Una propuesta de solución para automatizar la medición de aplicaciones orientadas a objeto

Cristina Greiner[1], Daniela Demchum[1], Gladys Dapozo[1]; Marcelo Estayno[2]

1 Departamento de Informática. Facultad de Ciencias Exactas y Naturales y Agrimensura
Universidad Nacional del Nordeste, Av.Libertad 5450, 3400, Corrientes, Argentina
{gndapozo,cgreiner}@exa.unne.edu.ar; erika.demchum@gmail.com
2 Departamento de Informática. Facultad de Ingeniería. Universidad Nacional de Lomas de Zamora, Ruta 4 Km 2, 1832 Lomas de Zamora, Buenos Aires, Argentina
mestayno@fibertel.com.ar

Resumen. La medida de la calidad del software es una necesidad para las empresas de software y servicios informáticos (SSI) dado que representa una ventaja estratégica al proporcionar el conocimiento de los procesos productivos y permitir mejoras en las tareas menos eficientes. En este trabajo se describe una propuesta de solución orientada a contribuir a la calidad del producto software automatizando el proceso de medición de aplicaciones orientadas a objetos, con el objetivo de brindar información 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 permita realizar predicciones y una adecuación del proceso de medición al contexto organizacional específico.

Palabras clave: Calidad del software, Calidad del producto, Métricas Orientadas a Objetos.

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, CMMI, ISO 15504 SPICE, 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 [1].

La medida de la calidad del software es una necesidad para las empresas de 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 modificar 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 [2].

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 de este producto se realiza mediante las métricas, para caracterizar numéricamente los distintos aspectos del 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 necesita una referencia cuantitativa que le ayude en 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 [3].

Por otra parte, la tendencia de la industria de software hacia la adopción del paradigma de la programación orientada a objetos no ha disminuido en los últimos años debido a la promoción de características deseables en el software, tales como la reutilización de código, encapsulación, abstracción y modularización, entre otros [4]. Paralelamente al desarrollo de las aplicaciones orientadas a objetos crece la necesidad de métricas que permitan medir los atributos del software que indican su calidad. Un estudio calcula que el ahorro de costo de mantenimiento es del 42% mediante el uso de métricas orientadas a objetos [5].

Este trabajo 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, siendo una entidad software todo elemento software sobre el que se puede aplicar un proceso de medición y que están caracterizadas por una serie de atributos (tamaño, tiempo, etc.). Para realizar la medición es necesario identificar tanto las entidades como los atributos a medir (Morasca en [6]).

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 [7]:

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*. Se elige un conjunto representativo de componentes o se evalúan 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 las 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 los valores de la métrica indican que la calidad del componente está en peligro. Los valores de la métrica 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.

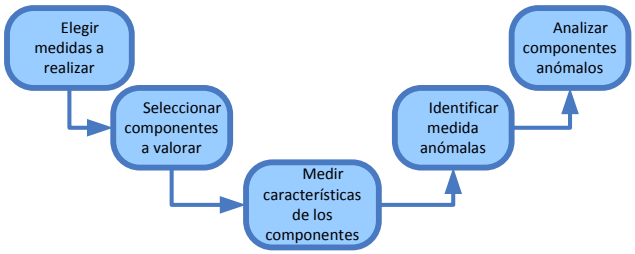

Figura 1. Proceso de medición del software

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.

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 calidad del trabajo realizado a nivel del proyecto [3].

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 modulo 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) [8].
- Métricas MOOD: PF (Polymorphism Factor), CF (Coupling Factor), MHF (Method Hiding Factor), AHF (Attribute Hiding Factor), MIF (Method Inheritance Factor), AIF (Attribute Inheritance Factor) [9].
- Lorenz y Kidd: SIX (Specialization Index per Class), LOC (Lines of Code per method), NOM (Number of Messages Send) [10].

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 [11]. Estas métricas de diseño, a las cuales suele aludirse con el nombre de métricas CK, son las siguientes:

1.3.1. DIT (Depth of Inheritance Tree):

La profundidad en 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.

1.3.2. 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 es un indicador 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 [8].

1.3.3. 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.

1.3.4. 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.

1.3.5. 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.

2 Metodología

En esta sección se describe la metodología utilizada para elaborar una propuesta de solución orientada a contribuir a la calidad del producto software automatizando el proceso de medición de aplicaciones orientadas a objetos.

