Ingeniería Industrial/ISSN 1815-5936/Vol. XXXVII/No. 3/septiembre-diciembre/2016/p. 278-285

ARTÍCULO ORIGINAL
GESTIÓN DEL CONOCIMIENTO

Pruebas de rendimiento a componentes de software utilizando programación orientada a aspectos

Performance tests to software components using aspect-oriented programming

Sandra Verona-Marcos[I], Yasiel Pérez-Díaz[II], Lisbán Torres-Pérez[III], Martha Dunia Delgado-Dapena[I], Cornelio Yáñez-Márquez[IV]

[I] Instituto Superior Politécnico José Antonio Echeverría, Cujae. Facultad de Ingeniería Informática. La Habana, Cuba
E-mail: sverona@ceis.cujae.edu.cu, marta@ceis.cujae.edu.cu
[II] Empresa de Servicios Petroleros. La Habana, Cuba
E-mail: yasiel.perez@geoserv.cupet.cu
[III] Complejo de Investigaciones Tecnológicas Integradas (CITI). La Habana, Cuba
E-mail: ltorresp@udio.cujae.edu.cu
[IV] Centro de Investigación en Computación del Instituto Politécnico Nacional, México, D. F. México
E-mail: coryanez@gmail.com

Recibido: 2 de febrero de 2016
Aprobado: 26 de mayo de 2016

RESUMEN
A pesar de la existencia de modelos y estándares internacionales que permiten evaluar los productos de software desarrollados, continúa siendo un problema para muchas organizaciones que desarrollan software. Las pruebas de software poseen gran importancia dentro del proceso de desarrollo del producto, debido al tiempo y esfuerzo que conlleva realizarlas. Para contribuir a disminuir el tiempo y esfuerzo que se utiliza para la realización de las pruebas se cuenta con herramientas que automatizan partedel proceso de evaluación. Sin embargo, la mayoría de las herramientas están diseñadas para evaluar sistemas de software tradicionales, obviando componentes más pequeños.Este trabajo documenta la creación de un *plugin* para realizar pruebas de rendimiento a componentes de software; utilizando el paradigma de programación orientada a aspectos en *Java*. Este *plugin* se considera relevante dada la necesidad de automatizar las pruebas de rendimiento, que en ocasiones pierden protagonismo ante otros tipos de pruebas como las funcionales.

Palabras clave: calidad de software, modelos de calidad, componentes de software, pruebas de rendimiento, programación orientada a aspectos.

ABSTRACT
In models and standard international existence spite that allow to evaluate the quality that it possess the developed software products, this it constitutes an aspect that continues being a problem for many organizations that develop software.
The software tests possess great importance inside the product development process, due to the time and effort that it bears to carry out them. To contribute the time and effort that it is used for the tests realization to diminish it is had tools that automate in some measure the evaluation process. However, a big number of the tools are de-signed to evaluate traditional software systems, obviating those smaller components that conform these systems.
This work is centered in the creation of a plugin to carry out performance tests to software components using aspect-oriented programming in Java, given the necessity to automate this

system test type that it lose protagonism before other tests types in many occasions like they could be the functional tests.

Key words: software quality, quality models, software components, performance tests, aspect-oriented programming.

I. INTRODUCCIÓN

El aumento de las tecnologías y el desarrollo de las telecomunicaciones han traído consigo que el mercado del software se encuentre actualmente en un gran momento. Teniendo en cuenta este auge, se hace imprescindible que las empresas que desarrollan *software* prioricen la calidad, ya que paralelo a la gran demanda de software, se encuentran los clientes que exigen que este posea cada vez mayor calidad. Los resultados de una empresa en la actualidad son medidos con mayor frecuencia en términos de calidad y satisfacción del cliente. Sin embargo, la compañía *Standish Group* en su último reporte CHAOS en el 2014, emitió la alarmante cifra de que solo el 16,2% de los proyectos analizados en grandes y medianas empresas desarrolladoras de *software* resultaron exitosos [1]. Entendiéndose como exitosos, aquellos proyectos que son entregados en tiempo, no exceden el presupuesto y que cumplen con los requisitos establecidos por el cliente, entre otros aspectos.

