Evaluación de mantenibilidad de un gestor de contenidos open source utilizando métricas de orientación a objetos

Julio Acosta[1]; Gladys Dapozo[1], Cristina Greiner[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
julio_acosta_01@hotmail.com, {gndapozo, cgreiner}@exa.unne.edu.ar
[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@gmail.com

Resumen: Los aspectos vinculados a la mantenibilidad contribuyen a la calidad del software y también a la decisión favorable de adoptarlo para su uso, en el caso particular del software libre. En este artículo se presentan los resultados de la evaluación de un gestor de contenidos de software libre con el objetivo de conocer el grado de mantenibilidad del mismo, aplicando métricas Orientadas a Objetos, utilizando la metodología Goal-Question-Metrics. De acuerdo a los resultados obtenidos se puede presumir que el software Joomla es un proyecto estable y maduro, en el que se observa un crecimiento sistemático a lo largo del tiempo, y mejoras en el aspecto específico de mantenibilidad evaluado.

Palabras claves: calidad de software, mantenibilidad, software libre

1. Introducción

1.1. Importancia de la mantenibilidad en el software libre

En la elección de una plataforma de desarrollo, lenguaje de programación, gestores de contenidos web etc., un factor crucial a la hora de decidir cuál adoptar es la calidad del soporte que brindan al usuario y la frecuencia de las actualizaciones que incorporan correcciones a los errores reportados por los usuarios. Este es un factor que las organizaciones tienen en cuenta en la decisión de adquirir software propietario en lugar de software libre.

En general, los desarrolladores de software con licencia propietaria lo hacen bajo procesos de desarrollo certificados por normas de calidad, por tanto pueden establecer un contrato de garantía de calidad con sus compradores, situación que no se presenta en el desarrollo de software libre no generan software dado que, en normalmente no existe una empresa constituida formalmente con miembros totalmente dedicados a la producción del software, sino que son desarrollos colaborativos en los cuales contribuyen cientos de programadores distribuidos por todo el mundo.

Sin embargo, el software libre ha crecido considerablemente debido precisamente a su filosofía de desarrollo abierta y distribuida, permitiendo generar cada vez software más confiable y mantenible, debido a las constantes actualizaciones que se lanzan ante los reportes de errores. En la actualidad, juega un importante papel en el mundo de Internet, donde más del 67% de los servidores son libres. Esto da cuenta de que

Internet tal y como se conoce no sería lo mismo sin software libre, o al menos sería más costoso [1].
En este contexto, interesa conocer las características del software libre en relación al atributo de mantenibilidad. De acuerdo con Herbold [2], la mantenibilidad describe aspectos no funcionales como testeabilidad, comprensibilidad, extensibilidad. Señala que si estas características se cumplen satisfactoriamente, se puede decir que el software será mantenible y por lo tanto, la detección como la corrección de errores será rápida y eficiente, entre otras ventajas.
Por su parte, la norma ISO 9126 sostiene que mantenibilidad es uno de los factores que todo software de calidad debe cumplir, y para lograrlo el software debe tener entre sus características la facilidad de análisis para la rápida detección de errores, facilidad de cambio y corrección, así también como la facilidad de pruebas.
De este modo, en un software mantenible, ante el reporte de un error, este debería ser rápidamente detectado y corregido.
Por tal motivo, conocer el grado de mantenibilidad de un producto software contribuye favorablemente en la decisión de utilización del mismo.
Para inferir la facilidad de mantenimiento es necesario realizar determinadas mediciones en el software. Según Fenton y Pfleeger [3], la medición es el proceso por el cual se asignan números o símbolos a los atributos de las entidades en el mundo real, de forma que los describen de acuerdo con reglas claramente definidas. Estas reglas son conocidas como “métricas". De acuerdo a los atributos de calidad observados, se obtendrá un conjunto de métricas que indicarán el grado en que un producto se ajusta a un estándar definido, comparándolo con un umbral previamente establecido.

1.2. El proceso de medición del software – Métricas OO

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.).
Las métricas para sistemas orientados a objetos (OO) hacen hincapié en los conceptos básicos del paradigma de programación OO, tales como el encapsulamiento, la ocultación de la información, la herencia, técnicas de abstracción y polimorfismo.
El ocultamiento de información suprime los detalles operativos de un componente de un programa, brindando solamente la información necesaria para acceder a ese componente. Un sistema OO bien diseñado debería impulsar al ocultamiento de información. Por tanto, aquellas métricas que proporcionen un índice del grado de ocultamiento proporcionarán un índice de la calidad del diseño OO.
Por su parte, la herencia es un mecanismo que hace posible que los compromisos de un objeto se difundan a otros objetos. La herencia se produce a lo largo de todos los niveles de la jerarquía de clases. Las métricas relacionadas a esta característica cuentan el número de descendientes (número de instancias inmediatas de una clase), número de predecesores (número de generalizaciones inmediatas), grado de anidamiento de la jerarquía de clases (profundidad de una clase dentro de una jerarquía de herencia) y otros. Son un indicador de la reutilización, así también como de la complejidad.

