Saber, Universidad de Oriente, Venezuela.Vol. 22 Nº 2: 185-192. (2010)

UTILIZACIÓN DE MÉTRICAS DE SOFTWARE PARA APOYAR LA SELECCIÓN DE FRAMEWORKS WEB PARA EL SISTEMA DE GESTIÓN DE DATOS ACADÉMICOS DE LA FACULTAD DE CIENCIAS UCV

USE OF SOFTWARE METRICS TO SUPPORT THE SELECTION OF FRAMEWORKS WEB FOR

Zulma González, Andrés Sanoja, Sergio Rivas

Escuela de Computación, Facultad de Ciencias, Universidad Central de Venezuela
Los Chaguaramos, 1041, Caracas, Venezuela.
E-mail: zulmacarol@gmail.com, andres.sanoja@ciens.ucv.ve, srivas@blues.ciens.ucv.ve

RESUMEN

La motivación de este trabajo fue proveer de criterios de decisión a líderes de proyecto de la Facultad de Ciencias de la UCV para la selección de herramientas tecnológicas Web para el desarrollo de un módulo del Sistema de Control de Estudios (CONEST), proponiéndose su medición utilizando métricas de software. El módulo fue desarrollado en dos versiones, utilizando diferentes *frameworks* de desarrollo Web, manteniendo los mismos requerimientos funcionales y el modelo de datos. La primera versión fue desarrollada utilizando el *framework* Web Ruby on Rails y la segunda, una selección de *frameworks* de desarrollo en Java: Struts, Hibernate, JavaMail y Axis2. Las métricas de software utilizadas fueron: *Líneas de Código Fuente* (LOC), *Líneas de Código Comentadas* (CLOC), *Complejidad Ciclomática* (CC), COCOMO básico y medidas *ad hoc* para medir rendimiento Web y standalone. Los resultados evidenciaron, para el caso de estudio presentado, diferencias entre el framework Ruby on Rails y los *frameworks* en Java relacionados con el tiempo de desarrollo y rendimiento de ejecución.

Palabras clave: *Frameworks* Web, medición, métricas de software, desarrollo Web.

ABSTRACT

The motivation of this work is to provide criteria oriented to the software leaders of the U.C.V Science Faculty for the selection of web technologies for the development of a module for the "Control de Estudios" System (CONEST), proposing to measure them by the use of software metrics. The module was developed in two versions, using different Web *frameworks*, having in common the same functional requirements and data model. The first version was developed using the Web *framework* Ruby-on-Rails and the second, a selection of Java *frameworks*: Struts, Hibernate, JavaMail and Axis2. The software metrics used are: *Lines of Code* (LOC), *Commented Lines of Code* (CLOC), *Ciclomatic Complexity* (CC), Basic COCOMO and *ad hoc* Web and standalone performance metrics. Based on our case study, the results allow to detect differences between the values of metrics in Ruby-on-Rails framework and Java *frameworks*, basicaly related to the development and performance time.

Key words: Web *frameworks*, measure, software metrics, Web development.

INTRODUCCIÓN

En 2006 la División de Control de Estudios de la Facultad de Ciencias de la Universidad Central de Venezuela se planteó el reto de automatizar el proceso de gestión de datos sobre inscripciones, calificaciones y otros procesos de los estudiantes de pregrado, con la finalidad de agilizar el proceso que por varios años se llevó a cabo de una manera semi-automatizada. Disponer de una herramienta Web para llevar a cabo la inscripción permitiría evitar las colas de los estudiantes, reducir los costos de la inscripción minimizando el uso del papel, y asegurar la calidad de los datos del estudiante.