Para evaluar la calidad que poseen los productos de software existen modelos y estándares entre los que se destacan: el modelo de McCall[2], el modelo de Boehm [3], el modelo propuesto por la norma internacional ISO 9126 adoptado como norma cubana NC-ISO/IEC 9126:2005 [4] y más recientemente las familias de normas ISO/IEC 25 000 [5].

La prueba es una actividad necesaria que permite determinar la calidad que poseen los productos desarrollados. Las pruebas de *software* juegan un papel importante en el control y aseguramiento de la calidad, impactando directamente en la satisfacción del cliente. Pressman en [6]expresó acerca de las pruebas de software que: "...son un elemento crítico para la garantía de la calidad y representan una revisión final de las especificaciones, del diseño y de la codificación".

Incluso las nuevas tecnologías utilizadas para la creación de sistemas cumplen con un proceso rigu-roso para comprobar la calidad de sus productos[7][8]. Una de estas tecnologías emergentes es el desarrollo de software basado en componentes (DSBC, por sus siglas)[9]. El DSBC no es más que el proceso de desarrollo que utiliza la integración de componentes para crear productos de software. Entendiéndose por componente de software según [10] … como "una unidad de composición de aplicaciones software, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio".

Los componentes de software presentan características propias que los diferencian del sistema de software tradicional, entre ellas se pueden encontrar, la ausencia de interfaz gráfica de usuario, la invariabilidad de sus servicios y una identidad bien definida [11].

Existen varias clasificaciones en cuanto a tipos de componentes de software, entre ellas se pueden encontrar: los componentes comerciales (COTS, por sus siglas en inglés), de código abierto (FOSS, por sus siglas en inglés) y servicios web [12].

Los componentes FOSS son productos de software que también se pueden encontrar en el mercado, sin embargo para su adquisición no es necesario que los usuarios paguen licencias comerciales. Entre los componentes FOSS se encuentran las bibliotecas de *software*. Las bibliotecas son aquellos productos de software que aportan funcionalidades específicas, las cuales permiten resolver problemas puntuales en un determinado sistema [13].

En los componentes de software se destaca la evaluación de la característica de calidad rendimiento, ya que se debe velar que en el proceso de interconexión de componentes no sea integrado un componente cuyos servicios presenten un rendimiento que pueda afectar la estabilidad del sistema. Una estrategia a seguir para evaluar el rendimiento en los componentes de software son las pruebas de rendimiento [11].

Existen varias herramientas para realizar pruebas de rendimiento como son: *JMeter*, *OpenSTA* y *Selenium*, entre otras. Pero, la mayoría están diseñadas para entornos web o requieren que los productos a evaluar posean interfaces gráfica de usuario, lo que constituye un inconveniente para los componentes de software que no las presenten, como es el caso de las bibliotecas. Las características de rendimiento que se proponen evaluar estas herramientas no se corresponden con las características de calidad propuestas para evaluar el rendimiento en componentes de software de manera general.

Para el desarrollo del presente trabajo los autores se centran en las bibliotecas de software desarrolladas en *Java* como ejemplo de componentes de software.
Es por ello que el objetivo del artículo lo constituye el desarrollo de un *plugin* para realizar pruebas de rendimiento a bibliotecas de software desarrolladas en *Java* con el apoyo del paradigma de la programación orientada a aspectos.

II. MÉTODOS

A continuación se definen elementos teóricos considerados importantes para presentar la propuesta del *plugin* para realizar pruebas de rendimiento a componentes de software utilizando programación orientada a aspectos.

