Ruby CodeCount™ Counting Standard

University of Southern California

Center for Systems and Software Engineering

March, 2011

Revision Sheet

Date	Version	Revision Description	Author
3/1/11	1.0	Original Release	CSSE

1.0 CHECKLIST FOR SOURCE STATEMENT COUNTS

PHYSICAL AND LOGICAL SLOC COUNTING RULES

Measurement Unit	Order of Precedence	Physical SLOC	Logical SLOC	Comments
Executable lines	1	One per line	See table below	Defined in 2.9
Non-executable lines				
Declaration (Data) lines	2	One per line	See table below	Defined in 2.4
Compiler directives	3	One per line	See table below	Defined in 2.5
Comments				Defined in 2.8
On their own lines	4	Not included (NI)	NI	
Embedded	5	NI	NI	
Banners	6	NI	NI	
Empty comments	7	NI	NI	
Blank lines	8	NI	NI	Defined in 2.7

Table 1 Physical and Logical SLOC Counting Counts

LOGICAL SLOC COUNTING RULES

No.	Structure	Order of Precedence	Logical SLOC Rules	Comments
R01	"*for*", "*while*" or "*if*" statement	1	Count once.	Looping and conditional statements are independent.
R02	*do* {…} *until* (…); statement	2	Count once.	
R03	Statements ending by a semicolon	3	Count once per statement, including empty statement.	
R04	Block delimiters, braces {..}	4	Count once per set except where "}" followed by semicolon or "{" follows "else"	
R05	Compiler directive	5	Count once per directive.	

Table 2 Logical SLOC Counting Rules

2.0 DEFINITIONS

2.1 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.

2.2 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.

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

2.4 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.

The following table lists Ruby keywords that denote data declaration lines:

`String`	`Int`	`Def`	`array`

Table 3 Data Declaration Types

NOTE: See Section 3 of this document for examples of data declaration lines.

2.5 Compiler directive - A statement that tells the compiler how to compile a program, but not what to compile.

A list of common Ruby directives is presented in the table below:

`RubyAddPath`	`RubyRequire`	`RubyHandler`	`RubySetEnv`

Table 4 Compiler Directives

NOTE: See Section 3 of this document for examples of compile directive lines.

2.6 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).

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

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

2.8 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, ? operator)
  - Iteration statements (for, while, do)
  - Empty statements (one or more ";")
  - Jump statements (return, goto, last, next, exit function)
  - Expression statements (function calls, assignment statements, operations, etc.)
  - Block statements

  NOTE: See Section 3 of this document for examples of control statements.

- An executable line of code may not contain the following statements:
  - Compiler directives
  - Data declaration (data) lines
  - Whole line comments, including empty comments and banners
  - Blank lines

3.0 EXAMPLES OF LOGICAL SLOC COUNTING

EXECUTABLE LINES

SELECTION STATEMENTS

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT
ESS1	if, elseif, else	if <boolean expression> [then]	if x != 0 then	1
	and	<statement>	print “non-zero”	1
	nested if	end	end	0
	statements			
		if <boolean expression> [then]	if x > 0	1
		<statement>	print “positive”	1
		else	else	0
		<statement>	print “negative”	1
		end	end	0
		if <boolean expression> [then]	if x == 0	1
		<statement>	print “zero”	1
		elsif <boolean expression> [then]	elsif x > 0	1
		<statement>	print “positive”	1
		else	else	0
		<statement>	print “negative”	1
		end	end	0
			i = 1 if x > 10	2
		NOTE: complexity is not considered, i.e. multiple “&&” or “||” as part of the expression.		
ESS2	case statements	case <expression>	case $num	1
		when <constant 1>	when 0..10	1
		<statement>	print “small num”	1
		when <constant 2>	when 11..100	1
		<statement>	print “large num”	1
		else	else	0
		<statement>	print “HUGE num”	1
		end	end	0
ESS3	unless statements	unless <expression> [then]	unless $big	1
		<statement>	print “small”	1
		else	else	0
		<statement>	print “big”	1
		end	end	0

ITERATIONS STATEMENTS				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**

EIS1	for-do	for <control> in <expression> [do]
		    <statement>
		end
			for i in [1, 2, 3] do
			    print i*2
			end
				1
				1
				0

EIS2	while-do	while <boolean expression> [do]
		    <statement>
		end
			while i < 10 do
			    print i
			end
				1
				1
				0

EIS3	until-do	until <boolean expression> [do]
		    <statement>
		end
			until i < 10 do
			    print i
			end
				1
				1
				0

JUMP STATEMENTS
(ARE COUNTED AS THEY INVOKE ACTION – PASS TO THE NEXT STATEMENT)

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT

EJS1	goto, label	goto *label*;
		.
		.
		label:
			loop1:
			    x++
			    if x < y
			        goto loop1
				0
				1
				1
				1

EJS2	return	return <expression>	if x < 0 return	2

EXPRESSION STATEMENTS

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT
EES1	function call	<function_name> <parameters>	readfile “filename”	1

EES2	assignment statement	<name> := <value>	x = y
			a = 1; b = 2; c = 3;
				1
				3

EES3	empty statement (is counted as it is considered to be a placeholder for something to call attention)	one or more “;” in succession	;	1 per each

BLOCK STATEMENTS

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT

EBS1	block = related statements treated as a unit	{
		    <statements>
		}
			{
			    i = 0
			    print i
			}
				1
				1
				1
				0

DECLARATION (DATA) LINES

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT
DDL1	variable declaration	<name>	President = [“Ford”, “Carter”]	1

COMPILER DIRECTIVES

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT
CDL1	directive types	<directive>	RubyRequires	1