Ruby CodeCount™ Counting Standard

University of Southern California

Center for Systems and Software Engineering

December, 2011

Revision History

Date	Version	Revision Description	Author
3/1/11	1.0	Original Release	CSSE
12/1/11	2.0	Updated Release	Sowmya Rao

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 2.4
Non-executable lines				
Directives	2	One per line	See table below	Defined in 2.5
Comments				Defined in 2.6
On their own lines	3	Not included (NI)	NI	
Embedded	4	NI	NI	
Banners	5	NI	NI	
Empty comments	6	NI	NI	
Blank lines	7	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	Block delimiters, braces {..}	3	Count once per set except where "}" followed by semicolon or "{" follows "else"	

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 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
  - Whole line comments, including empty comments and banners
  - Blank lines

2.5 Compiler Directive – A statement that tells the compiler how to compile a program, but not what to compile. Bash shell script does not contain any compiler directives.

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

The comment delimiter for Ruby 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.

NOTE: The ‘#’ character is also used for other purposes within Ruby, apart from delimiting comments.

2.7 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).

3.0 EXAMPLES OF LOGICAL SLOC COUNTING

EXECUTABLE LINES

SELECTION STATEMENTS

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT

ESS1	if, elsif, else and nested if statements	if <Boolean expression> [then]
		   <statements>
		end
		
		if <Boolean expression> [then]
		   <statements>
		else
		   <statements>
		end
		
		if <Boolean expression> [then]
		   <statements>
		elsif <Boolean expression> [then]
		   <statements>
		else
		   <statements>
		end
		
		<statement> if <Boolean expr>
		
		<statement LHS> if <Boolean expr>
		   <statement RHS1>
		else
		   <statement RHS2>
		end
		
		NOTE: complexity is not considered, i.e. multiple “&&” or “||” as part of the expression.
			if x != 0 then
			   print “non-zero”
			end
			
			if x > 0
			   print “positive”
			else
			   print “negative”
			end
			
			if x == 0
			   print “zero”
			elsif x > 0
			   print “positive”
			else
			   print “negative”
			end
			
			i = 1 if x > 10
			
			toss = if rand(2) == 1 then
			   "heads"
			else
			   "tails"
			end
				1
				1
				0
				
				1
				1
				0
				1
				0
				
				1
				1
				1
				1
				0
				1
				0
				
				2
				
				1
				1
				0
				1
				0

ESS2	case-when-else-end	case <expression>
		   when <constant 1>
		      <statements>
		   when <constant 2>
		      <statements>
		   else
		      <statements>
		End
			case $num
			   when 0..10
			      print “small num”
			   when 11..100
			      print “large num”
			   else
			      print “HUGE num”
			end
				1
				1
				1
				1
				1
				0
				1
				0

ESS3	unless statements	unless <expression> [then]
		   <statements>
		else
		   <statements>
		end
		
		<statements> unless <Boolean expr>
			unless $big
			   print “small”
			else
			   print “big”
			end
			
			print "Non-negative" unless x > 0
				1
				1
				0
				1
				0
				
				2

ITERATION STATEMENTS

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT

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

EIS2	while statements	while <Boolean expr> [do]
		    <statements>
		end
		
		<statement> while <Boolean expr>
		
		begin
		    <statements>
		end while <Boolean expr>
			while $i < $num
			    puts("Inside the loop i = #$i");
			    $i +=1;
			end
			
			puts $1 += 2 while $i < 10
			
			begin
			    puts("Inside the loop i = #$i");
			    $i +=1;
			end while $i < $num
				1
				1
				1
				0
				
				2
				
				1
				1
				1
				1

EIS3	until statements	until <Boolean expr> [do]
		    <statements>
		end
		
		<statement> until <Boolean expr>
		
		begin
		    <statements>
		end until <Boolean expr>
			until $i > $num
			        puts("Inside the loop i = #$i");
			        $i +=1;
			end
			
			puts $1 += 2 until $i > 10
			
			begin
			    puts("Inside the loop i = #$i");
			    $i +=1;
			end until $i > $num
				1
				1
				1
				0
				
				2
				
				1
				1
				1
				1

EIS4	each iterator	<collection>.each do <|variable|>
		    <statements>
		end
			a.each do |i|
			    puts i
			end
				1
				1
				0

