Memoria del 6to. Congreso Internacional de Investigación
en Ciencias Computacionales CIICC'99
22 a 24 de Septiembre de 1999, Cancún, México

Propuesta de Métricas Para Controlar la Mantenibilidad de los Entornos de Desarrollo de Cuarta Generación[1]

ANTONIO MARTÍNEZ[+1], MARIO PIATTINI[+]
[+]GRUPO ALARCOS
DEPARTAMENTO DE INFORMÁTICA
UNIVERSIDAD DE CASTILLA-LA MANCHA
RONDA DE CALATRAVA, 5
13071, CIUDAD REAL (ESPAÑA)
E-MAIL: {amartinez, mpiattin} @inf-cr.uclm.es

[1]EXCMA. DIPUTACION DE CIUDAD REAL
CALLE TOLEDO, 17
13001, CIUDAD REAL (ESPAÑA)

Resumen:

En la actualidad, los entornos de cuarta generación sustituyen cada vez más a los lenguajes de tercera generación como plataforma de desarrollo habitual de sistemas informáticos, por lo que se hace imprescindible controlar su complejidad y mantenibilidad. Una forma de realizar este control es mediante la utilización de métricas específicas para estos entornos, campo poco analizado dentro de la ingeniería del software.

Las métricas tradicionales como (tamaño, complejidad, etc.), fueron concebidas para ser aplicadas a los lenguajes de tercera generación, los cuales son más homogéneos que los lenguajes de cuarta generación. Debido a la heterogeneidad de los sistemas que constituyen los entornos de cuarta generación, se hace difícil aplicar métricas tradicionales a estos entornos.

Este artículo propone una clasificación de los lenguajes de cuarta generación en diversos sub-lenguajes, facilitando así la aplicación de distintas métricas.

Palabras Clave: Ingeniería del software empírica, Métricas, Lenguaje SQL

1 Este trabajo forma parte del proyecto MANTICA, parcialmente financiado por la CICYT y la Unión Europea (1FD97-0168).

1. Introducción

Muchas organizaciones que utilizan sistemas de información basados en lenguajes de tercera generación, como COBOL, están evolucionando sus sistemas a entornos de cuarta generación, que son más efectivos y utilizan técnicas modernas facilitando el mantenimiento. Estas herramientas han recibido varios nombres, entre otros, entornos de cuarta generación, lenguajes de cuarta generación (4GLs), generadores de aplicaciones, o sistemas de cuarta generación (4GSs), [9].

Debido a su creciente difusión, se hace cada día más imprescindible contar un conjunto de métricas que permita controlar la calidad de los sistemas desarrollados con este tipo de entornos; especialmente la mantenibilidad, que representa el mayor problema del desarrollo software ya que supone entre el 60 y el 80 por ciento de los costes del ciclo de vida [4], [15]. Las métricas del software son un buen medio para entender, monitorizar, controlar, predecir y probar el desarrollo software y los proyectos de mantenimiento [1] y también pueden ser utilizadas para que los profesionales e investigadores puedan tomar mejores decisiones [14].

Los ingenieros de software han propuesto grandes cantidades de métricas para productos, procesos y recursos software [13], [7]. Sin embargo, desde que McCabe propusiera su complejidad ciclomática [12]. Hasta ahora, la gran mayoría de las métricas estaban enfocadas a programas desarrollados en lenguajes de tercera generación, dejando a un lado las bases de datos y los entornos de cuarta generación.

Tres son los factores que influyen en la mantenibilidad: entendibilidad, modificabilidad y testeabilidad, los cuales a su vez están influenciados por la complejidad [9]. Sin embargo, una métrica general para medir la complejidad es como el "santo grial" [6]. Henderson-Sellers divide la complejidad en tres: computacional, psicológica y representacional, y para la psicológica define tres componentes: complejidad del problema, factores cognitivos humanos y complejidad del producto [8].

En este artículo se identifican distintos sub-lenguajes dentro de un lenguaje de cuarta generación, facilitando así la aplicación de métricas, que se ve dificultada debido a la heterogeneidad de estos entornos. También se presenta la adaptación de métricas tradicionales para medir el tamaño y la complejidad del sub-lenguaje procedimental y se definen tres métricas para el sub-lenguaje de manipulación de base de datos

En la sección 2 introducimos una caracterización de los entornos de cuarta generación, para realizar en la sección 3 una identificación de los sub-lenguajes. En la sección 4 describimos las métricas propuestas y, para terminar , en la sección 5 presentamos las conclusiones y las futuras líneas de investigación.

