IDL CodeCount™

Counting Standard

University of Southern California

Center for Systems and Software Engineering

October 22, 2014

Revision Sheet

Date	Version	Revision Description	Author
6/5/2014	1.0	Original Release	CSSE
8/11/2014	1.1	Updated and revised document	CSSE
9/23/2014	1.2	Updated and revised document	CSSE
10/22/2014	1.3	Updated and revised document	CSSE

Table of Contents

1. Definitions

1.1. **SLOC –** Source Lines of Code is a unit used to measure the size of software program. SLOC counts the program source code based on a certain set of rules. SLOC is a key input for estimating project effort and is also used to calculate productivity and other measurements.

1.2. **Physical SLOC –** One physical SLOC is corresponding to one line starting with the first character and ending by a carriage return or an end-of-file marker of the same line, and which excludes the blank and comment line.

1.3. **Logical SLOC –** Lines of code intended to measure "statements", which normally terminate by a semicolon, comma, or a carriage return. Logical SLOC are not sensitive to format and style conventions, but they are language-dependent.

1.4. **Data declaration line or data line –** A line that contains declaration of data and used by an assembler or compiler to interpret other elements of the program.

IDL uses implicitly defined types so that there are no data declaration statements.

1.5. **Compiler Directives –** A statement that tells the compiler how to compile a program, but not what to compile. In IDL, the character "@"at sign at the beginning of a line causes the IDL compiler to substitute the contents of the file whose name appears after the "@" for the line.

The following table lists the IDL characters that denote compiler directive lines:

@

Table 1 Compiler Directives

1.6. **Blank Line –** A physical line of code, which contains any number of white space characters (spaces, tabs, form feed, carriage return, line feed, or their derivatives).

1.7. **Comment Line –** A comment is defined as a string of zero or more characters that follow language-specific comment delimiter.

IDL single line comment delimiter is ";". A whole comment line may span one line and does not contain any compliable source code. An embedded comment can co-exist with compliable source code on the same physical line. Banners and empty comments are treated as types of comments.

1.8. **Executable Line of code –** A line that contains software instruction executed during runtime and on which a breakpoint can be set in a debugging tool. An instruction can be stated in a simple or compound form.

- An executable line of code may contain the following program control statements:
  - Selection statements (if, then, else, switch, case)
  - Iteration statements (for, foreach, repeat, until, while, do)
  - Error control statements
  - Jump statements (break, continue, goto)
  - Expression statements (function calls, assignment statements, operations, etc.)
  - Block statements (begin, end). Note that in "case" and "switch" statement blocks, case or switch expressions that include relational operators and evaluate to true or false can appear in selector expressions. In this case, the selector expression is also counted as an executable line of code.
- An executable line of code may not contain the following statements:
  - Compiler directives
  - Whole line comments, including empty comments and banners
  - Blank lines

Note that the IDL function "where" may include an array expression as its first input parameter. This array expression may be in the form of a statement which may include relational operators. In this case, we do not count the relational operator statement as a separate line of code. It is considered as belonging to the same code line as the "where" function.

2. Checklist for source statement counts

PHYSICAL SLOC COUNTING RULES			
MEASUREMENT UNIT	ORDER OF PRECEDENCE	PHYSICAL SLOC	COMMENTS
Executable Lines	1	One Per line	Defined in 1.8
Non-executable Lines			
Compiler Directives	2	One per line	Defined in 1.5
Comments			Defined in 1.7
On their own lines	3	Not Included (NI)	
Embedded	4	NI	
Banners	5	NI	
Empty Comments	6	NI	
Blank Lines	7	NI	Defined in 1.6

LOGICAL SLOC COUNTING RULES				
NO.	STRUCTURE	ORDER OF PRECEDENCE	LOGICAL SLOC RULES	COMMENTS
R01	“for”, “foreach”, “repeat”, “while”, or “if” statement	1	Count Once	Loops and conditionals are independent statements.
R02	repeat (…) until (…) statement	2	Count Once	The subject statement can also be in the form of a “begin … endrep” block.
R03	Line terminated by new line character and last symbol is not ellipsis “$”	3	Count Once	End of command
R04	Block delimiters, “begin”, “end”	4	Count once per “begin … end” block.	“begin … end” blocks used with R01 and R02 are not counted.
R05	Compiler Directive	5	Count once per directive	

3. Examples

Executable Lines

Selection Statement

ESS1 – if, elseif, else and nested if statements

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

if <boolean expression> then begin
   <statements>
endif
	if (A EQ 2) then begin
	   print, ‘A = ‘, A
	endif
		1
		1
		0

if <boolean expression> then begin
   <statements>
endif else begin
   <statements>
endelse
	if (A EQ 2) then begin
	   print, ‘A = ‘, A
	endif else begin
	   if A NE 2 then print, ‘A != 2’
	endelse
		1
		1
		0
		1
		0

