Análisis de proyectos de Programación Orientada a Objetos utilizando métricas de *software* como medio para determinar necesidades didácticas

Object-Oriented Programming project analysis with software metrics as a means to determine instructional needs

Ramón Ventura Roque Hernández[1]*, Juan Manuel Salinas Escandón[1], Adán López Mendoza[1], Salvador Mota Martínez[1], Óscar Flores Rosales[1]

Roque Hernández, R. V., Salinas Escandón, J. M., López Mendoza, A., Mota Martínez, S., Flores Rosales, O. Análisis de proyectos de Programación Orientada a Objetos utilizando métricas de *software* como medio para determinar necesidades didácticas. *Investigación y Ciencia de la Universidad Autónoma de Aguascalientes*. Número 61: 19-25, enero-abril 2014.

RESUMEN

El artículo presenta los resultados de un estudio cuyo objetivo es identificar si se requiere brindar un enfoque diferente a la enseñanza de la Programación Orientada a Objetos (POO) a los alumnos irregulares tomando como base las características de sus proyectos finales. Se hizo un estudio de las métricas de software en proyectos de 38 alumnos divididos en dos grupos: regulares e irregulares. Se utilizó una prueba estadística *t* para medir la diferencia de las métricas entre ambos grupos. Solamente el porcentaje de comentarios y el número de clases utilizadas tuvieron una diferencia significativa. El estudio concluye que no hay una diferencia importante entre las características de ambos grupos, por lo que no se requiere un enfoque diferente para impartir el curso de POO a alumnos irregulares.

Palabras clave: métricas de *software*, Programación Orientada a Objetos, enseñanza de la programación.
Keywords: software metrics, object oriented programming, programming teaching.

Recibido: 16 de junio de 2013, aceptado: 26 de febrero de 2014

[1] **Facultad de Comercio, Administración y Ciencias Sociales, Universidad Autónoma de Tamaulipas.**
* **Autor para correspondencia: rvhernandez@uat.edu.mx**

ABSTRACT

This paper presents the results of a study aimed at identifying if teaching object-oriented programming (OOP) to irregular students requires a different approach based on the characteristics of their final projects. A study of software metrics in 38 students' projects who were divided in two groups, regular and irregular was conducted. A statistical *t* test was used to measure the difference of the metrics between the groups. Only the percentage of comments and the number of classes had a significant difference. The study concludes that there is no significant difference between the characteristics of both groups, so a different approach in teaching the OOP course to irregular students is not required.

INTRODUCCIÓN

Con la finalidad de determinar si los alumnos irregulares que cursan la materia de Programación Orientada a Objetos (POO) necesitan un enfoque de enseñanza diferente al de los alumnos regulares, se realizó un estudio para conocer las características de los proyectos de *software* que cada grupo de estudiantes es capaz de realizar. Se buscó establecer, con base en un proyecto final, si había diferencias estadísticas significativas entre los productos de los alumnos regulares y los productos de los alumnos irregulares.

Con ayuda del *software* Source Monitor (Campwood, 2013) se obtuvieron los valores para diferentes métricas aplicadas al trabajo de 38 estudiantes. Los valores obtenidos fueron analizados después con el *software* SPSS (Valderrey, 2010) para elaborar una estadística descriptiva de los proyectos presentados por cada grupo y determinar el nivel de significancia entre cada una de las métricas para ambos grupos utilizando una prueba *t*. De las métricas aplicadas (porcentaje de comentarios, número de archivos utilizados, líneas de código en el proyecto, sentencias, clases, métodos por clase, número de llamadas por método, sentencias por método, complejidad promedio, complejidad máxima, profundidad promedio y profundidad máxima), solamente el número de clases incorporadas en el proyecto y el porcentaje de comentarios resultaron significativas estadísticamente. Se consideraron estos resultados, se emitieron algunas recomendaciones y se concluyó que no existen diferencias importantes entre los proyectos presentados por alumnos regulares y los presentados por alumnos irregulares.

La justificación para realizar esta investigación surgió de la necesidad de proporcionar a los alumnos los medios más adecuados para acceder al contenido de la materia de POO. Los profesores percibían que el ritmo de trabajo de los grupos irregulares era más lento porque tardaban un poco más en cubrir los temas y los alumnos tenían más problemas para aprobar los exámenes, pero sabían que no podían omitir contenido ni prácticas para ganar tiempo. El porcentaje de alumnos reprobados en grupos con alumnos irregulares solía ser elevado; por ejemplo, en los años 2010 y 2011, en promedio casi el 50% de los alumnos en estos grupos reprobó el curso. En casos extremos se observó hasta un 67% de reprobación.

