Métricas de Complejidad del Software*

Miguel-Angel Sicilia

Abstract

Definición y tipos de Métricas de Complejidad con ejemplos para calcular la complejidad.

Son todas las métricas de software que definen de una u otra forma la medición de la complejidad; Tales como volumen, tamaño, anidaciones, costo (estimación), agregación, configuración, y flujo. Estas son los puntos críticos de la concepción, viabilidad, análisis, y diseño de software.

Los 2 tipos de métrica para calcular la complejidad es:

- Complejidad ciclomática de McCabe[1]
- Ciencia del Software de Halstead[2]

1 Complejidad ciclomática de McCabe

La complejidad ciclomática se basa en el recuento del número de caminos lógicos individuales contenidos en un programa. Para calcular la complejidad del software, Thomas McCabe utilizó la teoría y flujo de grafos. Para hallar la complejidad ciclomática, el programa se representa como un grafo, y cada instrucción que contiene, un nodo del grafo. Las posibles vías de ejecución a partir de una instrucción (nodo) se repesentan en el grafo como aristas. Por ejemplo, el código que se muestra a continuación con 2 sentencias selectivas anidadas genera el siguiente grafo:

```
1 if (condicion){

2  if (condicion){

3      A;

       B;

   } else {
```

*Version 1.8: Jan 9, 2009 2:33 am US/Central
†http://creativecommons.org/licenses/by/2.0/
[1]McCabe,T.J., y A.H. Watson, "Solftware Complexity", Crosstalk.
[2]Halstead, M., "Elements of Software Science", Holland.

```
4       C;

        D;

5   }

6 }
```

Si se realizase el grafo, se observaría que se encuentran 3 caminos posibles para llegar de la sentencia 1 a la sentencia 6:
Camino 1 (si ambos IF's son verdad): Sentencias 1, 2, 3, 6
Camino 2 (si el primer IF es verdad y el segundo es falso): Sentencias 1, 4,6
Camino 3 (si el primer IF es falso): Sentencias 1, 6
Este programa tiene una complejidad ciclomática de 3.
La complejidad ciclomática se puede calcular de otras maneras. Se puede utilizar la fórmula:

$$\mathbf{v}(\mathbf{G}) = \mathbf{e} - \mathbf{n} + \mathbf{2}$$

donde e representa el número de aristas y n el número de nodos.
Otra forma de calcular la complejidad ciclomática consiste en aplicar la siguiente fórmula:

$$\mathbf{v}(\mathbf{G}) = \textbf{nmero de regiones cerradas en el grafo} + \mathbf{1}$$

1.1 Ejemplo de Calcular la Complejidad Ciclomática

Calcular la complejidad ciclomática del método de ordenación de la Burbuja siguiendo el siguiente código:

```
Public static void bubbleSort2 (Sequence S) {

 Position prec, succ;

 int n = S.size();

 for (int i = 0; i < n; i++) {

  prec = S.first();

  for (int j=1; j < n - i; j++) {

   succ = S.after(prec);

   if (valAtPos(prec) > valAtPos(succ))

    S.swapElements(prec, succ);

   rec = succ;

  }
```

```
    }

  }
```

El código da como resultado el siguiente grafo:

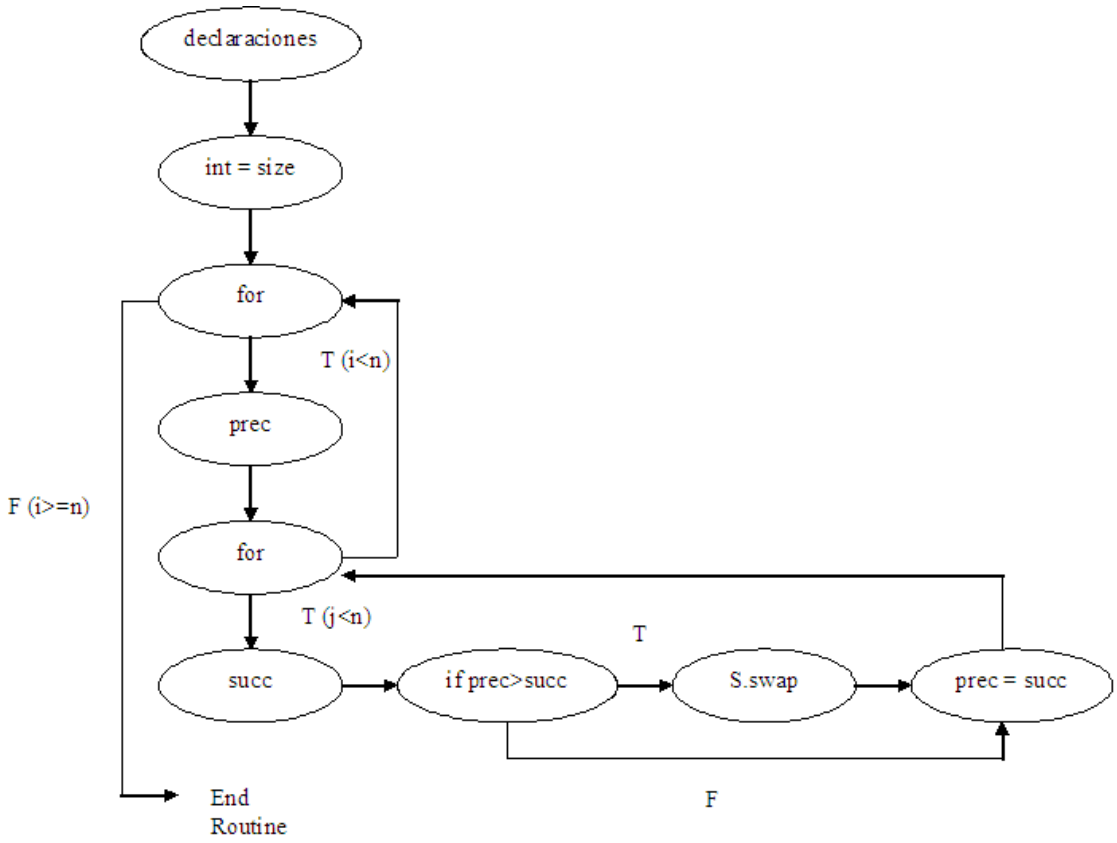

Figure 1

$$\mathbf{v(G) = e - n + 2}$$

v(G) = 12 - 10 + 2 = 4
Complejidad ciclomática = 4

2 Ciencia del software de Halstead

Durante el final de los años 70 y principios de los 80, Maurice Halstead desarrolla un conjunto de métricas llamadas Halstead Software Science, métricas basadas en el cálculo de palabras clave (reservadas) y variables.

Su teoría está basada en un simple cuenta (muy fácil de automatizar) de operadores y operandos en un programa:

- Los operadores son las palabras reservadas del lenguaje, tales como IF-THEN, READ, FOR,...; los operadores aritméticos +, -, *,..... los de asignación y los operadores lógicos AND, EQUAL TO,....
- Los operandos son las variables, literales y las constantes del programa.

Halstead distingue entre el número de operadores y operandos únicos y el número total de operadores y operando. Por ejemplo, un programa puede tener un READ, siete asignaciones y un WRITE; por lo tanto tiene tres únicos operadores, pero nueve en total operadores, y de manera idéntica se procede con los operandos. Se utiliza la siguiente notación:

- n1 - número de operadores únicos que aparecen en un programa
- N1 - número total de ocurrencias de operadores
- n2 - número de operandos únicos que aparecen en un programa
- N2 - número total de ocurrencias de operandos

Las métricas de la Ciencia del Software para cualquier programa escrito en cualquier lenguaje pueden ser derivadas de estas cuatro cuentas. A partir de ellas han sido elaboradas diferentes medidas para diversas propiedades de los programas, tales como longitud, volumen, etc...

Por ejemplo, consideremos el siguiente trozo de programa:

```
if (N<2){

    A=B*N;

    System.out.println("El resultado es : " + A);

}
```

A partir de aquí se deduce:
N2 = 6 (N, 2, A, B, N, A)
N1 = 6 (if, {}, system.out.println, =, *, <)
n2 = 4 (N, A, B, 2)
n1 = 6 (if, {}, system.out.println, =, *, <)
Halstead permite obtener una medida de la longitud, N, de un programa, que es calculada como:

$$\mathbf{N = N1 + N2}$$

N es una simple medida del tamaño de un programa. Cuanto más grande sea el tamaño de N, mayor será la dificultad para comprender el programa y mayor el esfuerzo para mantenerlo. N es una medida alternativa al simple conteo de líneas de código. Aunque es casi igual de fácil de calcular, N es más sensible a la complejidad que el contar el número de líneas porque N no asume que todas las instrucciones son igual de fácil o de difícil de entender.

La medida de longitud, N, es usada en otra estimación de tamaño de Halstead llamada volumen. Mientras que la longitud es una simple cuenta (o estimación) del total de operadores y operandos, el volumen da un peso extra al número de operadores y operandos únicos. Por ejemplo, si dos programas tienen la misma longitud N pero uno tiene mayor número de operadores y operandos únicos, que naturalmente lo hacen más difícil de entender y mantener, este tendrá un mayor volumen. La fórmula es la siguiente:

$$\mathbf{volumen\ V = N\ x\ log2\,(n)}$$

donde $\mathbf{n = n1 + n2}$

El esfuerzo es otra medida estudiada por Halstead que ofrece una medida del trabajo requerido para desarrollar un programa. Desde el punto de vista del mantenimiento, el esfuerzo se puede interpretar como una medida del trabajo requerido para comprender un software ya desarrollado.

La fórmula es la siguiente:
esfuerzo

$$\mathbf{E} = \frac{\mathbf{V}}{\mathbf{L}}$$

donde el volumen V es dividido por el nivel del lenguaje L. Éste indica si se está utilizando un lenguaje de alto o bajo nivel. Por ejemplo, una simple llamada a un procedimiento podría tener un valor L de 1; el COBOL podría tener 0,1 y el ensamblador podría tener un L de 0,01. Así pues el esfuerzo aumenta proporcionalmente con el volumen, pero decrece con la utilización de lenguajes de alto nivel.

Atendiendo a varios estudios empíricos, el esfuerzo, E, es incluso una medida mejor de la entendibilidad (comprensión) que N.

2.1 Ejemplo para calcular las medidas de Halstead

Calcular las medidas de Halstead de longitud y esfuerzo para el siguiente algoritmo:

```
{

  for (i=2;i<=n;i++)

    for (j=1;j<=i;j++)

      if (x[i] < x[j])

      {

        aux = x[i];

        x[i] = x[j];

        x[j] = aux;

      }

}
```

Para realizar los cálculos de longitud y el volumen, vamos a realizar antes otros cálculos:
Operadores:
{..} → 2
for(;;) → 2
= → 5
if → 1
; → 3
(..) → 1
< → 1
< = → 2
++ → 2
[] → 4
El número total de operadores (n1) son 10 y la cantidad de operadores (N1) que hay son 23.
Operandos:
i → 7

n → 1
j → 6
x → 6
aux → 2
El número total de operandos (n2) son 5 y la cantidad total (N2) son 22.
Por tanto, la longitud es

$$\mathbf{N = N1 + N2}$$

N = 23 + 22 = 45
El volumen es

$$\mathbf{V = N \text{ x } log2\,(n)}$$

V = 45 x log2 (10+5)= 175.8