La abstracción es un mecanismo que permite al diseñador centrarse en los detalles esenciales de algún componente de un programa (dato o proceso) dejando de lado los detalles de nivel inferior. Cuando los niveles de abstracción van elevándose, se obtiene una visión más general de un concepto u objeto. De manera inversa, niveles menores de abstracción, proporcionan una visión más específica de un concepto u objeto. Dado que una clase es una abstracción que se puede visualizar con distintos niveles de detalles, y de muchas maneras diferentes, las métricas OO representan la abstracción en términos de medidas de una clase.
Diversos estudios identificaron la correlación entre métricas OO, como las métricas CK (Chidamber y Kemerer), y la localización de defectos [4], la relación entre acoplamiento de objetos y atributos de calidad externos con la propensión a fallos, y la aplicación de medición de acoplamiento para tareas de mantenimiento del software [5]. Por su parte, Cartwright and Shepperd [6] analizaron un sistema desarrollado en C++ (de 1333 kloc) y observaron que las clases que participan en jerarquías de herencia de muchos niveles tienen mayor densidad de defectos que otras clases. Por tanto, las métricas que indican profundidad en el árbol de herencia y número de hijos pueden ser usadas para identificar clases que son complejas y propensas a fallo [4].
Por su parte Arisholm [7] menciona que las métricas de acoplamiento son indicadores significativos de propensión a cambios al igual que las clases de gran tamaño [8].
De igual modo, existe una variedad de métricas OO que son un buen indicador de mantenibilidad, y que están discriminadas en función del aspecto a medir (diseño, complejidad, etc.).
A continuación una breve reseña de las métricas utilizadas en este trabajo, que permiten determinar el grado de mantenibilidad.

1.2.1. Métricas MOOD (Metrics for Object oriented Design)

- **MHF (**Method Hiding Factor) - Proporción de métodos ocultos: Es la proporción de la suma de los *métodos* privados en todas las clases respecto al número total de métodos definidos en el sistema. Se propone como una medida de encapsulamiento y cantidad relativa de información oculta. Como se señaló, un mayor grado de ocultamiento da cuenta de la calidad del diseño. En [9] se ha demostrado empíricamente que cuando se incrementa el valor de esta métrica, la densidad de defectos y el esfuerzo necesario para corregirlos disminuye.
- **AHF** (Attribute Hiding Factor) - Proporción de atributos ocultos: De manera similar, representa la proporción de la suma de las invisibilidades de los *atributos* en todas las clases respecto al número total de atributos definidos en el sistema. Se propone como una medida de encapsulamiento. Es recomendable que todos los atributos sean privados, por lo que, idealmente, la proporción de invisibilidad debe ser el 100%.
- **MIF** (Method Inheritance Factor) - Proporción de métodos heredados: Representa la proporción de la suma de todos los *métodos* heredados en todas las clases respecto al número total de métodos (localmente definidos más los heredados) en todas las clases. Esta métrica da indicios del nivel de reuso, pero a la vez puede señalar una disminución en la comprensibilidad.
- **AIF** (Attribute Inheritance Factor) - Proporción de atributos heredados: Señala la proporción del número de *atributos* heredados respecto al total de atributos. Tiene idéntica interpretación que MIF.

1.2.2. Métricas a nivel de acoplamiento

- **CBO** (Coupling Between Objects) - Acoplamiento entre Objetos: Para una clase determinada indica el número de clases a las cuales está ligada. Existe dependencia entre dos clases cuando una de ellas usa métodos o variables de la otra. Para el cálculo de esta métrica no se consideran las clases relacionadas por herencia. Un alto nivel de acoplamiento dificulta la comprensibilidad. Chidamber y Kemerer [10] suponen que es un indicador del esfuerzo necesario para el mantenimiento y en el testeo. Consideran como umbral un límite superior de 5.

1.2.3. Métricas a Nivel de Clases

- **DIT (**Depth of Inheritance Tree**)** - Profundidad en árbol de herencia: Mide la distancia entre un nodo y la raíz en una jerarquía de herencia. En el nivel cero de la jerarquía se encuentra la clase raíz. Esta métrica permite predecir la complejidad de una clase y el potencial de reuso. Altos niveles de herencia indican objetos complejos y bajos niveles implican código escrito de manera funcional. Como umbral se considera un límite superior de 6, valor tomado de otra herramienta de medición.
- **WMC** (Weighted Methods per Class) - Métodos ponderados por clase: Mide la complejidad de una clase. El número de métodos y su complejidad es un indicador razonable de la cantidad de esfuerzo necesaria para implementar y comprobar una clase. Además, cuanto mayor sea el número de métodos, más complejo será el árbol de herencia, (todas las subclases heredan el método de sus predecesores). La medida de la complejidad de los métodos se debe normalizar. Si todos los métodos se consideran igualmente complejos, esta métrica se obtiene como la suma de métodos definidos en una clase. Clases con un gran número de métodos requieren más tiempo y esfuerzo para desarrollarlas y mantenerlas. Por otra parte, a medida que el número de métodos crece para una clase dada; es más probable que se vuelva cada vez más específico de la aplicación, limitando por tanto su potencial de reutilización. Por todas estas razones, WMC debería mantener un valor tan bajo como sea razonable. Lorenz y Kidd [10] proponen que los resultados de las mediciones deben estar entre 20 y 40.