Por otra parte, se percibía que los alumnos de ambos grupos carecían de buenas prácticas de programación, ya que en muchos casos desarrollaban sus aplicaciones con un enfoque de prueba y error orientados más a la obtención de la funcionalidad inmediata que a una buena arquitectura interna. Si bien es cierto que el contenido temático de la materia de POO no es explícito en cuanto a las buenas prácticas que se deben cultivar entre los estudiantes, la caracterización de la asignatura indica que la materia debe proporcionar al estudiante

> las competencias necesarias para abordar el estudio de cualquier lenguaje orientado a objetos, metodología de análisis y diseño orientado a objetos, de los sistemas gestores de bases de datos y en general de cualquier materia basada en el modelo orientado a objetos,

al mismo tiempo que determina la competencia específica de "Diseñar e implementar objetos de programación que permitan resolver situaciones reales y de ingeniería". Con el objetivo de preparar al alumno para enfrentar estos escenarios complejos, durante el desarrollo del curso se abordan de manera complementaria temas que promueven las buenas prácticas como la modularización, el uso de patrones, la creación de librerías, la correcta documentación y la especificación de requerimientos y diseños con UML.

Sin embargo, no se sabía si a pesar de todo, los alumnos regulares e irregulares al final de un curso normal de POO eran capaces de producir *software* comparable en características; se necesitaba esta información para determinar si las estrategias didácticas que hasta ese momento eran iguales, deberían de cambiar.

Una búsqueda bibliográfica demostró que las percepciones de los maestros no eran infundadas, pues se trataba de una problemática que se presentaba recurrentemente con grupos de programación y que podía ser abordada como un problema de investigación. Conforme a Cardell (2011), la evidencia documentada por diversos estudios sugiere que existe un pobre desempeño de aprendizaje en los cursos de programación para principiantes; sin embargo, los métodos para medir la habilidad para generar *software* todavía son un problema de investigación. Habría que considerar dentro de la problemática anterior que en el desarrollo de proyectos de programación en un ambiente escolar, el tamaño y complejidad de los proyectos difiere notablemente de los de un profesional de la industria. Conforme a esto, se debería involucrar a los estudiantes en proyectos más amplios y complejos con la finalidad de que obtengan experiencia relevante que les será necesaria cuando egresen y se integren a la industria del *software* (Clark, 2005).

Frentiu, Laza y Horia (2003) consideran que un proyecto pequeño desarrollado por estudiantes contiene al menos unos cientos de sentencias y otros más podrían llegar a 3000. Ellos proponen entre otras, las siguientes métricas de *software* para analizar estos proyectos: líneas de código, número de comentarios, número de clases, número de métodos

para todas las clases, promedio de métodos por clase y profundidad de herencia. Afirman que los estudiantes son escépticos a desarrollar un buen estilo de programación, no les gusta escribir comentarios, realizar un buen diseño o implementar algoritmos en pseudocódigo; se conforman con que el programa "funcione".

Además de lo antes expuesto, Edwards (2004) menciona que existe una diferencia en cuanto al enfoque metodológico para resolver un problema, ya que los estudiantes de Ciencias de la Computación o Informática utilizan en su mayoría el ensayo y error como herramienta de uso cotidiano para arreglar y depurar código —que para fines de aprendizaje de un lenguaje de programación podría ser adecuado—, sin embargo, tienden a utilizarlo por demasiado tiempo. Conforme a Cardell (2011), si se toma en consideración que los cursos de programación para estudiantes que inician tienen dos características importantes como resultado del aprendizaje: una de ellas es la habilidad para comprender el código de programación y otra es la habilidad para escribir código de programación, queda pendiente dentro de la investigación en esta área cómo medir esas características.

Para la presente investigación se tomaron las métricas de los proyectos finales de los alumnos como indicadores del trabajo que ellos son capaces de realizar. Las métricas para este estudio se eligieron porque representan conceptos y buenas prácticas que se abordaron durante las sesiones del curso, además de que en la literatura existen antecedentes que las refieren (Cardell, 2011). Estas métricas están basadas en atributos físicos del código de programa y proporcionan una medida cuantitativa útil para evaluar las características del *software*. Aunque se pudieron incorporar otras métricas como los puntos de función -*function points*- (Durán, 2003), este estudio no las consideró desde un principio porque se deseaba enfatizar las características técnicas internas y estructurales del código de los proyectos, ya que el contenido del curso está centrado en la perspectiva arquitectónica del *software* que el estudiante desarrolla. Por otra parte, los alumnos al término del curso desconocen aún la metodología para trabajar con puntos de función.