Se planteo el proyecto CONEST (Sistema de Gestión Académica de la Facultad de Ciencias de la Universidad Central de Venezuela) cuyo objetivo fue dar solución a los problemas anteriormente presentados. En esta dependencia en los últimos años, tanto en docencia como en investigación, la influencia de soluciones asociadas al lenguaje de programación Java y la plataforma J2EE ha sido alta, por lo que la primera opción en herramientas de desarrollo fue los frameworks Java (Johnson y Russo 1991). Por otro lado un grupo de investigadores venía trabajando con el lenguaje de programación Ruby y el framework Web Ruby-on-Rails (o simplemente Rails) lo cuál generó una discusión sobre cual en definitiva sería la opción más adecuada para desarrollar el sistema.

En este contexto uno de los factores clave es la selección de la herramientas tecnológica a utilizar. Dado que no se puede asegurar que una misma herramienta pueda dar la mejor solución a todos los problemas, se debe proveer de datos sobre las bondades que ofrecen

Recibido: marzo 2008. Aprobado: febrero 2010.
Versión final: junio 2010

y limitaciones que tienen las herramientas disponibles; siempre hay un riesgo asociado sobre una selección de este tipo. Una manera de minimizar el riesgo es conocer mejor las características de cada herramienta, el contexto donde mejor aplica y las experiencias previas. Una manera de presentar las características de una herramienta es midiéndola y obteniendo valores para dichas características, con base en las cuales los líderes de proyecto pueden tomar decisiones y determinar, en cierto grado, la factibilidad de utilizar una herramienta para un determinado problema y dentro de un determinado contexto.

En el desarrollo Web se han hecho populares los framework Ruby on Rails (Heinemeier 2005) y los frameworks Java: Struts, Hibernate, JavaMail y Axis2. Para presentar las características de éstas tecnología nos apoyamos en métricas de software (Offutt 2002, Ruhe *et al.* 2003, Fewster *et al.* 2001).

Este artículo describe un estudio cuya finalidad fue buscar una forma de proporcionar información de valor agregado a líderes de proyecto de la Facultad de Ciencias en la toma de decisiones de la herramienta a utilizar en sus desarrollos de software en el contexto del Sistema CONEST. Se expone el resultado de la medición, valoración y aproximaciones de métricas de software en un caso de estudio usando ambos tipos de *frameworks*.

MATERIALES Y MÉTODOS

Basándonos en Pressman (2005) y Olsina (2007) nos enfocamos en determinar las diferencias o similitudes que existía entre ambas opciones por lo que seleccionamos un conjunto de métricas que nos permitieron, de una manera cuantitativa, valorar las características de una misma solución de software implementada en ambos tipos de herramientas. Los valores de las métricas debían estar contextualizadas coherentemente para ambas opciones, es decir, su significado debía ser similar tanto en una opción como en la otra; esto fue tomado como criterio para la selección de las métricas a utilizar. Tanto para el lenguaje de programación Java como para lenguaje Ruby se encontraron herramientas de medición que proporcionaban valores válidos, pero no significativos ya que se especializan para cada lenguaje en particular. Necesitábamos herramientas que midieran "con la misma vara" ambas opciones para poder garantizar resultados significativos, tratando de garantizar cierta igualdad de condiciones en la medición.

Se desarrollaron, en iteraciones separadas, las soluciones para cada opción: una utilizando el framework Ruby on Rails y en otra los frameworks Java; destacando que ambas soluciones tenían los mismos requerimientos funcionales y se basaron en el mismo modelo conceptual de datos (Ambler 2003). Se desarrollaron herramientas de medición, siguiendo heurísticas, para medir cualidades internas de las aplicaciones Web, en este caso: eficiencia, confiabilidad, mantenibilidad y rendimiento (Offutt 2002). En la Figura 1 se presenta gráficamente la visión general de la solución propuesta.

Varias son las opiniones que se dicen sobre ambos tipos de frameworks, cada partidario defiende su opción, sin embargo las opiniones confluyen en la agilidad de Ruby on Rails, contra la solidez y eficiencia de Java. Por tanto la selección de métricas a utilizar esta orientada a presentar valores que permitieran afirmar o negar dichas apreciaciones. Como resultado concreto nos propusimos realizar una selección que nos permitiera medir y visualizar los resultados de las misma. A continuación se presenta un resumen de las métricas de software seleccionadas para medir nuestro caso de estudio y las consideraciones particulares para su uso, cálculo y aproximación.