1.3. Gestores de contenidos Web

Actualmente es imprescindible tanto para instituciones, empresas y personas particulares tener un sitio web para brindar los diferentes servicios o productos, estos sitios requieren mostrar diferentes contenidos los cuales pueden ser administrados por los gestores de contenidos (CMS - Content Management System), cuya funcionalidad es administrar contenidos en un medio digital.
Un CMS es una herramienta para crear, editar, gestionar y publicar contenido digital en diversos formatos (como texto, gráficos, vídeo, documentos, etc.), principalmente en sitios web, ya sea en Internet o en una intranet. El gestor de contenidos genera páginas dinámicas interactuando con el servidor para generar la página web bajo petición del usuario, con el formato predefinido y el contenido extraído de la base de datos del servidor. Esto permite gestionar, bajo un formato estándar, la información del servidor, reduciendo el tamaño de las páginas para descarga y reduciendo el costo de gestión del sitio con respecto a una página estática [11].

Una herramienta CMS generalmente contendrá una interfaz basada en formularios, a los que habitualmente se accede con el navegador, donde se pueden agregar fácilmente los contenidos, que luego aparecerán en la página en los lugares donde se ha indicado al darlos de alta. Por lo tanto, un CMS estará compuesto de dos partes: un back y un front, siendo el back la parte donde los administradores publican las informaciones y el front la parte donde los visitantes visualizan las mismas.
En la última década se experimentó un crecimiento importante de los CMS, convirtiéndose en plataformas de desarrollo web completas. La gran mayoría de los CMS son libres, y entre los más populares se encuentran WordPress, Joomla, Website Tonight, Blogger, Homestead y Drupal [12]. WordPress y Joomla son sistemas de gestión de contenidos de gran volumen de información que permiten administrar y crear sitios web desde cero y rápidamente.

1.4. Herramienta PHP Depend

Es una herramienta que permite obtener distintas métricas de atributos de calidad de software mediante el análisis de código estático. A partir del código fuente genera una estructura de datos interna fácilmente procesable, denominada AST (Árbol de Sintaxis Abstracta), que representa las diferentes sentencias y elementos utilizados en el código fuente analizado [13]. Mide y reporta valores que representan aspectos de calidad, que constituyen métricas de software. Realiza dos tipos de análisis:

a) **Overview Pyramid (OP):** muestra una visualización general del proyecto [14], agrupando las métricas implementadas en tres categorías: Herencia (*Inheritance*), Acoplamiento (*Coupling*) y Tamaño y Complejidad (*Size&Complexity*), que se presentan gráficamente en una figura piramidal con la estructura que se muestra en la Figura 1.Cada categoría incluye las siguientes métricas[15]:
   - Inheritance
     ANDC: Promedio de clases derivadas por clase.
     AHH: Promedio de altura de herencia por clase.
   - Coupling:
     CALLS: Cantidad de llamadas a métodos diferentes.
     FANOUT: Cantidad de clases colaboradoras que utiliza una clase.
   - Size&Complexity:
     NOP: Cantidad de paquetes.
     NOC: Cantidad de clases.
     NOM: Cantidad de métodos.
     LOC: Cantidad de líneas de código.
     CYCLO: Suma la complejidad ciclomática de cada método del proyecto.

   En cada peldaño de la pirámide se ubica el valor del cociente de normalización aplicado a la métrica de la fila inferior (Ver Figura 1). Por ejemplo, en el caso de NOM (*number of methods*) se divide por NOC (*number of clases*) para adaptar dicha medición al tamaño del proyecto que se está evaluando.

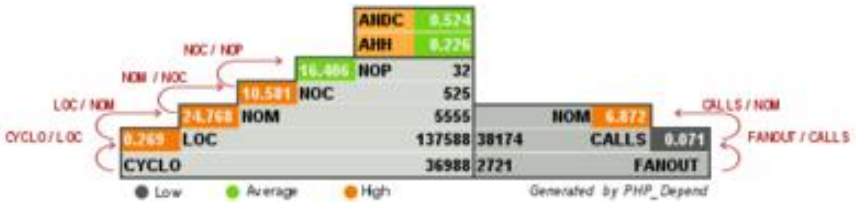

Figura 1. Valores de las métricas y cocientes de normalización en la pirámide

A su vez, los valores tienen un fondo de color que permite visualizar rápidamente las categorías de cumplimiento en cuanto a los valores aceptables, según la escala de intervalos que se muestran en la Tabla 1. Esta escala clasifica como Promedio a aquellas mediciones cuyos resultados son mayores que el indicado como Bajo y menores que el indicado como Alto, para cada medición respectivamente. Los valores umbrales indicados como Bajo y Alto son los que se encuentran por defecto en la herramienta PHPDepend.
Los valores clasificados como Bajos se referencian con fondo de color gris oscuro, los clasificados como Altos con fondo color anaranjado y los valores Promedio con fondo de color verde.

