Aplicación informática para obtener la Complejidad Ciclomática de algoritmos que utilicen estructura *if* y *while*[*]

Ana Delia Harriette Cabrera

Dainerys Sanamé Álvarez

Carrera: Ingeniería Informática

Instituto Superior Minero Metalúrgico (Cuba).

Resumen: Se logró el cálculo ciclomático para algoritmos que utilicen estructura *if* y *while*. La misma visualiza las tres formas que se emplean para obtener dicho cálculo con las estructuras especificadas. Actualmente esta técnica de prueba de caja blanca es realizada a mano por los ingenieros del software, debido a esta problemática surge la necesidad de sistematizar dicha técnica. Al realizar diferentes pruebas a la aplicación se logró obtener resultados que validaron su eficiencia. Los resultados adquiridos en el cálculo de la Complejidad Ciclomática definen el número de caminos independientes dentro de un fragmento de código.

Palabras clave: Algoritmos; cálculo ciclomático; estructura *if*; estructura *while*.

[*] Trabajo presentado en el fórum científico estudiantil 2014 de la carrera Ingeniería Informática. Tutorado por el ing. Edgar Núñez Torres.
Recibido: 13 abril 2014 / Aceptado: 30 mayo 2014.

Cyclomatic Complexity for algorithms using the *if* and *while* structure

Abstrac: The cyclomatic calculation was made for algorithms using the if and while structure. This describes the three ways that are used to calculate the specified structures. This clear box testing technique is carried out by software engineers manually. This explains the need to systematize the technique. The testing results proved the technique is efficient. The cyclomatic complexity results defined the number of independent paths through a fragment code.

Keywords: Algorithms; cyclomatic measure; *if* structure; *while* structure.

Introducción

Las pruebas del software son un elemento crítico para la garantía de la calidad del software mismo y representa una revisión final de las especificaciones, del diseño y de la codificación. Las pruebas son unos de los procesos de ingeniería del software que se consideran más destructivos que constructivos. Estas requieren que se descarten ideas preconcebidas sobre la corrección del software que se acaba de desarrollar y se supere cualquier conflicto de intereses que aparezcan cuando se descubran errores (Jacobson *et al.*, 1999).

Objetivos de las pruebas de software

> La prueba es el proceso de ejecución de un programa con la intención de descubrir un error.
> Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.

Prueba de caja blanca

Las pruebas de caja blanca también conocidas como pruebas de caja de cristal o pruebas estructurales se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente.

El testeador escoge distintos valores de entrada para examinar cada uno de los posibles flujos de ejecución del programa y cerciorarse de que se devuelven los valores de salida adecuados. Al estar basadas en una implementación concreta, si esta se modifica, por regla general, las pruebas también deberán rediseñarse.

Aunque las pruebas de caja blanca son aplicables a varios niveles unidad, integración y sistema, habitualmente se aplican a las unidades de software. Su cometido es comprobar los flujos de ejecución dentro de cada unidad (función, clase, módulo, etc.) pero también pueden testear los flujos entre unidades durante la integración, e incluso, entre subsistemas, durante las pruebas de sistema.

A pesar de que este enfoque permite diseñar pruebas que cubran una amplia variedad de casos de prueba, podría pasar por alto partes incompletas de la especificación o requisitos faltantes, pese a garantizar la prueba exhaustiva de todos los flujos de ejecución del código analizado.

Principales técnicas de diseño de pruebas de caja blanca

-Pruebas de flujo de control
-Pruebas de flujo de datos
-Pruebas de bifurcación (*branch testing*)
-Pruebas del camino básico.

La prueba del camino básico es una técnica de prueba de caja blanca propuesta inicialmente por Tom McCabe. El método del camino básico permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecuta, por lo menos una vez, cada sentencia del programa.

Complejidad Ciclomática

La Complejidad Ciclomática es una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Es una de las métricas de software de mayor aceptación, ya que ha sido concebida para ser independiente del lenguaje.

El objetivo es, a partir de un fragmento de código donde se utilicen las estructuras *if* y *while*, obtener el cálculo ciclomático utilizando cualquiera de las tres formas que se describen posteriormente. Además de identificar los nodos referentes a cada línea de código, así como la matriz de los mismos donde se verifica la conectividad entre ellos; todo esto a través de una aplicación informática, permitiéndole al programador de cualquier software ahorro de recursos y tiempo en este proceso.

