PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Reporte Técnico RT 08-12

CFGViewer: Visualizador Gráfico del Flujo de Control

Diego Vallespir Luis Costa Pablo Falero Federico Tejera

Setiembre 2008

CFGViewer: Visualizador Gráfico del Flujo de Control
Vallespir, D., Costa, L.., Falero, P., Tejera, F.
ISSN 0797-6410
Reporte Técnico RT 08-12
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República

Montevideo, Uruguay, 2008

CFGViewer
Visualizador Gráfico del Flujo de Control

Diego Vallespir
Laboratorio Cero Defectos
Grupo de Ingeniería de Software
Instituto de Computación
dvallesp@fing.edu.uy

Luis Costa
Instituto de Computación
luis.catu@gmail.com

Pablo Falero
Instituto de Computación
pfalero@gmail.com

Federico Tejera
Instituto de Computación
ftejera@gmail.com

Setiembre 2008

Abstract

En este trabajo se presenta una herramienta CASE que desarrollamos en el Instituto de Computación de la Facultad de Ingeniería del Uruguay. La misma presenta de forma gráfica el grafo de flujo de control de métodos escritos en el lenguaje Java. Además, se comunica con una herramienta de cubrimiento de código de forma de obtener los distintos caminos ejecutados por un conjunto de casos de prueba y presentarlos gráficamente en el grafo de flujo de control. Estas herramientas, funcionando en conjunto, ayudan al programador durante las pruebas unitarias de caja blanca.

Palabras Clave: *Ingeniería de Software, Lenguajes, Modelos y Ambientes de Programación, Cubrimiento de Código, Pruebas Unitarias.*

Universidad de la República
Facultad de Ingeniería
Instituto de Computación
Grupo de Ingeniería de Software

Technical Report
InCo-Pedeciba

Índice

1. Introducción

En los últimos tiempos los desarrolladores de software usan Ambientes Integrados de Desarrollo (IDE) para la construcción de aplicaciones. Estos están formados por un conjunto de herramientas que ayudan a la construcción de software. Estas herramientas se conocen como Herramientas de Ayuda por Computadora a la Ingeniería de Software (CASE *Tools*). Existen muchas herramientas CASE para ayudar en la fase de programación. Esto se debe a que en dicha fase se suelen introducir muchos defectos. En este trabajo se presenta una herramienta CASE para ser usada durante esa fase.

El grafo de flujo de control (CFG) de un programa es un grafo dirigido que describe los distintos caminos lógicos que puede ejecutar un programa. Está constituido por un conjunto de nodos conectados por aristas que indican la dirección del flujo. Se representan dos tipos de nodos: los bloques de sentencias y las bifurcaciones. Normalmente, las bifurcaciones (sentencias de control) se representan con rombos y los bloques de sentencias que no son de control se representan con rectángulos.

El CFG puede ser de utilidad para los programadores en dos contextos diferentes:

1. **Durante la programación.** Permite conocer si la lógica de un programa es sencilla o compleja. Una medida de complejidad asociada al CFG es, por ejemplo, la complejidad ciclomática [4].

2. **Durante las pruebas unitarias.** El CFG puede ser de ayuda al momento de desarrollar casos de prueba de caja blanca.

Este grafo es costoso de construir de forma manual para cada método o programa que se desarrolla. Por lo tanto, es interesante que el CFG se construya de forma automática y que el mismo sea integrado como otra herramienta al IDE utilizado. En este trabajo se presenta la herramienta **CFGViewer** que desarrollamos en el Instituto de Computación de la Facultad de Ingeniería del Uruguay. Esta herramienta presenta de forma gráfica el CFG de métodos escritos en el lenguaje Java. También desarrollamos un plug-in para el IDE Eclipse [3, 1].

En el Instituto de Computación de la Facultad de Ingeniería del Uruguay se desarrolló una herramienta, llamada **Paraguas**, que analiza los siguientes tres tipos de cubrimiento de código: cubrimiento de métodos, cubrimiento de sentencias y cubrimiento de caminos paraguas [5]. CFGViewer utiliza una API que brinda Paraguas para obtener los caminos ejecutados por un conjunto de casos de prueba. Los datos de los caminos se presentan luego de forma gráfica en el CFG. De esta forma es que CFGViewer sirve de apoyo durante las pruebas unitarias de caja blanca.

