Editors
Editorial Policy
Peer Review
Copyright
Author Instructions

Author Index
Subject Index
Search Database
Full-text Search

Digital Resources
Reviews

Register

Home

Information Research

an electronic journal

Information Research: an electronic journal, **is published by the Department of Information Studies, University of Sheffield.**

ISSN 1368-1613

If you find Information Research useful, please sign in and we'll notify you of future issues.

Best viewed with

This page is maintained by Professor Tom Wilson.
t.d.wilson@sheffield.ac.uk Last up-date: 2nd July 1999. For statistics, click here: 

Characterization of Fourth Generation languages (4GLs)

Antonio Martínez and Mario Piattini
Grupo Alarcos, Departamento de Informática
University of Castilla-la Mancha
Ronda de Calatrava, 5
13071, Ciudad Real (Spain)

Abstract

Because of their basic features, the Fourth Generation Languages (4GL´s) have contribute profits to the development of applications. 4GL´s environment are more user- friendly systems, they improve on productivity in relation to traditional languages, they make easy the development through "templates" and easier allow the creation of real prototypes that transform the cycle of the live of the applications, and the facility of building portable applications with considerable possibilities of adaptation.

The heterogeneity of the sentences that constitute the 4GL make it difficult to apply traditional metrics, such as McCabe´s cyclomatic number (McCabe, 1976).

We propose here a classification of the fourth generation languages in different sub-languages in order to make the application of metrics possible.

1. Introduction to fourth generation languages (4gls)

Many of the organisations which use management information systems are now aware that computer systems constructed using third generation languages, such as COBOL and FORTRAN, can be more effectively produced and maintained using modern productivity-enhancing tools. These tools have been given various names, including fourth-generation languages (4GL's), application generators, or more recently fourth-generation systems. The term fourth generation is open to a wide variety of interpretations — no two definitions ever have exactly the same implications. True 4GL's imply that all phases of application design and development are catered for, not just the coding phase, which is after all a relatively small proportion of the total effort involved in developing and maintaining a major application system (Holloway, 1990).

There is one central theme, common to all the definitions of 4G environment, and that is significant improvements in productivity over conventional methods of writing application software.

Fourth Generation Languages (4GLs) have been found appropriate for a large class of commercial applications, and allow rapid system development. In some cases such development can be extremely fast, which permits a significant change in the methodology used. Fourth Generation Languages have greatly enlarged the community of direct users and manipulators of data. Many of these tools are centred around some form of data dictionary, often actively used by the system. This begins to address the need for a single source of such routines, and has potential for greatly improving data integrity by eliminating the chance of multiple, inconsistently implemented routines.

4GLs have revolutionised the approach to software production in a number of ways. One of the key changes consists in making possible user participation in systems design. The ability to specify an application very rapidly at a workstation and subsequently to amend it has enabled the end-user to become a real partner in the software design process. Another key

area of change is in the reduction of errors in writing code to access files and handle screens. This is a major source of errors in conventional code. However, with a 4GL, this code is no longer the responsibility of the programmer. The benefits here derive from substantially reducing the number of these very common errors.

Overall, the use of 4GLs appears to lead to more reliable code, with less manpower. The 4GL provides good documentation in a uniform style, which helps to ensure that as the applications evolve reliability of the code is maintained.

Traditionally, we classify the 4GLs in two groups:

a. *Final user languages* (information centre): They are centred in ease and flexibility of use.

b. *Professional Languages* (development centre): To develop sophisticated applications and to permit rapid prototype-construction applications. These kinds of languages appear with Base Data Management Systems (Cobb, 1985).

2. Identification of sub-languages.

The rising development of applications in 4GL makes necessary a maintainability control of the written code in 4GL. Maintainability is considered to be influenced by understandability, modifiability and testability (Li and Chen, 1987); which depend on the size, length and complexity of Fourth Generation Language programs.

It is a fact that 4GL languages are heterogeneous, as they include sentences of different natures, so it is impossible to give overall metrics for 4GL programs. We propose to identify different sublanguages:

A. 4GL Sentences.

A1) Procedural control sentences.

A2) Visual control sentences.

A3) Exception handling sentences.

B. SQL Sentences.

B1) Database definition sentences.

B2) Database manipulation sentences

B3) Security control sentences.

B4) Transaction control sentences.

We define each of the sublanguages with examples in different 4GL (CA- OpenIngres, Visual Basic, Delphi).