Una diferencia significativa en las métricas de los dos grupos analizados indicaría que los alumnos regulares e irregulares aplican los conceptos de la materia en distinta medida y podrían necesitar enfoques distintos al momento de abordar los contenidos de la materia para enfatizar unos aspectos más que otros.

MATERIALES Y MÉTODOS

Para este estudio se analizaron 38 proyectos de *software,* los cuales fueron entregados por los alumnos del curso Programación Orientada a Objetos como parte de su trabajo final. El curso proporciona un panorama introductorio a la POO con C# (Gaddis, 2013) con temas que abordan: 1) historia y generalidades del paradigma, 2) conceptos básicos (clases, objetos, atributos, métodos, constructores, destructores, sobrecarga), 3) herencia, 4) polimorfismo y 5) acceso a datos. Para el final de cada curso, todos estos temas fueron abordados en la misma profundidad y con ejemplos y prácticas iguales. El profesor encargado de conducir a todos los alumnos durante el curso fue siempre el mismo.

En el estudio realizado se clasificaron como "alumnos regulares" aquéllos que no han cursado la materia anteriormente ni han reprobado alguna de las asignaturas previas requisito para poder cursar POO. De esta manera, un "alumno irregular" es aquel que, o bien, ya cursó POO (y no la aprobó), o bien, es la primera vez que cursa POO, pero ha reprobado un prerrequisito de esta materia en semestres anteriores.

Todos los alumnos tuvieron 1½ semanas para desarrollar un proyecto final libre: cada estudiante decidió el área de aplicación y los detalles de la interfaz gráfica. Durante ese tiempo, hubo revisiones y asesorías periódicas con cada uno. Cuando se fijó la fecha de entrega, los alumnos habían desarrollado ya durante las sesiones de clase un proyecto sencillo con las funciones básicas de acceso a datos como: altas, bajas, consultas y listados mediante una arquitectura por capas orientada a objetos. Para la implementación de la base de datos en el proyecto final se utilizó Microsoft Access y se pidió incorporar por lo menos una tabla con cuatro campos con diferentes tipos de datos, también se solicitó incluir todas las funciones básicas de acceso a datos que durante las sesiones del curso se habían estudiado. En el proyecto se debía utilizar un enfoque orientado a objetos y una arquitectura de capas (presentación, objetos de negocio, acceso a datos). No hubo una restricción en cuanto al número de clases, al número de líneas de código utilizadas en el proyecto, o a alguna otra de las métricas utilizadas en esta investigación.

INVESTIGACIÓN Y CIENCIA DE LA UNIVERSIDAD AUTÓNOMA DE AGUASCALIENTES

Número 61: 19-25, enero-abril 2014

Figura 1. Programación Orientada a Objetos utilizando métricas de *software*.
Imagen de Juan Manuel Salinas Escandón.

Los alumnos realizaron una especificación funcional de requerimientos con UML, un manual técnico y un manual de usuario para su proyecto. Todo esto fue entregado como documentación anexa a su trabajo final. Para ser considerados en el presente estudio, los proyectos debieron satisfacer en su implementación al menos el 70% de los requisitos funcionales plasmados en la especificación.

Los 38 proyectos finales se analizaron con el *software* SourceMonitor para obtener los valores de las métricas para cada uno. SourceMonitor calcula valores a nivel proyecto y a nivel archivo. Para este trabajo se utilizaron los valores del proyecto completo, que en algunos casos representan medias aritméticas de las métricas aplicadas a unidades de menor nivel como lo son los métodos o sentencias en todo el proyecto. Las métricas que se analizaron en este estudio son definidas por el *software* SourceMonitor de la siguiente manera:

Número de líneas de código. Las líneas de código *(lines of code,* LOC) representan el número de líneas físicas en el proyecto, mientras que las líneas efectivas de código *(effective lines of code,* eLOC) ignoran los comentarios. Una métrica similar es el número de sentencias *(number of commands,* NOC), que cuentan el número de sentencias, son fáciles de computar y poseen correlación con métricas más complejas (Porkoláb y Sillye, 2004).