Una vez calculada la Complejidad Ciclomática de un fragmento de código se puede determinar el riesgo que supone utilizando los rangos definidos en la siguiente tabla:

Tabla 1. Evaluación del riesgo de la Complejidad Ciclomática

Complejidad Ciclomática	**Evaluación del riesgo**
1-10	Programa simple, sin mucho riesgo
11-20	Más complejo, riesgo moderado
21-50	Complejo, Programa de alto riesgo
50	Programa no testeable, muy alto riesgo

A partir del análisis de muchos proyectos McCabe encontró que un valor 10 es un límite superior práctico para el tamaño de un módulo. Cuando la complejidad supera dicho valor se hace muy difícil probarlo, entenderlo y modificarlo. La limitación deliberada de la complejidad en todas las fases del desarrollo ayuda a evitar los problemas asociados a proyectos de alta complejidad. El límite propuesto por McCabe, sin embargo, es fuente de controversias. Algunas organizaciones han utilizado el valor 15 con bastante éxito.

Cálculo de Complejidad Ciclomática

Primero introducir una sencilla notación para la representación del flujo de control, denominada grafos de flujo de control de un programa.

V(G) = Complejidad Ciclomática.

A = Número de aristas del grafo. Una arista conecta dos vértices si una sentencia puede ser ejecutada inmediatamente después de la primera.

N = Número de nodos del grafo correspondientes a sentencias del programa.

P = Número de nodos predicados. Estos son los que en el grafo de flujo poseen más de una conexión con otro nodo.

Definidos estas notaciones, la Complejidad Ciclomática puede calcularse de la siguiente manera:

Primera forma: La cantidad de regiones del grafo de flujo debe de coincidir con el cálculo ciclomático de la matriz de grafo.
Segunda forma: $V(G) = A - N + 2$
Tercera forma: $V(G) = P + 1$

La Complejidad Ciclomática puede ser aplicada en varias áreas incluyendo:

Análisis de riesgo en desarrollo de código: mientras el código está en desarrollo, su complejidad puede ser medida para estimar el riesgo inherente.

Análisis de riesgo de cambio durante la fase de mantenimiento: la complejidad del código tiende a incrementarse a medida que es mantenido durante el tiempo.

Midiendo la complejidad antes y después de un cambio propuesto puede ayudar a decidir cómo minimizar el riesgo del cambio.

Planificación de pruebas: el análisis matemático ha demostrado que la Complejidad Ciclomática indica el número exacto de casos de prueba necesarios para probar cada punto de decisión en un programa.

Reingeniería: provee conocimiento de la estructura del código operacional de un sistema. El riesgo involucrado en la reingeniería de una pieza de código está relacionado con su complejidad.

Grafo de flujo o grafo del programa

Representa el flujo del control lógico mediante las siguientes notaciones:

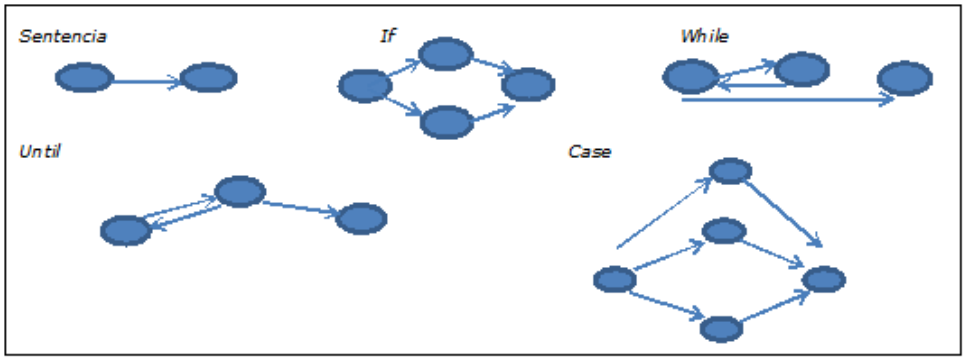

Figura 1. Notación de grafo de flujo.

Cada uno de los círculos representa una o más sentencias, sin bifurcaciones en LPD o código fuente.

Matriz de grafo

El procedimiento para obtener el grafo de flujo, e incluso la determinación de un conjunto de caminos básicos, es susceptible de ser mecanizado. Para desarrollar una herramienta de software que ayude en la prueba del camino básico puede ser bastante útil una estructura de datos denominada matriz de grafo.

Una matriz de grafo es una matriz cuadrada cuyo tamaño es igual al número de nodos del grafo de flujo. Cada columna y cada fila corresponde a un nodo en específico y las entradas de la matriz corresponden a las conexiones entre los nodos.