2. Caracterización de los Entornos de Cuarta Generación (4GL)

El término 4GL se utiliza para describir un paquete software, que permite simplificar el desarrollo de aplicaciones software.

Tradicionalmente los 4GL se clasifican en dos grandes grupos:

- Lenguajes para usuario final, lo que se ha denominado centro de información o infocentro (information center) que nacieron con la idea de independizar a los usuarios del control del departamento de informática. Se centran sobre todo en la facilidad de uso y

flexibilidad.

- Lenguajes para el informático (analista/programador), es decir para el centro de desarrollo (development center), cuyo fin es facilitar el desarrollo de aplicaciones sofisticadas, y permitir la construcción rápida de aplicaciones por medio del prototipado.

Nosotros nos centramos en esta segunda clase de lenguajes para el posterior desarrollo de este artículo.

El concepto de 4GL nos permite ver que si realizamos un análisis de las aplicaciones que se están utilizando actualmente, un gran número de ellas realizan un número pequeño de operaciones sobre ficheros. Por ejemplo, una operación normal es leer un registro de un fichero, aplicar algún criterio simple de comparación para determinar si hacer algún cambio y por tanto reescribirlo o para imprimirlo.

Desde este análisis se detecta un número de operaciones repetidas. Los diseñadores de la mayoría de los productos 4GL han realizado plantillas para estas operaciones. Plantillas que tienen espacios libres en las cuales pueden insertarse partes especificas de información, tal como el nombre del fichero a procesar.

Los 4GL adaptan mejor sus prestaciones para aplicaciones donde se hacen relativamente procesos simples con gran cantidad de datos, sin embargo para aplicaciones donde se hacen procesos con pequeñas cantidades de datos decae su eficiencia. El 4GL usa interacción implícita para superar los problemas de procesar ficheros, relevando a los programadores de la tarea de los posibles errores que se puedan producir en controlar la secuencia de leer y escribir durante las fusiones y actualizaciones de ficheros. Similarmente muchas de las decisiones acerca de la necesidad de definiciones de datos en un programa COBOL están también hechos a medida por el uso de mecanismos típicos implícitos.

La consecuencia de coger los 4GLs esta bajo la responsabilidad para generar muchas de estas características de los programas es que el rango de estas está limitado por las facilidades dadas por el suministrador en sus plantillas. Sin embargo, la mayoría de los suministradores superan esta limitación por la provisión de una interfaz de lenguaje procedimental. Esto permite al programador las características del código en un lenguaje convencional no representable con las facilidades de los 4GL.

Una de las características de los 4GLs es la facilidad que tienen los usuarios para el diálogo y obtener la información necesaria para completar las plantillas. La mayoría de las técnicas estándar usadas en diseñar diálogos con el computador son usadas por 4GL. Entre los estilos más populares están "*form-filling*", cuestiones y preguntas y menú de selección. Hay también un pequeño grupo de 4GLs que ha desarrollado plantillas de lenguaje procedimental de muy alto nivel.

Las aplicaciones creadas con 4GL hace que no exista aislamiento desde otro software y datos en un sistema. Consecuentemente, es necesario permitir el intercambio de datos. Por ejemplo, con un 4GL, es muy fácil recuperar un conjunto de datos para una operación de negocios desde una base de datos corporativa. Sin embargo, para realizar un completo examen de los datos sería útil permitir la transferencia de los datos a una hoja de cálculo. Esto puede hacerse usando 4GL y crear un fichero en un formato legible para que se lea en la hoja de cálculo. Una aproximación similar puede hacerse transfiriendo una lista de nombres, direcciones y otros datos recuperados desde la base de datos a un procesador de texto para formar parte de un conjunto de cartas para ser enviadas a los clientes.

Finalmente, uno de los aspectos más importantes para desarrollar aplicaciones con

4GLs, es que el paquete 4GL construye automáticamente aplicaciones que cubre necesidades de los usuarios, así como la documentación referida a la aplicación al mismo tiempo. Esto asegura que el mantenimiento sea más fácil y rápido de realizar.