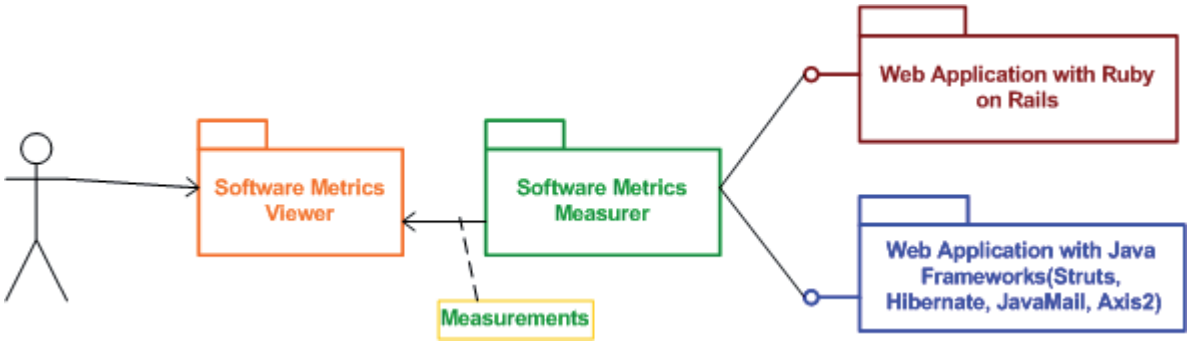

Figura 1: Visión General de la Solución

Métricas Simples de Código Fuente

Las métricas simples de código fuente permiten medir las características internas de los componentes de software. Para el cálculo de estas métricas se dio formato al código, de manera que cada instrucción correspondiera con una línea del archivo del programa. Se aplicaron las siguientes métricas simples de código fuente: Total Líneas de Código (Lines of Code, LOC) se tomaron los valores recomendados para una clase o archivo entre [5, 1000] y Líneas de Comentarios de Código (Comment Lines of Code, CLOC) tomando el porcentaje recomendado para una clase o archivo entre 20% y 40%.

Métrica para medir la Complejidad

Las métricas de complejidad pueden emplearse para predecir información crítica sobre la fiabilidad y mantenimiento de software. Se aplicaron las siguientes métricas para medir la complejidad: Complejidad Ciclomática (Cyclomatic Complexity CC) (McCabe 1976 1994, Rosenberg 1997). Para llevar a cabo el cálculo de esta métrica se aplicó el mismo criterio de formato para el código fuente: una instrucción por línea de archivo. Se recorre de manera secuencial el archivo de código fuente detectando las bifurcaciones y condicionales presentes en el texto del programa. Es importante resaltar que el valor obtenido es una aproximación al número ciclomático.

Métrica Orientada a Objetos

La métrica *WMC* (Métodos Ponderados por Clase o Weighted Methods per Class) mide el esfuerzo que se requiere para implementar o dar mantenimiento a una determinada clase. Esta métrica es definida como la suma de los números ciclomáticos de todos los métodos definidos en una clase: $WMC(C) = \sum V(m), \forall m / m \in M$ y M es el conjunto que contiene los métodos (m) definidos en la clase C (Chidamber 1993).

Modelo Constructivo de Costos (COCOMO)

El Modelo Constructivo de Costos (Constructive Cost Model) (Bohem, 1981). COCOMO es una jerarquía de modelos de estimación de software que incluye submodelos básico, intermedio y detallado. Para el cálculo de la métrica se utilizó la ecuación de COCOMO en modo básico. Los valores resultantes fueron calculados evaluando la ecuación en modo básico de COCOMO con los valores obtenidos en las métricas LOC.