Esta solución incluye la elaboración de una aplicación que toma los datos generados por herramientas disponibles en el mercado, que realizan el cálculo de las métricas seleccionadas, y genera informes que permiten a los equipos desarrolladores de software detectar los componentes (clases o métodos) que se encuentran por encima de los valores recomendados y que pueden generar fallas a futuro o afectar su comprensión y mantenibilidad.

Además, con el objetivo de mantener un registro histórico como recurso organizacional, los valores resultantes de la medición se almacenan en una base de datos que permitirá, posteriormente, la realización de análisis estadísticos que contribuyan a incrementar el conocimiento del producto y del proceso de desarrollo de software, y la adecuación de los valores recomendados de las métricas a los contextos específicos de desarrollo.

La metodología para el desarrollo de la propuesta tomó como referencia las etapas del proceso de medición propuesto por Sommerville [7], descritos en el apartado 1.1:

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 [8]. 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 una primera etapa en esta línea de trabajo.
   Para realizar una prueba de los módulos que calculan las métricas, se tomó el código de una aplicación open source denominada **jEdit.**
   jEdit es un editor de texto para programadores, escrito en Java, que soporta resaltado de sintaxis para más de 130 lenguajes: ActionScript , ASP, C, C++, C#, COBOL, ColdFusion, CSS, CVS, Fortran, Foxpro, HTML, Java, JavaScript, JSP, Latex, Lisp, Pascal, PHP, Prolog, Python, Ruby, Smalltalk, VBScript, XML, entre otros. Es software libre con código fuente completo, bajo los términos de la licencia GPL 2.0. La aplicación completa cuenta con 945 clases de java y 176.560 líneas de código. Se encuentra disponible en: http://sourceforge.net/projects/jedit/files/.

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

Con el objeto de realizar una valoración previa, se seleccionaron las herramientas *Eclipse metrics plugin* y *RefactorIt*, ambos complementos del entorno de desarrollo (IDE) Eclipse. Se descargó cada uno de los componentes de la página correspondiente, se los adicionó a Eclipse y se calcularon las métricas sobre la aplicación mencionada en el paso 2.

- **Cálculo con Eclipse metrics plugin**

  Habilitando la vista de las métricas (*Window\Show View\Metrics View*), este componente brinda un amplio conjunto de métricas. No permite seleccionar las métricas a calcular, pero sí establecer un orden de aparición en el resultado del cálculo (*Window\Preferences\Metrics Preferences*). También es posible establecer rango de confianza (*Window\Preferences\Metrics Preferences\Safe Ranges*).

  Una vez calculadas las métricas, son mostradas en la pantalla en el orden previamente establecido. Los valores que se encuentran fuera del rango aparecen coloreados de rojo, lo cual permite identificarlos de forma rápida. Las métricas calculadas pueden ser exportadas a un archivo con formato XML.

- **Cálculo con RefactorIT**

  Una vez incorporado el plugin a Eclipse, aparece el menú *RefactorIT* en la barra de menús. Este componente permite seleccionar cuáles métricas se desea calcular, y a la vez variar los intervalos de confianza. Por otra parte, brinda un breve comentario del significado de cada una de las métricas (*RefactorIt\Metrics*).

  Una vez calculadas las métricas, los valores son mostrados de forma organizada. Los datos que aparecen en rojo indican que los valores obtenidos

están fuera del rango establecido. El resultado del cálculo puede ser exportado a diversos formatos (Texto Plano, CSV, XML, HTML).

Para el cálculo se establecieron como rangos de confianza requeridos por las herramientas, los valores umbrales derivados de la literatura. Se compararon los resultados de los cálculos de las métricas con ambas herramientas. Dado que ambas ofrecían valores coincidentes, se seleccionó RefactorIT para esta propuesta, 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 rangos de confianza; brinda diversos formatos para exportar resultados y, además de las métricas CK, implementa otras.

4. Para los dos últimos pasos *(Identificar las mediciones anómalas y Analizar los componentes anómalos*) se propone el desarrollo de una aplicación para la automatización de la medición, que permite tomar los datos generados por el módulo *RefactorIT*, luego de evaluar las métricas CK, y brindar la información que se requiere para cumplir los objetivos de esta etapa.