2.1. 4GL Sub-languages

This sub-language contains the building blocks sentences (if, do while, case, do until) (see figures 1, 2 and 3.)

```
If status = 'n' then
        If empsum = 0 then
                Message 'Please enter employee number';
                Sleep 3;
        Else
                Callframe NewEmsp
        Endif;
Endif;
```

Figure 1. CA-OpenIngres Procedural Control Sentences

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

Figure 2. Visual Basic Procedural Control Sentences

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

Figure 3. Delphi Procedural Control Sentences

2.1.2. Visual Control Sentences.

This sublanguage is defined as covering display sentences and forms-control sentences to manipulate both forms field and local variables. (See figures 4, 5 and 6.)

```
Set_forms frs (activate (nextfield) = 1,

     activate (previousfield) = 1,

     activate (Keys) = 1);
```

Figure 4. CA-OpenIngres Visual Control Sentences.

```
MsgBox ("Are you there?", MB_SÍNO + MB_ICONOALTO)
```

Figure 5. Visual Basic Visual Control Sentences

```
MessageDlgPos ('Are you there?', mtConfirmation, mbYesNoCancel, 0, 200, 200);
```

Figure 6. Delphi Visual Control Sentences.

2.1.3. Exception handling sentences

This sublanguage is defined as containing the sentences and functions to retrieve information about an application that is running. (See figures 7, 8 and 9.)

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

Figure 7. OpenIngres Exception handling sentences

```
On error Goto CantReadFile
Open "\CURRENCY.TXT"
...
CantReadFile::
        MsgBox "Have you created the file CURRENCY.TXT ?"
        ...
```

Figure 8. Visual Basic Exception handling sentences.

```
Try
    Rewrite(f);
    Except On exception Do
        Begin
            ShowMessage('Impossible write in the file' + Name + '.');
        ErrorWrite := True;
        End;
End;
```

Figure 9. Delphi Exception handling sentences.

2.2.1. Database definition sentences.

This sublanguage is defined as containing sentences that can create, modify and destroy a variety of database objects, such as tables, views, indexes and database procedures. (See figure 10.)

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

Figure 10. CA-OpenIngres/SQL database definition sentences

2.2.2. Database manipulation sentences

This sublanguage is defined as containing the sentences that enable you to manipulate data

in the tables. (See figure 11.)

```
Select ename from employees
Where edept isnull and hiredate = date ('today')
```

Figure 11. CA-OpenIngres/SQL database manipulation sentences

2.2.3. Security control sentences

This sublanguage is defined as containing the sentences that enable you to control access to database objects, roles and DBMS resources. (See figure 12.)

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

Figure 12. CA-OpenIngres/SQL security control sentences

2.2.4. Transaction control sentences

This sublanguage contains the sentences that help to manage transactions (e.g. rollback).

3. Metrics for 4GLs

Different kinds of metrics can be defined for 4GL's. At the moment, some work has been done in order to estimate the development effort and to correlate it to program size (Dolado, 1997; Verner and Tate, 1988), but more work is necessary in order to control the quality of 4GL programs.

The problem is, as we have mentioned, that the heterogeneity of 4GL statements invalidates global measures so different sublanguages must be taken into account in order to get a more precise measure. In this way, different measures for size, length, complexity, cohesion and coupling could be defined for all 4GL sublanguages, and summarized in one global measure. For example, application program size could be defined as:

$$SIZE = w_{PC}PCS + w_{VC}VCS + w_{EH}EHS + w_{DD}DDS + w_{DM}DMS + w_{SC}VCS + w_{TC}TCS$$

Where PCS, VCS, EHS, DDS, DMS, VCS and TCS are procedural control size, visual control size, exception handling size, database definition size, database manipulation size, security control size and transaction control size respectively. w_{PC}, w_{VC}, w_{EH}, w_{DD}, w_{DM}, w_{SC}, w_{TC} are 4GL-dependent weights, which must be adjusted for each different 4GL (CA-OpenIngres, Delphi, Visual Basic, Open Road, Powerbuilder, etc.) and also for each organization. As an example we will present the procedural control sentences and the database manipulation sentences.

In Martinez and Piattini (1998) we provide a first approximation to these metrics, adapting several other classical metrics as: lines of code, depth of nesting, McCabe cyclomatic complexity, Bieman and Ott's cohesion (1994), Fenton and Pfleeger's coupling (1996), etc.