Métricas de Rendimiento Web

Mide el tiempo de respuesta de una página Web. Para medir el rendimiento Web se realizaron peticiones a páginas Web tomando la diferencia entre los tiempos final e inicial de una petición Web. En los experimentos realizados en (González 2007) se calcularon los tiempos de varias peticiones Web y se promediaron sus valores.

Rendimiento Standalone

El rendimiento *standalone* se midió tomando la diferencia entre los tiempos final e inicial de la ejecución de un procedimiento, siguiendo un enfoque caja negra. En los experimentos realizados se promediaron resultados de varias ejecuciones. Para el cálculo de las métricas de rendimiento se desarrolló una aplicación Proxy la cual genera peticiones y marca los tiempos de respuesta. Las aplicaciones Web o procedimientos *standalone* son vistos como caja negra.

Caso de Estudio

La investigación se basó en dos aplicaciones de software desarrollados en las tecnologías Web a medir, estas aplicaciones Web proporcionan soluciones a un mismo problema y utilizan el mismo repositorio de datos. Las aplicaciones resuelven requerimientos del Sistema de Gestión Académica de la División de Control de estudios de la Facultad de Ciencias de la Universidad Central de Venezuela (CONEST).

La aplicación desarrollada es una aplicación Web estándar la cual posee una Base de Datos relacional medianamente compleja (60+tablas) a la cual se realizaron un número significativo de consultas, además requería la implementación de servicios Web, gestión de algunos procesos complejos, emisión de reportes en formato PDF y HTML, implementación de envío de correos, entre otras funcionalidades generales.

La aplicación Web Rails fue desarrollada con el lenguaje de programación Ruby versión 1.8.5 y con el *framework* Ruby on Rails versión 1.2.3. Mientras que la aplicación Web Java fue desarrollada con Java versión 1.5 y con los frameworks: Struts versión 1.3.8, Hibernate versión 3.2.2, JavaMail versión 1.4 y Axis2 versión 1.1.1.

Cada aplicación está dividida en dos componentes de software: Un componente encargado de la implementación de requerimientos del Módulo Administrativo y de Servicio al estudiante de CONEST: CONEST RAILS,

desarrollada con Rails, y CONEST JAVA, desarrollada con Java, y un componente que permite implementar la autenticación de usuario por medio de un servicio Web: CONEST USUARIO, desarrollada con Rails, y CONEST USUARIO JAVA, desarrollada con Java. En la Figura 2 y 3 se observan la estructura de los caso de estudio Rails y Java, respectivamente.

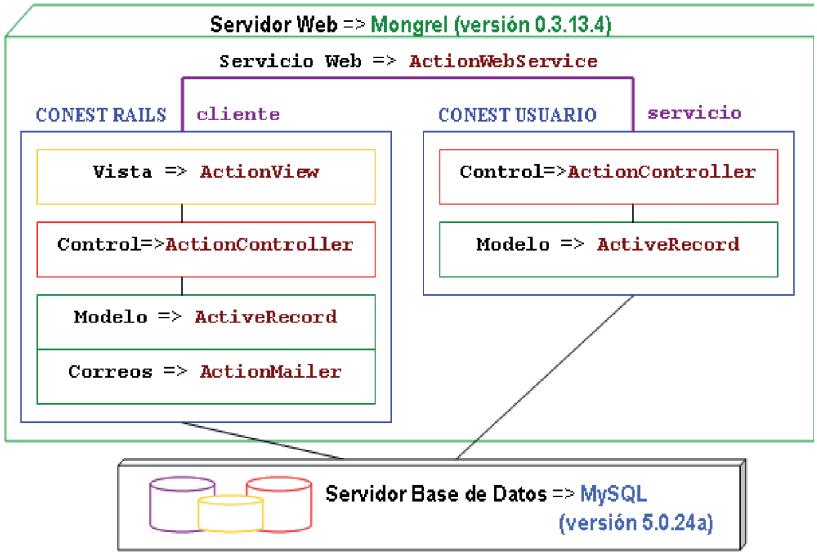

Figura 2: Caso de estudio Rails.