NOTE: complexity is not considered, i.e. multiple “&&” or “||” as part of the expression.	if (x NE 0) && (x > 0) then begin
	   x = x + 4
	endif
		1
		1
		0

ESS2 – case/switch and nested statements

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

case/switch <expression> of
   1: <constant 1>
      <statements>
   2: <constant 2>
      <statements>
   3: <statement>
      <statements>
   else:
      <statements>
endcase /endswitch
	Case/switch input_num of
	   1:
	      print, ‘one’
	   2:
	      print, ‘two’
	   (x GE 1):
	      print, ‘three’
	   else:
	      print, ‘Enter a value between 1 and 3’
	endcase /endswitch
		1
		0
		1
		0
		1
		1
		1
		0
		1
		0

Iteration Statement

EIS1 – for

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
for <index> = <start> do <increment>	for k = 0, n-1 do begin	1

<statements>
endfor
	C = A[k]
	endfor
	
	for x = 1, 4 do print, x, x+2
		1
		0
		
		2

EIS2 – while

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

while <boolean expression> do
    <statements>
endwhile
	n = 1
	while (n < 100) do
	    n = n + 1
	endwhile
		1
		1
		1
		0

EIS3 – repeat

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

repeat begin
      <statements>
endrep until <statements>
	A = 1
	B = 10
	repeat begin
	      A = A * 4
	endrep until A GT B
		1
		1
		0
		1
		1

EIS4 – foreach

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

foreach <elm>, <list>, <index> do begin
    <statements>
endforeach
	list = LIST(77.97, ‘Hiromi’, [2,4,6])
	foreach elm, list, index do begin
	   print, ‘Value = ‘, elm
	endforeach
		1
		1
		1
		0

JUMP Statement

EJS1 – goto

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

goto	If (i == 0) then
	    goto, JUMP1
	endif
	JUMP1: print, ‘Do Something’
		1
		1
		0
		1

EJS2 – break

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

break	if (i > 10) then
	    break
	endif
		1
		1
		0

EJS3 – continue

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

continue	If(i < 5) then
	   continue
	endif
		1
		1
		0

EXPRESSION Statement

EES1 – function call

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

function <function_name> ::P1, P2, …, Pn
    <statements>
   return, <statement>
end
	function Init_X::P1, P2
	      x = P1 + P2
	      return, x
	end
		1
		1
		1
		0

EES2 – procedure call

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

pro <procedure_name> ::P1, P2, …, Pn
    <statements>
end
	pro Init_X::P1, P2
	      x = P1 + P2
	end
		1
		1
		0

EES3 – assignment statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

<name> = <value>	X = [1 2 3 4]
	Y = X
		1
		1

COMPILER DIRECTIVES

CDL1 – directive types

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
@(<file name>)	@do_something	1

4. Cyclomatic Complexity

Cyclomatic complexity measures the number of linearly independent paths through a program. It is measured for each function, procedure, or method according to each specific program language. This metric indicates the risk of program complexity and also determines the number of independent test required to verify program coverage.

The cyclomatic complexity is computed by counting the number of decisions plus one for the linear path. Decisions are determined by the number of conditional statements in a function. A function without any decisions would have a cyclomatic complexity of one. Each decision such as an if condition or a for loop adds one to the cyclomatic complexity.

The cyclomatic complexity metric v(G) was defined by Thomas McCabe. Several variations are commonly used but are not included in the UCC. The modified cyclomatic complexity counts select blocks as a single decision rather than counting each case. The strict or extended cyclomatic complexity includes boolean operators within conditional statements as additional decisions.

Cyclomatic Complexity	Risk Evaluation
1-10	A simple program, without much risk
11-20	More complex, moderate risk
21-50	Complex, high risk program
> 50	Untestable program, very high risk

For IDL, the following table lists the conditional keywords used to compute cyclomatic complexity.

Statement	CC Count	Rationale
if	+1	if adds a decision
else	0	Decision is at the if statement
switch	+1 per item	Each item adds a decision – not the switch, if condition is a statement, then an additional decision is added
switch else	0	Decision is at the item statements
case	+1 per item	Each item adds a decision – not the case, if condition is a statement, then an additional decision is added
case else	0	Decision is at the item statements
for	+1	for adds a decision at loop start
foreach	+1	foreach adds a decision at loop start
while	+1	while adds a decision at loop start
repeat	+1	while adds a decision at loop start