Estándar ISO 9126

Desde hace varios años los ingenieros de software han buscado un estándar que unifique las características para medir la calidad de un producto de software. En el año 1991 se creó un estándar para la evaluación de la calidad de los productos de software nombrado ISO 9126, luego fue actualizado en el año 2001 [14]. En el 2005 fue adoptada por Cuba como norma cubana (NC, por sus siglas) llamándose a partir de ese momento NC-ISO/IEC 9126:2005 [4].
La norma ISO-9126 está estructurada en cuatro partes las cuales se enumeran a continuación [14]:
a. Modelo de calidad.
b. Medidas externas.
c. Medidas internas.
d. Medidas de la calidad en uso.
En la primera parte de la norma llamada Modelo de calidad se especifican seis características de calidad que pueden ser de tipo interna y externa. Estas características se dividen en subcaracterísticas que son el resultado de atributos internos del software.
Las medidas internas miden el software en sí mismo. Las externas miden el comportamiento del sistema para un contexto basado en los detalles de hardware. Las medidas en uso se enfocan en los efectos del uso del software, en un contexto de uso específico [14][15].

Componentes de Software

Existen varias definiciones acerca del término componente de software, el Instituto de Ingeniería del Software (SEI, por sus siglas en inglés) lo define como: “una implementación opaca de funcionalidad, sujeta a composición por terceros y que cumple con un modelo” [16].
Los componentes de software presentan algunas características que los hacen ser representativos, entre ellas se pueden encontrar [17]:
Identificables: deben tener una identificación que permita acceder fácilmente a sus servicios.
Auto contenido: un componente no debe requerir de la utilización de otros para realizar la función para la cual fue diseñado.
Reemplazables: se puede remplazar por nuevas versiones u otro componente que lo remplace y mejore.
Con acceso solamente a través de su interfaz: debe asegurar que estas no cambiaran a lo largo de su implementación.
Sus servicios no varían: las funcionalidades ofrecidas en su interfaz no deben variar, pero su implementación sí.
Bien Documentados: un componente debe estar correctamente documentado para facilitar su búsqueda si se quiere actualizar o integrar con otros.
Ser genéricos: sus servicios deben servir para varias aplicaciones.
Reutilizados dinámicamente: pueden ser cargados en tiempo de ejecución en una aplicación.
Independiente de la plataforma: deben funcionar independientemente del hardware, software o sistema operativo.
Para el enlace de los componentes dentro de un sistema se utilizan modelos de componentes. Un modelo de componentes define la forma de sus interfaces y el mecanismo para interconectarlos.

Modelo de calidad para evaluar componentes de software

De manera general para la evaluación de la calidad de los componentes de software no se cuenta con un modelo de calidad que contemple las particularidades de los componentes de *software* descritos anteriormente.
Existen varios trabajos que abordan esta problemática pero aún no se ha llegado a un consenso refe-rente a este tema. Algunos autores han desarrollado propuestas en este sentido. Tal es el

caso de la propuesta presentada por [11], basada en el estándar ISO 9126. La propuesta adapta varias de las características de calidad del estándar a un modelo que responde a las características propias de los componentes COTS.
A continuación se muestra en la tabla 1 las características y subcaracterísticas de calidad propuestos en el modelo presentado por [11] para tiempo de ejecución.

Tabla 1. Modelo para evaluar calidad en componentes de Software en tiempo de ejecución

Característica	Subcaracterística
Rendimiento	Comportamiento Temporal Utilización de Recursos
Funcionalidad	Corrección Seguridad
Fiabilidad	Recuperabilidad

Como se puede observar en la tabla 1 este modelo define un conjunto de características que a su vez se dividen en subcaracterísticas. Las subcaracterísticas se dividen en dos grupos. Un grupo en el que están las subcaracterísticas que tienen sentido ser evaluadas en tiempo de ejecución y el otro grupo para aquellas que se obtienen durante el ciclo de vida. La eficiencia o rendimiento, como también se le conoce en la literatura, las subcaracterísticas de calidad propuestas por los autores se obtienen durante el tiempo de ejecución.
En la tabla 2se muestran cada una de las subcaracterísticas que pueden ser obtenidas en tiempo de ejecución y los atributos propuestos por los autores para evaluar de calidad. Se puede observar en la tabla 2 que para la característica eficiencia una de las subcaracterísticas para evaluar la calidad es el comportamiento temporal. Para evaluar esta subcaracterística se proponen los atributos tiempo de respuesta y capacidad de emisión. El tiempo de respuesta constituye una medida para evaluar los tiempos de respuesta de un producto de software cuando se le realiza una petición. La capacidad de emisión permite conocer el flujo de datos en un tiempo determinado mientras se realiza una petición al producto. La otra subcaracterística está relacionada con la utilización de recursos de hardware. Los atributos de calidad propuestos por los autores para evaluar esta subcaracterística son los requisitos de memoria y la capacidad o utilización de espacio en disco. El atributo requisito de memoria permite conocer cómo se está utilizando la memoria dinámica del producto mientras se ejecutan sus funcionalidades. La capacidad en disco por su parte evalúa los requisitos de almacenamiento está-tico para el producto de software.