En resumen, CFGViewer permite al programador visualizar rápidamente el CFG de cualquier método y conocer el cubrimiento de código alcanzado, según el criterio de cubrimiento de caminos paraguas, al ejecutar un conjunto de casos de prueba. Además, la herramienta provee una librería para que pueda ser integrada a cualquier IDE, en particular, se desarrolla y prueba un *plug-in* para Eclipse.

En la sección 2 se presenta un ejemplo que es usado para describir las funcionalidades de la herramienta CFGViewer. En la sección 3 se presentan las funcionalidades más relevantes de CFGViewer. Por último, en la sección 4 se presentan las conclusiones y el trabajo a futuro.

2. Un Ejemplo Sencillo

En esta sección se presenta un ejemplo sencillo, que se usa en las secciones siguientes, para mostrar las funcionalidades de CFGViewer en combinación con Paraguas. Este ejemplo simula la lectura de un archivo. El archivo está representado por un arreglo de *Strings*.

```
String[]
```

Cada uno de los String del arreglo representa una línea del archivo. La siguiente es una sentencia Java que construye uno de estos archivos con dos líneas. La primer línea del archivo es *Primera línea*. La segunda línea del archivo es *Esta es 2 línea*.

```
String[] archivo = {"Primera línea","Esta es 2 línea"};
```

El método constructor de la clase que simula la lectura de archivos recibe un archivo y un carácter separador de datos. Se considera que cada línea del archivo contiene múltiples datos que están separados por una única aparición del carácter separador.

```
public SimulacionLecturaArchivo(char separa, String[] arch)
```

La clase *SimulacionLecturaArchivo* tiene un método que lee de a una línea del archivo y devuelve un array de Strings. Cada uno de los String del array se corresponden con un dato de la línea leída. Dicho método tiene la siguiente firma:

```
public String[] readLine()
```

Cada vez que es invocado lee una línea del archivo y cambia el puntero de lectura a la línea siguiente. Cuando no hay más líneas en el archivo el método devuelve `null`. En la figura 1 se presenta el archivo de ejemplo que se mencionó anteriormente y tres invocaciones sucesivas al método *readLine*. El carácter separador pasado a la constructora de la clase *SimulacionLecturaArchivo* es el "espacio en blanco".

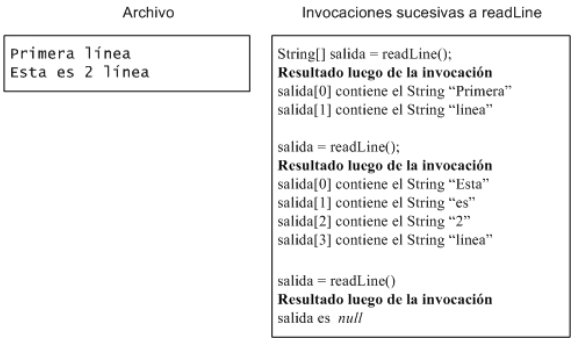


Figura 1: Archivo y Lecturas con *readLine*

A continuación se presenta el código de la clase SimulacionLecturaArchivo.

```
public class SimulacionLecturaArchivo {
    private char separador;
    private String[] archivo;
    int indexArchivo = 0;

    public SimulacionLecturaArchivo(char separa, String[] arch){
        separador = separa;
        archivo = arch;
    }

    public String[] readLine() {
        if (indexArchivo == archivo.length){
            return null;
        }
        String linea = archivo[indexArchivo];
        ArrayList<String> arrayLinea = new ArrayList<String>();
```

```
        int indexSeparador = linea.indexOf(separador);
        int index = 0;
        while (indexSeparador != -1){
            arrayLinea.add( linea.substring(index, indexSeparador) );
            index = indexSeparador + 1;
            indexSeparador = linea.indexOf(separador, index);
        }
        if (index < linea.length()){
            arrayLinea.add( linea.substring(index, linea.length()) );
        }
        indexArchivo++;
        String[] aux = new String[1];
        return arrayLinea.toArray(aux);
    }
}
```

3. Funcionalidad y Características de CFGViewer

La herramienta CFGViewer permite:

- Visualizar el CFG de métodos escritos en lenguaje Java.
- Comunicarse con la herramienta Paraguas para:
  - Instrumentar código fuente.
  - Ejecutar un conjunto de casos de prueba contra código instrumentado.
  - Visualizar el cubrimiento alcanzado según el criterio de cubrimiento de caminos paraguas.

En la figura 2 se muestra una posible configuración de Eclipse para usar CFGViewer. En dicha figura se muestran las vistas “Package Explorer”, el editor de código Java y las vistas provistas por la herramienta CFGViewer: “CFG View” y “Cover View”.