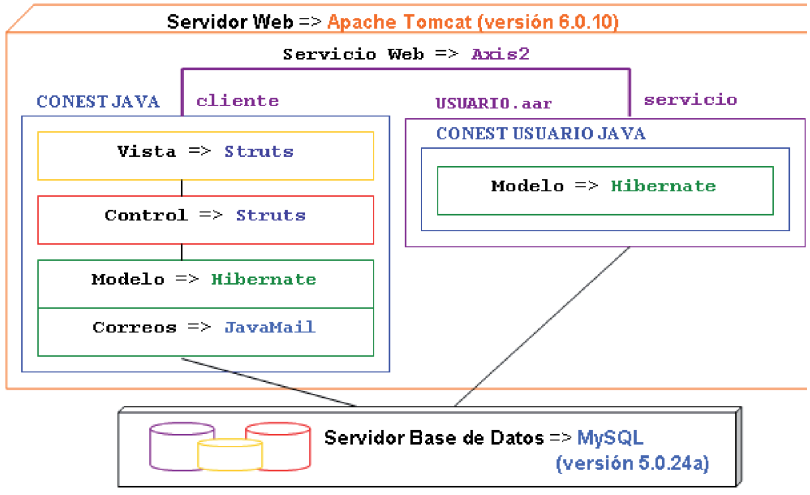

Figura 3: Caso de estudio Java.

RESULTADOS

El valor de LOC de la aplicación Web Rails (CONEST RAILS) es igual a 1280 líneas de código fuente y comentarios, y el valor de LOC la aplicación Web Java (CONEST JAVA) es igual a 3949 líneas. En estos resultados se observa una relación de 3 a 1 en Java y Rails, respectivamente. El *framework* Ruby on Rails posee varias características que permitieron que CONEST RAILS tuviera el menor valor de LOC como por ejemplo la utilización de un conjunto de convenciones, múltiples funciones de alto nivel (como los métodos find de ActiveRecord), *helpers,* entre otras. En la Figura 4 se observa un gráfico de barras para representar los resultados obtenidos de las mediciones de la métrica LOC sobre los componentes del caso de estudio.

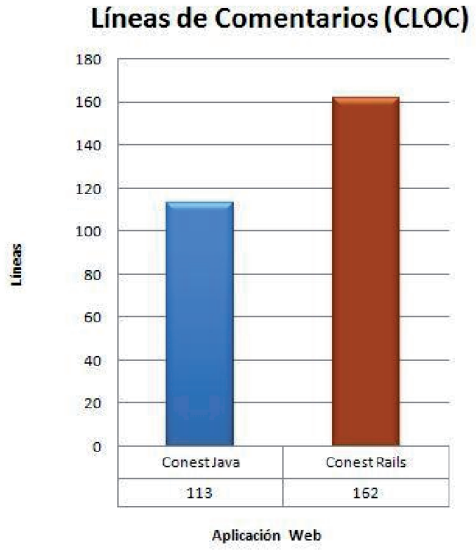

Figura 4: Gráfico de resultados de LOC.

El valor de CLOC de la aplicación Web Rails (CONEST RAILS) es igual a 162 líneas de comentarios, y el valor de CLOC la aplicación Web Java (CONEST JAVA) es igual a 113 líneas. A pesar que en los componentes del caso de estudio se documentaron de manera similar, la aplicación Web Rails presentó un valor más elevado de comentarios, esto es debido a que casi toda la estructura básica de la aplicación Web Rails se obtiene mediante generadores los cuales crean archivos adicionales, incluyendo comentarios. Como por ejemplo el archivo **environment.rb**, el cual al ser generado, incluyen varias líneas de comentarios. En la Figura 5 se observa un gráfico de barras para representar los resultados obtenidos de las mediciones de la métrica CLOC.