Tabla 2. Atributos de calidad para evaluar rendimiento en componentes de software

Subcaracterística	Atributo
Comportamiento Temporal	Tiempo de respuesta Capacidad de emisión
Utilización de Recursos	Requisitos de memoria Utilización de disco

Pruebas de rendimiento a componentes de software

Existen diversos alcances para las pruebas de software que permiten agruparlas por niveles. Un nivel de prueba permite describir el alcance de la prueba de software que se desea realizar, se pueden identificar principalmente dos niveles para las pruebas.
El primer nivel, llamado nivel bajo son consideradas todas las pruebas que se realizan a los componentes individuales [18]. Dentro de este nivel se encuentran las pruebas de unidad que no son más que las pruebas que miden el funcionamiento de un módulo o componente y se centran en la lógica del procesamiento interno [18]. En el mismo nivel puede encontrarse las pruebas de integración, cuyo objetivo es tomar componentes a los que se les aplicó una prueba de unidad y construir una estructura de programa que determine el diseño.
El otro nivel, llamado nivel alto es dedicado a pruebas orientadas al producto completo. Dentro de este nivel se encuentran las pruebas de sistema, que abarcan una serie de pruebas diferentes cuyo propósito principal es ejercitar profundamente el sistema de cómputo [6].
Entre las pruebas de sistema se encuentran las pruebas de rendimiento. Este tipo de prueba se

realiza, desde una perspectiva, para determinar lo rápido que realiza una tarea un sistema en condiciones particulares de trabajo. También puede ser utilizada para validar y verificar algunos atributos de calidad, tales como la escalabilidad, fiabilidad y el uso de los recursos [6].
En un producto de software el rendimiento se mide como la capacidad del sistema de utilizar los recursos de hardware de forma eficiente, las pruebas de rendimiento tienen como objetivo estresar el sistema realizando demandas fuera de los límites para los que fue diseñado el *software*.
Los componentes de software se encargan fundamentalmente de proveer servicios a los sistemas. Las pruebas de rendimiento, buscan monitorear que los servicios mencionados anteriormente, se brinden minimizando en lo posible el tiempo y el porciento de utilización de recursos de hardware. Estas pruebas reportan las medidas necesarias para evaluar la calidad en este aspecto ofrecido por el componente [19].

Pruebas de rendimiento como conceptos transversales

Las pruebas de rendimiento en cualquier producto de software presentan un comportamiento transversal a la ejecución de las funcionalidades. Cualquier herramienta que realice este tipo de pruebas aunque no utilice el paradigma orientado a aspectos actúa de manera transversal al sistema. El detalle está en que la herramienta para poder evaluar los elementos de rendimiento necesita que se ejecuten las funcionalidades del sistema. Pero, en la ejecución de una funcionalidad dentro de un sistema intervienen muchas otras funcionalidades que se activan consecutivamente detrás de la llamada a un método determinado, por tanto la herramienta estaría evaluando muchas funcionalidades que se encuentran en diferentes partes del sistema. La figura 1 demuestra esta afirmación.