Descripción de la aplicación de automatización de medición

La aplicación toma la información generada por la herramienta de cálculo de las métricas, cuyos datos son:

- Locación: es el nombre del elemento medido
- Tipo: es el tipo de elemento (clase, interface, enum, etc)
- WMC: valor de la métrica para el elemento
- RFC: valor de la métrica para el elemento
- DIT: valor de la métrica para el elemento
- NOC: valor de la métrica para el elemento
- LCOM: valor de la métrica para el elemento

Las funcionalidades previstas son:

a) Seleccionar el formato del archivo a leer (CSV, XML, HTML).
b) Ingresar rango de valores de confianza utilizados en la medición, para almacenarlos con los datos históricos.
c) Aplicar filtro a los datos (ejemplo, sólo las clases, y sólo aquellas que tengan algún valor de métrica fuera del rango establecido).
d) Elaborar un informe detallado que incluya las clases con posibles dificultades, y alguna sugerencia para su revisión, basado en los valores de las métricas.
e) Elaborar un informe resumen, que indique porcentajes de clases con posibles dificultades, en el total de la aplicación analizada, de modo de tener una idea aproximada de la calidad del software
f) Persistir los datos, en una base de datos históricos, que permita generar un recurso organizacional, con el propósito a futuro de realizar un refinamiento de las métricas y de su significado, según las necesidades organizacionales.

La interfaz de la aplicación es la siguiente:

Figura 2: Interfaz de la aplicación

El diseño de los datos históricos incluye los siguientes elementos:

- Nombre del sistema/aplicación, Fecha de medición, Cantidad de clases de la aplicación/sistema
- Locación: es el nombre del elemento medido
- Tipo: es el tipo de elemento (clase, interface, enum, etc)
- WMC: valor de la métrica, Límite inferior y superior usado para WMC
- RFC: valor de la métrica, Límite inferior y superior usado para RFC
- DIT: valor de la métrica, Límite inferior y superior usado para DIT
- NOC: valor de la métrica, Límite inferior y superior usado para NOC
- LCOM: valor de la métrica, Límite inferior y superior usado para LCOM

Síntesis de la solución propuesta:

Considerando los elementos descriptos en los apartados anteriores, para evaluar la complejidad de programas orientados a objetos realizados en Java, la solución propuesta consiste en:

1. Incorporar el código Java de la aplicación OO a evaluar en el IDE Eclipse.
2. Realizar la medición usando el módulo RefactorIT
3. Exportar los valores resultantes
4. Ejecutar la aplicación propuesta para emitir los informes y persistir los datos en la base de datos histórica
5. Analizar los resultados obtenidos e implementar mejoras correctivas, si fuera necesario.

4 Conclusiones

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. Por lo tanto, esta solución contribuye a facilitar la medición del software, dado que propicia un marco de trabajo que permite 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 permita realizar predicciones y una adecuación del proceso de medición al contexto organizacional específico.

Como trabajo futuro se propone adicionar un indicador de fallas que surja de los reportes del software en producción, asociado a las clases que las produjeron, con la finalidad de relacionar la complejidad del diseño de clases con probables fallas y tener, además, una medida del esfuerzo que implica la corrección y que afecta la mantenibilidad. Asimismo, se propone ampliar la medición abarcando código fuente de otros lenguajes de programación del paradigma OO.

Referencias

1. 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, Diciembre 2004, 11-20.
2. 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). www.issi.uned.es/CalidadSoftware/Noticias/PonIng2005.rtf
3. Pressman, R. "Ingeniería de Software. Un enfoque práctico". McGraw-Hill. 2005.
4. Avello, D.G., A. Cernuda del Río. "Reflexiones y Experiencias sobre la Enseñanza de POO como único Paradigma. JENUI 2003. Cádiz, Spain.
5. Khaled El Emam,: "A Primer on OO Measurement", 1530-1435/05 IEEE, Proceeding of the Seventh International Software Metrics Symposium.
6. Piattini M., García O., Caballero I. "Calidad de los sistemas informáticos".RA-MA Editorial, España, MADRID, 2007.
7. Sommerville, Ian. Ingeniería del Software. Séptima Edición. Madrid : Pearson Educación, S.A., 2005. 84-7829-074-5
8. Rodríguez, D., Harrison, R., "Medición en la Orientación a Objeto", in Medición para la Gestión en la Ingeniería del Software, Dolado, J. and Fernández, L., Eds., RA-MA, 2000, ISBN 84-7897-403-2.
9. Rosenberg, Linda H. Applying and Interpreting Object Oriented Metrics. [En línea] http://satc.gsfc.nasa.gov/support/STC_APR98/apply_oo/apply_oo.html.
10. Etzkorn, Letha y Delugach, Harry. s.l. Towards a Semantic Metrics Suite for Object-Oriented Design. IEEE, 2000. 07695-0774-3.
11. Chidamber, S., Kemerer, C., "A Metrics Suite for Object-Oriented Design." M.I.T. Sloan School of Management E53-315, 1993.