datos, de la arquitectura, de las interfaces y de los componentes. El verificador o *tester* necesita una referencia cuantitativa como soporte para la selección de los casos de prueba y de sus objetivos.

Las métricas técnicas facilitan una base para que el análisis, diseño, codificación y prueba puedan ser conducidos más objetivamente y valorados más cuantitativamente [5].
Por otra parte, el paradigma de Programación Orientada a Objetos (POO) se ha afianzado en la industria del software, debido a la promoción de características deseables en el software, tales como la reutilización de código, encapsulación, abstracción y modularización, entre otros [6]. Paralelamente al desarrollo de las aplicaciones Orientadas a Objetos crece la necesidad de métricas que permitan medir los atributos del software asociados a su calidad. Algunas estimaciones indican que el ahorro de costo de mantenimiento es del 42% mediante el uso de métricas orientadas a objeto [7].
En este trabajo se describe la aplicación de una metodología de evaluación de atributos de calidad a una aplicación OO, escrita en Java, cuyo objetivo es brindar información a los desarrolladores para detectar situaciones que pudieran afectar la comprensión y mantenibilidad del software, como así también, posibilitar la generación de un registro histórico de los valores de las métricas que permita realizar predicciones, contribuyendo al proceso de estimación en la gestión de proyectos software.
El mismo forma parte del proyecto "Modelos y métricas para la evaluación de la calidad de software", cuyo objetivo es contribuir a la mejora en la calidad del producto y del proceso, mediante la transferencia de técnicas y herramientas hacia las pymes de software de la región NEA (Nordeste Argentino), como medio para aumentar la competitividad de las mismas y promover la industria del software en la región.

1.1. El proceso de medición del software

El objetivo de todo proceso de medición es recopilar indicadores cuantitativos sobre entidades software, entendiéndose por *entidad software* todo *elemento software* sobre el que se puede aplicar un proceso de medición y que están caracterizadas por una serie de atributos. Para realizar la medición es necesario identificar tanto las entidades como los atributos a medir (Morasca en [8]).

La Figura 1 muestra el proceso de medición del software dentro de un proceso de control de calidad. Los pasos claves en este proceso son [9]:

1. *Seleccionar las medidas a realizar.* Se deben formular las preguntas que la medición intenta responder y definir las mediciones requeridas para resolver estas preguntas.
2. *Seleccionar los componentes a evaluar.* Puede elegirse un conjunto representativo de componentes o bien evaluar los componentes particularmente críticos.
3. *Medir las características de los componentes.* Se miden los componentes seleccionados y se calculan los valores de las métricas. Normalmente, esto comprende procesar la representación del componente (diseño, código, etc.) utilizando una herramienta de recogida de datos.
4. *Identificar las mediciones anómalas.* Una vez que se obtienen las mediciones de los componentes, se comparan entre sí y con mediciones previas registradas en una base de datos de mediciones.
5. *Analizar los componentes anómalos.* Una vez identificados los componentes con valores anómalos, se examinan para decidir si esos valores indican que la calidad del componente está en peligro. Valores anómalos para la complejidad, por ejemplo, no significan necesariamente que el componente tenga una calidad deficiente, sino que representan un llamado de atención que debe analizarse para determinar si existen problemas con la calidad del producto.

Los datos recogidos se mantienen como un recurso organizacional y deben conservarse registros históricos de todos los proyectos aun cuando los datos no se hayan utilizado durante un proyecto particular. Una vez que se haya creado una base de datos suficientemente grande de mediciones, se hace la comparación de los proyectos, y las métricas específicas se refinan de acuerdo con las necesidades organizacionales.