Número de sentencias. Esta métrica define una sentencia como aquéllas finalizadas por punto y coma (;). También son contados: 1) las ramificaciones tales como *if, for* y *while,* 2) los métodos y los atributos, 3) las sentencias de control de excepción tales como *try, catch* y *finally,* 4) las directivas de preprocesamiento como *#define* y *#undef*. Todas las demás directivas de preprocesador son ignoradas.

Porcentaje de líneas comentadas. Es el porcentaje de líneas comentadas que utilizan el estilo C (/*…*/) o el estilo C++ (//…). Kuipers, Heitlager y Visser (2007) mencionan que la importancia de líneas comentadas como métrica reside en la noción lógica de que una sección de código bien documentada es más fácil de mantener que una sección de código sin documentar; el índice de mantenimiento de Oman y Hagemeister *(Maintainability Index)* la considera opcional. Para este estudio se tomó en cuenta el porcentaje de líneas comentadas ignorando los comentarios de los encabezados y pies de página.

Porcentaje de líneas de documentación. Es el resultado de contabilizar las líneas que contienen documentación cuando son indicadas por "///" al inicio de una línea y relacionarlas con el número de líneas en el proyecto.

Número de clases, interfaces, estructuras. Esta métrica contabiliza todas las clases, interfaces y estructuras definidas en un proyecto. En el caso especial de las clases parciales, cada una se contabiliza separadamente en cada archivo donde reside.

Número de archivos utilizados. Se refiere al número de archivos con extensión ".cs" contenidos en el proyecto. Se toman en cuenta únicamente los archivos creados explícitamente por el alumno y se omiten los archivos que son auto generados por la herramienta de desarrollo.

Número de métodos por clase. Esta métrica es un promedio general de los métodos en clases, interfaces y estructuras en un proyecto. Se obtiene dividiendo el número total de métodos entre el número total de clases, interfaces y estructuras.

Número de llamadas por método. Esta métrica se obtiene dividiendo el número total de llamadas a otros métodos dentro de un proyecto entre el número de métodos en ese proyecto.

Número de sentencias por método. Se obtiene dividiendo el número total de sentencias dentro de todos los métodos del proyecto entre el número de métodos de ese proyecto.

Complejidad. Esta métrica registra el número de rutas de ejecución en un método o función. Cada método o función tiene una complejidad de 1 + 1 por cada rama en las sentencias *if, else, for, foreach, while*. Las sentencias conocidas como "*if* aritméticos" (prueba? valorVerdadero : valorFalso) incrementan 1 a la complejidad total. También cada operador "&&" y " || " agrega 1 a la cuenta. Cada sentencia *catch* o *except* dentro de un bloque *try* incrementan en uno la complejidad del proyecto; sin embargo, las sentencias *try* o *finally* no modifican ese valor. Para este estudio, se utilizó la complejidad modificada, la cual consiste en agregar 1 a la cuenta de complejidad por cada sentencia *switch* e ignorar para este conteo las sentencias *case*.

Complejidad máxima. Es el valor de complejidad más grande que se observó en los métodos del proyecto.

Complejidad promedio. Se calcula con el promedio aritmético de todos los valores de complejidad de los métodos o funciones del proyecto.

Profundidad. Se refiere a la colocación de bloques de código dentro de otros, lo que da lugar a bloques anidados; éstos se introducen principalmente con las sentencias de control como "*if*" y "*while*". Conforme se aumenta la profundidad, el código resulta más difícil de leer porque con cada nuevo nivel de anidamento se deben evaluar más condiciones para saber el momento en el que el código será ejecutado.

Profundidad máxima. Se refiere al mayor nivel de anidamiento registrado en el proyecto.

Profundidad promedio. Es el promedio ponderado de la profundidad de bloques de código anidados en un proyecto.

Durante el análisis, SourceMonitor fue configurado para: 1) utilizar métricas para el lenguaje C#, 2) analizar únicamente los archivos con extensión ".cs" que el alumno codificó directamente (como clases y formularios excluyendo el código autogenerado por la herramienta de desarrollo), 3) usar la métrica de complejidad modificada en lugar de la complejidad tradicional, la cual trata las sentencias *switch* de manera diferente como se explica en la definición de esta métrica, 4) ignorar los comentarios y documentación de cabecera y pie de página, si existieran.

Una vez que se obtuvieron los valores de las métricas para cada proyecto, se utilizó el *software* SPSS para registrarlos y analizarlos categorizados en dos grupos: alumnos regulares (REG) y alumnos irregulares (IRREG), de acuerdo a los criterios expuestos previamente.