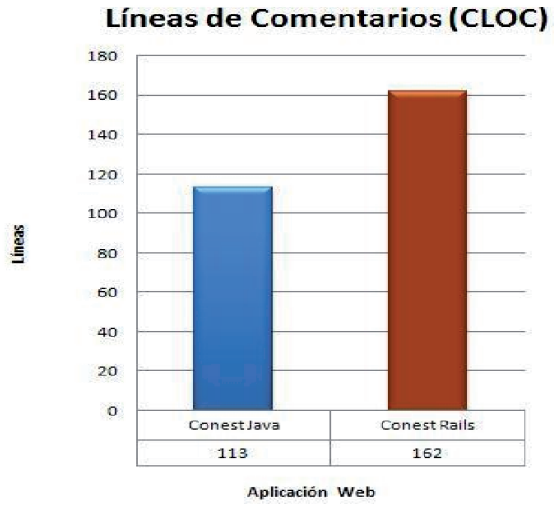

Figura 5: Gráfico de resultados de CLOC.

La complejidad ciclomática de la aplicación Rails (CONEST RAILS) es igual a 213, y el número ciclomático de la aplicación Java (CONEST JAVA) es igual a 669; al igual que el LOC se observa un factor de 3 a 1 en Java y Rails, respectivamente. Estos resultados indican que la aplicación Java es más compleja que la aplicación Rails, en la Figura 6 se observa un gráfico de barras para representar los resultados obtenidos de las mediciones de la métrica CC.

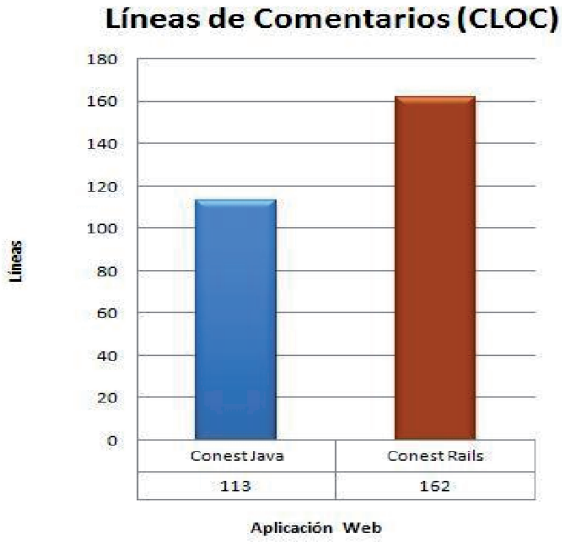

Figura 6: Gráfico de resultados de CC.

Por medio de COCOMO se pudieron estimar algunas características asociadas con el proceso de desarrollo de las aplicaciones Web que componen el caso de estudio. COCOMO permitió determinar el esfuerzo, tiempo y número de desarrolladores que se requieren para implementar cada aplicación considerando la naturaleza o el tipo de proyecto. Las aplicaciones Web del caso de estudio son proyectos de tipo medio y con base en está clasificación, en el Tabla 1, se presentan los resultados obtenidos.

Tabla 1: Resultados de COCOMO.

Aplicación Web	Esfuerzo Personal (personas-mes)	Tiempo de Desarrollo (meses)	Número de Personas
Java	15.9383	3.6319	4.3884
Rails	4.5129	2.3353	1.9325

Los resultados de las métricas anteriores sugieren que el desarrollo en Ruby on Rails requiere menos esfuerzo, que el desarrollo Web en Java. Una de las

características de Ruby on Rails que permitió esto es el conjunto de componentes integrados (ActionController, ActiveRecord, ActionMailer y ActionWebService) los cuales proveen facilidades de desarrollo similares a las encontradas en los *frameworks* Struts, Hibernate, JavaMail y Axis2, esta característica permite ahorrar tiempo en la implementación ya que no es necesario realizar un proceso de integración de varias tecnologías. En el desarrollo en Java se apreció la necesidad de una cantidad de tiempo importante en los archivos de configuración y recolección de bibliotecas.