3. Sub-lenguajes de un Entorno de Cuarta Generación

El creciente desarrollo de aplicaciones en 4GL hace necesario un control del mantenimiento del código escrito en 4GL. El mantenimiento se considera por influir en el entendimiento, modificabilidad y testeabilidad; que depende del tamaño, longitud y complejidad de los lenguajes de cuarta generación.

Es una realidad que los lenguajes de cuarta generación 4GL son heterogéneos, porque incluyen sentencias de diferentes naturaleza. El resultado es que no hay una definición establecida de las líneas de código en un 4GL [6].

Nosotros proponemos estos sub-lenguajes:

A) Sentencias 4GL
   A1) Sentencias de control procedural
   A2) Sentencias de control Visual
   A3) Sentencias de manejo de excepciones

B) Sentencias SQL
   B1) Sentencias de definición de base de datos
   B2) Sentencias de manipulación de base de datos
   B3) Sentencias de control de seguridad
   B4) Sentencias de control de transacciones.

A continuación definimos cada uno de los sub-lenguajes con ejemplos en 4GL (CA-OpenIngres, Visual Basic, Delphi).

3.1. Sublenguajes 4GL

3.1.1. Sentencias de control procedimental

Este sub-lenguaje contiene las sentencias de construcción de bloques (if, do while, case, do until, etc). Ver figuras 1, 2 y 3.

```
If status = 'n' then
        If empsum = 0 then
                Message 'Por Favor entre el número de empleado';
                Sleep 3;
        Else
                Callframe NewEmsp
        Endif;
Endif;
```

Fig. 1. Sentencias de control procedimental de CA-OpenIngres

```
If Isnull(x) and Isnull(y) then
        Z = null
Else
        Z = 0
End If
```

Fig. 2. Sentencias de control procedimental de Visual Basic

```
If x > 1.5 then
        Begin
        k := k + 1;
        z := x + y;
        end;
else
        begin
        z := 1.5;
        r := calcula_suma (x,y,z);
        end;
```

Fig. 3. Sentencias de control procedimental de Delphi

3.1.2. Sentencias de Control Visual

Este sub-lenguaje contiene las sentencias de visualización y sentencias de control de pantalla para la manipulación de campos de pantalla y variables locales.
Ver figuras 4,5 y 6.

```
Set_forms frs (activate (nextfield) = 1,
        activate (previousfield) = 1,
            activate (Keys) = 1);
```

Fig. 4. Sentencia de control visual CA-OpenIngres.

```
MsgBox ("Esta Aquí?", MB_SÍNO + MB_ICONOALTO)
```

Fig. 5. Sentencia de control visual de Visual Basic

```
MessageDlgPos ('Estas Aquí?', mtConfirmation, mbYesNoConcel, 0, 200, 200);
```

Fig. 6. Sentencias de control visual de Delphi.

3.1.3. Sentencias de Manejo de Excepciones

Este sub-lenguaje contiene las sentencias y funciones que obtienen informacion acerca de la aplicación que se está ejecutando. Ver Figura 7,8 y 9.

```
Status = callproc inquire_file (handle = fileno, filename = byref (filename),
                                Offset = byref (offsetno));
```

Fig. 7. Sentencias de manejo de excepciones de CA-OpenIngres .

```
On error Goto CantReadFile
Open "\CURRENCY.TXT"
....
CantReadFile::
        MsgBox "No se ha creado el fichero CURRENCY.TXT ?"
        ...
```

Fig. 8. Sentencias de manejo de excepciones de Visual Basic.

```
Try
    Rewrite(f);
    Except On exception Do
      Begin
        ShowMessage('No es posible grabar en el fichero' + Nombre + '.');
        ErrorEscritura := True;
      End;
End;
```

Fig. 9. Sentencias de manejo de excepciones de Delphi.