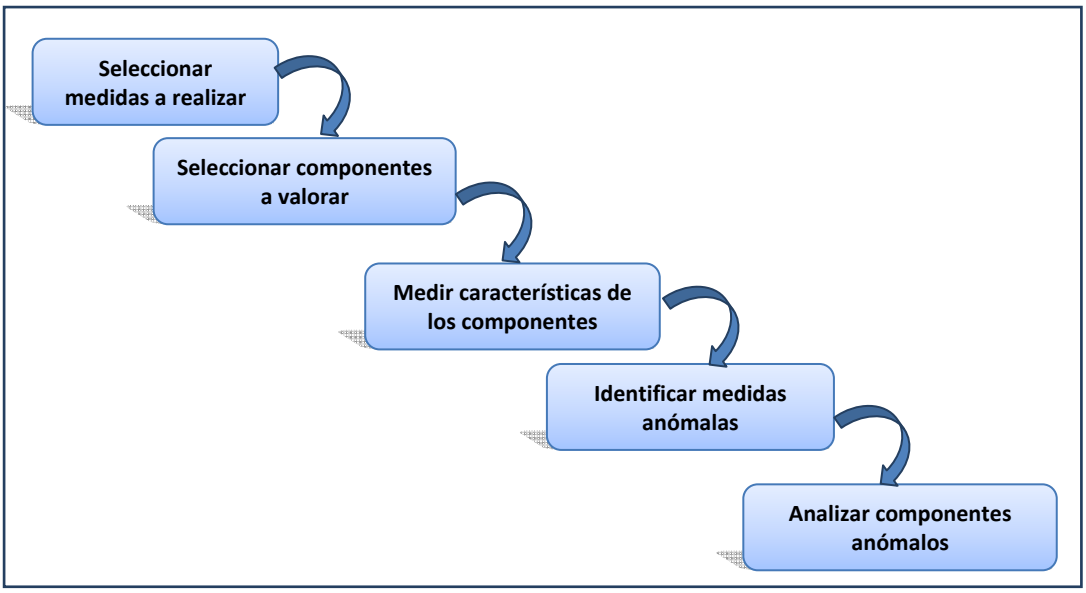

Figura 1. Proceso de medición del software

1.2. Métricas Orientadas a Objetos (OO)

Las métricas para sistemas OO hacen hincapié en los conceptos básicos del paradigma de programación OO, tales como el encapsulamiento, la herencia y polimorfismo. Los objetivos principales de las mismas son: comprender mejor la calidad del producto, estimar la efectividad del proceso y mejorar la eficiencia en la gestión del proyecto [5].
Las características que distinguen al software OO del software convencional, y en las cuales se centran las métricas, son las siguientes:

- *Encapsulamiento*: engloba las responsabilidades de una clase, incluyendo sus atributos y operaciones. Influye en las métricas, dado que cambia el enfoque de las mediciones de un módulo simple a un paquete de datos y procesos.
- *Ocultación de la información*: implica no mostrar los detalles operacionales de un componente de programa. Un sistema OO bien diseñado debe implementar ocultación de información. Consecuentemente las métricas que proporcionan una indicación del grado de ocultamiento logrado suministran un indicio de la calidad de diseño OO.
- *Herencia*: mecanismo que habilita las responsabilidades de un objeto para propagarse a otros objetos. Favorece la reutilización, por lo tanto las métricas deben indicar si el sistema tiene un grado adecuado de jerarquías y niveles de herencia.
- *Técnicas de abstracción de objetos*: La abstracción es un mecanismo que permite al diseñador concentrarse en los detalles esenciales de un componente de programa, prestando poca atención a los detalles de bajo nivel. Las métricas OO permiten evaluar la calidad de las abstracciones, indicando posible necesidad de mejora.

Las métricas OO que surgen de la bibliografía especializada son las siguientes:

- Métricas CK (Chidamber y Kemerer): DIT (Depth of Inheritance Tree), NOC (Number of Children), CBO (Coupling Between Objects), RFC (Response For a Class), WMC (Weighted Methods per Class), LCOM (Lack of Cohesion in Methods) [10].
- Métricas MOOD: PF (Polymorphism Factor), CF (Coupling Factor), MHF (Method Hiding Factor), AHF (Attribute Hiding Factor), MIF (Method Inheritance Factor), AIF (Attribute Inheritance Factor) [11].

- Lorenz y Kidd: SIX (Specialization Index per Class), LOC (Lines of Code per method), NOM (Number of Messages Send) [12].