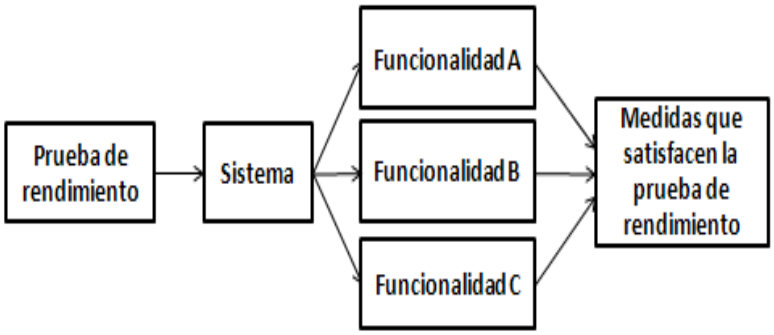

Fig. 1. Evaluación del rendimiento en un sistema como asunto transversal a la ejecución de las funcionalidades

En una biblioteca de software sucede muy parecido, la diferencia fundamental radica en que la biblioteca no está diseñada para ejecutar sus propios servicios. Sin embargo a través de un mecanismo externo que permita implementar y ejecutar automáticamente las funcionalidades de la biblioteca se puede resolver esta diferencia como se observa en la figura 2.

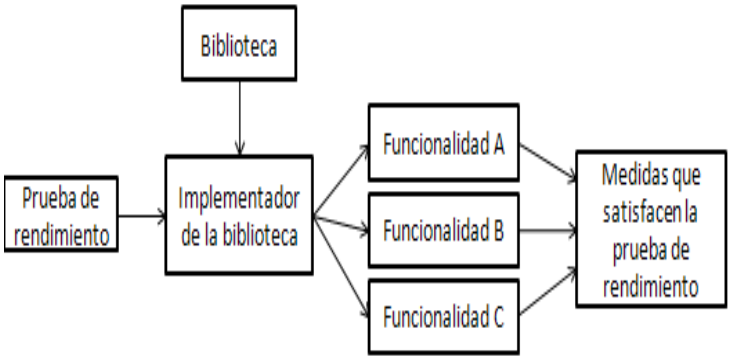

Fig. 2. Evaluación del rendimiento en una biblioteca como asunto transversal a la ejecución de las funcionalidades

Basado en este esquema y tomando como referencia el proceso básico propuesto por [20] para la implementación de los conceptos transversales, se realizó una adaptación al contexto de pruebas de rendimiento. De esta forma se obtiene el modelo de proceso básico para pruebas de rendimiento utilizando un lenguaje AOP, como se muestra en la figura 3.

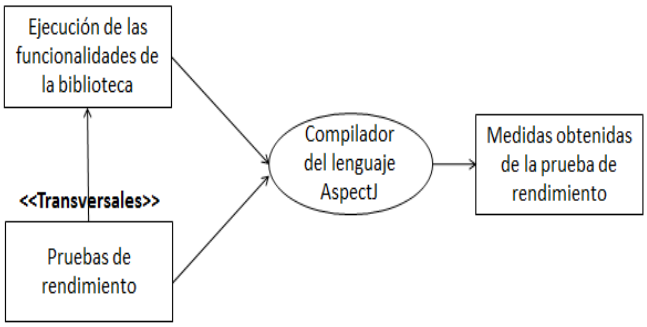

Fig. 3. Proceso de pruebas de rendimiento utilizando el enfoque AOP

Implementación de los Aspectos

Es necesario explicar cómo se implementaron los aspectos contenidos en el módulo *aspects*, los cuales permitieron interceptar la ejecución de estas funcionalidades utilizando el lenguaje *AspectJ*.

Se identificó e implementó el aspecto *PerformanceTesterAspect*. Dentro del aspecto se implementó el punto de corte *testerExecution*, el cual intercepta la ejecución de todas las funcionalidades públicas de la clase objetivo *TestCase* (la clase de prueba que se crea con el *plugin*).

Con el punto de corte se garantiza tener puntos de acceso a los instantes de ejecución de cada funcionalidad por separado. A través de un aviso de tipo *before*se tiene acceso al instante inicial de ejecución de cada método, permitiendo activar las variables de rendimiento implementadas. Luego con un aviso de tipo *after* se determina el instante en que cada funcionalidad ha terminado su ejecución.