3.2. Sentencias SQL

3.2.1. Sentencias de Definición de Base de Datos.

Este sub-lenguaje contiene las sentencias que pueden crear, modificar y destruir objetos de una base de datos, tales como tablas, vistas, índices y procedimientos de base de datos. Ver Figura 10.

```
Create table dept (dname char(10),
                   localizacion char(10),
                   budget money,
                   Expenses money,
                   constraint check_amount check
                              (budget > 0 and expenses • budget));
```

Fig. 10. Sentencias de definición de bases de datos de CA-OpenIngres

3.2.2. Sentencias de Manipulación de Base de Datos

Este sub-lenguaje contiene las sentencias que permiten manipular los datos en las tablas. Ver Figura 11.

```
Select enombre from empleados
Where edept isnull and hiredate = date ('today')
```

Fig. 11. Sentencias de manipulación de bases de datos de CA-OpenIngres

3.2.3. Sentencias de Control de Seguridad

Este sub-lenguaje contiene las sentencias que permiten el control de acceso para objetos de base de datos, reglas y recursos de DBMS. Ver Figura 12.

```
Grant select, update (depart) on table employees
to acconting_supervisor with grant option;
```

Fig. 12. Sentencias de control de seguridad en CA-OpenIngres

3.2.4. Sentencias de Control de Transacciones

Este sub-lenguaje contiene las sentencias que garantizan las siguientes características de una transacción: atomicidad, consistencia, isolación y durabilidad.(P. Ej. Rollback).

4. Métricas para 4GL

Diferentes tipos de métricas han sido definidos para 4GL. Hasta el momento algunos trabajos han sido desarrollados para estimar el esfuerzo de desarrollo y la correlación de él con el tamaño de un programa [5], [15]. Pero es necesario trabajos para controlar la calidad de los programas 4GL.

A continuación desarrollamos las métricas del sub-lenguaje de manipulación de base de datos (particularizando para la sentencia SELECT), incluyendo la adaptación de las métricas tradicionales (líneas de código, número ciclomático de McCabe) para el sub-lenguaje de control procedimental.

4.1 Métricas para el Sub-lenguaje de Control Procedimental

En [10]damos una primera aproximación de las métricas para este sublenguaje, adaptando varias métricas clásicas como líneas de código y número ciclomático de McCabe.

4.2 Métricas para el Sub-lenguaje de Manipulación de Base de Datos

Proponemos tres diferentes métricas para este sub-lenguaje que las particularizamos para la sentencia SELECT.

Métrica NT

Expresa el número de tablas que contiene la sentencia SELECT.
Ver Figura 12.

```
Select hora from prueba where num_ficha='0959' and fecha ='181298';
```

Fig. 12. Ejemplo sentencia SELECT

Podemos caracterizar la sentencia SELECT en base al valor NT=3

Métrica NA

Número de anidamientos de la sentencia SELECT.

Ver Figura 13.

```
select num_ficha, fecha
      from prueba
      where num_ficha not in (select num_ficha
                              from prueba
                             where fecha='171298'
                               and control='SM'
                               and estado='A'
                               and hora in (select hora
                                            from prueba
                                           where fecha='171298'
                                             and control='SM'
                                             and tipo='A0'
                                           )
                             )
   and fecha>'131298'
```

Fig. 13. Ejemplo sentencia SELECT

Podemos caracterizar la sentencia SELECT en base al valor NA=3

Métrica A

Dentro de la sentencia SELECT nos da si hay agrupamiemto (A = 1) o no lo hay (A = 0).

Ver Figura 14.

```
select num_ficha, fecha, count(hora) as n_fichajes
      from prueba
      where num_ficha='0800'
   group by num_ficha, fecha
```

Fig. 14. Ejemplo sentencia SELECT

Podemos caracterizar la sentencia SELECT en base al valor A=1.

A continuación se muestra un ejemplo de la sentencia SELECT completo.

Ver Figura 15.

```
select f.p0_nom_nomi, p.num_ficha, p.fecha
    from prueba p, fper020 f, hor_personal h
    where p.nif not in (select h.nif
                        from prueba p, fper020 f, hor_personal h
                       where p.num_ficha=h.num_ficha
                         and f.p0_nif=h.nif
                         and p.nif=f.p0_nif
                         and p.fecha='171298'
                         and p.control='SM'
                         and p.estado='A'
                         and f.p0_sexo='V'
                         and p.hora in (select hora
                                        from prueba p, fper020 f,
                                             hor_personal h
                                        where p.num_ficha=h.num_ficha
                                         and f.p0_nif=h.nif
                                         and p.nif=f.p0_nif
                                         and p.fecha='171298'
                                         and p.control='SM'
                                         and p.tipo='A0'
                                         and h.saldot=0
                                       )
                      )
and p.fecha='151298'
and p.num_ficha=h.num_ficha
and p.nif=f.p0_nif
and f.p0_nif=h.nif
group by f.p0_nom_nomi, p.num_ficha, p.fecha
```

Fig. 15. Ejemplo sentencia SELECT

Podemos caracterizar la sentencia SELECT en base a los valores NT=3, NA=3 y A=1.