Estas métricas consideran distintos niveles de granularidad, algunas consideran al sistema en su totalidad, otras se enfocan en las clases y otras en los métodos.

1.3. Métricas CK (Chidamber y Kemerer)

Uno de los conjuntos de métricas de software 00 a los que se hace más ampliamente referencia es el propuesto por Chidamber y Kemener [13]. Estas métricas de diseño, a las cuales suele aludirse con el nombre de métricas CK, son las siguientes:

a) DIT (Depth of Inheritance Tree):

La profundidad del árbol de herencia mide el máximo nivel en la jerarquía de herencia. DIT es la cuenta directa de los niveles en la jerarquía de herencia. En el nivel cero de la jerarquía se encuentra la clase raíz. Representa la medida de la complejidad de una clase: la complejidad del diseño y el potencial reuso. Esto es debido a que cuanto más profunda se encuentra una clase en la jerarquía, mayor será la probabilidad de heredar un mayor número de métodos.

El uso de la herencia es visto como un compromiso ya que:

- Altos niveles de herencia indican objetos complejos, los cuales pueden ser difíciles de testear y reusar.
- Bajos niveles en la herencia pueden indicar que el código está escrito en un estilo funcional sin aprovechar el mecanismo de herencia proporcionado por la orientación a objetos.

b) NOC (Number of Children)

El número de hijos es el número de subclases subordinadas a una clase en la jerarquía, es decir, el número de subclases que pertenecen a una clase. Es un indicador del nivel de reuso, la posibilidad de haber creado abstracciones erróneas y del nivel de test requerido.
Aunque un mayor número de hijos representa una mayor reutilización de código, presenta los siguientes inconvenientes:

- Mayor probabilidad de usar incorrectamente la herencia creando abstracciones erróneas.
- Mayor dificultad para modificar una clase ya que afecta a todos los hijos que tienen dependencia con la clase base.
- Se requiere mayor número de recursos para testear.

Es un potencial indicador de la influencia que una clase puede tener sobre el diseño del sistema. Si el diseño tiene una alta dependencia en la reusabilidad a través de herencia, puede ser mejor dividir la funcionalidad en varias clases [10].

c) RFC (Response for a Class)

La respuesta de una clase es el cardinal del conjunto de todos los métodos que pueden ser invocados en respuesta a un mensaje a un objeto de la clase o por algún método en la clase. Esto incluye todos los métodos accesibles dentro de la jerarquía de la clase. En otras palabras, cuenta las ocurrencias de llamadas a otras clases desde una clase particular.
RFC es una medida de la complejidad de una clase a través del número de métodos y de la comunicación con otras clases, ya que incluye todos los métodos llamados desde fuera de la clase. Es un indicador de los recursos necesarios para testeo y depuración. Cuanto mayor es RFC, más complejidad tiene el sistema ya que se puede invocar un mayor número de métodos como repuesta a un mensaje.

d) WMC (Weighted Methods per Class)

WMC mide la complejidad de una clase. Clases con un gran número de métodos requieren más tiempo y esfuerzo para desarrollarlas y mantenerlas, ya que influirán en las subclases que heredan todos sus métodos. Además, estas clases tienden a ser específicas de la aplicación, limitando su posibilidad de reuso.

e) LCOM (Lack of Cohesion in Methods)

La falta de cohesión en los métodos establece en qué medida los métodos hacen referencia a atributos. LCOM es una medida de la cohesión de una clase teniendo en cuenta el número de atributos comunes usados por diferentes métodos, indicando la calidad de la abstracción hecha en la clase.
Un valor alto de LCOM implica falta de cohesión, es decir, escasa similitud de los métodos. Esto puede indicar que la clase está compuesta de elementos no relacionados, incrementando la complejidad y la probabilidad de errores durante el desarrollo.
Es deseable una alta cohesión en los métodos dentro de una clase lo que implica que ésta no pueda ser dividida, lo cual fomenta la encapsulación.