RESULTADOS

Como producto del análisis, primero se obtuvo la estadística descriptiva mostrada en la Tabla 1, en donde se observa cada métrica y los datos de cada grupo: el número de alumnos incluidos, la media aritmética del valor de esa métrica, la desviación estándar y el error estándar de la media.

Con el *software* SPSS también se realizó una prueba *t* para la comparación de ambos grupos. El resultado se presenta en la Tabla 2, en donde se destaca el nombre de dos métricas con un asterisco (*): porcentaje de comentarios y número de clases en el proyecto; los valores de ambas resultaron significativamente diferentes entre los alumnos regulares e irregulares. El resto de las métricas analizadas no tuvieron una diferencia estadística significativa entre ambos grupos.

Tabla 1. Estadística descriptiva de los grupos

	Grupo	N	Media	Desviación estándar	Error estándar de la media
Archivos	REG	27	6.96	2.87	.55
	IRREG	11	5.09	1.92	.57
Líneas de código	REG	27	538.46	271.64	52.27
	IRREG	11	412.27	181.93	54.85
Sentencias	REG	27	313.74	138.80	26.71
	IRREG	11	230.18	94.45	28.47
% Comentarios	REG	27	3.27	2.66	.51
	IRREG	11	1.35	1.45	.43
% Documentación	REG	27	.01	.05	.01
	IRREG	11	.00	.00	.00
Clases	REG	27	6.96	2.87	.55
	IRREG	11	4.91	1.92	.57
Métodos por clase	REG	27	8.26	1.21	.23
	IRREG	11	7.95	1.36	.41
Llamadas por método	REG	27	2.02	.56	.10
	IRREG	11	1.92	.58	.17
Sentencias por método	REG	27	3.86	1.67	.32
	IRREG	11	3.42	.73	.22
Complejidad máxima	REG	27	6.11	6.82	1.31
	IRREG	11	4.00	2.14	.64
Profundidad máxima	REG	27	4.67	1.10	.21
	IRREG	11	4.27	.64	.19
Profundidad promedio	REG	27	1.94	.42	.08
	IRREG	11	1.79	.20	.06
Complejidad promedio	REG	27	1.46	.14	.02
	IRREG	11	1.40	.17	.05

Resumen estadístico de las métricas analizadas con el *software* SPSS.

Tabla 2. Prueba estadística de comparación de grupos

	Prueba *t* para la igualdad de medias						
	t	Grados de libertad	Sig. (2-colas)	Diferencia de medias	Error estándar de la diferencia	Intervalo de confianza de la diferencia (95%)	
						Inferior	Superior
Archivos	1.97	36	.05	1.87	.94	-.04	3.79
Líneas de código	1.41	36	.16	126.19	89.41	-55.14	307.53
Sentencias	1.82	36	.07	83.55	45.79	-9.32	176.44
% Comentarios*	2.85	32.571	.00	1.92	.67	.55	3.29
% Documentación	.63	36	.53	.01	.01	-.02	.04
Clases*	2.17	36	.03	2.05	.94	.13	3.97
Métodos por clase	.69	36	.49	.31	.44	-.60	1.22
Llamadas por método	.46	36	.64	.09	.20	-.32	.50
Sentencias por método	.83	36	.40	.44	.52	-.62	1.50
Complejidad máxima	.99	36	.32	2.11	2.11	-2.17	6.39
Profundidad máxima	1.09	36	.27	.39	.35	-.33	1.12
Profundidad promedio	1.13	36	.26	.15	.13	-.12	.42
Complejidad promedio	1.09	36	.28	.06	.05	-.05	.17

Resultados obtenidos de la prueba estadística de comparación de grupos realizada con el *software* SPSS.

DISCUSIÓN

En los resultados obtenidos en este estudio destaca el hecho de que los alumnos irregulares utilizaron menos clases en sus proyectos que los alumnos regulares. Sin embargo, los alumnos regulares también incorporaron pocas clases en sus proyectos. En ambos grupos se recomienda enfatizar los primeros temas del programa de la materia, en especial los relacionados con "clases" y "abstracción"; con esta iniciativa se busca que los alumnos fortalezcan su capacidad para identificar candidatos para convertirse en clases en los problemas planteados y puedan realizar tantas abstracciones como sean necesarias. Asimismo, el tema de "varios formularios en un proyecto", si bien no es parte de la materia de POO en su estado más puro, es imprescindible para el desarrollo de los proyectos finales de la materia y debe tener también una atención especial pues impacta en la métrica que contabiliza el número de clases.