A una matriz de grafo también se le añade un peso de enlace a cada entrada de la matriz. Este peso de enlace es 1 si existe una conexión y 0 si no existe conexión. Estos pasos de enlaces tienen otras propiedades:

El tiempo de procesamiento asociado al recorrido de un enlace.
La memoria requerida durante el recorrido de un enlace.
Los recursos requeridos durante el recorrido de un enlace.

Implementación de la solución

En la implementación del sistema se utilizó el entorno de programación NetBeans 7.3 por las facilidades que brinda para el trabajo con el lenguaje de programación Java, el cual fue seleccionado para el desarrollo de la aplicación por las características ventajosas que brinda en comparación con otros lenguajes de programación (Eckel, 2008).

NetBeans es un software en el cual se pueden crear programas en un lenguaje de programación determinado, de manera rápida y fácil. Es una herramienta libre y gratuita. Permite programar aplicaciones, principalmente en Java, pero también admite otros lenguajes como PHP. Algo muy importante de NetBeans es que es compatible con diversos sistemas operativos, tal como Windows, Mac, Linux o Solaris, además de tener una fácil instalación (Pressman, 2005).

Resultados

Dado un fragmento de código y al dar clic en la opción *Run* la aplicación da como resultado los correspondientes nodos del grafo de flujo; posteriormente, se puede calcular la Complejidad Ciclomática mediante cualquiera de las tres formas propuestas en la aplicación, para tener mayor seguridad sobre si el resultado del cálculo es correcto o no, sugerimos realizar el cálculo ciclomático mediante las tres formas.

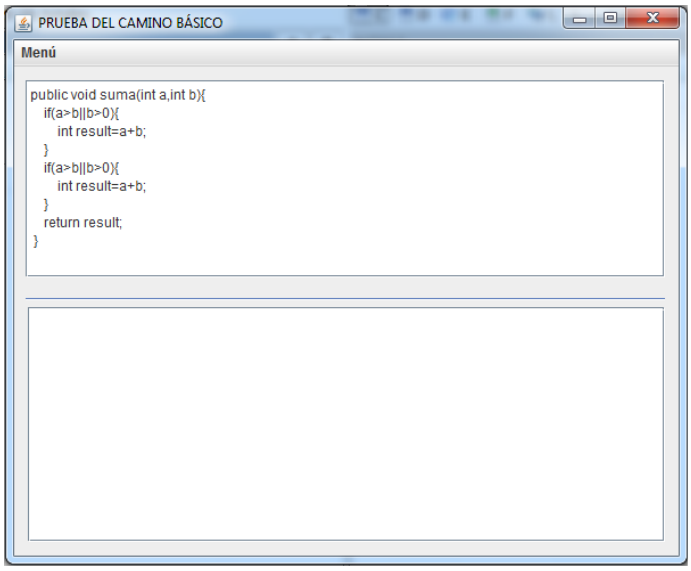

Figura 2. Ventana principal donde se introduce el fragmento de código.

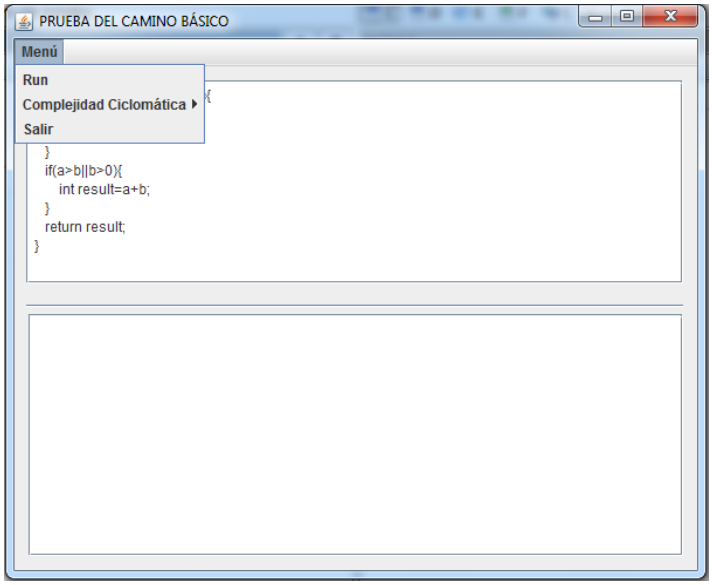

Figura 3. Opciones del menú.