Tabla1: Escala de valores aceptables para las métricas analizadas

Métrica	Bajo	Promedio	Alto
CYCLO/LOC	0,16	0,20	0,24
LOC/NOM	7	10	13
NOM/NOC	4	7	10
NOC/NOP	6	17	26
CALLS/NOM	2,01	2,62	3,2
FANOUT/CALLS	0,56	0,62	0,68
ANDC	0,25	0,41	0,57
AHH	0,09	0,21	0,32

b) Abstraction Instability Chart (AI). La flexibilidad y extensibilidad propias de la POO son factores que hacen a la calidad del software. Estas características dependen en gran medida de un adecuado nivel de acoplamiento entre objetos, lo cual contribuye a la mantenibilidad del software. Por tal motivo, es deseable reducir las dependencias entre las clases. Esto podría lograrse mediante el uso de clases abstractas e interfaces, en lugar de implementación concreta en la aplicación, lo cual implica algún tipo de contrato. De este modo se flexibiliza el código al permitir a una aplicación implementar sus propias clases para cumplir el contrato. Esta es una característica de la POO que reduce el riesgo de que al modificar una clase o paquete, esto repercuta en el resto del producto.
Esta opción de la herramienta [16] indica la calidad del diseño en términos de extensibilidad, reutilización y mantenibilidad, en función de las dependencias y la abstracción de paquetes, basado en la propuesta de Martin [17]. Este análisis establece una relación entre la proporción de clases abstractas y la estabilidad de la clase, es decir, trata de definir un valor de estabilidad de acuerdo a la proporción de clases abstractas de cada paquete.

Las métricas que se consideran en este análisis son:

- Ca - Acoplamiento aferente: número de paquetes que dependen de clases dentro del paquete analizado. Es un indicador de cómo influye algún cambio en el paquete en el resto del proyecto analizado.
- Ce - Acoplamiento eferente: número de paquetes de los cuales dependen clases del paquete analizado. Es un indicador de cuan sensible a cambios en otro lugar del proyecto es un paquete.
- I-Instability: Es la proporción entre Ce y el acoplamiento total (Ce/(Ce+Ca)). I=0 indica máxima estabilidad del paquete, es decir no depende de nadie. I=1 indica dependencia total de otros paquetes.
- A-Abstractness: Es la proporción entre clases abstractas (AC) y el total de las clases (AC/(AC+CC)). A=0 implica que no existen clases abstractas en el paquete mientras que A=1 implica un paquete compuesto solo de clases abstractas.

Es deseable que todos los paquetes se ubiquen sobre la diagonal. La Figura 3 muestra la disposición de los paquetes según las métricas del proyecto analizado.

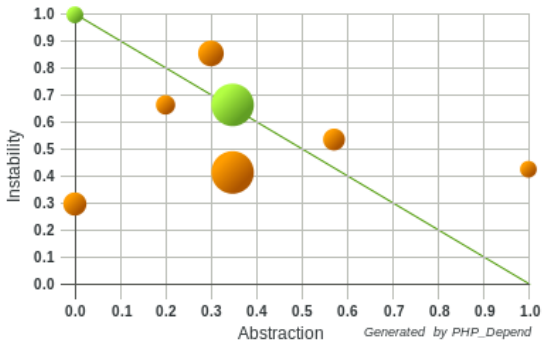

Figura3. Abstraction Instability Chart (AI)

1.5. GQM (Goal-Question-Metric)

GQM contribuye a identificar, documentar y analizar un número reducido de métricas con la intención de mejorar un objetivo, producto, proceso o sus recursos.
El método se basa en tres niveles:

- Nivel conceptual: Se identifica un objetivo de calidad que constituye el propósito de la medición en relación a una entidad-producto, proceso o recurso.
- Nivel operacional: Divide el objetivo en una serie de preguntas que caracterizan a la entidad.
- Nivel cuantitativo: Especifica el conjunto de métricas necesarias para poder responder a las preguntas planteadas en el segundo nivel.

El funcionamiento del método se basa en el refinamiento progresivo de un conjunto de objetivos de negocio (G-Goals) que se establecen como partida. Tomando dichos objetivos como entrada y mediante el planteamiento de preguntas (Q-Questions), se obtienen finalmente un conjunto de métricas (M-Metrics) específicas que permitirán medir los objetivos enunciados.