Con estos puntos de acceso a los extremos inicio y final, dentro de la ejecución de los métodos se-leccionados, se garantiza la evaluación de las medidas de rendimiento tiempo de respuesta, capacidad de emisión y la utilización de memoria a través del uso de clases, métodos y variables. Por ejemplo para el caso del tiempo de respuesta se utilizaron los métodos *start*() y *stop*(), los cuales permiten iniciar y detener una variable de tipo tiempo. La llamada al método *start*() se realiza dentro del aviso *before*, garantizando que antes de la ejecución de una funcionalidad determinada, se inicialice la variable de tiempo. Por su parte la ubicación del método *stop*() dentro del aviso *after* permite que al concluir la ejecución esa variable contenga el tiempo transcurrido desde el inicio de la ejecución hasta el final, donde se supone que la funcionalidad proporciona una respuesta. De esta manera al finalizar la ejecución de todas las funcionalidades de la clase se obtienen los tiempos de respuesta por cada una de ellas.

Para el caso de la memoria sucede parecido, con el uso de interfaces nativas del lenguaje se puede acceder al estado de la memoria dentro de la máquina virtual de *Java*. Por tanto utilizando el aviso *after* se puede controlar el estado de la memoria luego de cada ejecución, permitiendo conocer el consumo de este recurso de manera separada.

III. RESULTADOS

A continuación se exponen los resultados de la solución desarrollada utilizando un caso de estudio como mecanismo de validación de la propuesta implementada.

Se utilizó la biblioteca *ImageJ* desarrollada en *Java*, la cual entre sus funcionalidades fundamentales contiene el tratamiento de imágenes digitales. Generalmente, los productos de software que gestionan el tratamiento, procesamiento y manipulación de imágenes exigen un consumo excesivo de los recursos de hardware como es el caso de la memoria dinámica. Esta característica presente en la biblioteca *ImageJ* fue determinante en su selección para la validación de la propuesta de solución.

Para realizar la prueba de rendimiento a la biblioteca *ImageJ* se diseñaron cuatro escenarios de prueba. En cada escenario se utilizan los mismos valores de entrada variando el formato *(.JPG, .GIF, .PNG y .BMP*) de las imágenes que se utilizan en la muestra. Se realizaron tres iteraciones

por cada escenario de prueba y se promediaron los resultados de cada escenario para dar un resultado final.
Luego de realizar la prueba para todos los escenarios creados, sujeto a los parámetros de entrada establecidos, se obtuvieron resultados cuyos valores de las medidas evaluadas pertenecen a los promedios de las evaluaciones realizadas para las tres iteraciones de prueba. En la tabla 3 se muestran los resultados obtenidos en uno de los escenarios de prueba evaluados.

Tabla 3. Resultados de un escenario de prueba

Nombre del servicio	Tiempo de ejecución (segundos)	Cantidad de memoria utilizada (mb)	Capacidad de emisión (mb/s)	Capacidad en disco (kb)
filter All Images	1,584	9,879	16,276	6,256
resizing	0,204	9,814	2,001	6,256
Cutter Image	0,126	11,339	1,396	6,256
convert	0,663	5,384	3,700	6,256
combining	0,249	13,073	3,218	6,256

Con los resultados obtenidos los probadores pueden arribar a conclusiones como:
- El método *filterAllImages* necesitó mayor tiempo de ejecución de forma general y es el que mayor flujo de trasmisión de datos posee.
- Las operaciones *combining* y *cutterImage* son las de mayor consumo de memoria.
- El formato *.PNG* exige mayores requisitos de memoria, por el contrario el que exige menores recursos es el formato *.JPG*.

De manera general se puede concluir que el tratamiento de imágenes exige altos requisitos de memoria para un funcionamiento adecuado.