Para determinar si la aplicación se ajusta a criterios de calidad, el valor de las métricas se contrasta con valores "umbrales" o valores recomendados de acuerdo a la naturaleza de la medición. Esto permite detectar clases o métodos que difieren sustancialmente de unos valores medios, por lo que son candidatos para ser revisados o reescritos.

2. METODOLOGÍA

En esta sección se describe la metodología seguida para validar un conjunto de métricas, orientadas a la evaluación de una aplicación OO, desarrollada en Java, que se encuentra implementada como parte de un sistema mayor en un organismo judicial de la provincia de Corrientes, contando con la autorización expresa de sus propietarios.

2.1. Descripción de la aplicación:

La aplicación denominada JAdmin tiene como objetivo la administración de las tareas de Mesa de Ayuda del Sistema de Administración de Expedientes del mencionado organismo.
Para su desarrollo se utilizaron herramientas *open source*, tales como el entorno de desarrollo Netbeans y el lenguaje de programación Java.
A continuación se presenta la descripción técnica de la aplicación:

Nombre de la Aplicación: JAdmin
Líneas de código: 6744.
Cantidad de clases: 34.
Versión Actual: 1.0
Plataforma de Desarrollo: Java (JDK 1.6)
Sistemas Operativos: Windows XP (32 y 64 bits), Windows Vista, Windows 7 y Distribuciones Linux.
Aplicaciones externas requeridas: Visor de documentos pdf, Máquina virtual de Java correspondiente a la plataforma (JRE 1.6 o JDK 1.6)
Extensibilidad: La aplicación tiene la facilidad de adaptarse a cambios de especificación, quedando abierta para la incorporación de funcionalidades.
Portabilidad: Tiene la aptitud de transferirse a diferentes entornos de hardware y software.

2.2 Evaluación de la aplicación JAdmin

La metodología de evaluación se basa en el proceso de medición del software descrito anteriormente e ilustrado en la Figura 1. Cada uno de los componentes del sistema se analiza por separado y los diversos valores de las métricas se comparan entre sí y, si existen, con los datos históricos de medición recogidos en los proyectos previos. Las medidas anómalas se utilizan para centrar el esfuerzo de garantía de calidad en los componentes que tienen problemas de calidad. Los pasos seguidos son:

1. *Seleccionar las medidas a realizar*. De las características del software susceptibles de medición, en este trabajo se propone medir la complejidad de las clases. Por tal motivo, se seleccionaron las métricas CK, que permiten indagar cuán bien están definidas las clases y el sistema, lo cual tiene un impacto directo en la mantenibilidad del mismo, tanto por la comprensión de lo desarrollado como por la dificultad de modificarlo con éxito. Los objetos complejos, por otra parte, pueden ser difíciles de testear y reusar. Este conjunto de métricas identifica características dentro de las clases, destacando diferentes aspectos de sus abstracciones y ayudando a descubrir clases que podrían necesitar ser rediseñadas [10]. Trabajan a distintos niveles de granularidad, como por ejemplo a nivel de jerarquías de herencia, a nivel de clases y a nivel de métodos.
2. *Seleccionar los componentes a evaluar*. La propuesta se orienta a la evaluación del producto software, utilizando como objeto de evaluación el código fuente de programas Java, en este caso de estudio particular.
   Para realizar la evaluación se tomó el código de la aplicación JAdmin mencionada anteriormente.
3. *Medir las características de los componentes.* Se realizó un relevamiento de las herramientas open source que implementan el cálculo de las métricas de diseño OO. Las características de las herramientas relevadas se resumen en la tabla1.