2. Evaluación de gestores de contenidos

Los gestores de contenidos web open source constituyen un estándar de facto en el desarrollo de sitios web para muchos desarrolladores.
En este trabajo se seleccionó CMS Joomla! por ser uno de los más utilizados. Es un CMS de código abierto y es distribuido bajo licencia GPL. Permite crear sitios web elegantes, dinámicos e interactivos en los que se pueden incluir publicación de noticias, blogs, directorios de enlaces o documentos para descargar sin necesidad de conocimientos técnicos especiales o de complejos lenguajes de programación. También facilita la creación de sistemas que funcionen en redes cerradas (intranets) [18].
Para el análisis de calidad del CMS Joomla, orientado al factor mantenibilidad, se consideraron las tres últimas versiones lanzadas al mercado:

- Joomla!1.7 (sin soporte actual)
- Joomla! 2.5 (con soporte hasta diciembre del 2013)
- Joomla! 3 (con soporte hasta abril del 2013)

El objetivo del análisis es inferir el grado de mantenibilidad mediante la generación de indicadores que aporten información sobre el costo de mantenimiento y el esfuerzo de actualización y corrección de errores que podrían tener las futuras versiones aun no lanzadas.

2.1. Metodología de evaluación

Para obtener información acerca del grado de mantenibilidad del software que se evalúa, se realizó el cálculo de métricas OO que aportan información vinculada a la facilidad de mantenimiento. Se utilizó la herramienta PHDepend dado que presenta características que la hacen apropiada para este estudio validada en trabajos anteriores [19]. Un primer análisis se orientó a analizar el grado de estabilidad del software y otro análisis se enfocó en los aspectos específicos de la mantenibilidad.

2.1.1. Análisis comparativo global

Se realizó un primer análisis, utilizando la herramienta PHPDepend, tomando como referencia las etapas del proceso de medición de Morasca en Sommerville [20].

1. *Seleccionar las medidas a realizar:* Dado que la medición está orientada a obtener un análisis comparativo global, se realizaron mediciones sobre tamaño, uso de herencia, acoplamiento, abstracción y estabilidad del software.
2. *Selección de los componentes a evaluar*: Se seleccionaron para la evaluación las tres últimas versiones del CMS Joomla!
3. *Medir las características de los componentes*: Para la medición se seleccionó la herramienta open source PHPDepend, dado que funciona independientemente del IDE utilizado para la construcción del software, y presenta informes completos de una variedad de métricas OO. PHPDepend analiza cada elemento del software medido y genera los informes Overview-Pyramyd (OP) y Abstraction Inestability Chart (AI).

4. *Identificar las mediciones anómalas:*
   Los gráficos de las Figura 4, 5 y 6 muestran los resultados del análisis de OP para cada versión evaluada. Los valores se encuentran coloreados según su relación con la escala propuesta: Bajo (gris oscuro), Promedio (verde) y Alto (anaranjado), tal como se indica al pie de la pirámide.

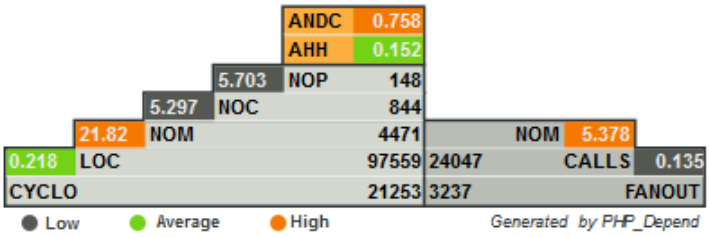

Figura 4. Análisis OP Joomla! 1.7

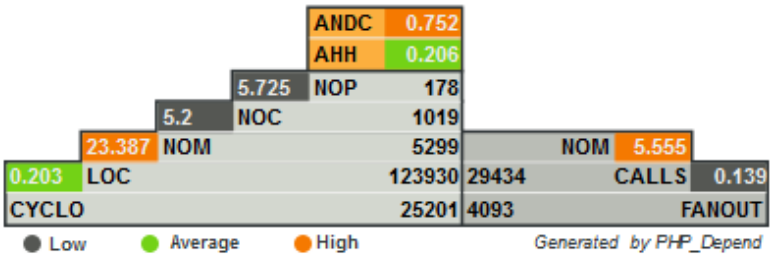

Figura 5. Análisis OP Joomla! 2.5

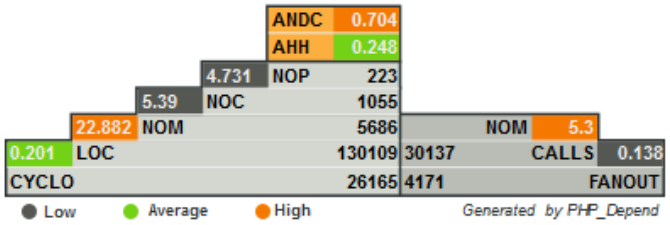

Figura 6. Análisis OP Joomla! 3

Respecto al análisis de estabilidad, en la Figura 7, 8 y 9 se muestran los gráficos resultantes para las versiones analizadas:

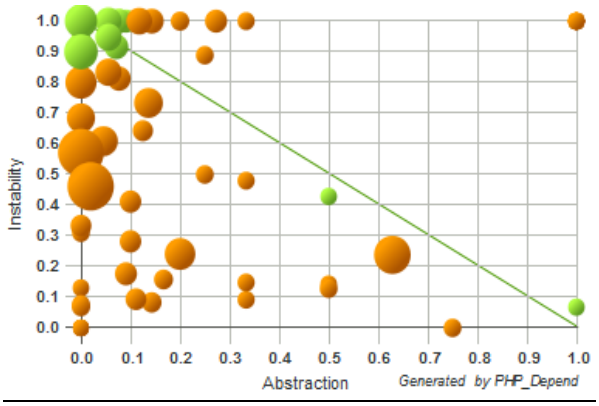

Figura 7: Abstraction Instability Chart (AI) Joomla! 1.7

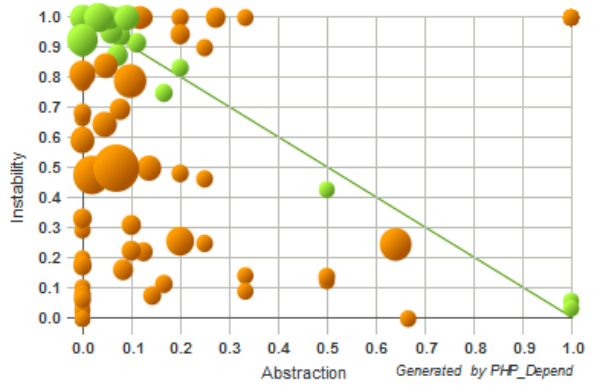

Figura 8: Abstraction Instability Chart (AI) Joomla! 2.5

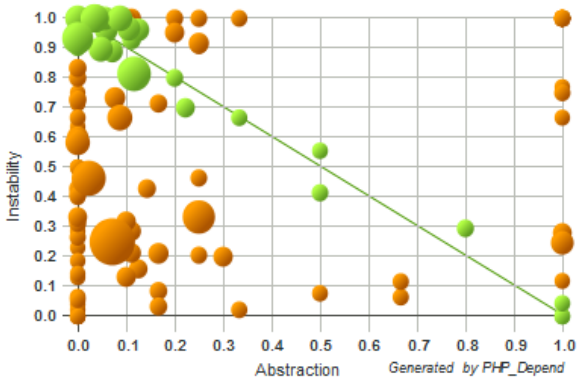

Figura 9: Abstraction Instability Chart (AI) Joomla! 3

5. Análisis de los resultados

Del análisis OP se concluye que no existen grandes variaciones en los valores de las mediciones, a pesar de haber experimentado un importante crecimiento entre la versión 1.7 y 3, de más de 30000 líneas de código. Del análisis de estabilidad (AI), se observa que a medida que el proyecto fue evolucionando y creciendo en tamaño, los paquetes se distribuyeron mejor, acercándose a la diagonal donde es deseable que se ubiquen. Esto indica que se fue mejorando la flexibilidad y mantenibilidad.

2.1.2. Análisis de mantenibilidad

Para precisar la medición orientada a los aspectos de mantenibilidad se utilizó el método GQM definiendo el objetivo, las preguntas y las métricas, según la estructura de la Tabla 2.

Tabla 2: Método GQM

Objetivo	Preguntas	Métricas
Conocer las características de mantenibilidad del software de gestión de contenidos **Joomla!**	**Q1**: ¿Cuál es la densidad de defectos del software?	**M1: MHF** (Proporción de métodos ocultos). Cuando incrementa MHF la densidad de defectos y el esfuerzo necesario para corregirlos disminuye.
	Q2: ¿Cuál es el grado de comprensibilidad del sistema?	**M2: MIF** (Proporción de métodos heredados). El uso de la herencia es visto como un compromiso entre la reusabilidad y la comprensibilidad.
		M3: AIF (Proporción de atributos heredados).
		M4: DIT (Profundidad en árbol de herencia).
	Q3: ¿Cuál es el grado de complejidad del software?	**M5: CBO** (Acoplamiento entre Objetos). Es un indicador del esfuerzo en el mantenimiento y testeo.
		M6: DIT (Profundidad en árbol de herencia). Es una medida de la complejidad de una clase y su potencial de reuso.
		M7: WMC (Métodos ponderados por clase). Es una medida de complejidad de una clase.

2.1.3. Procedimiento para el cálculo de las métricas

Para obtener los valores de las métricas requeridas, se utilizaron los valores resultantes de la herramienta PHPDepend, y se realizó una serie de cálculos adicionales dado que no todas las métricas están consideradas en esta herramienta. Para ello se procedió a la elaboración de algoritmos codificados en el lenguaje PHP que leen el archivo XML generado por la herramienta PHPDepend y realizan los cálculos necesarios para cada una de las métricas que se explican a continuación.

