

# Herramientas de análisis y medición de código administrado de Visual Studio® 2008.

## **Introducción**

*En este trabajo presentamos los resultados de una prueba que realizamos con las herramientas de análisis y medición de código administrado de Visual Studio® 2008. Las herramientas de análisis buscan defectos en el código mientras que las de medición determinan la complejidad y la facilidad de dar mantenimiento a éste. Aplicamos el análisis a dos piezas de código abierto, Database Commander y SQL Buddy, que están enfocadas a crear un entorno para acceder a diversos backends de bases de datos, y describimos los datos cualitativos que obtuvimos acerca de qué tan fácil instalamos las herramientas para analizar los proyectos designados y que tan rápidamente encontramos errores empleándolas.*

## **Herramientas de análisis de código**

Las herramientas de análisis llevan a cabo diversas verificaciones para determinar la presencia de defectos en el código, analizando los ensamblados y reportando cierta información acerca de aquellos tal como violaciones de reglas de programación y diseño establecidas en los Lineamientos de Diseño del .NET Framework. Estas herramientas representan las verificaciones que se llevan a cabo durante el análisis como advertencias. Los mensajes de advertencias identifican cualquier problema de diseño o programación relevante y cuando es posible, dan información acerca de cómo corregirlo.

Las advertencias antes mencionadas indican violaciones de reglas en bibliotecas de código administrado y están organizadas en una serie de áreas tales como diseño, desempeño, seguridad, etc. Las herramientas de análisis de código tienen la capacidad de permitir al usuario indicar que una determinada advertencia no es aplicable, con lo cual se informa a los desarrolladores o a quienes revisen el código posteriormente que esa advertencia ya fue investigada.

Finalmente estas herramientas permiten que uno se asegure de que el código que esta siendo protegido esté limpio, mediante el establecimiento de una política de protección que requiera que el análisis de código sea ejecutado; de esta manera solamente se podrá proteger una pieza de código, si ésta pasó exitosamente aquel.

## **Herramientas de medición de código**

Las herramientas de medición son empleadas para generar métricas del código, referentes a su complejidad y la facilidad para darle mantenimiento, que permiten a los desarrolladores entender mejor lo que están desarrollando. Al emplear las métricas, los desarrolladores pueden entender qué tipos y/o métodos deben ser escritos nuevamente o probados con mayor profundidad. Asimismo los equipos de desarrollo pueden identificar riesgos, entender mejor el estado actual del proyecto y dar seguimiento al progreso del mismo. Sin embargo, estas métricas no sirven a menos que tengan sentido para la persona que las está usando, en este caso el desarrollador, por lo que a continuación se presenta una breve descripción del significado de algunas de las cinco métricas ofrecidas por las herramientas.

- **Acoplamiento de clases.** Indica el número total de dependencias de otros tipos que un determinado nivel tiene. Este número no incluye tipos primitivos o incorporados, tales como Int32 Object o String, y cuando es mayor, es más probable que los cambios realizados en otros tipos se propaguen al elemento que está siendo analizado. Un valor mas bajo de acoplamiento de clases generalmente indica un candidato para reutilización.
- **Complejidad ciclomática.** Mide el número total de rutas individuales a lo largo del código, en cada nivel, y se calcula contando el número ramas (tales como switch, foreach y ciclos for) mas uno. Esta

métrica generalmente es una buena indicación del número de pruebas unitarias que se requerirán para obtener una completa cobertura del código.

- Índice de facilidad de mantenimiento. Es un número que va de 0 a 100, se calcula para cada miembro y nivel de tipo, e indica su facilidad de mantenimiento en general. Para el caso de espacios de nombres y ensamblados, es un promedio del índice de facilidad de mantenimiento para todos los tipos contenidos dentro de aquellos. Este índice se calcula en base a otras métricas incluyendo la complejidad ciclomática y el número de líneas de código e incluye un indicador icónico. Un número menor indica que el código es complejo y difícil de darle mantenimiento. El rango menor, 0-9 inclusive, muestra un círculo rojo, mientras que el rango intermedio, 10-19 inclusive, muestra un triángulo amarillo y por último, una caja verde indica alta facilidad de mantenimiento con valores entre 20 y 100 inclusive.