Herramienta	Tipo	Tipo de Licencia	Entrada	Formato exportable	Métricas	Aceptación intervalos
Eclipse metrics plugin	Plugin	Common Public	Fuentes	XML	CK,LK, RM	Sí
Chidamber and Kemerer Java Metrics	Autónoma y Plugin (Ant)	Creative Commons	Binarios	XML	CK	No
Eclipse Plugin to calculate CKmetrics	Plugin	Public Domain	Fuentes	-	CK	No
RefactorIt	Autónoma y Plugin	GPL	Fuentes	CSV, XML, HTML y TXT	CK,LK, RM	Sí
Eclipse UnitMetrics	Plugin	Common Public	Fuentes	-	CK,LK, RM	Sí

Tabla 1: Características de herramientas de medición

Se realizaron pruebas con las herramientas, *Eclipse metrics plugin* y *RefactorIt*, ambos complementos del entorno de desarrollo (IDE) Eclipse. Se compararon los resultados de los cálculos de las métricas con ambas herramientas y, dado que ambas ofrecían valores coincidentes, se seleccionó RefactorIT por las siguientes características: no implica un costo adicional al cliente, está disponible como plugin de Eclipse y NetBeans, y como programa *standalone;* tiene una baja curva de aprendizaje; permite modificar los valores umbrales; brinda diversos formatos para exportar resultados y, además de las métricas CK, implementa otras métricas.

Para la instalación y ejecución de las herramientas se realizaron los siguientes pasos:

a) Descargar e instalar el Kit de Desarrollo de Java (JKD), disponible en http://www.java.com/es/download/
b) Descargar e instalar el entorno de desarrollo Eclipse disponible en http://www.eclipse.org/downloads/
c) Descargar e instalar el plugins Refactorit para Eclipse disponible en http://sourceforge.net/projects/refactorit/
d) Crear un proyecto en Eclipse e importar el código fuente de la aplicación a medir. Para tomar las medidas de las métricas, la aplicación a medir debe estar libre de errores.
e) Ejecutar la herramienta de medición Refactorit, realizando las siguientes acciones:
   i. Seleccionar el proyecto a evaluar
   ii. Ir al menú Refactorit y seleccionar la opción "Métricas".
   iii. Seleccionar las métricas CK y verificar el valor umbral de las mismas
   iv. Ejecutar el cálculo de las métricas
   v. Exportar los valores obtenidos. Los formatos provistos son: html, xml, cvs, txt.

Los valores resultantes de la medición se exportaron a un archivo .csv y los mismos se encuentran detallados en la Tabla 2. En cada fila se muestran los valores de cada una de las métricas, correspondientes a cada clase de la aplicación evaluada. Las celdas sombreadas corresponden a las clases cuyos valores de métricas superan los umbrales prefijados. Esta información permite luego acceder a las clases factibles de mejora.

Location Type		**WMC**	**RFC**	**DIT**	**NOC**	**LCOM**
administracion.BaseDato	Class	109	89	1	0	0,908
administracion.Buscar	Class	3	34	6	0	1
administracion.CambiarRadicacion	Class	13	80	6	0	0,906
administracion.CargaDatos	Class	5	13	1	0	0
administracion.CargarFeriados	Class	9	70	6	0	0,833
administracion.CargarTrap	Class	12	64	6	0	0,845
administracion.Circunscripcion	Class	7	3	1	0	0,6
administracion.Ciudad	Class	14	14	1	0	0,75
administracion.ConOrgEquiv	Class	7	0	1	0	0,6
administracion.ConsultarConexionJFrame	Class	25	80	6	0	0,92
administracion.EjecucionMasivaJFrame	class	7	51	6	0	0,6
administracion.Expediente	class	21	3	1	0	0,904
administracion.Feriado	class	8	0	1	0	0,4
administracion.FrameAltaUsuarioAdmin	class	7	46	6	0	0,75
administracion.GestionMovimiento	class	16	62	6	0	0,886
administracion.ListaExptesJDialog	class	4	44	6	0	0,667
administracion.ListaMovimientosJDialog	class	4	44	6	0	0,667
administracion.Movimiento	class	8	1	1	0	0,556
administracion.Organismo	class	33	4	1	0	0,944
administracion.Servidor	class	6	0	1	0	0,5
administracion.Usuario	class	5	0	1	0	0,333
principal.Inicio	class	24	112	6	0	0,951
trap.BaseDato	class	22	31	1	0	0,81
trap.BasePDF	class	7	28	2	0	1
trap.CargaDatos	class	3	14	1	0	0
trap.Circunscripcion	class	7	3	1	0	0,6
trap.Ciudad	class	15	21	1	0	0,722
trap.ElegirCiudad	class	12	80	6	0	0,917
trap.Expediente	class	12	3	1	0	0,78
trap.Inicio	class	12	52	6	0	0,933

trap.ListaPDF	class	9	33	2	0	0
trap.Organismo	class	8	3	1	0	0,667
trap.PDF	class	22	59	1	2	0,823
trap.Servidor	class	7	0	1	0	0,6