- **MHF**: El factor de ocultamiento de métodos no es una métrica implementada directamente por PHPDepend por lo que se obtuvo a través de NOM (número de métodos) y NPM (número de métodos no privados). Mediante la resta de NOM-NPM se obtiene el número de métodos privados de una clase. Luego se sumó el total de métodos privados y se dividió por NOM para obtener la proporción de métodos privados con respecto al total, logrando así el valor de MHF, tal como se indica en la siguiente fórmula, donde i es la cantidad de clases y j la cantidad de paquetes.

$$\frac{\sum_{j=1}^{m}(\sum_{i=1}^{n}(NOM_i - NPM_i))_j}{\sum_{j=1}^{m}\left[\sum_{i=1}^{n} NOM_i\right]_j}$$

- **MIF:** De la misma forma, para obtener la proporción de métodos heredados, se obtuvo la cantidad de métodos heredados utilizando las métricas NOAM (número de métodos añadidos) y NOM, realizando la resta NOM - NOAM. Luego se obtuvo la proporción de métodos heredados, obteniendo el valor de MIF.

$$\frac{\sum_{j=1}^{m}(\sum_{i=1}^{n}(NOM_i - NOAM_i))_j}{\sum_{j=1}^{m}\left[\sum_{i=1}^{n} NOM_i\right]_j}$$

- **AIF:** para el cálculo de la proporción de atributos heredados se utilizaron las métricas VARS (cantidad de atributos de una clase) y VARSI (Cantidad de atributos heredados). VARS solo contabiliza la cantidad de atributos agregados en una clase, por lo que para obtener la cantidad total de atributos fue necesario sumar VARS+VARSI. Para obtener la proporción de atributos heredados, se divide VARSI/(VARS+VARSI).

$$\frac{\sum_{j=1}^{m}(\sum_{i=1}^{n}(VARSI_i))_j}{\sum_{j=1}^{m}\left[\sum_{i=1}^{n} VARS_i + VARSI_i\right]_j}$$

- **CBO, DIT y WMC** son métricas implementadas por la herramienta utilizada, por lo que se sumó el valor total arrojado por la medición para

cada una de las clases y luego se dividió por la cantidad de clases para establecer la media.

2.1.4. Resultados de las mediciones

En la tabla 3 se muestran los valores de las métricas para cada una se las versiones del software evaluado.

Tabla 3: Resultados de las métricas en GQM

Métrica	Joomla! 1.7	Joomla! 2.5	Joomla! 3
M1:MHF	0,18585	0,21073	0,21757
M2:MIF	0,65873	0,65187	0,68753
M3:AIF	0,83127	0,83325	0,82829
M4:CBO	3,77251	3,95388	3,88531
M5:DIT	1,65521	1,81256	1,48246
M6:WMC	24,46682	24,07753	24,13839

Los valores de las métricas y el cumplimiento o no de los estándares recomendados, permiten responder a las tres preguntas que aportarán información para lograr el cumplimiento del objetivo planteado.

a) **Q1**: ¿Cuál es la densidad de defectos del software?
   M1) MHF: Se observa que la proporción de métodos ocultos con respecto a los públicos aumenta de 18% a aproximadamente 22% en las distintas versiones. Como se señaló, se demostró empíricamente que cuando se incrementa el valor de esta métrica, la densidad de defectos y el esfuerzo necesario para corregirlos disminuye.
   Respuesta: Se puede concluir que disminuye la densidad de defectos del proyecto Joomla y, consecuentemente, el esfuerzo para corregirlos.

b) **Q2:** ¿Cuál es el grado de comprensibilidad del sistema?
   M2) MIF: Se observa que la proporción de métodos heredados con respecto al total de métodos es de casi 70% y se ha mantenido estable en todas las versiones. Alto nivel de MIF implica un buen nivel de reuso.
   M3) AIF: De manera similar, se observa una alta proporción en la herencia de atributos, lo que indica también buen nivel de reuso.
   M4) DIT: La profundidad en el árbol de herencia tiene valores que no superan el umbral, lo que indica un adecuado nivel de complejidad.

Respuesta: Se observa un alto nivel de herencia en métodos y atributos, lo cual indica buena reutilización. Valores altos en métricas de herencia podrían significar posibles costos en la comprensibilidad del sistema. Sin embargo, si se considera la métrica que señala la profundidad en el árbol de herencia (DIT), se observa que ésta se mantiene dentro de los valores recomendados, por lo cual se puede sostener que se mantiene el equilibrio entre reuso y complejidad. Se puede inferir que la conjunción de estos valores señala un buen grado de comprensibilidad del sistema.

c) **Q3:** ¿Cuál es el grado de complejidad del software?
M5) CBO: Se observa que su valor no sobrepasa el umbral, indicando un nivel de acoplamiento conveniente, lo cual se traduce en un adecuado nivel de complejidad en las relaciones de los objetos.
M6) DIT: niveles muy altos en la profundidad en el árbol de herencia pueden indicar complejidad en las clases. Sin embargo, se observa que los valores obtenidos están dentro del umbral recomendado.
M7) WMC: Este valor, que señala la complejidad de los métodos, y por ende la de las clases, se encuentra dentro del rango recomendado.
Respuesta: Las tres mediciones observadas para esta pregunta tienen valores dentro de los umbrales deseados, por lo que se concluye que la aplicación no posee un alto grado de complejidad.