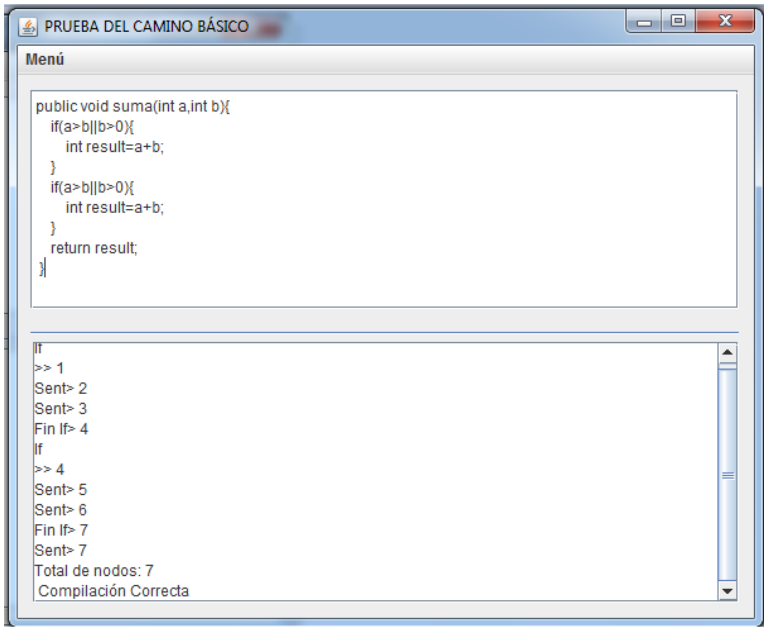

Figura 4. Después de haber ejecutado el Run.

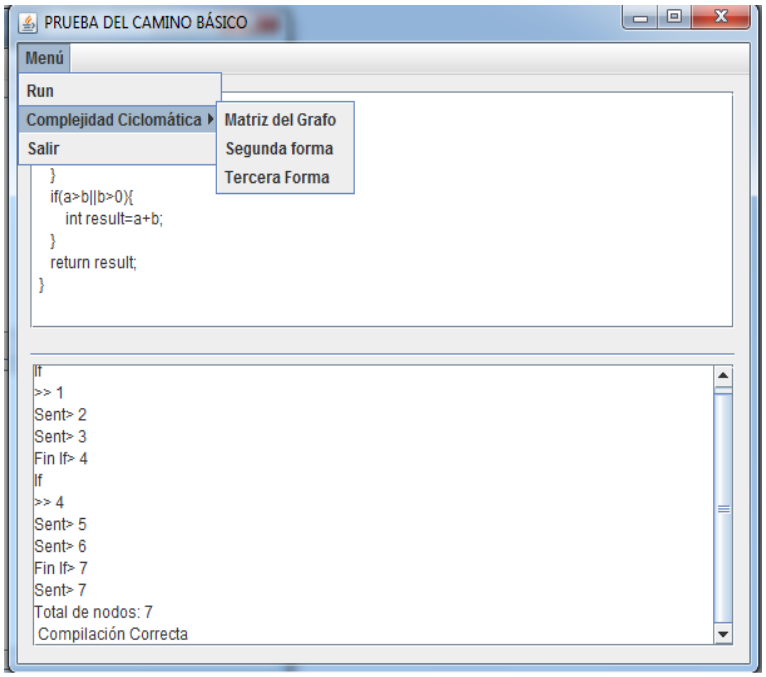

Figura 5. Vista de las tres opciones para realizar la Complejidad Ciclomática.

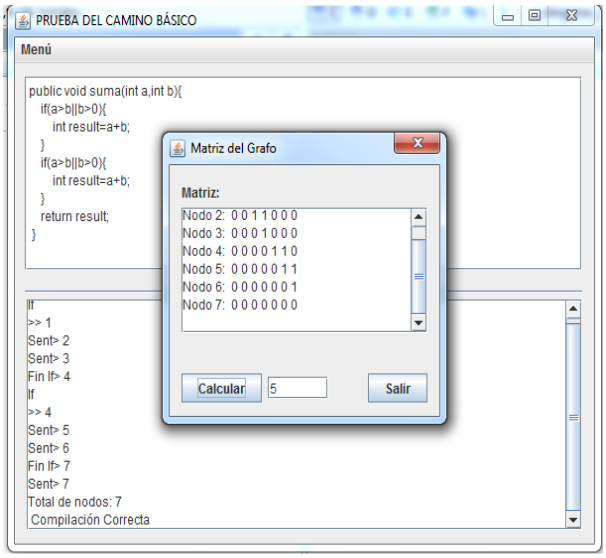

Figura 6. Complejidad ciclomática mediante la matriz del grafo.