5. Conclusiones y Trabajos Futuros

Los entornos de cuarta generación están adquiriendo una gran importancia en algunas

instalaciones imponiéndose a los lenguajes de tercera generación gracias a sus ventajas, entre las que destaca la mayor productividad.

Pensamos que es necesario elaborar métricas para controlar la mantenibilidad de los sistemas informáticos desarrollados con 4GL, bien adaptando métricas clásicas, bien proponiendo nuevas métricas.

En este momento estamos definiendo métricas para cada uno de los sub-lenguajes identificados en un entorno de cuarta generación. Tambien estamos verificando las propiedades que cumplen estas métricas en diferentes marcos formales como los propuestos por [18], [17] [1], [2].

Como la verificación formal no es suficiente, tambien validamos estas métricas empíricamente utilizando tanto experimentos controlados con estudiantes como casos reales en diferentes organizaciones, que permiten confrontar los valores de las métricas con el coste real de mantenimiento.

En un futuro queremos abordar también la definición de métricas para entornos avanzados de cuarta generación, basadas en las propuestas de [3],[8].

6. Referencias

1. Briand, L.C., Morasca, S. and Basili, V. (1996). Property-based software engineering measurement. *IEEE Transactions on Software Engineering,* 22(1): 68-85.

2. Briand, L.C. and Morasca, S. (1997). Towards a Theoretical Framework for measuring software attributes. *Proceeding of the Fourth International Software Metrics Symposium,* 119-126.

3. Chidamber, S.R. and Kemerer, C.F. (1991). A metrics suite for object-oriented design. *IEEE Trans. on Software Engineering* 20, 6, 476-493.

4. Card, D.N. y Glass, R.L. (1990). *Measuring Software Design Quality*. Englewood Cliffs. USA.

5. Dolado, J.J. (1997). A Study of the Relationships among Albrecht and Mark II Function Points, Lines of Code 4GL and Effort. *J. systems software,* 37:161-173.

6. Fenton, N. (1994). Software Measurement: A Necessary Scientific Basis. *IEEE Transactions on Software Engineering*, 20 (3): 199-206.

7. Fenton, N.and Pfleeger, S. L. (1997). *Software Metrics: A Rigorous Approach* 2nd. edition. London, Chapman & Hall.

8. Henderson-Sellers, B. (1996). *Object-Oriented Metrics - Measures of complexity*. Prentice-Hall, Upper Saddle River, New Jersey.

9. Holloway, S. (ed.) (1990). *Fourth-Generation Systems, their scope application and methods of evaluation*. London: Chapman and Hall.

