The United Kingdom Software Metrics Association

# *10th Anniversary Conference*

# "Software Measurement in Practice"

**Albany Conference Centre**
**Forte Posthouse Bloomsbury**
**London UK**
**29th & 30th October 1998**

United Kingdom UKSMA Software Metrics Association

"United Kingdom Software Metrics Association" is a trading name of the United Kingdom Function Point Users Group Ltd.

# UKSMA 10TH ANNIVERSARY CONFERENCE
# SOFTWARE MEASUREMENT IN PRACTICE
# CONFERENCE PROCEEDINGS

## Contents

# Validation of Fourth-Generation Language Metrics[1]

Antonio Martínez[+1], Mario Piattini[+]
[+]Grupo Alarcos
Departamento de Informática
University of Castilla-La Mancha
Ronda de Calatrava, 5
13071, Ciudad Real (Spain)
e-mail: {amartinez, mpiattin} @inf-cr.uclm.es

[1]Excma. Diputacion de Ciudad Real
Calle Toledo, 17
13001, Ciudad Real (Spain)

**Abstract**

In this paper, we propose a measurement set for the CA-OpenIngres/4GL, by adapting classical metrics (size, length, complexity, cohesion, and coupling) which can be generalized for other relational database fourth generation languages. These measures are characterized using the mathematical framework developed by Briand et al. (1996) and Morasca and Briand (1997). Note that the proposed measures and metrics have been conceived to be applied in traditional languages, which are more homogeneous than 4GL ones. These measures do not take into account the 4GL languages include sentences ( sub-languages) of different nature.

We propose a classification of the fourth generation languages in sub-languages ( procedural control sentences, visual control sentences, exception handling sentences, database objects manipulation sentences, data manipulation sentences, security control sentences, transaction control sentences). The application of these measures would provide a more accurate vision of the characteristics of the evaluated code, and we provide a validation of these measures that satisfy the properties of Briand et al. (1996).

**Keywords:** Fourth-generation languages, database measurement, CA-OpenIngres.

[1] This work is part of the MANTICA project.

## 1.- INTRODUCTION

Since the seventies software engineers have been proposing all sorts of metrics for software products, processes and resources Fenton and Pfleeger (1996). These metrics were defined in order to control some internal attributes of software elements (e.g. complexity, consistency, modularity) which influence external attributes (e.g. maintainability, testability, portability, understandably) that concern software development centres.

Classical quality models as McCall et al. (1977) proposed to consider factors (external attributes), criteria (internal attributes composing factors) and metrics (criteria measurement) in order to assess software quality. Also quality frameworks as those proposed by IEEE or ISO, and process improvement initiatives as SPICE, CMM, BOOTSTRAP, etc. demands metrics definition and control in order to improve software quality.

Unfortunately, almost all the metrics proposed since McCabe's cyclomatic number McCabe (1976) (by far the most referenced complexity metric) until now, focused on third-generation programming languages like COBOL, and few of them are applied to fourth-generation languages.

Many organizations which use management information systems are now aware that computer systems constructed using third generation languages such as COBOL can be more effectively produced and maintained using modern productivity-enhancing tools. These tools have been given various names, including fourth-generation languages (4GLs), application generators, or move recently fourth-generation systems (4GSs), Holloway (1990).

This paper proposes some metrics in order to measures database 4GL programs, defined in the framework proposed by Briand et al. (1996) and Briand and Morasca (1997). The rest of the paper is organised as follows: Section 2 characteristics of 4GLs and summaries the elements of CA-OpenIngres/4GL. Metrics proposed are presented in Section 3 . In section 4, we describe the validation of the metrics. Finally, we provide our conclusions and future works in Section 5.

## 2. - CHARACTERIZATION OF FOURTH-GENERATION LANGUAGE (4GL)

### 2.1. –CHARACTERISTICS OF 4GLS

The term 4GL is used to describe a software package which aims to simplify the process of developing software applications. The concept of the 4GL is reliant upon the fact that an analysis of applications in use today would show that a very large number of them are carrying out a small number of routine file-processing operations. For example, a common pattern is to read a record from a file, apply some comparatively simple test criteria to it to determine whether to make some change and then rewrite it or output it to a printer.

From this analysis a number of recurring patterns can be detected. What the designers of most 4GL products have done is to provide pre-written templates for these operations. Templates have gaps into which specific pieces of information, such as the name of the file to be processed can be inserted. The emphasis in a 4GL is, therefore, on what is to be done rather than how it is to be done, since the supplier has already made those decisions in creating his templates.