3.1.1 Procedural control size (PCS)

The procedural control size (PCS) is defined as the sum of the lines of code (LOC).

$PCS = \Sigma LOC$

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. In appendix A, a complete example for an Ingres program is shown.

3.1.2. Procedural control length (PCL)

We use depth of nesting to measure the length of a program. Depth of nesting of a program can be defined using graph theory (Fenton and Pfleeger, 1996). Program can be modeled by flowgraphs. We calculate the application length adding 1 to maximum of the depth of nesting of its program. In appendix A, a complete example for an Ingres program is shown.

3.1.3. Procedural control complexity (PCC)

The metric used to measure the complexity in procedural control sentences is McCabe's cyclomatic complexity (McCabe, 1976):

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

Where:

G: is the graph of the program 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.

In appendix A , a complete example for an Ingres program is shown.

3.2 Database manipulation sentences metrics

We have studied different kinds of metrics for this sublanguage (Martinez and Piattini, 1999). The best metrics for characterised the SELECT sentence are:

3.2.1 Database manipulation size (DMS)

Metrics NT

Expresses the number of tables that the SELECT sentence contains. (See figure 13.)

$$DMS = \Sigma NT$$

```
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
```

Figure 13. SELECT sentence example

We can caracterise SELECT sentence in base to the values NT=3.

5.Conclusions and future work

Fourth Generation Languages are acquiring big importance in some installations, substituting slowly existing Third Generation Languages.

Metrics are useful mechanisms in improving the quality of software products, specially maintenance, which is the most important problem of software development, ranging between 60 and 90 percent of life- cycle costs (Card and Glass, 1990; Pigoski, 1997). Software measurement is widely recognised as an effective means to understand, monitor, control, predict and improve software development and maintenance projects (Briand et al., 1996). Measurement is used not only for understanding, controlling, and improving development, but also for determining the best ways to help practitioners and researchers.

Maintainability is achieved by means of three factors: understandability, modifiability and testability, which are in turn influenced by complexity (Li and Cheng, 1987). However, a general complexity is *"the impossible holy grail"* (Fenton, 1994). Henderson- Sellers (1996) distinguishes three types of complexity: computational, psychological and representational, and for psychological complexity he considers three components: problem complexity, human cognitive factors and product complexity. The last one is our focus.

At this moment, we are defining metrics for each one of the sublanguages that we have identified in this paper. We are also verifying these measures in different formal frameworks as Zuse (1998), Weyuker (1988) and Briand et al. (1996) (1997).

As formal verification is not enough, we are also validating these metrics empirically using both controlled experiments with students and real cases in different organisations. These metrics are confronted with maintenance cost, moreover limits for these measures (as programme guidelines) are to be established in order to control application production.

Future research will consists in the definition of metrics for object-oriented 4GL, based on those proposed by Chidamber and Kemerer (1994) and Henderson-Sellers (1996).

References

- Bieman J.M., and Ott L.M. (1994) "Measuring Functional Cohesion". *IEEE Trans. On Software Engineering* 20, 8,644-657.
- Briand, L.C., Morasca, S. and Basili, V. (1996). Property-based software engineering measurement. *IEEE Transactions on Software Engineering,* 22(1): 68-85.
- 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.
- Card, D.N. y Glass, R.L. (1990). *Measuring Software Design Quality* . Englewood Cliffs. USA.
- Codd, E.F. (1970). A Relational Model of Data for Larged Shared Data Banks. *CACM* , 13 (6), 377-387.
- Chidamber, S.R. & Kemerer, C.F. (1991). A metrics suite for object-oriented design. *IEEE Trans. On Software Engineering* 20 (6) 476-493.
- 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.
- Fenton, N. (1994). Software Measurement: A Necessary Scientific Basis. *IEEE Transactions on Software Engineering* , 20 (3): 199-206.
- Fenton, N. & Pfleeger, S.L. (1996). *Software Metrics: A Rigorous Approach and Practical Approach.* London: International Thomson Computer Press.
- Henderson-Sellers, B. (1996). *Object-oriented Metrics - Measures of complexity.* Upper Saddle River, New Jersey: Prentice-Hall.
- Holloway, S. (ed.) (1990). *Fourth-Generation Systems, their scope application and methods of evaluation.* London: Chapman and Hall.
- Li, H.F. and Chen, W.K. (1987). An empirical study of software metrics. *IEEE Trans. on Software Engineering* , 13 (6): 679-708.
- Martinez, A. and Piattini, M. (1998). Validation of 4GL metrics, proceedings *in United Kingdom Software Metrics Association (UKSMA),* 13-22.
- Martínez A., and Piattini M. (1999) " Experimental validation of SQL metrics", accepted *in FESMA.*
- McCabe, T.J. (1976). A complexity measure. *IEEE Trans. Software Engineering* 2(5): 308-320.
- Pigoski, T.M. (1997). *Practical Software Maintenance* . Wiley Computer Publishing. New York, USA.
- Verner J. and Tate G.(1988) "Estimating Size and Effort in Fourth-Generation Development". *IEEE Trans. On Software Engineering* . July, 15-22 .
- Weyuker, E.J. (1988). Evaluating software complexity measures. *IEEE Transactions on Software Engineering* 14(9):1357-1365.
- Zuse, H. (1998). "A framework of software measurement" , Ed. Walter De Gruyter.