EIS5	collect iterator	<collection> = <collection>.collect
		
		<collection> =
		<collection>.collect{|variable| expr}
			b = a.collect
			
			c = a.collect{|x| 10*x}
				1
				
				2

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

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT
EJS1	throw statement	throw <:labelname>	throw :greeting	1
		throw <:labelname> <condition>	throw :greeting if TIME == 0	2
EJS2	catch statement	catch <:labelname> do	catch :greeting do	1
		<statements>	puts(“Good morning!”);	1
		end	end	0
EJS3	return statement	return <expr>	def test2	1
			i = 100; j = 200; k = 300	3
			return i, j, k;	1
			end	0
		<condition> return	if x < 0 return	2
EJS4	break statement	break	if i > 2 then	1
			break	1
			end	0
EJS5	next statement	next	if i < 2 then	1
			next	1
			end	0
EJS6	redo statement	redo	if i < 2 then	1
			redo	1
			end	0
EJS7	retry statement	begin	begin	1
		<statements>	nil; # exception raised	1
		rescue	rescue	1
		<statements>	nil; # handles error	1
		retry	retry # restart from begin block	1
		end	end	0
		retry <condition>	for i in 1..5	1
			retry if i > 2	2
			puts "Value of local variable is #{i}"	1
			end	0

EXPRESSION STATEMENTS

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT
EES1	Assignment statement	<name> = <value> <name1> = <name2>	x = 3; x = y; $num = 10 @cust_name = name @@no_of_customers = 4 PI = 3.14159	2 1 1 1 1

EES2	Empty statement (counted as it's a placeholder for something)	One or more ';', but not following another statement	while i < 10 do
			   puts("Hello!");
			   ;
			end
				1
				1
				1
				0

EES3a	Function calls – general	<function_name> <parameters>	puts("Hello!")	1

EES3b	Function calls – special	raise
		
		require
		
		include
			begin
			   puts 'I am before the raise.'
			   raise 'An error has occurred.'
			   puts 'I am after the raise.'
			rescue
			   puts 'I am rescued.'
			end
			
			require "Week"
			
			class Decade
			   include Week
			   no_of_yrs=10
			   def no_of_months
			      puts Week::FIRST_DAY
			      number = 10*12
			      puts number
			   end
			end
				1
				1
				1
				1
				1
				1
				0
				
				1
				
				1
				1
				1
				1
				1
				1
				1
				0
				0

CLASS AND MODULE STATEMENTS				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**

ECS1	class	class <class_name>
		    <statements>
		end
			class Customer
			   @@no_of_customers = 0
			end
				1
				1
				0

ECS2	def	def <method_name>[var = value]
		    <statements>
		end
			def hello
			     puts "Hello Ruby!"
			end
				1
				1
				0

ECS3	undef	undef <method_name>	undef hello	1
ECS4	alias	alias <new_method> <old_method> alias <new_glob_var> <old_glob_var>	alias greeting hello alias $angle $argument	1 1

ECS5	super		class Employee < Sample
			    def initialize(fname, lname, position)
			        super(fname,lname)
			        @position = position
			    end
			    def to_s
			        super + ", #@position"
			    end
			end
				1
				1
				1
				1
				0
				1
				1
				0
				0

ECS6	module	module <module_identifier>
		    <statements>
		end
			module Trig
			    PI = 3.141592654
			    def Trig.sin(x)
			        nil; # Code for sine of x
			    end
			    def Trig.cos(x)
			        nil; # Code for cosine of x
			    end
			end
				1
				1
				1
				1
				0
				1
				1
				0
				0

BLOCK STATEMENTS

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT
EBS1	yield statements	yield [var1, var2, …]	def test1	1
			yield	1
			end	0
			def test2	1
			yield 5	1
			end	0
EBS2	do-end statements	<method_invocation> do	test1 do	1
		<statements>	puts "You are in the block"	1
		end	end	0
EBS3	{ } delimiters	<method_invocation> {	test2 {	1
		<statements>	|i| puts "You are in the block #{i}"	1
		}	}	0
EBS4	BEGIN and END blocks	BEGIN {	BEGIN {	1
		<statements>	puts "Initializing Ruby Program"	1
		}	}	0
		END {	END {	1
		<statements>	puts "Terminating Ruby Program"	1
		}	}	0
EBS5	begin-rescue-else-ensure-end	begin	begin	1
		<statements>	puts "I'm not raising exception"	1
		rescue	rescue Exception => e	1
		<statements>	puts e.message	1
		else	puts e.backtrace.inspect	1
		<statements>	else	0
		ensure	puts "Congratulations-- no errors!"	1
		<statements>	ensure	1
		end	puts "Ensuring execution"	1
			end	0

OPERATORS AND PSEUDO-VARIABLES				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**
EOP1	defined? operator	defined? [parameter] (parameter = variable, method_call, super, yield)	defined? foo defined? $_ defined? puts defined? puts(bar) defined? super defined? yield	1 1 1 1 1 1

EOP2	nil	<variable> = nil;
		(functions as a variable with a logic value false)
		
		nil
		(functions as a placeholder)
			@name = nil;
			
			def Trig.sin(x)
			   nil # Code for sine of x
			end
				1
				
				1
				1
				0