CFG View es la encargada de presentar el CFG del método que se seleccione. En la figura 2 se muestra la selección del método *readLine* en la vista Package Explorer, el código fuente de dicho método en la vista del editor y el CFG asociado al método en la vista CFG View. En esta última vista se puede ver que algunas de las aristas, que tienen como nodo origen un rombo, están etiquetadas. Los rombos representan sentencias de control. En el ejemplo están representadas con un rombo dos sentencias *if* y una sentencia *while*. Las aristas etiquetadas que salen de estos rombos muestran hacia dónde fluye el flujo de control. Esto equivale a tener aristas etiquetadas para cuando la evaluación de la decisión es *true* y para cuando es *false*. Se etiqueta la mínima cantidad de aristas posible de forma de entender el flujo.

Cover View es la vista encargada de presentar el cubrimiento de código logrado en cierta ejecución de un conjunto de casos de prueba. En la figura 2 se muestra que existen siete caminos paraguas, para el método seleccionado, en este caso *readLine*, y que dos de ellos han sido cubiertos al ejecutar un conjunto de casos de prueba.

3.1. Representación del CFG

Para obtener el CFG de un método simplemente se selecciona dicho método en la vista Package Explorer del Eclipse. La vista CFG View se refresca automáticamente mostrando el CFG del método seleccionado. La figura 3a presenta el CFG del método *readLine* de la forma que lo presenta originalmente CFGViewer.

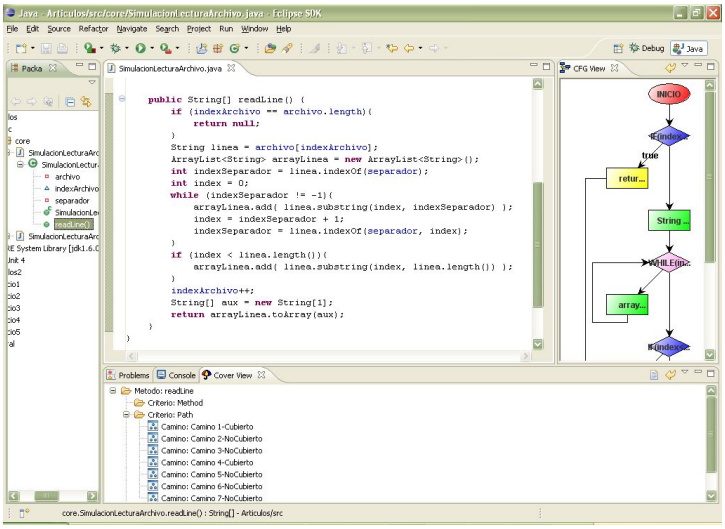


Figura 2: Posible Configuración de Eclipse con CFGViewer

Al posicionar el *ratón* sobre un elemento del CFG se despliega una etiqueta con texto. La etiqueta contiene el código que representa el elemento. En la figura 3b se presenta la etiqueta que se despliega si se está parado con el *ratón* en el nodo que contiene el texto "String...".

Si bien la herramienta tiene una representación lo más adecuada posible del CFG el programador tiene la posibilidad de mover los distintos elementos (nodos y aristas). En la figura 3c se presenta el CFG del método *readLine* pero con sus elementos ordenados de forma diferente. Esto se realiza simplemente "arrastrando" los elementos con el *ratón*.

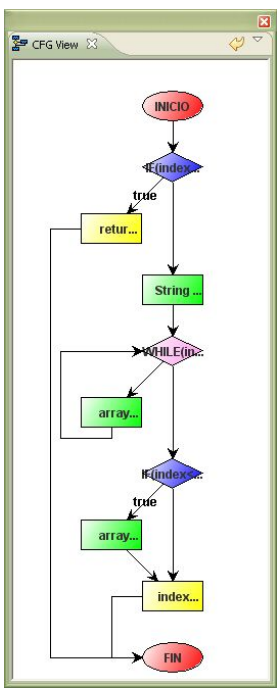
3.2. Instrumentación

CFGViewer presenta una interfaz para la instrumentación de código que provee Paraguas. Instrumentar código refiere a poner *marcas* en el mismo. Estas, al momento de ejecutar un conjunto de casos de prueba (CCP), brindan información acerca de qué se ha ejecutado respecto a un criterio de cubrimiento preestablecido. Por ejemplo, si se considera el criterio de cubrimientos de sentencias, el código instrumentado debe brindar información sobre cuáles sentencias fueron ejecutadas y cuáles no lo fueron. La interfaz presentada es sencilla y por razones de espacio no se presenta una figura. Lo que se hace a través de CFGViewer es pedirle a Paraguas que instrumente un proyecto de Eclipse; es decir, se instrumentan todas las clases Java que están en el proyecto seleccionado.