The 4GL is best suited to data-intensive applications where relatively simple operations are being carried out on large amounts of data rather than complex processing of small volumes of data. The 4GLs use implicit iteration to overcome the problems of file processing, by relieving programmers of the error-prone task of controlling the sequence of reads and writes during file merges and updates. Similarly many of the decisions about data definitions needed in a COBOL program are also made by the use of implicit typing mechanisms.

The consequence of the 4GL taking responsibility for many of these features of the program is that the range of programs that can be generated is necessarily limited by the facilities provided by the supplier in his templates. However, most suppliers overcome this limitation by provision of a procedural language any features not representable within the facilities of the 4GL.

One of the characteristics of 4GLs that users are most conscious of is the style of dialogue used to obtain the information needed to complete the templates. Most of the standard techniques used in designing computer dialogues are used by at least one 4GL. Among the most popular styles are form-filling, question and answer and menu selection. There are also a small group of 4GLs which have developed forms of Very High Level Procedural Language. These languages employ small numbers of powerful statements that each embody a number of assumptions about how that particular statement is to be executed. The result is a program of comparatively few statements which is claimed to be quick to write and easy to understand.

Applications created with a 4GL do not exist in isolation from other software and data in the system. Consequently, it is necessary to be able to interchange data. For example, with a 4GL it is very easy to retrieve sets of data concerning the operation of a business from a corporate database. However to carry out a fuller examination of the data it would be very useful to be able to transfer the data to a spreadsheet. This can achieved by using the 4GL to create a file in a suitable format for the spreadsheet to read. A similar approach can be taken to transferring a list of names, addresses and other data retrieved from a database to word-processor to form part of a set of letters to be sent to customers.

Finally, one of the most important aspects of applications developed using 4GLs is that the 4GL package, in building an application in response to a user's needs, also builds the documentation at the same time. The 4GL application is effectively self-documenting. This ensures that subsequent maintenance should be easier and quicker to carry out.

## 2.2. – CA-OPENINGRES

The basic architecture of the CA-OpenIngres system is divided into three facilities: the Data Manager, the query languages and the user interfaces (see figure 1).