## Instalación de la Visual Studio® 2008 Team Suite

Después de descargar una versión de evaluación por 90 días de la Visual Studio® 2008 Team Suite, iniciamos el proceso de instalación. En virtud de que quisimos hacer el proceso de instalación lo mas ligero y ágil posible, seleccionamos solamente aquellas opciones que eran absolutamente necesarias para que las herramientas de análisis y medición de código funcionaran sobre proyectos de C#. Sin embargo, el proceso de instalación no concluyó sin problemas, la instalación del .NET Framework versión 3.5 falló. La bitácora de instalación no nos dio suficiente información para determinar cual era el problema, sin embargo pensamos que probablemente hubo un conflicto entre el mencionado componente y otro no identificado que ya estaba instalado en el equipo. La solución fue crear una máquina virtual desde cero instalando únicamente el sistema operativo y Visual Studio®.

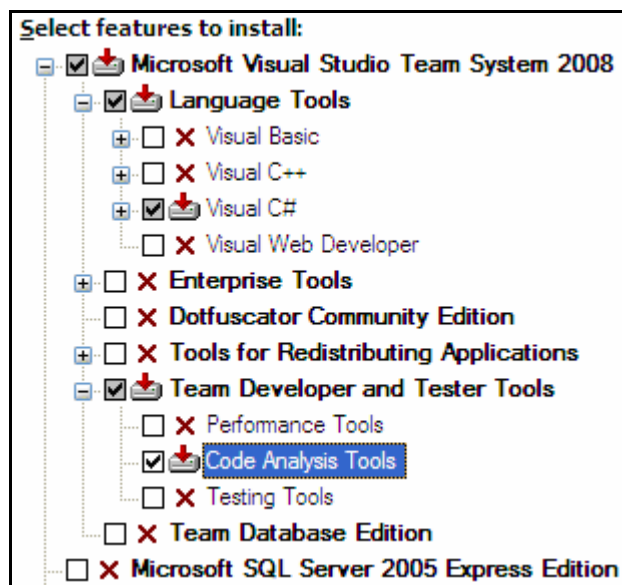
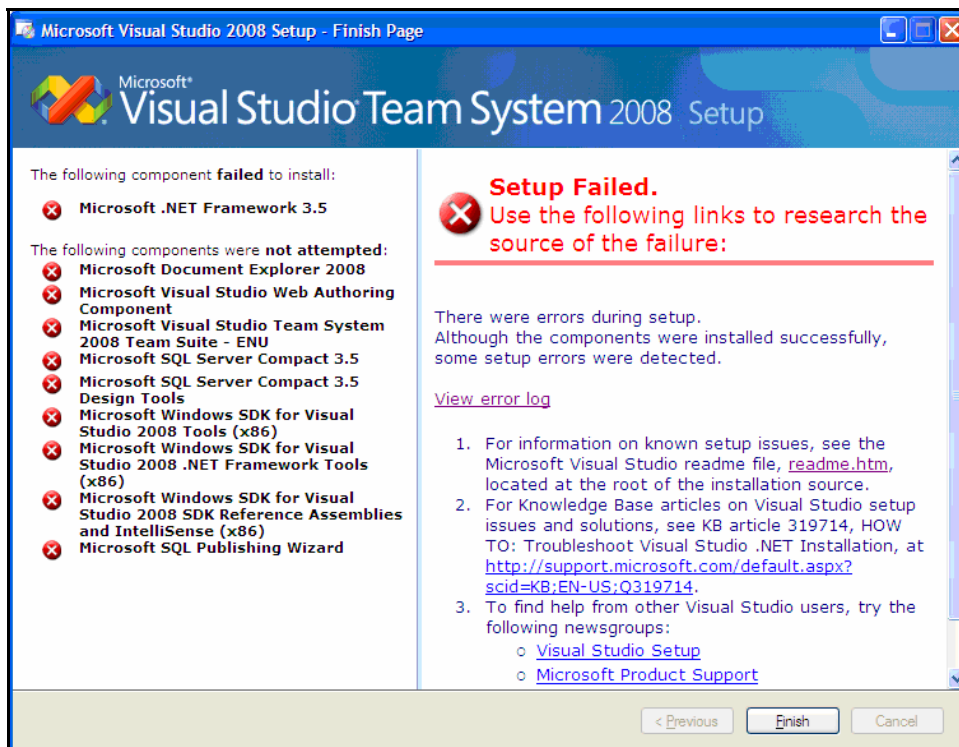