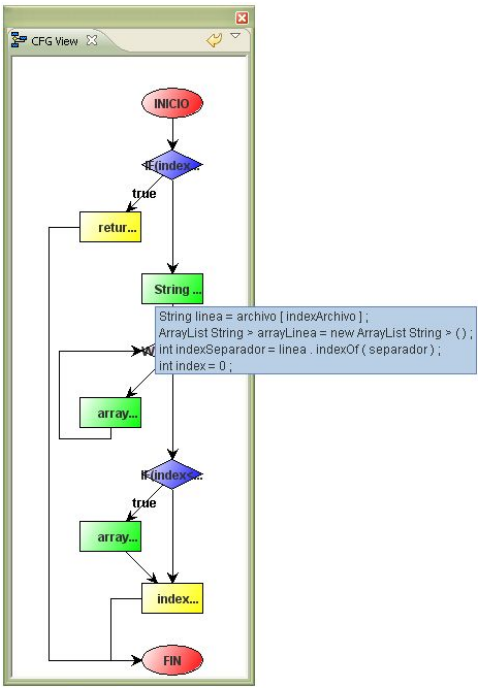
Paraguas soporta tres tipos de cubrimiento de código: de métodos, de sentencias y de caminos paraguas. Desde el punto de vista de CFGViewer lo más interesante es el cubrimiento de caminos paraguas ya que este se puede representar usando el CFG. Por esto, en este trabajo sólo se presenta dicho cubrimiento.

Los caminos paraguas son todos los caminos ejecutables que no ingresan más de dos veces en los bucles del CFG. Para cumplir con el 100 % del criterio de cubrimiento paraguas se debe contar con un CCP que logre ejecutar al menos una vez cada uno de los caminos paraguas. Detalles de este criterio de cubrimiento y la herramienta Paraguas se presentan en [5].

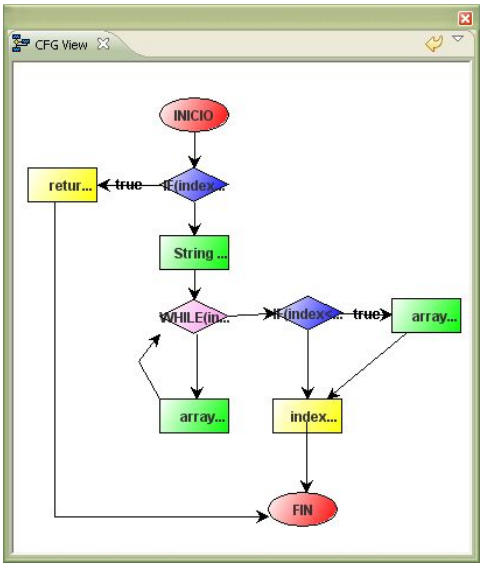
Luego que el código es instrumentado, la herramienta Paraguas guarda información acerca de cuáles son los caminos paraguas. Estos caminos se generan para cada uno de los métodos de las clases del proyecto instrumentado.



(a) CFG Original



(b) CFG con Etiqueta



(c) CFG Movido

Figura 3: Funcionalidades de Visualización del CFG

CFGViewer consulta esta información y presenta los caminos paraguas en Eclipse. Para visualizarlos simplemente basta con seleccionar un método en la vista “Package Explorer”. Luego de seleccionado se despliegan los caminos paraguas en la vista “Cover View”. La forma sugerida de configurar esta vista se presentó en la figura 2. En la figura 4(a) se presentan los caminos paraguas para el método *readLine*. Estos caminos figuran todos como *no cubierto* debido a que aún no se ha ejecutado ningún CCP.

Cada uno de los caminos paraguas se puede visualizar en el CFG. Para esto simplemente hay que seleccionar el camino en la vista “Cover View” y el mismo se despliega en la vista “CFG View”. Los caminos 1, 2, 5 y 7 se presentan en la figura 4(b-e).