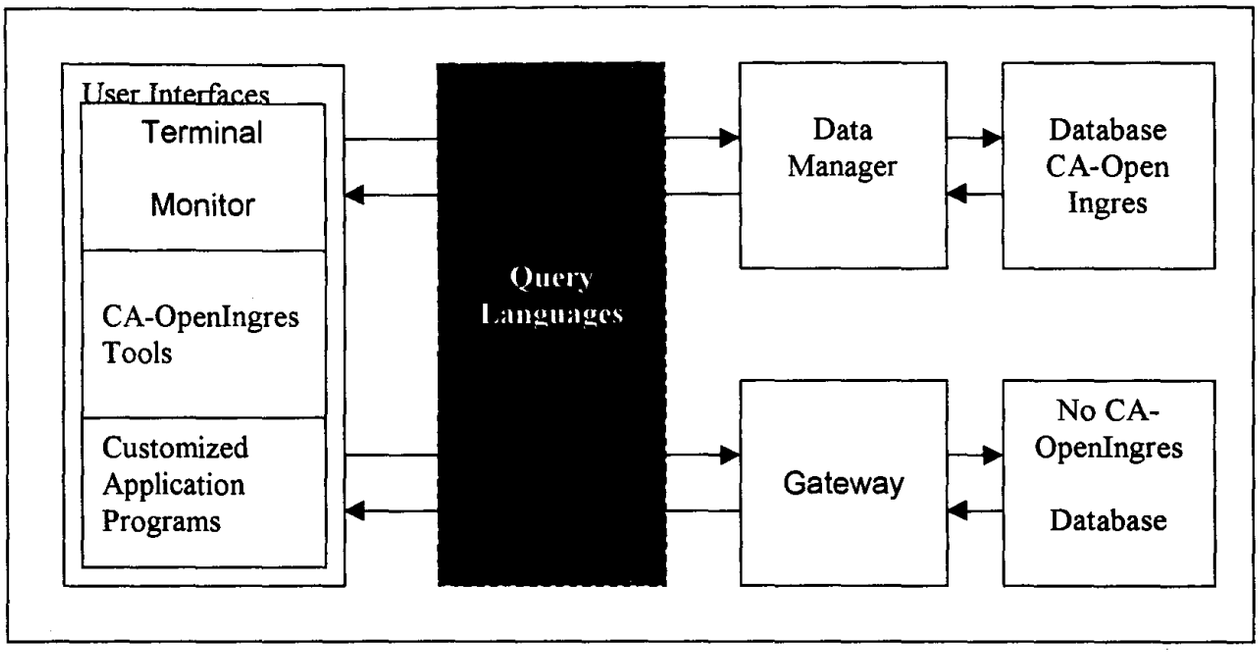



Figure 1: the basic architecture

### 2.2.1.- The data manager

The Data Manager accepts query language instructions and performs the specified operations on the database. All basic CA-OpenIngres tasks, such as data updates and retrievals, are performed by the Data Manager. However, the user never interacts directly with the Data Manager. Instead, the user must give instructions to CA-OpenIngres through one of the CA-Openingres tools.

### 2.2.2.- The query language

The query language passes instructions to the Data Manager from a user interface. There are many user interfaces that request different database tasks. One user interface, the Terminal Monitor, allows the user to enter data and instructions for the Data Manager by entering query language statements. Other user interfaces allow the user to enter data and instructions for the Data Manager by using CA-OpenIngres forms and menus. These forms-based subsystems then send the appropriate query language statements to the Data Manager. CA-OpenIngres provides these query languages: SQL (standard relational query language, Open SQL, and QUEL (the original CA-OpenIngres query language).

### 2.2.3.-The user interfaces

A user interface enables the user to give instructions to the Data Manager. The user interface accepts instructions from the end user and forwards them to the Data Manger via the query language. After the operations are performed by the Data Manager, the user interface displays the results to the end user.

**The Terminal Monitor** allows users to issue query language statements directly to the Data Manager.

**CA-OpenIngres tools**, provided by CA-OpenIngres, include QBF (Query-By-Forms), RBF (Report-By-Forms), VIFRED (Visual-Forms-Editor), and Vision. These forms-based application subsystems free the user from having to memorize the query language syntax and provide a working environment that is easy to learn and use.

**Customized Application Programs.** Programmers can create customized application programs by using the CA-OpenIngres ABF (Application-By-Forms) and 4GL (Fourth Generation Languages) tools.

**CA-OpenIngres Gateways** enable you to access data that is not stored in a CA-OpenIngres database. Using CA-OpenIngres Gateways, you can access the non-CA-OpenIngres data as if it were stored in a CA-OpenIngres database.

## 3.- PROPOSED OF METRICS 4GL

### 3.1.- Basic Definitions

Before introducing the measures for the set of concepts we intend to study, we provide basic definitions related to CA-OpenIngres/4GL.

**Applications**: An application is a sequence of related frames and procedures that solves a particular problem. It is an interface that provides the information and tools the user needs to interact with a database. An application can include the following components:

**Frames**: The basic operational units of an application. The end user interacts with the application through forms and menus defined within the frame structure.

**Procedures**: Separate modules that perform specific operations.

**Global Variables**: Variables global in scope for the application.

**Constants**: Named values global in scope for the application.

Applying with the framework proposed by Briand et al. (1996), and back generalization by Briand and Morasca (1997) CA-OpenIngres application can be defined as a modular

systems A=<E,R,M> where:

E: the set of executable sentences of CA-OpenIngres/4GL.

The 4GL languages are heterogeneous, they include sentences of different nature. We propose classification of the 4GL languages in different sentences nature (sub-languages).
In this paper we propose the following sub-languages:

A) Sentences 4GL
   - A1) Procedural control sentences
   - A2) Visual control sentences
   - A3) Exception handling sentences

B) Sentences SQL
   - B1) Database objects manipulation sentences
   - B2) Data manipulation sentences
   - B3) Security control sentences
   - B4) Transaction control sentences

R: is a binary relation on E ( $R \subseteq E^*E$), defined as the set of control flows from one statement to another.

M: is a collection of frames or procedures (modules) disjoint of A such that
A frame or procedure can be defined as a 2-tuple $f=<E_f,R_f>$ where $E_f \subseteq E$, $R_f \subseteq E_f * E_f$ and $R_f \subseteq R$.

### 3.2 Metrics

#### 3.2.1- Size

The metrics is used to measure size in CA-OpenIngres/4GL application is LOC (line of code).