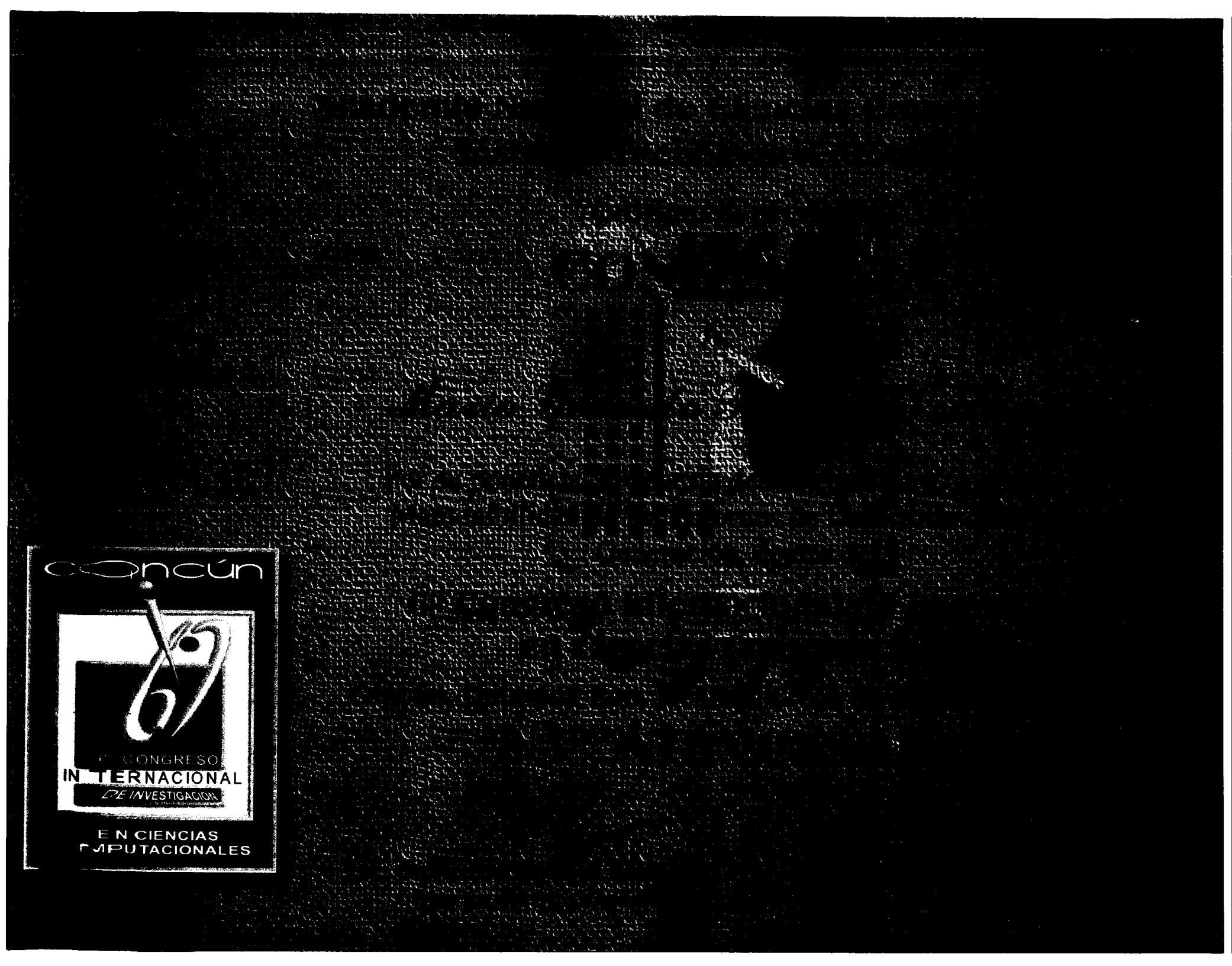
10. Li, H.F. and Cheng, W.K. (1987). An empirical study of software metrics. *IEEE Trans. on Software Engineering,* 13 (6): 679-708.

11. Martínez, A. and Piattini, M. (1998). Validation of 4GL metrics. *Proc. of the Software Measurement in Practice, 10th Anniversary Conference*. United Kingdom Software Metrics Association, Londres, octubre 1998, X 1-19.

12. McCabe, T.J. (1976). A complexity measure. *IEEE Trans. Software Engineering* 2 (5), 308-320.

13 Melton, A. (ed.) (1996). *Software Measurement*. London, International Thomson Computer

Press.
14. Pfleeger, S. L. (1997). Assessing Software Measurement. *IEEE Software*. March/April, 25-26.
15. Pigoski, T.M. (1997). *Practical Software Maintenance*. Wiley Computer Publishing. New York, USA.
16. Verner J. and Tate G.(1988) "Estimating Size and Effort in Fourth-Generation Development". *IEEE Trans. On Software Engineering*. July 15-22.
17. Weyuker, E.J. (1988). Evaluating software complexity measures. *IEEE Transactions on Software Engineering* 14(9), 1357-1365.
18. Zuse, H. (1998). *A Framework of Software Measurement*. Berlin, Walter de Gruyter.


cancún
CONGRESO
INTERNACIONAL
DE INVESTIGACION
EN CIENCIAS
COMPUTACIONALES