El camino 1 hace *false* el primer *if* y el método termina. El camino 2 ejecuta dos veces el bloque dentro del *while* y luego hace *true* el segundo *if*. El camino 3 es similar pero hace *false* el segundo *if*. Los caminos 4 y 5 son similares a los caminos 2 y 3 pero el bloque dentro del *while* sólo se ejecuta una vez. Los caminos 6 y 7 hacen *true* el primer *if* y *false* el *while*, luego, se comportan diferente en el último *if*. Como se puede ver estos son todos los caminos que ejecutan como máximo dos veces el cuerpo del *while*; por definición son todos los caminos paraguas.

3.3. Ejecución de Pruebas

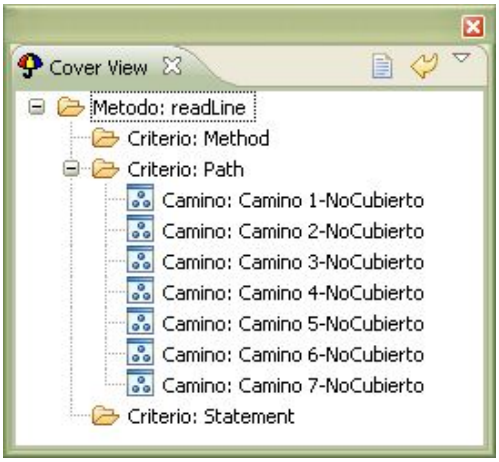
Paraguas acepta la ejecución de pruebas tanto con JUnit como ejecutadas desde un método *main*. CFGViewer provee interfaces para ambas formas de ejecución de código instrumentado.

Luego de ejecutar un CCP, CFGViewer presenta los datos de Paraguas acerca de los caminos paraguas cubiertos. Es decir, aquellos caminos, entre los caminos paraguas, que han sido ejecutados por el CCP. En la vista “Cover View” aparecen los caminos marcados con la palabra *Cubierto* en caso que se hayan cubierto con el CCP.

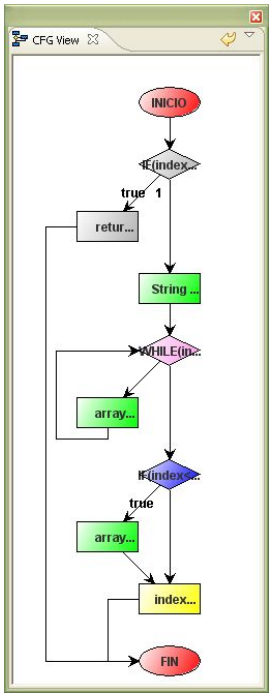
A continuación se presenta un CCP que usa JUnit para probar el método *readLine*. Se crea un archivo con 2 líneas. Con ese archivo y el espacio en blanco como carácter separador se crea un objeto de la clase *SimulacionLecturaArchivo*. Luego se invoca 3 veces a *readLine*. Al invocar la tercera vez se espera como resultado el valor *null* debido a que ya se han leído las únicas 2 líneas del archivo. Este ejemplo es el mismo que se presentó informalmente en la figura 1.

```
@Test
public void testReadLine() {
    String[] archivo = {"primera línea", "Esta es 2 línea"};
    SimulacionLecturaArchivo sla =
                new SimulacionLecturaArchivo (' ', archivo);
    String[] linea = sla.readLine();
    assertEquals("primera", linea[0]);
    assertEquals("línea", linea[1]);
    linea = sla.readLine();
    assertEquals("Esta", linea[0]);
    assertEquals("es", linea[1]);
    assertEquals("2", linea[2]);
    assertEquals("línea", linea[3]);
    linea = sla.readLine();
    assertTrue(linea == null);
}
```

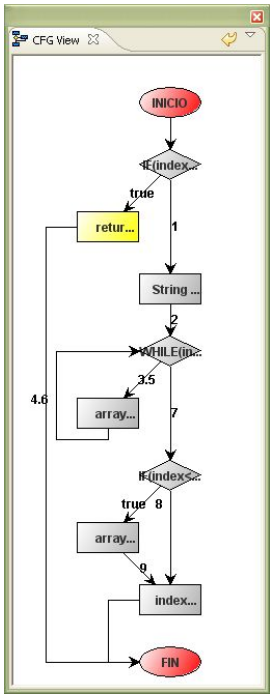
Este CCP cubre solamente 2 caminos de los 7 caminos paraguas. Estos caminos son el camino 1 y el camino 3. En la figura 5 se presenta la salida que despliega CFGViewer en “Cover View” luego de la ejecución del CCP.