*Application size (AS).* The application size (AS) is defined as the sum of the size of each frame or procedure (FS) of the application.

$$AS = \Sigma FS$$

*Frame or procedure size (FS).* The frame or procedure size (FS) is defined as the sum of the lines of code of each sub-languages.

$$FS = \Sigma LOC$$

*LOC (line of code).* The definition of LOC is controversial Fenton and Pfleeger (1996) we adopted the following definition of line of code: A line of code is any line of program text that contain executable statements finished in a semicolon excluding blank lines, comment lines, data declarations, program headers, non-executable statements and compile directives.

The statements taking more than one line count as only one line.

### 3.2.2.- Length

We use depth of nesting to measure the length of a CA-OpenIngres /4GL application. Depth of nesting of a frame or procedure can be defined using graph theory (Fenton and Pfleeger, 1996).
CA-OpenIngres frames and procedures can be modeled by flowgraphs.
We calculate the application length adding 1 to maximum of the depth of nesting of its frames and procedures.

### 3.2.3.- Complexity

*Application complexity (AC).* The application complexity AC is defined as:

$$AC=\Sigma FC$$

*Frame or procedure complexity (FC).* The frame or procedure complexity (FC) is defined as:

$$FC= W_{CSCP} * v(G) + W_{CVCS} * CVCS + W_{CEHS} * CEHS + W_{CDOS} * CDOS + W_{CDMS} * CDMS + W_{CSCS} * CSCS + W_{CTCS} * CTCS$$

Where:

$W_{CSCP}$ is complexity procedural control sentences weight
$W_{CVCS}$ is complexity visual control sentences weight
$W_{CEHS}$ is complexity exception handling sentences weight
$W_{CDOS}$ is complexity database objects manipulation sentences weight
$W_{CDMS}$ is complexity data manipulation sentences weight
$W_{CSCS}$ is complexity security control sentences weight
$W_{CTCS}$ is complexity transaction control sentences weight

In the following we provide details about the above terms.

1. Procedural control sentences

The metric used to measure the complexity in CA-OpenIngres/4GL procedural control sentences is McCabe's cyclomatic complexity (McCabe, 1976).

$$\mathbf{V(G)=|R| - |E| + 2p}$$

Where:

G: is the graph of the frame or procedure composed with the procedural control sentences
|R|: is the number of edges in the graph
|E|: is the number nodes
p: is the number of connected components of G.

2. Visual control sentences

The metric used to measure the complexity in CA-OpenIngres/4GL visual control sentences is the following:

$$CVCS = \Sigma\ NSLV$$

Where:

CVCS: complexity visual control sentences
NSLV: number of parameters of the visual control sentences

3. Exception handling sentences

The metric used to measure the complexity in CA-OpenIngres/4GL exception handling sentences is the following:

$$CEHS = \Sigma\ NSME$$

Where:

CEHS: complexity exception handling sentences
NSME: number of parameters of the exception handling sentences

4. Database objects manipulation sentences

The metric used to measure the complexity in CA-OpenIngres/4GL definition sentences is the following:

$$CDSS = \Sigma\ (\ NE + NP + \Sigma\ CC + CP)$$

Where:

$CC = (\ \Sigma\ CS + \Sigma\ CSSQL)$
$CS = CP$
CDSS: complexity of database objects manipulation sentences
NE: number of elements of the database objects manipulation sentences
NP: number of parameters of the database objects manipulation sentences
CC: complexity of the clauses of the database objects manipulation sentences SQL
CS: complexity of sentence
CSSQL: complexity of sentence SQL
CP: complexity of predicate of the database objects manipulation sentences. We calculate by Halstead's software science.

5. Data manipulation sentences

The metric used to measure the complexity in CA-OpenIngres/4GL data manipulation sentences is the following:

CDMS= Σ (NE + NP + Σ CC + CP)

Where:

CC = ( Σ CS + Σ CSSQL)

CS = CP

CDMS: complexity of data manipulation sentences

NE: number of elements of the data manipulation sentences

NP: number of parameters of the data manipulation sentences

CC: complexity of the clauses of the data manipulation sentences

CS: complexity of sentence

CSSQL: complexity of sentence SQL

CP: complexity of predicate of the data manipulation sentences. We calculate by Halstead's software science.