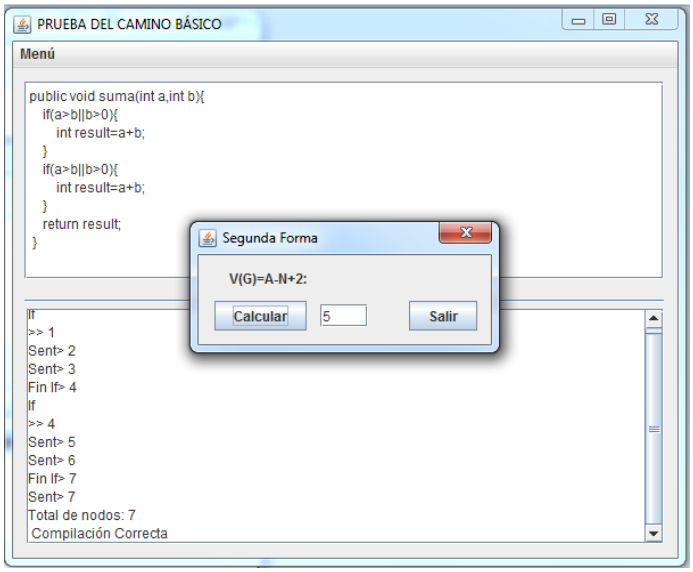

Figura 7. Complejidad ciclomática mediante la segunda forma.

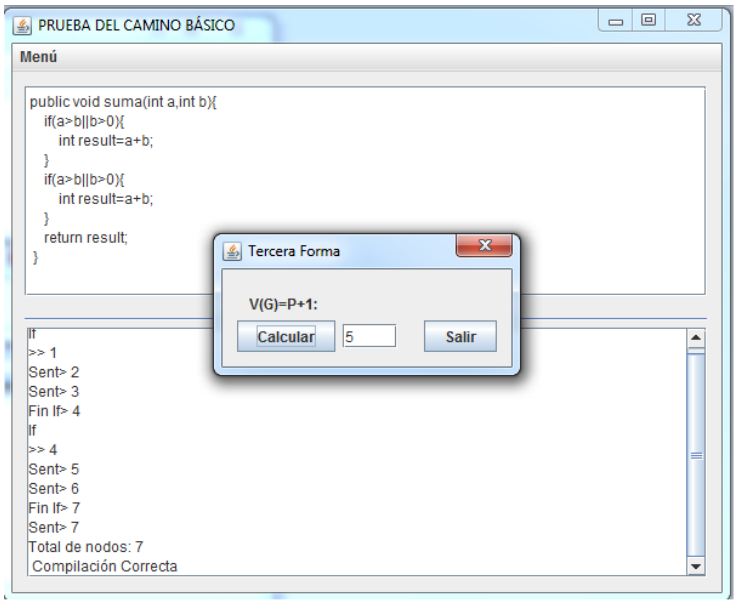

Figura 8. Complejidad ciclomática mediante la tercera forma.

Conclusiones

La aplicación implementada permite optimizar el proceso de cálculo de la Complejidad Ciclomática para cualquier fragmento de código que utilice estructura *if* y *while*, de manera que se pueda utilizar en el ISMMM o en otras instituciones que trabajen con esta herramienta de gestión de software.

La inclusión de dicha herramienta en la universidad es útil para el trabajo tanto de profesores como estudiantes para desarrollar sus habilidades mediante esta aplicación.

Recomendaciones

Realizar un análisis para las demás sentencias de código y así poder obtener el cálculo ciclomático.

Al aplicar la primera forma del cálculo ciclomático realizar el dibujo del grafo de flujo de manera dinámica para verificar si la cantidad de regiones del grafo coinciden con el cálculo ciclomático obtenido de la matriz del grafo.

Referencias bibliográficas

ECKEL, B. 2008: *Thinking in Java*. Primera Parte. 3th Edition, Beta. Editorial Félix Varela, Cuba.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. 1999: *The Unified Software Development Process*. Addison-Wesley Longman, Amsterdam.

PRESSMAN, R. 2005: *Ingeniería de Software. Un enfoque práctico*. Parte 1. Editorial Félix Varela, La Habana.