Ilustración 1 Opciones de instalación de la Visual Studio Team Suite



**Ilustración 2** Asistente de instalación de la Visual Studio Team Suite

Después de completar la instalación del entorno, instalamos la documentación del mismo. El proceso fue sencillo, bastó con invocar un asistente de instalación por separado.

### **Database Commander**

Después de descargar el código fuente del DBCommander, encontramos varios problemas. En virtud de que el código fuente del DBcommander no fue escrito originalmente con Visual Studio® 2008, fue necesario invocar el asistente de conversión de soluciones que viene incluido. Lo anterior produjo algunos errores, ninguno de los cuales era crítico; por ejemplo la imposibilidad de respaldar archivos con extensión .MDX era un error trivial, ya que esta clase de archivos constituyen un tipo de base de datos.



**Ilustración 3** Asistente de conversión de soluciones

Después de la conversión, intentamos compilar el proyecto para lo cual fue necesario compilar un proyecto dependiente (que estaba incluido en el código fuente pero no en la solución) que proveía un control para la aplicación. Una vez que concluimos lo anterior, fuimos capaces de compilar el código fuente en Visual Studio® 2008 por primera vez.

Ya con el proyecto cargado, intentamos invocar el análisis. En base a una entrada de blog que identificamos, sabíamos que era necesario intentar obtener primero las métricas de código. A pesar de que un principio no entendimos por completo el significado de las métricas, fue gratificante observar que los resultados se produjeron muy fácil y rápidamente. El hecho de que todos los proyectos se hayan mostrado en color verde nos transmitió confianza acerca del estado del código fuente.

Hierarchy	Maintainability Index
DBCommander (Debug)	80
YariSoft.DBCommander	84
YariSoft.DBCommander.Misc	82
YariSoft.DBCommander.Modals	68
DBUtil (Debug)	70
YariSoft.DBUtil	70
Utils (Debug)	79
YariSoft.Exceptions	65
YariSoft.Utils	80

**Ilustración 4 Métricas del código del Database Commander**

Cuando miramos las métricas en Visual Studio® 2008, se nos dificultó un poco entender exactamente que significan, partiendo del hecho de que se presentan como un árbol, en donde el nivel del proyecto constituye la raíz, y cada nivel de este puede representar un tipo diferente. Es decir, las métricas tienen un significado diferente en función al tipo de un nivel en particular.

Posteriormente probamos el análisis de código. Esta función está disponible seleccionando el proyecto de inicio en explorador de proyectos y la opción “Ejecutar Análisis de Código” del menú de Análisis. De esta manera se ejecuta el análisis que verifica el código contra los Lineamientos de Diseño de Microsoft® que mencionamos anteriormente. El resultado de este análisis se muestra en la siguiente ilustración. ¡Este análisis generó 159 advertencias! solo tomando en cuenta el proyecto de arranque. Sin embargo, nos tranquilizó el hecho de que no se encontraron errores.