6. Security control sentences

The metric used to measure the complexity in CA-OpenIngres/4GL security control sentences is the following:

CTCS= Σ NCCS

Where:

CTCS: complexity security control sentences

NCCS: number of clauses of the security control sentences

7. Transaction control sentences

The metric used to measure the complexity in CA-OpenIngres/4GL transaction control sentences is the following:

CSCT= Σ NCCT

Where:

CSCT: complexity transaction control sentences

NCCT: number of clauses of the transaction control sentences

### 3.2.4.- Cohesion

Bieman and Ott(1994) weak functional cohesion is used for measuring CA-OpenIngres frame or procedure is cohesion.

$$WFC(P) = \frac{|G(SA(P))|}{|tokens(P)|}$$

Where:

$P$ is a frame or procedure

*WFC* is the weak functional cohesion

*Data slice* is the sequence of all those data tokens

*Slice abstraction* is the set of data slices

*Glue in a slice abstraction of a P, G(SA(P))* is defined as the set of data tokens may belong to more than one data slice.

*Data tokens* considered in Bieman and Ott(1994) are analogous in CA-OpenIngres/4GL(i.e. variable and constant definitions and references).

The cohesion of a CA-OpenIngres application (ACH) is measured as the total number of frames and procedures having functional cohesion divided by the total number of frames and procedures.

### 3.2.5.- Coupling

For measure coupling we are using (Fenton and Pfleeger,1996) definition. The coupling between two frames or procedures is:

$$C(f,f') = i + \frac{n}{n+1}$$

Where

*i* is the number corresponding to the "worst type of coupling" ( from 5 for content coupling to 0 no coupling)

*n* the number of interconnections between frames and procedures (global variables and formal parameters).

We take the sum of coupling of all frames and procedures of an application as the coupling of the application (ACO).

## 4.- VALIDATION

In this section we use the properties proposed by Briand et al. (1996) in order to characterise the metrics defined in the previous section.

### 4.1.- AS Metric

A size metric is characterised by the following properties (Briand et al. 1996):

1. Nonnegativity. The size of a system is nonnegative.
2. Null value. The size of a system is null if it has no elements.
3. Module additivity. The size of a system is equal to the sum of the sizes of two of its modules such that any element of the system is an element of either a module or the other.

Proving these properties for the AS metric is easy:

1. Seeing the different expression that compose the AS metric, it is impossible to obtain a negative value.
2. If we have no elements the LOC=0 and FS=0, so AS=0.
3. If we have two frames or procedures disjoint (its have no common elements), the AS will be the sum of both.

4.2.- AL Metric

For length metrics, the properties given by Briand et al. (1996) are:

1. Nonnegativity. The length of a system is nonnegative.
2. Null value. The length of a system is null if has no elements.
3. Nonincreasing monotonicity for connected components. Let S be a system and m be a module of S such that m is represented by a connected component of the graph representing S. Adding relationships between elements of S does not increase the length of S.
4. Nondecreasing monotonicity for nonconnected components. Let S be a system and m1 and m2 be two modules of S such that m1 and m2 are represented by two separate connected components of the graph representing S. Adding relationships from elements of m1 to elements of m2 does not decrease the length of S.
5. Disjoint modules. The length of a system made of two disjoint modules m1, m2 is equal to the maximum of the lengths of m1 and m2.

The demonstration of these properties for AL metric is the following:

1. The depth of a tree never can be negative.
2. If we have no attributes (E=0), we have no tree, we have no depth and AL=0.
3. If we add relationships between elements of a tree the depth does not vary. This will be only possible by adding elements to the tree.
4. If we add relationships between elements of two trees we cannot decrease the depth of the resulting tree because the depth of the first tree stays equal and we add a relation which has a depth 1 as minimum (if it's a leaf tree).
5. The depth of a tree is given by the component which has more levels from the root to the leaves.

4.3.- AC Metric

For complexity metrics, the properties given by Briand et al. (1996) are:

1. Nonnegativity. The complexity of a system is nonnegative.

2. Null value. The complexity of a system is null if it has no relations.
3. Symmetry. The complexity of a system does not depend on the convention chosen to represent the relationships between its elements.
4. Module Monotonicity. The complexity of a system is no less than the sum of the complexities of any two of its modules with no relationships in common.
5. Disjoint Module Additivity. The complexity of a system composed of two disjoints modules is equal to the sum of the complexities of the two modules.

AC metric verifies these properties because:

1. Seeing the expression that composes the AC metric, it is impossible to obtain a negative value.
2. If there is no relations then FC=0 and AC=0.
3. The definition of AC is the same disregarding the direction of the reference.
4. If the modules are no disjoint, this means that between elements of both modules there is a relation, then FC crease, so AC never decrease.
5. Every module will have a value for FC. When modules are disjoint neither a frame or procedure will be common to both modules, so the result of AC of the application will be the sum of the FC of the two modules, and so AC will be the sum of the AC of the modules.