Para medir el Rendimiento Web de las aplicaciones del caso de estudio se seleccionaron las páginas Web más representativas en complejidad y diversidad de procesamientos. Las páginas Web seleccionadas realizaban las siguientes tareas: 1) realizar un número significativo de consultas a la base de datos, 2) enviar correos electrónicos y 3) interactuar con el servicio Web implementado en cada caso.

Una aplicación *standalone* denominado Medidor realizó 10 lotes de 200 peticiones cada uno de la página Web index de cada aplicación, para evaluar la variación de los valores obtenidos en el tiempo. En la Figura 7 se observan gráficamente los resultados obtenidos en este experimento.

Los resultados del Rendimiento Web observados en la Figura 7, sugieren que el rendimiento de la aplicación Web Java mejora a medida que las peticiones se realizan sobre recursos solicitados previamente. Mientras que en Rails los resultados del rendimiento Web oscilaron cerca de un mismo valor. Uno de los factores por los cuales CONEST JAVA presenta mejor rendimiento Web es por la utilización de una caché de primer nivel presente en el *framework* Hibernate, caso contrario a CONEST RAILS. Se observó que ambas herramientas presentaron una diferencia en tiempo de ejecución y es que en el caso de los frameworks se vio impactada negativamente la misma por falta de espacio en el *heap*. Esta problemática fue simplemente observada y como solución se iteró sobre el código fuente para refactorizar los elementos que causaban dicho problema.

Las mediciones realizadas a la aplicación Web Rails sugieren que posee características que apoyan el desarrollo, por ejemplo este *framework* posee un conjunto de componentes integrados que facilitan el desarrollo Web lo que disminuye el proceso de integración de varias tecnologías, además Rails utiliza un conjunto de convenciones y utilidades de programación que permiten disminuir el valor de la métrica LOC y los tiempos de configuración.

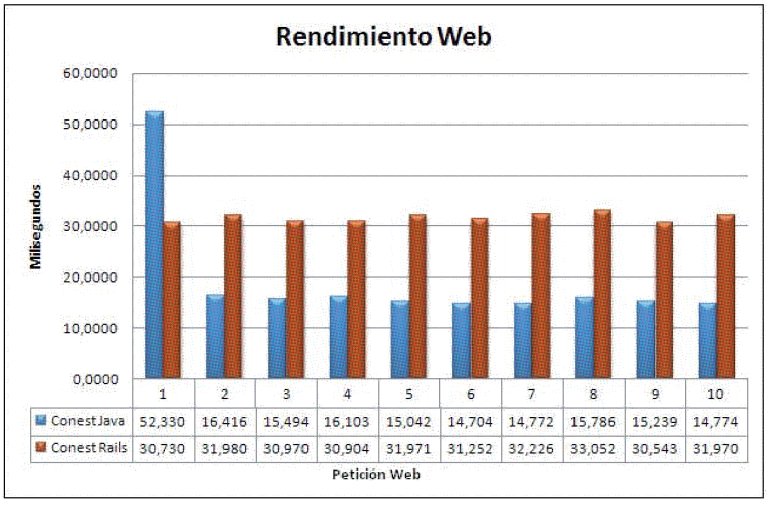

Figura 7: Gráfico de resultados de rendimiento Web.

DISCUSIÓN

Evaluando de forma general los resultados de ambas opciones de desarrollo Web, se evidenció en el caso de estudio que ambas soluciones constituyen una opción viable y la decisión se centra en sí se desea explotar la facilidad de desarrollo o buen rendimiento. Aún cuando el desempeño de Rails es suficiente para trabajar aceptablemente, la especialización de los *frameworks* en Java hace que las aplicaciones funcionen más eficientemente.