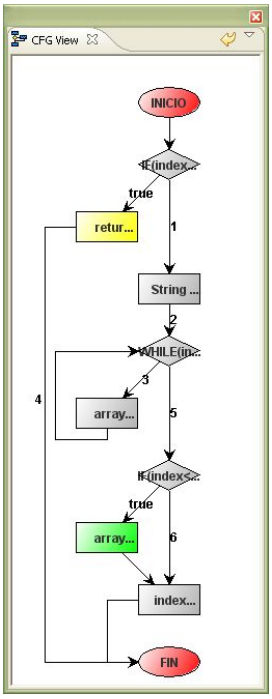
(a) Caminos



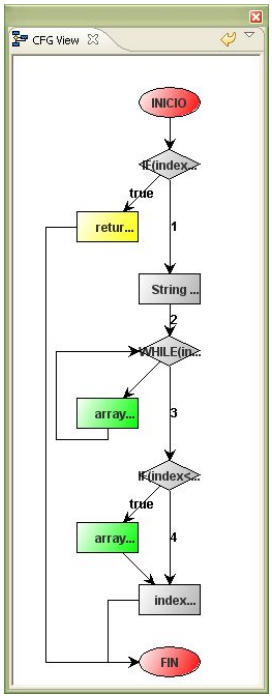
(b) Camino 1



(c) Camino 2



(d) Camino 5



(e) Camino 7

Figura 4: Caminos Paraguas de *readLine*

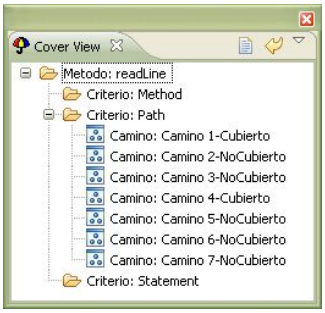


Figura 5: Caminos Cubiertos por el CCP

3.4. Usos de CFGViewer-Paraguas

Algunos de los usos de estas herramientas funcionando en conjunto son los siguientes:

- El CFG de un método muestra la complejidad lógica del mismo. Esto puede ayudar al programador ya que rápidamente puede notar una lógica demasiado compleja y optar por mejorar el método.
- A partir del gráfico se pueden hacer recorridas de código guiadas por el CFG. Estas se pueden apoyar también en las etiquetas de texto que muestra CFGViewer.
- El contar con los caminos paraguas cubiertos y los que no da posibilidades de realizar pruebas de caja blanca muy exigentes. El programador puede construir su CCP de caja negra de forma de tener cubierta la funcionalidad del método. Luego, ejecuta dicho conjunto y analiza el cubrimiento alcanzado respecto al criterio caminos paraguas. Por último, aumenta su CCP de forma de cumplir con el criterio.

4. Conclusiones y Trabajo a Futuro

Se presenta una herramienta que permite visualizar el CFG de un método de una clase desarrollada en lenguaje Java. Se muestra dicha herramienta funcionando como *plug-in* de Eclipse y en comunicación con la herramienta Paraguas para obtener cubrimientos de código. Ambas herramientas se encuentran disponibles para bajar en una página web.[1]

Es importante indagar acerca del rendimiento del criterio paraguas respecto a la cantidad de defectos que encuentra. También es necesario conocer el costo asociado a cumplir 100 % con este criterio. Para esto se pretenden hacer experimentos formales en Ingeniería de Software. Uno de estos experimentos pretende comparar el rendimiento del Personal Software Process (PSP) [2] con y sin el uso obligatorio del criterio de cubrimiento caminos paraguas. Esto se prentende realizar durante 2009. También parece interesante el uso de CFGViewer como apoyo a las recorridas de código. Este trabajo aún no está planificado.

Referencias

[1] E. Clayberg and D. Rubel. *Eclipse: Building Commercial-Quality Plug-ins*. Addison-Wesley Professional, 2 edition, 2006. 1

[2] W. S. Humphrey. *A Discipline for Software Engineering*. Addison-Wesley, 1995. 4

[1] http://www.fing.edu.uy/~dvallesp/wiki/field.php?n=Research.Software

[3] J. McAffer and J.-M. Lemieux. *Eclipse Rich Client Platform: Designing, Coding, and Packaging Java Applications*. Addison-Wesley Professional, 2005. 1

[4] T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4), Diciembre 1976. 1

[5] S. Pastorino, J. I. Costa, and L. Capillera. Paraguas: Herramienta de cubrimiento de código para caminos. In *Proyecto de Grado*. Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Julio 2008. 1, 3.2