Tabla 2. Valores resultantes de las mediciones de la aplicación JAdmin.

4. *Identificar las mediciones anómalas y Analizar los componentes anómalos.* Los datos de las mediciones fueron procesados con una planilla de cálculo para un análisis global de los resultados. Fueron contrastados con los valores umbrales que se detallan en la Tabla 3, los cuales fueron tomados de los establecidos por defecto en la herramienta de medición. Este es un criterio inicial, dado que no se ha encontrado en la literatura consultada valores fuertemente recomendados.

Métrica	Denominación	Límite inferior	Límite superior
WMC	weighted-methods-per-class	1	50
RFC	response-for-class	0	50
DIT	depth-in-treee	2	5
NOC	number-of-children-in-tree	0	10
LCOM	Lack-of-cohesion	0	0.2

Tabla 3. Umbrales considerados para cada una de las métricas en RefactorIt

El procedimiento realizado para cada una de las métricas evaluadas fue el siguiente:

- En función de los valores umbrales se establecieron dos categorías: **Crítico**, que corresponde a los valores por fuera del rango establecido en cada métrica, y **Aceptable**, a los que están comprendidos dentro del rango especificado.
- Se calcularon las frecuencias absolutas y relativas de las clases para cada categoría mencionada. Para ello, se utilizó la función Frecuencia de la planilla Excel.

Mediante el procedimiento anterior se obtuvieron los valores para cada una de las métricas, que permitieron el análisis que se describe a continuación:

a) DIT (Profundidad del Árbol de Herencia): Esta métrica es la cantidad de niveles en la jerarquía de herencia.

Rango DIT	Frec Abs	%
Crítico (0 y 1)	19	56%
Aceptable (Entre 2 y 5)	2	6%
Crítico (Mayor que 5)	13	38%

Tabla 4. Porcentaje de clases dentro de cada categoría para DIT

Lo que se observa en la Tabla 4 es que el 56% de las clases se encuentra en niveles bajos de herencia, lo cual permite suponer que el código está escrito mayormente en estilo funcional, es decir, sin la utilización de las ventajas de la herencia.
Por otra parte, el 38% de las clases obtuvo valores por encima del umbral recomendado, lo que indicaría niveles altos de complejidad.
En general, los valores obtenidos permiten inferir que el aprovechamiento de la herencia, característica de la POO, no es óptimo y para mejorar este valor se debería verificar en la representación del diseño de la aplicación (diagrama de clase, código fuente) que existan estructuras y comportamientos que sean factibles de generalizar, de modo de factorizar y establecer una mayor profundidad en la jerarquía de herencia. Sin embargo, el valor puede representar un modelo ajustado a la realidad, que debe permanecer sin modificaciones. De aquí la importancia del juicio del experto para resolver esta situación, el valor de la métrica en este

sentido no es taxativo, sólo advierte de un potencial desaprovechamiento de las características distintivas de este paradigma.
Asimismo, del análisis posterior del código fuente de la aplicación, se detectó que los objetos de tipo Ventanas, Diálogos, y demás clases que conforman la interfaz gráfica y que heredan de las clases predefinidas de la librería SWING, son las que se encuentran en niveles profundos de herencia. En este caso, esta situación no representa un potencial riesgo, dado que se debe utilizar la jerarquía tal cual la ofrece el lenguaje de programación utilizado.
En este sentido, conviene tener presente también que las aplicaciones construidas a partir de frameworks suelen presentar niveles de herencia altos ya que las clases son creadas a partir de una jerarquía existente. También se verifica que, en lenguajes como Java o Smalltalk, las clases siempre heredan de la clase Object, lo que añade uno a DIT automáticamente por el contexto específico.

b) **NOC (Número de Hijos):** El número de hijos es el número de subclases subordinadas a una clase en la jerarquía, es decir, el número de subclases que pertenecen a una clase.