2.1.5. Comparación de los resultados obtenidos con otras mediciones.

En [19] se realizó una medición de atributos POO en frameworks de desarrollo PHP con el objetivo de obtener una aproximación de cuáles de ellos implementan mejor criterios de calidad (en términos de atributos POO). Se realizaron mediciones sobre cada uno de ellos, utilizando el mismo conjunto de métricas y umbrales. Se obtuvo un ranking de calidad, que permite orientar la selección, con la idea de que utilizar un framework que cumple criterios de calidad, contribuirá a la calidad del producto que se desarrolla.
La medición realizada en el presente trabajo, que compara distintas versiones de un mismo producto, en las cuales se mantienen la arquitectura y gran parte de la organización del sistema, tiene la ventaja de permitir el aporte de sugerencias de mejora de calidad para algún atributo en particular del software analizado.

3. Conclusión

Los resultados aportados por las métricas OO aplicados a tres versiones diferentes del gestor de contenidos open source Joomla! permiten presumir que el software responde a las características de un proyecto estable y en continuo crecimiento. En tanto, los resultados de la medición GQM brindan valores favorables que permiten inferir que el software es mantenible por tanto, no sería costosa la implementación de nuevas versiones.

Referencias

1. Cúlebro, J.; Gómez H., Tórres Sanchez, S. “Software libre vs software propietario Ventajas y desventajas". http://www.rebelion.org/docs/32693.pdf.
2. Herbold, S.; Grabowski, J.; Waack. S. Calculation and optimization of thresholds for sets of software metrics. Empir Software Eng (2011)
3. Fenton N. E., Pfleeger S. L. Software Metrics. A rigorous and Practical Approach. 2nd Edition. ITP, International Thomson Compute Press, 1997.
4. Kpodjedo, S.; Ricca, F.; Galinier, P.; Guéhéneuc, Y.; Antoniol, G. Design evolution metrics for defect prediction in object oriented systems. Empir Software Eng (2011)
5. Revelle, M.; Gethers, M.; Poshyvanyk, D. Using structural and textual information to capture feature coupling in object-oriented software. Empir Software Eng (2011).
6. Cartwright M, Shepperd M (2000) An empirical investigation of an object-oriented software system. IEEE TransSoftwEng 26(8):786–796
7. Arisholm, E., Briand, L.C., Føyen, A. (2004) Dynamic coupling measurement for object-oriented software. IEEE TransSoftwEng 30(8):491–506.
8. Lindvall M (1998) Are large C++ classes change-prone? An empirical investigation. Software Pract Ex 28 (15):1551–1558
9. Brito e Abreu F., Melo W. Evaluating the impact of Object-Oriented Design on Software Quality”. Proceedings of 3rd International Software Metrics Symp., Berlin, 1996.
10. Chidamber, S. R., Kemerer, C. F. A metric suite for object oriented design. IEEE Transactions on Software Engineering, pp. 467–493, 1994
11. Betetta, J., Castro Díaz, M., Flores, C., Palavecino, R. Evaluación de las característica y comparación de los Sistemas de Gestión de Contenidos. CACIC 2010. http://sedici.unlp.edu.ar/handle/10915/19284
12. Content Management System distribution http://www.empiremedia.com/content-management-system-distributions
13. What is PHP_Depend? http://pdepend.org/documentation/what-is-php-depend.html
14. Overview Pyramid. http://pdepend.org/documentation/handbook/reports/overview-pyramid.html
15. Lanza, M.; Marinescu, R. Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems. Object-Oriented Metrics in Practice. Springer-Verlag Berlin Heidelberg; ISBN 978-3-540-24429-5. 2006.
16. Abstraction Instability Chart. http://pdepend.org/documentation/handbook/reports/abstraction-instability-chart.html
17. Martin, R.C. OO Design Quality Metrics - An Analysis of Dependencies. http://www.objectmentor.com/resources/articles/oodmetrc.pdf. 1994.
18. Joomla Spanich: http://www.joomlaspanish.org/
19. Acosta, J., Greiner, C., Dapozo, G., Estayno, M. Medición de atributos POO en frameworks de desarrollo PHP. CACIC 2012. http://sedici.unlp.edu.ar/handle/10915/23734
20. Sommerville, I. Ingeniería del Software. 7ma. Ed. Pearson Educación (2005).