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 ,
```

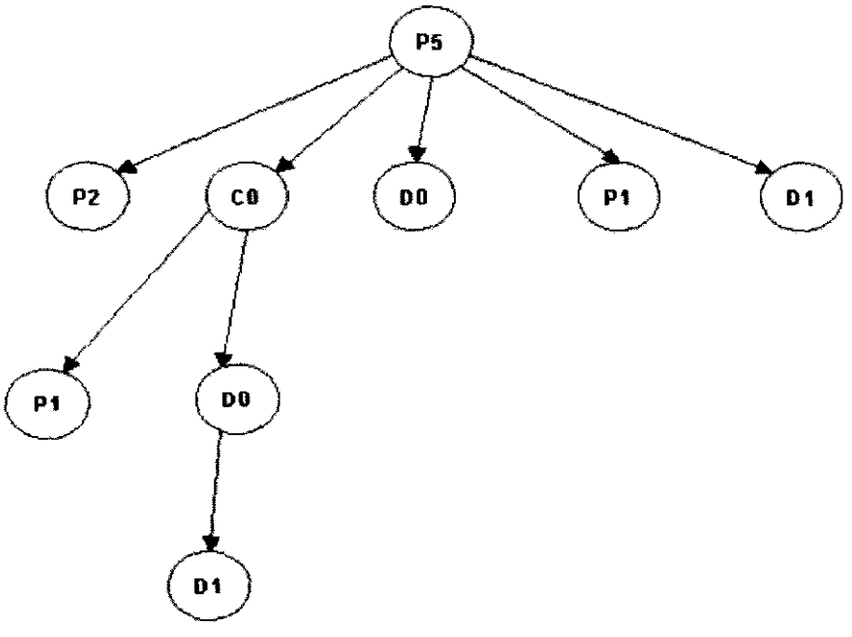
```
            reverse(cod_servicio)=1);
   resume next;
end /* nom_servicio */

/************************/
/**** PROCEDURE ****/
/************************/

procedure inicia_campos()=
begin
   set_forms field " (reverse(cod_servicio)=0 ,
            reverse(nom_servicio)=0 );
end
```

1. SIZE PROGRAM (PCS)

LOC of procedural control sentences = 29

2- LENGTH (DEPTH OF NESTING)

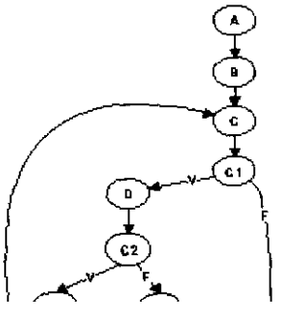
The tree of the frame is the following:

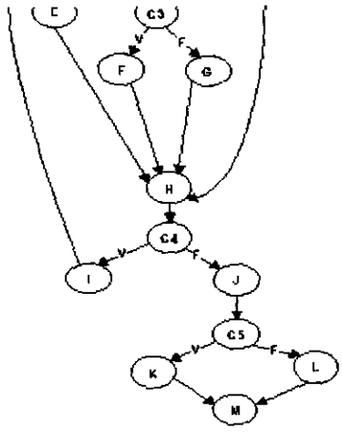


The depth of nesting is = 5

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: 6

```
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


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;


callproc beep();
message 'ERROR --> NO introducidos datos necesarios';
sleep 2;
    set_forms field " (reverse(cod_servicio)=1 );
resume field cod_servicio;


  (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.');
```