Por otra parte, tanto en alumnos regulares como en alumnos irregulares, el uso de los comentarios y la documentación interna de los programas fue muy pobre, esto confirmó lo encontrado previamente en Frentiu *et al*. (2003). Se recomienda, durante la clase, poner en relieve la importancia del uso de comentarios y documentación en el código fuente con el propósito de concientizar a los estudiantes para que escriban código de fácil lectura, entendimiento y mantenimiento.

CONCLUSIONES

En este artículo se reportó un experimento basado en métricas de *software* para determinar si los alumnos irregulares necesitan un enfoque diferente de enseñanza al de los alumnos regulares en Programación Orientada a Objetos. Sus trabajos finales se tomaron como evidencia y fueron analizados y comparados. Se obtuvo como resultado que de las 13 métricas calculadas, solamente 2 resultaron con diferencia estadística significativa: porcentaje de comentarios y número de clases utilizadas en el proyecto. Sin embargo, se observó que en ambos grupos el número de clases pudo ser mayor por las características del proyecto solicitado y que el porcentaje de comentarios fue muy bajo. Se concluye que, con base en las métricas aplicadas a los proyectos realizados por los alumnos, no hay suficiente evidencia para concluir que el enfoque de enseñanza deba ser diferente para los alumnos regulares que para los irregulares; sin embargo, sí deberían tomarse en cuenta algunas

recomendaciones para ambos grupos: 1) enfatizar los temas de "clases" y "abstracción", 2) motivar a los alumnos a considerar el uso de varios formularios en un proyecto y 3) subrayar la importancia de escribir código de fácil entendimiento mediante el uso de comentarios y documentación interna.

Es importante considerar que la investigación realizada centra su análisis únicamente en las métricas de *software* de los proyectos finales de los estudiantes. Aunque se mantuvieron constantes las condiciones de cátedra, los temas abordados, las prácticas realizadas, las características solicitadas para el proyecto y el profesor conductor del curso, podrían existir factores particulares de los estudiantes, del profesor o del entorno que pudieran influir en las características de los proyectos entregados. El estudio mostrado en este artículo representa un trabajo en estado inicial. Como trabajo futuro se plantea repetir la investigación con otros grupos para aumentar la validación de los resultados y considerar el grado de generalización de los mismos, aplicar otras métricas para determinar si hay diferencias significativas entre ambos grupos y conducir entrevistas con los participantes de los experimentos para registrar sus experiencias en el desarrollo del proyecto. También resultaría relevante comparar los resultados de este estudio con aquéllos obtenidos cuando todos los alumnos desarrollen un mismo proyecto con características determinadas completamente por el profesor.

LITERATURA CITADA

- CARDELL, R. How can software metrics help novice programmers? *13th Australasian Computing Education Conference*, Vol. 114, 55-62, 2011.
- CLARK, N. Evaluating student teams developing unique industry projects. *Australasian Conference on Computing Education Conference*, Vol. 42, 21-30, 2005.
- DURÁN, S. Puntos por Función. Una métrica estándar para establecer el tamaño del software. *Boletín de política informática, XXVI*(6): 41-52, 2003.
- EDWARDS, S. Using Software Testing to Move Students from Trial-and-Error to Reflection-in-Action, *SIGCSE 04 Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, 26-30, 2004.
- FRENTIU, M., LAZA, I., y HORIA, F. On individual projects in software engineering Education. *Revista Studia Univ. Babes Bolyai, Informatica, XLVIII*(2): 83-94, 2003.
- GADDIS, T. *Starting out with Visual C# 2012*. (3ª ed.). EE UU: Addison-Wesley, 2013.
- KUIPERS, T., HEITLAGER, I., y VISSER, J. A Practical Model for Measuring Maintainability. *6th International Conference on the QUATIC 07 (Quality of Information and Communications Technology)*, 30-39, 2007.
- PORKOLÁB, Z. y SILLYE, A. Comparison of Object-Oriented and Paradigm Independent Software Complexity Metrics. *6th International Conference on Applied Informatics*, 435-444, 2004.
- VALDERREY, P. SPSS 17 *Extracción del conocimiento a partir del análisis de datos.* México: Alfaomega, 2010.

De páginas electrónicas

- CAMPWOOD. Campwood Software. De: http://www.campwoodsw.com/sourcemonitor.html, 1 jun. 2013.