Rango NOC	Frec Abs	%
Aceptable (Entre 0 y 10)	34	100%
Critico (Mayor que 10)	0	0%

Tabla 5. Porcentaje de clases dentro de cada categoría para NOC

Los valores obtenidos para la métrica NOC, indicados en la Tabla 5, muestran que el 100% de las clases presenta valores recomendados de "hijos". Esto permite inferir que se realizó una correcta especialización.
En caso de obtener valores críticos, correspondería analizar estas clases con el propósito de determinar si las especializaciones realizadas son adecuadas, siempre teniendo en cuenta el juicio del experto.

c) **RFC (Respuesta por Clase):** es una medida de la complejidad de una clase relacionada con la cantidad de métodos y de la comunicación con otras clases.

Rango RFC	Frec Abs	%
Aceptable (Entre 0 y 50)	23	67,65%
Crítico (Mayor que 50)	11	32,35%

Tabla 6. Porcentaje de clases dentro de cada categoría para RFC

Según los valores que se muestran en la Tabla 6, el 32,35% de las clases está por encima del umbral aceptable, lo que estaría indicando la presencia de clases con una alta complejidad.
Observando el código fuente se pudo comprobar que, si bien el porcentaje de clases que superan el umbral de aceptable no es tan elevado, dentro de este grupo existen clases con valor alto de RFC. Por ejemplo, la clase *Inicio.java* tiene RFC=112 (ver Tabla 2). En estos casos, puede ser difícil probar el comportamiento de las clases y depurar problemas de comportamiento, ya que se requiere la comprensión de las interacciones que los objetos de esas clases pueden tener con el resto del sistema. Por lo tanto, la recomendación es revisar las clases con un elevado nivel de RFC, con el propósito de verificar la modularidad, buscando un adecuado equilibrio entre la cohesión y el acoplamiento.

d) **WMC (Peso de los Métodos por Clase):** mide la complejidad de una clase, en función de la cantidad de métodos que posee. Clases con un gran número de métodos requieren más tiempo y esfuerzo para desarrollarlas y mantenerlas.

Rango WMC	Frec Abs	%
Aceptable (Entre 1 y 50)	33	97,05%
Crítico (Mayor que 50)	1	2,95%

Tabla 7. Porcentaje de clases dentro de cada categoría para WMC

De los valores mostrados en la Tabla 7, el 97% de las clases tienen valores de complejidad aceptables.
De la observación del código se desprende que sólo una clase tiene el valor de WMC = 109, es *administracion.BaseDato* (ver Tabla 2). De igual manera, se recomienda revisar el código de la clase mencionada, dado que es muy utilizada dentro de la aplicación, con idéntica recomendación que en el caso anterior, en referencia a la modularidad.

e) **LCOM (Métrica de Falta de Cohesión):** mide la cohesión de una clase en función del número de atributos comunes usados por diferentes métodos, indicando la calidad de la abstracción hecha en la clase.

Rango LCOM	**Frec Abs**	**%**
Aceptable (entre 0 y 0,2)	3	8,82%
Crítico (Mayor que 0.2)	31	91,18%

Tabla 8. Porcentaje de clases dentro de cada categoría para LCOM

Los valores que dan cuenta del nivel de cohesión se muestran en la Tabla 8. Se observa que el 91,18% de las clases muestra falta de cohesión, lo cual significa que la mayoría de los métodos no acceden a la mayoría de los atributos.
Esta situación puede deberse a que las clases están compuestas por elementos no relacionados, incrementando la complejidad y la probabilidad de errores durante el desarrollo y/o mantenimiento. Los valores obtenidos en esta métrica podrían considerarse críticos. Se debe realizar una revisión de las clases tratando de mejorar la cohesión en las mismas, en concordancia con lo sugerido en el análisis de las dos métricas analizadas previamente.