IV. DISCUSIÓN

El caso de estudio desarrollado como escenario de la validación del *plugin* implementado para el *IDE Eclipse* permite que desde el propio entorno el desarrollador tenga acceso a definir y testear el rendimiento de un componente de software. Además es destacable el empleo para ello de la programación orientada a aspectos como paradigma que facilita la automatización de la tarea. Esta facilidad redunda en: una disminución del esfuerzo a la realización de las pruebas de rendimiento, mayor calidad de los componentes al final de su desarrollo para que sean utilizados por múltiples soluciones que requieran de la funcionalidad específica que estos provean. La comprobación la validez de esta propuesta con una biblioteca de tratamiento de imágenes da una medida que es factible de aplicarse con elementos que requieran elevados consumos de hardware y tiempo. Es válido destacar que para obtener un resultado fiable en la ejecución no automatizada de pruebas de rendimiento se hace necesario poseer conocimientos específicos en esta área.

V. CONCLUSIONES

Una vez terminado el trabajo se puede concluir que:
1. La evaluación del rendimiento es muy importante en el desarrollo de software de forma general y particularmente en los componentes de software.
2. De igual manera, se puede alegar que la evaluación de la característica de calidad rendimiento puede ser tratada como un asunto transversal en la ejecución de las funcionalidades de los componentes de software.

3. Con la realización del caso de estudio se pudo evidenciar que con el empleo del paradigma de la programación orientada a aspectos es posible automatizar el proceso de evaluación de la característica de calidad rendimiento en los componentes de software.
4. Se encuentra a disposición de la comunidad científica un componente desarrollado en forma de *plugin* para la evaluación de la característica de calidad rendimiento en componentes de software.

V. REFERENCIAS

1. Standish Group. CHAOS Manifiesto. 2014. [Citado 20 de septiembre de 2015] Disponible en: http://www.standishgroup.com/chaos-news
2. McCall P, Walters G. Factors in Software Quality: US Rome Air Development Center. 1977.
3. Boehm JB, Kaspar J, Lipow M, et al. Characteristics of Software Quality. 1978.
4. Oficina Nacional de Normalización. Ingeniería de software, calidad del producto. En: Parte 1: Modelo de calidad. 2005.
5. ISO. Systems and software engineering. In: Systems and software Quality Requirements and Evaluation. 2011.
6. Pressman RS. Ingeniería del Software: Un enfoque práctico. México D.F: McGraw-Hill: McGraw Hill; 2010.
7. Flores M. Testing-based process for component substitutability. Software Testing, Verification & Reliability. 2012.
8. Xin X. A Regression Testing Approach of COTS Component Based Software. In: Proceedings of the 2013 Fifth International Conference on Multimedia Information Networking and Security; 2013.
9. Shang H, Jiang L. The Development Process of Component-Based Application Software. In: Proceedings of the 2011 International Conference of Information Technology; 2011.
10. Szyperski C. Component Software: Beyond Object-Oriented Programming. Arizona, United States Addison-Wesley Educational Publishers Inc; 2002.
11. Bertoa AV. Atributos de Calidad para Componentes COTS. presented at the IDEAS. La Habana, Cuba; 2002.
12. Papazoglou M. Web services: principles and technology. Edinburgh, UK2008.
13. Panunzio TV. A component-based process with separation of concerns for the development of embedded real-time software systems. Journal of Systems and Software. 2014;96:105-21.
14. ISO. Software engineering - Product quality -Part 1: Quality Model. ISO-9126-1. Geneva. Switzerland: 2001.
15. Scalone F. Estudio comparativo de los modelos y estándares de calidad del software [Tesis de máster]. Buenos Aires, Argentina: Universidad de Buenos Aires; 2006.
16. SEI. Community Software Architecture Definitions. 2015.
17. Rojas JC. Introducción y principios básicos del desarrollo de software basado en componentes. Colombia2004. ISBN. DOI [Citado
18. Ochoa JC. Metodología para testing de software basado en componentes [Tesis de grado] 2010.
19. Tian J. Software quality engineering: testing, quality assurance, and quantifiable improvement. En: Hoboken. New Jersey: John Wiley & Sons; 2005.
20. Hernández LR. Un modelo para la implementación de la seguridad de una aplicación web con el uso de la programación orientada a aspectos. [Tesis de maestría]. La Habana, Cuba: Facultad de Ingeniería Informática, Instituto Superior Politécnico “José Antonio Echeverría”; 2002.