4.4.- ACH Metric

For cohesion metric the properties given by Briand et al. (1996) are:

1. Nonnegativity and Normalization. The cohesion of a module or modular system belong to a specified interval.
2. Null value. The cohesion of a module or modular system is null if R=0 or IR=0.
3. Monotonicity. Let be two modular systems (with the same set of elements) such that there exist two modules (with the same set of elements) belonging to the modular system. Adding intramodule relationships does not decrease (module | modular system) cohesion.
4. Cohesive modules. Let be two modular systems (with the same underlying system). The two modules m1 and m2 are replaced by the module m3, union of m1 and m2. If no relationships exist between the elements belonging to m1 and m2 then the cohesion of a (module | modular system ) obtained by putting together two unrelated modules is not greater than the (maximun cohesion of the two original modules | the cohesion of the original modular system)

ACH metric verifies these properties because:

1. Seeing the expression that composes the ACH metric, it is impossible to obtain a negative value. ACH belong to a specified interval [0,max]
2. If there is no relations then R=0 or IR=0, so ACH=0.
3. If add intramodule relationships then ACH not decrease because the numerator crease.

4. If m1 and m2 are replaced by the module m3 union of m1 and m2, and no relationships exist between the elements belonging to m1 and m2 then the expression the ACH is not greater than the maximun cohesion of the two original.

4.5.- ACO Metric

For coupling metric the properties given by Briand et al. (1996) are:

1. Nonnegativity. The coupling of a module is nonnegative.
2. Null value. The coupling of a module is null if OuterR(m)=0.
3. Monotonicity. Let be two modular systems (with the same set of elements) such that there exist two modules then adding intermodule relationships does not decrease coupling.
4. Merging coupling. Let be two modular systems MS and MS' such that m1 belong to MS and m2 belong MS' and m3 not belong to MS. The two modules m1 and m2 are replaced by the module m3, whose elements and relationships are the union of those of m1 and m2., then the coupling of a [module | modular system] obtained by merging two modules is not greater than the [sum of the couplings of the two original modules | coupling of the original modular system], since the two modules may have common intermodule relationships.
5. Disjoint module additivity. Let be two modular systems MS and MS' (with the same underlying system). The two modules m1 and m2 are replaced by the module m3, union of m1 and m2. If no relationships exist between the elements belonging to m1 and m2. The coupling of a [ module |modular system] obtained by merging two unrelated modules is equal to the [ sum of the coupling of the original modules | coupling of the original modular system]

ACO metric verifies these properties because:

1. Seeing the different expression that compose the ACO metric, it is impossible to obtain a negative value.
2. If we have no relations between the modules n=0 and i=0, so C(f , f')=0 and ACO=0.
3. If we add relationships, between modules, n increase and i increase, so C(f , f') increase and ACO does not decrease.
4. The coupling of a modular system obtained by merging two modules may have common intermodule relationships is not greater than the sum of the coupling of the two original modules, because the common intermodule relationships not belong the union (Briand and Morasca, 1997).

## 5.- CONCLUSIONS AND FUTURE WORKS

We have presented a first approximation some measures for CA-OpenIngres/4GL, were proposed using the mathematical framework developed by Briand et al. (1996) and Briand and Morasca (1997). These measures can be supplemented with other metrics, however we believe they constitute an approximation for controlling application quality.

In this moment a tool is being developed for obtain these measures automatically.

We want to validate these measures confronting them with maintenance costs. Also, limits for these measures must be established in order to control application production (as programmer's guidelines).

Validation of the measures were performed in a real environment (e.g. Spanish Public central and local administrations) using empirical study.

Future research will be the definition of Objects Oriented metrics, based on those proposed by Chidamber and Kemerer (1991) and Henderson-Sellers (1996) because the next version (2.0) of CA-OpenIngres include an objects Oriented 4GL.