Error List	
0 Errors	159 Warnings
Description	
CA1704 : Microsoft.Naming : Correct the spelling of 'Yari' in namespace name 'YariSoft.DBCommander'.	1
CA1704 : Microsoft.Naming : Correct the spelling of 'Yari' in namespace name 'YariSoft.DBCommander.Misc'.	2
CA1704 : Microsoft.Naming : Correct the spelling of 'Yari' in namespace name 'YariSoft.DBCommander.Modals'.	3
CA2210 : Microsoft.Design : Sign 'DBCommander.exe' with a strong name key.	4
CA1014 : Microsoft.Design : Mark 'DBCommander.exe' with CLSCompliant(true) because it exposes externally visible types.	5
CA1017 : Microsoft.Design : Because 'DBCommander.exe' exposes externally visible types, mark it with ComVisible(false) at the assembly level and then mark all types within the assembly that should be exposed to COM clients with ComVisible(true).	6
CA1824 : Microsoft.Performance : Because assembly 'DBCommander.exe' contains a ResX-based resource file, mark it with the NeutralResourcesLanguage attribute, specifying the language of the resources within the assembly. This could improve lookup performance the first time a resource is retrieved.	7

**Ilustración 5 Resultado del análisis del código del Database Commander**

Después de ejecutar el mismo análisis en cada uno de los otros dos proyectos, obtuvimos un total de 509 advertencias.

La primera advertencia se debía a que el nombre de un ensamblado estaba mal escrito. Al hacer click derecho sobre el error se nos permitió seleccionar la opción mostrar Ayuda del Error. Lo anterior nos proporcionó una descripción acerca de cómo corregir la advertencia, lo que involucro agregar la palabra mal escrita al diccionario personalizado contra el cual se realizan las verificaciones. El diccionario personalizado es un archivo llamado CustomDictionary.xml. En el ejemplo de la ayuda, fue interesante ver que el archivo todavía tiene referencias al sitio anterior de FXCop. Todo este proceso nos pareció incómodo. Lo que esperábamos era encontrar una opción que nos permitiera agregar una palabra directamente al diccionario personalizado.

## SQL Buddy

Tal y como ocurre en el caso del DBCommander, el código fuente de SQL Buddy no fue escrito empleando Visual Studio® 2008, de modo que fue necesario hacer una conversión. Afortunadamente, el proceso de conversión fue iniciado automáticamente cuando abrimos la solución de SQL Buddy la primera vez y concluyó sin errores.

Conversion Report - SqlBuddy		
Time of Conversion: Sunday, December 02, 2007 8:39 AM		
Project: SqlBuddy		
Filename	Status	Errors
SqlBuddy.csproj	Converted	0
Conversion Issues - SqlBuddy.csproj:		
Project converted successfully		
Scan complete: Upgrade not required for project files.		
1 file		
	Converted: 1	0
	Not converted: 0	

Ilustración 6 Reporte de conversión del código de SQL Buddy

La solución está compuesta de dos proyectos, el principal y el de instalación. Después de llevar a cabo la conversión de la solución eliminamos el proyecto de instalación ya que no era de interés. Una vez hecho esto fuimos capaces de compilar el código fuente en Visual Studio® 2008 por primera vez.

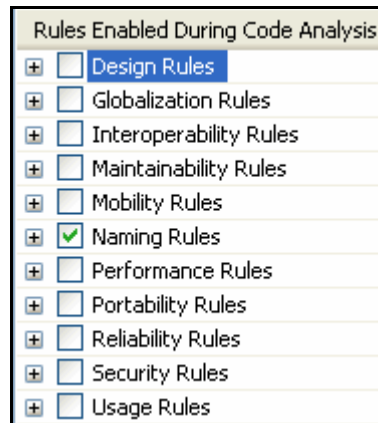
Para calcular las métricas de código simplemente accedimos al menú Analizar y seleccionamos la opción “Calcular Métricas de Código para SQL Buddy”. La ventana de métricas de código muestra los datos que son generados por el análisis. Decidimos enfocarnos exclusivamente en el índice de facilidad de mantenimiento para el proyecto.