Tomando en cuenta los resultados obtenidos de la experimentación presentada en este trabajo, los líderes

de proyecto de la División de Control de Estudios de la Facultad de Ciencias de la UCV, establecieron que para la aplicación que se quería desarrollar no era determinante el tiempo de respuesta, pero si la posibilidad de construir soluciones en un relativo corto tiempo y que pudiesen ser modificadas y mantenidas por diversos grupos independientes de desarrollo; dicho departamento se basa en los resultados realizados por los estudiantes que realizan sus Trabajos Especiales de Grado y pasantías.

CONCLUSIONES

Con el desarrollo del presente trabajo se pudo dar respuesta a la inquietud presentada en la División de Control de Estudio de la Facultad de Ciencias de la UCV, sobre la selección del framework a utilizar para el Proyecto CONEST basándose en los resultados de la experimentación presentada. Adicionalmente permitió conocer detalles y característica importantes de ambas configuraciones, lo cual queda como referencia para futuros trabajos.

Los resultados permitieron guiar esta decisión de selección en particular. Se llevaron a cabo, los cálculos y aproximaciones de métricas aquí presentados se realizaron mediante heurísticas y permitieron obtener una visión general, así como una prueba de concepto y validación preliminar. Sin embargo, para futuros trabajos es necesario realizar cálculos más formales y validaciones más rigurosas sobre los valores de las métricas de Software.

Se pudo constatar que las suposiciones sobre las ventajas y desventajas observables en cada herramienta, corresponden con los resultados obtenidos. Esto permite asegurar que el enfoque fue acertado y que es posible continuar este trabajo como parte de una linea de investigación. Sin embargo, se debe tomar en cuenta el contexto y por el momento los resultados aquí presentados se circunscriben y son válidos solamente a este caso de estudio.

Finalmente es recomendable incluir otros frameworks que permitan visualizar las características de otras herramientas Web para conformar un catálogo que resuma diversos casos de estudio y sirva como guía para la toma de decisiones. Asimismo, la utilización de otras métricas de software también permitiría medir y evaluar otras características de las tecnologías.

REFERENCIAS BIBLIOGRÁFICAS

Ambler S. 2003. Agile Database Techniques: Effective Strategies For The Agile Software Developer. John Wiley & Sons, Inc. New York, Ny, Usa.

Boehm 1981. B. Software Engineering Economics. Prentice Hall.

Chidamber S., Kemerer C. 1993. A Metrics Suite For Object-Oriented Design.

Ewster R., Y Mendes E. 2001. Measurement, Prediction and Risk Analysis for Web Applications, metrics, pp.338, Seventh International Software Metrics Symposium (METRICS'01).

González Z. 2007. Medición y visualización de métricas de Software para *frameworks* Web. Caso de Estudio: Ruby on Rails y Frameworks en Java. Escuela de Computación. Facultad de Ciencias. Trabajo Especial de Grado. Universidad Central de Venezuela.

Heinemeier D., Tomas D. 2005. Agile Web Development with Rails. Dallas: The Pragmatic Programmers LLC.

Johnson R., Russo, V. 1991 Reusing Object-Oriented Designs. 91-1696.

Krasner G., Pope 1988. S. A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. Journal of Object-Oriented Programming, 1(3):26-49, Agosto/Septiembre 1988.

Mccabe T. 1976. A Software Complexity Measure.

Mccabe, T., Watson, A. 1994. Software Complexity. Crosstalk Jounal.

Offutt J. 2002. Quality Attributes of Web Software Applications. IEEE Software, 19 (2): 25-32.

Olsina L. 2007. Soporte de Información Contextual en un marco de Medición y Evaluación. Memorias del X Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software. Caracas.

Pressman R. 2005. Ingeniería de Software, Un enfoque práctico (Sexta edición). Madrid: Mc Graw Hill.

Rosenberg. L., Lawrence H. 1997. Software Quality Metrics for Object-Oriented Environments.

Crosstalk Jounal.

Ruhe M., Jeffery R., Wieczorek I. 2003. Cost estimation for web applications. In Proceedings of the 25th international Conference on Software Engineering (Portland, Oregon, May 03 - 10, 2003). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 285-294.