# Appendix A

```
/*****************************************************************/
/*                                                               */
/*   SOURCE -------> cont413.osq                                 */
/*   DATE -------> 26/Abril/96                                   */
/*   DESCRIPTION ---> Altas Servicios                            */
/*                                                               */
/*****************************************************************/
initialize ()=
declare
   mes=integer not null,
   campo=integer1 not null,
   errorIng=integer not null,
   tecla=integer not null,
   filas=integer not null,
   error=integer1 not null,
   inicia_campos=procedure returning none
begin
   set_forms frs(timeout=300);
   set_forms frs (activate(nextfield)=1,
            activate(previousfield)=1,
            activate(keys)=1);
   callproc inicia_campos();
   set_forms field " (reverse(cod_servicio)=1);
   redisplay;
   resume field cod_servicio;
end

'Ayuda', key frskey1=
begin
   callproc inicia_campos();
   if campo=1 then
            message 'Codigo del Servicio.';
            sleep 2;
     set_forms field " (reverse(cod_servicio)=1 );
            resume field cod_servicio;

   elseif campo=2 then
            message 'Nombre descriptivo del Servicio.';
            sleep 2;
     set_forms field " (reverse(nom_centro)=1 );
            resume field nom_centro;
   endif; /* campo */

end /* Ayuda */

'Grabar', key frskey4=
begin
   callproc inicia_campos();
   if (cod_servicio='' or
            nom_servicio='') then
            callproc beep();
            message 'ERROR --> NO introducidos datos necesarios';
            sleep 2;
     set_forms field " (reverse(cod_servicio)=1 );
            resume field cod_servicio;
   endif;
   insert into hor_servicios
            (cod_servicio, nom_servicio, cod_centro)
    values
            (:cod_servicio, :nom_servicio, :cod_centro);
   errorIng=callproc error1c
            (filasE4=byref(filas);
             operacionE1='INSERT';
             programaE2='cont413';
             notasE3='Inserto Servicio en la tabla.');
   if errorIng>0 then
            redisplay;
```

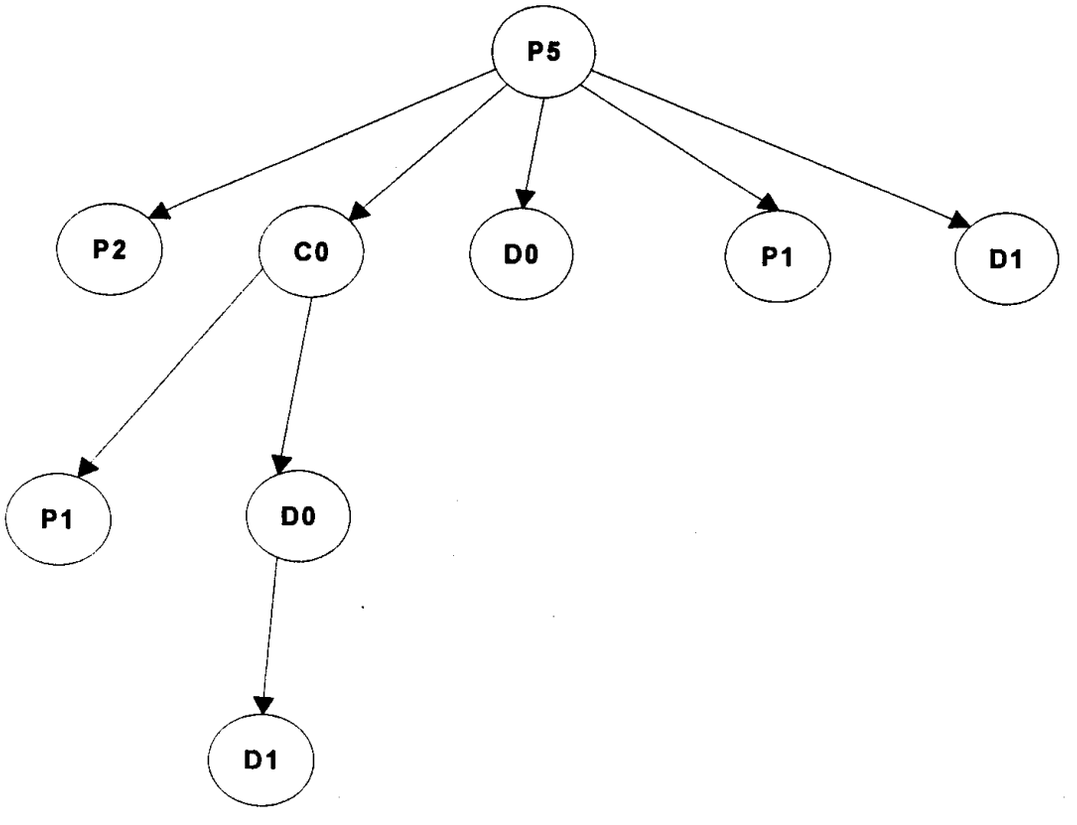
```
            rollback;
            return 1;
    else
            commit;
    endif;
    return 0;
end /* Ejecutar */

'Salir'. key frskey3 (activate=0)=
begin
    return 1;
end /* Salir */
/*************************************************************************/
/*************************Validacion de campos*********************/
/*************************************************************************/
field 'cod_servicio'=
begin
    campo=1;
    if cod_servicio!='' then
      cod_servicio=danumero(cod_servicio,3);
            select nom_servicio
             from hor_servicios
            where cod_centro=:cod_centro
              and cod_servicio=:cod_servicio;
      errorIng=callproc error1c
              (filasE4=byref(filas);
               operacionE1='SELECT';
               programaE2='cont413';
               notasE3='Compruebo si YA existe el Servicio.');
      if errorIng>0 then
              redisplay;
              rollback;
            elseif filas>0 then
              rollback;
              callproc beep();
              message 'ERROR --> Servicio YA existe.';
              sleep 3;
              cod_servicio='';
              nom_servicio='';
              resume;
      else
              commit;
      endif;/* errorIng */
    endif;
    set_forms field '' (reverse(cod_servicio)=0 ,
                reverse(nom_servicio)=1);
    resume next;
end /* cod_servicio*/

field 'nom_servicio'=
begin
    campo=2;
    set_forms field '' (reverse(nom_servicio)=0 ,
                reverse(cod_servicio)=1);
    resume next;
end /* nom_servicio */

/************************/
/**** PROCEDURE    ****/
/************************/
procedure inicia_campos()=
begin
    set_forms field '' (reverse(cod_servicio)=0 ,
                reverse(nom_servicio)=0 );
end
```