Hierarchy	Maintainability Index
SqlBuddy (Debug)	82
SqlBuddy.App.WinApi	96
ICSharpCode.SharpDevelop.Gui.HtmlControl	95
SqlBuddy.App.Presenters	91
System.Windows.Forms	91
SqlBuddy	90
SqlBuddy.GUI.SynopsisControl	84
SqlBuddy.App.DatabaseSchema	84
SqlBuddy.App.Commands	81
SqlBuddy.App	81
SqlBuddy.GUI.Util	80
SqlBuddy.GUI.SessionUI	78
SqlBuddy.GUI.Selector	78
SqlBuddy.GUI.NavigationPanel	76
SqlBuddy.GUI	75
SqlBuddy.GUI.Menus	67
SqlBuddy.src.GUI	65
SqlBuddy.GUI.Options	60

Ilustración 7 Métricas del código del SQL Buddy

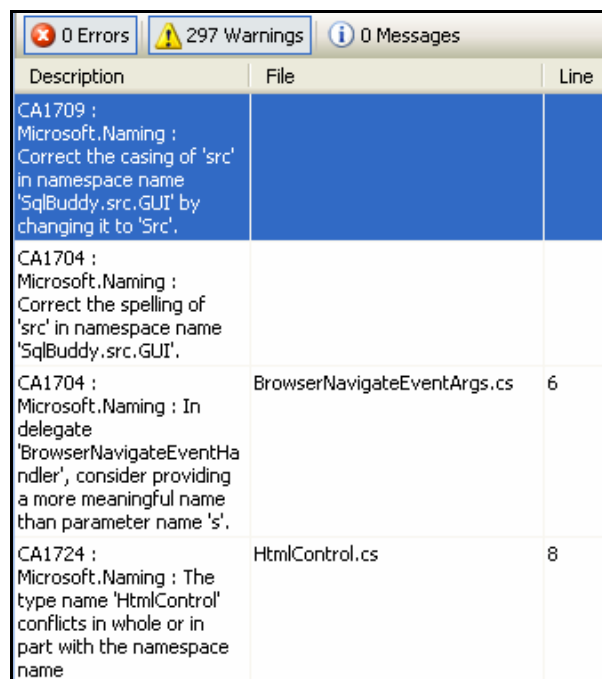
El lector habrá notado que en los resultados, la columna de facilidad de mantenimiento contiene un icono que acompaña al resultado numérico. Como ya mencionamos, un icono verde indicia un grado relativamente alto de facilidad de mantenimiento, mientras un icono amarillo indica un grado moderado de facilidad de mantenimiento, por último que un icono rojo indica baja facilidad de mantenimiento y un área potencialmente problemática. Los resultados del índice de facilidad de mantenimiento indican que el código fuente del SQL Buddy tiene relativamente un alto grado de facilidad de mantenimiento y no contiene áreas que puedan ser problemáticas.

Antes de analizar el código del proyecto decidimos activar únicamente la categoría de reglas de nomenclatura.