4. CONCLUSIONES Y TRABAJOS FUTUROS

Los modelos de calidad consideran la medida de la calidad del software como una exigencia para la obtención de productos y servicios que satisfagan las necesidades y expectativas de los usuarios. Medida y calidad son relacionados de forma directa por estos modelos.
En este trabajo se muestran los resultados obtenidos mediante la aplicación de una metodología de medición de atributos de calidad al código fuente de un programa OO, escrito en Java.
De los valores resultantes de las métricas aplicadas al código fuente de la aplicación caso de estudio (JAdmin), se advierte que la misma requiere de una revisión del diseño para lograr un mejor aprovechamiento de las características de la POO, tal como se ha indicado en el análisis de resultado de cada una de las métricas.
La metodología propuesta es válida para aplicaciones de cualquier tamaño. Cuando se trate de programas con un elevado número de clases, para facilitar su análisis, la planilla de cálculo permite el filtrado y selección de las clases críticas, es decir, aquellas que superen los umbrales establecidos.
Como trabajo futuro se propone sistematizar este proceso de medición, mediante una herramienta que almacene los valores de las métricas de manera de generar un marco de trabajo que permita a los desarrolladores detectar situaciones que puedan afectar la comprensión y mantenibilidad del software, como así también, generar un registro histórico de los valores de las métricas que posibilite realizar predicciones contribuyendo a una gestión más eficiente de proyectos de software.

Referencias

[1] SEI (Software Engineering Institute). http://www.sei.cmu.edu/
[2] ISO (Internacional Organization for Standardization). http://www.iso.org/iso/home.htm
[3] Fernández Sanz, L.; Lara Bercial, P. “Mejora de la calidad en desarrollos orientados a objetos utilizando especificaciones UML para la obtención y precedencia de casos de

prueba". Revista de Procesos y Métricas de las Tecnologías de la Información (RPM) ISSN: VOL. 1, Nº 3, 11-20. (2004)
[4] Hernández Ballesteros, J.F.; Minguet Melián, J. M. "La Medida de la Calidad del Software como Necesidad y Exigencia en Modelos Internacionales (CMMI, ISO 15504, ISO 9001)". Departamento de Ingeniería de Software y Sistemas Informáticos, E.T.S. de Ingeniería Informática de la Universidad de Educación a Distancia (UNED). (2005). www.issi.uned.es/CalidadSoftware/Noticias/PonIng2005.rtf
[5] Pressman, R. "Ingeniería de Software. Un enfoque práctico". Ed- McGraw-Hill. (2005)
[6] Avello, D.G.; Cernuda del Río, A. "Reflexiones y Experiencias sobre la Enseñanza de POO como único Paradigma. Jornadas de Enseñanza Universitaria de Informática (JENUI). Cádiz, España. (2003)
[7] Khaled El Emam. "A Primer on OO Measurement", 1530-1435/05 IEEE, Proceeding of the Seventh International Software Metrics Symposium. (2001)
[8] Piattini, M.; García, O.; Caballero, I. "Calidad de los sistemas informáticos". Ed. RA-MA España. (2007)
[9] Sommerville, I. "Ingeniería del Software". Pearson Educación. Séptima Edición. Madrid. (2005)
[10] Rodríguez, D., Harrison, R., "Medición en la Orientación a Objeto". Cap.4 de Medición para la Gestión en la Ingeniería del Software. Dolado, J. and Fernández, L., Ed. RA-MA. (2000)
[11] Rosenberg, L.H. "Applying and Interpreting Object Oriented Metrics". Disponible en http://www.literateprogramming.com/ooapply.pdf
[12] Etzkorn, L.; Delugach, H. "Towards a Semantic Metrics Suite for Object-Oriented Design". IEEE, 07695-0774-3. (2000)
[13] Chidamber, S.; Kemerer, C. "A Metrics Suite for Object-Oriented Design". M.I.T. Sloan School of Management E53-315. (1993)