## 1.- SIZE APPLICATION (AS)

LOC of procedural control sentences = 29
LOC of visual control sentences = 17
LOC of exception control sentences = 0
LOC of definition sentences = 0
LOC of data manipulation sentences = 2
LOC of security control sentences = 0
LOC of transaction control sentences = 4
**FS= 52 and AS=52**

## 2- LENGTH (DEPTH OF NESTING)

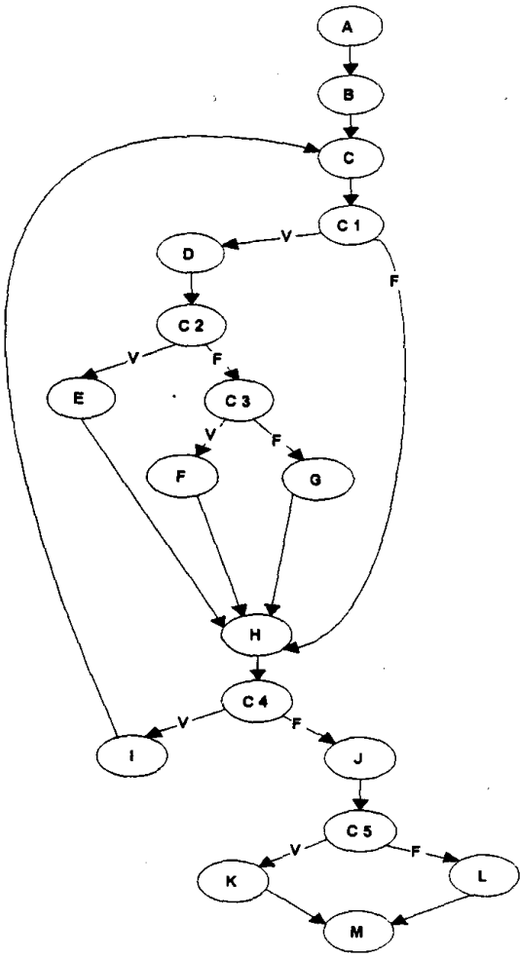
The tree of the frame is the following:




The depth of nesting is = 5

## 3- COMPLEXITY

### 3.1 Procedural control sentences




$V(G) = |R| - |E| + 2p$

$p=1$

$|R| = 22$

$|E| = 18$

**$V(G) = 22-18+2 = 6$**

Complexity procedural control sentences: $W_{CSCP}$ * 6
Complexity visual control sentences : $W_{CVCS}$ * 22
Complexity exception handling sentences: 0
Complexity definition sentences : 0
Complexity data manipulation sentences : $W_{CDMS}$ * 84
Complexity security control sentences : 0
Complexity transaction control sentences : $W_{CTCS}$ * 4

$FC = W_{CSCP} * 6 + W_{CVCS} * 22 + W_{CDMS} * 84 + W_{CTCS} * 4$
$AC = W_{CSCP} * 6 + W_{CVCS} * 22 + W_{CDMS} * 84 + W_{CTCS} * 4$