**Ilustración 8 Selección de categorías de reglas para el análisis de SQL Buddy**

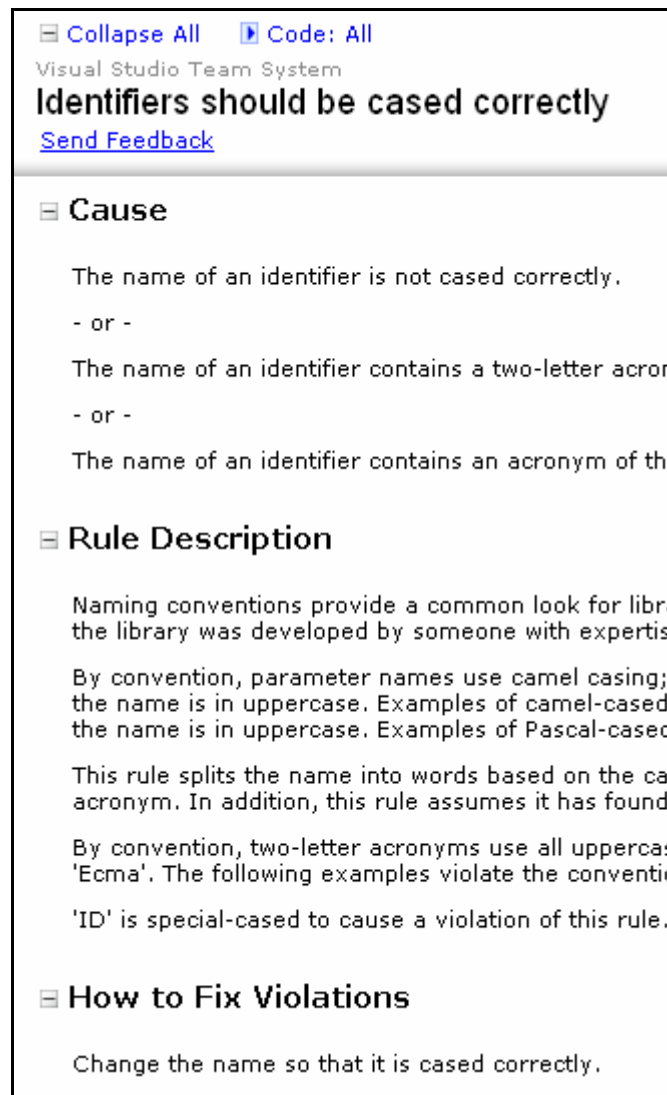
La ejecución del análisis de código fue una operación muy intuitiva, desde el menú *Analizar* seleccionamos la opción “Ejecutar el análisis de código en SQL Buddy”. Después de que el análisis fue concluido, la ventana lista de errores fue desplegada mostrando todas las advertencias. Esa ventana incluye la descripción de la advertencia, el archivo y el número de línea donde está localizada.



**Ilustración 9 Resultado del análisis del código del SQL Buddy**

Al dar doble clic sobre la advertencia se nos llevó al archivo y a la línea de código que contiene el problema. El análisis produjo 297 advertencias. Consideramos que este es número alto de advertencias, tomando en cuenta el hecho de que la cantidad de código fuente de SQL Buddy es relativamente pequeña. A partir de estos resultados pudimos concluir que el código fuente tiene un bajo nivel de apego a las reglas de nomenclatura de los Lineamientos de Diseño del .NET Framework.

Las herramientas de análisis de código incluyen información detallada acerca de cada advertencia incluyendo: el código específico que provoca la generación de la advertencia, una descripción de los problemas detrás de la advertencia, una explicación de cómo cambiar el código fuente para satisfacer la regla y prevenir que se genere la advertencia, una descripción de cuando es conveniente suprimir una advertencia.



**Ilustración 10** Página de ayuda de la advertencia relacionada con el uso de letras mayúsculas y minúsculas

Para mirar esta información hicimos click derecho en una advertencia y seleccionamos la opción “Mostrar Ayuda del Error”. Gracias a esta información, la tarea de entender el significado de las advertencias fue muy fácil. Por otro lado notamos que un número pequeño de las primeras advertencias en la lista tenían valores en blanco para el archivo y el número de línea. Al principio no entendimos porqué, sin embargo posteriormente descubrimos que el problema estaba relacionado con varios archivos. Las advertencias antes mencionadas estaban relacionadas con el uso de letras mayúsculas y minúsculas en el nombre de algunos espacios de nombres.

Después de analizar las explicaciones detalladas de estas advertencias y el código fuente, llegamos a la conclusión de estas advertencias eran válidas y decidimos corregirlas, para hacerlo empleamos la capacidad de refactorización del entorno. Finalmente, ejecutamos el análisis de código nuevamente y verificamos que las advertencias ya no aparecieran. En este caso encontramos que el proceso fue fácil y las capacidades del entorno que empleamos resultaron ser muy intuitivas.

## Conclusión

Las herramientas que usamos son intuitivas y fáciles de usar. La realización de la mayoría de las operaciones básicas tales como iniciar un análisis, mirar los resultados del análisis, encontrar el código que tiene el problema y corregir los problemas, fue sencilla. La documentación del producto es muy abundante lo cual facilita el entendimiento de las advertencias. La funcionalidad de métricas de código es muy poderosa ya que nos permitió tener un panorama claro acerca del nivel de facilidad de mantenimiento del código fuente a pesar de que no estábamos familiarizados con él. Adicionalmente, las herramientas de análisis y medición de código de Visual Studio® 2008 per se no requieren prácticamente ningún tipo de configuración. Asimismo la conversión de los proyectos fue automática. Encontramos y corregimos advertencias sobre reglas de nomenclatura en los espacios de nombres del código fuente de SQL Buddy. Por último, estas herramientas pueden hacer una excelente combinación con las herramientas de pruebas unitarias ya que soportan el establecimiento de una política de protección de código.

Por otro lado, las herramientas de análisis y medición de código únicamente pueden ser aplicadas a ensamblados administrados, lo cual constituye una limitación importante ya que no pueden procesar código producido en otros lenguajes predominantes tales como C++ o Java. Las herramientas de Visual Studio® 2008 están empotradas en el entorno de Visual Studio® la cual es un producto grande y complejo que en ocasiones puede ser difícil de instalar. Llevar a cabo la selección de un subconjunto de alertas relevantes para un proyecto en particular no es una tarea trivial ya que requiere conocimiento de los Lineamientos de Diseño de Microsoft®. Por último, no existe la noción de severidad dentro de las advertencias obtenidas que pueda ayudar al desarrollador a decidir por donde comenzar a corregir las advertencias.

Por último como respuesta a nuestra pregunta inicial acerca de la facilidad para llevar a cabo la instalación, las herramientas de análisis de Visual Studio® 2008 son por mucho las mas fáciles de instalar de todas las herramientas que hemos usado hasta el momento. La funcionalidad que dichas herramientas incorporan aparenta ser muy robusta. En virtud del análisis de código generalmente se basa en los Lineamientos de Diseño de Microsoft®, no existe ningún lenguaje para expresar reglas, que nosotros hayamos descubierto, que permitan agregar verificaciones al análisis que se efectúa. De modo que, en dependencia de cuales atributos de calidad esté tratando de dar seguimiento y mejorar en su proyecto, las herramientas a análisis de código de Visual Studio® 2008 pueden ser de mucha utilidad.

## Referencias

- [1] <http://blogs.msdn.com/fxcop/archive/2007/10/03/new-for-visual-studio-2008-code-metrics.aspx>
- [2] <http://dbcommander.sourceforge.net/>
- [3] [http://msdn2.microsoft.com/en-us/library/czefa0ke\(VS.71\).aspx](http://msdn2.microsoft.com/en-us/library/czefa0ke(VS.71).aspx)
- [4] <http://msdn2.microsoft.com/en-us/vstudio/products/aa700831.aspx>
- [5] <http://sqlbuddy.sourceforge.net/>

---

Rajah James (rjames (en) andrew (punto) cmu (punto) edu) es ingeniero senior en Yellowstone Hotel Systems y está trabajando en la última versión de su software principal OpenBook; antes trabajó durante seis años para Rockwell Software y Rockwell Automation, donde estuvo involucrado con sus soluciones de manufactura e integración empresarial basadas en .NET. Rajah tiene un posgrado en ingeniería de software de la Universidad Carnegie Mellon.

Rigoberto Calleja Cervantes (rigobertoc (en) alumni (punto) cmu (punto) edu) tiene cuatro años de experiencia como consultor en ingeniería de software, impartiendo cursos y participando en proyectos de adopción de herramientas de gestión del ciclo de vida de desarrollo de software. Rigoberto tiene un posgrado en ingeniería de software de la Universidad Carnegie Mellon.

El presente artículo es una cortesía de sus autores para el portal [www.xtrategy.com.mx](http://www.xtrategy.com.mx)