

Ruby Code Count™ Counting Standard

University of Southern California

Center for Systems and Software Engineering

December , 2016

Revision Sheet

Date	Version	Revision Description	Author
7/6/2016	1.0	Original Release	Derek Lengenfelder
Date	Version No.	Click here to Description.	Name

Table of Contents

No.			Contents	Page No.
1.0	Definitions			4
	1.1	SLOC		4
	1.2	<u>Physi</u>	cal SLOC	4
	1.3	Logic	al SLOC	4
	1.4	<u>Data</u>	declaration line	4
	1.5	Comp	oiler directive	4
	1.6	<u>Blank</u>	<u>line</u>	4
	1.7	Comr	nent line	4
	1.8	Execu	itable line of code	4
2.0	Checklist fo	or source	statement counts	6
3.0	<u>Examples of</u>	of logical S	LOC counting	7
	3.1	Execu	table Lines	7
		3.1.1	Selection Statements	7
		3.1.2	<u>Iteration Statements</u>	8
		3.1.3	Jump Statements	9
		3.1.4	Expression Statements	11
		3.1.5	Block Statements	12
		3.1.6	Class and Module Statements	13
		3.1.7	Operator and	14
			<u>Pseudo-variables</u>	
	3.2	Comp	iler directives	15
4.0	Complexity	L		16
5.0	Cyclomatic	Complex	ity	17

1. Definitions

- SLOC Source Lines of Code is a unit used to measure the size of software program. SLOC counts the 1.1. 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 (C/C++, Java, C#) or a carriage return (VB, Assembly), etc. 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 a compiler or assembler to interpret other elements of the program. Ruby does not contain any data declarations.
- 1.5. Compiler Directives - A statement that tells the compiler how to compile a program, but not what to compile.

The following table lists the Ruby keywords that denote compiler directive lines:

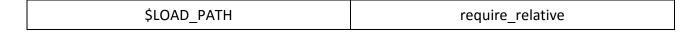


Table 1 Ruby Compiler Directive

- 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. Ruby comment delimiters are "#" and "=begin", ending with "=end". A whole comment line may span one line and does not contain any compilable source code. An embedded comment can coexist 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.
- 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, ? operator, case)
 - Iteration statements (for, while, loop-do-break, begin-end-while)

- Jump statements return, next, break, retry, redo, yield)
- Expression statements (function calls, assignment statements, operations, etc.)
- **Block 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

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				
Declaration (Data) lines	2	One per line	Defined in 1.4	
Compiler directives	3	One per line	Defined in 1.5	
Comments			Defined in 1.7	
On their own lines	4	Not included (NI)		
Embedded	5	NI		
Banners	6	NI		
Empty comments	7	NI		
Blank lines	8	NI	Defined in 1.6	

	LOGICAL SLOC COUNTING RULES				
NO.	STRUCTURE	ORDER OF PRECEDENCE	LOGICAL SLOC RULES	COMMENTS	
R01	"for", "while" or "if" statement	1	Count once	"while" is an independent statement	
R02	begin-end-while OR loop do-condition- break-end	2	Count once	The "end" keyword is not counted.	
R03	Statements ending by a semicolon	3	Count once per statement, including empty statement	Semicolons within "for" statement are not counted. Semicolons used with R01 and R02 are not counted	
R04	Block delimiters	4	Count once per block	Function definition is counted once	
R05	Compiler Directive	5	Count once per directive		

3. Examples

EXECUTABLE LINES		
	SELECTION Statement	
ESS1 - if, else if, else and neste GENERAL EXAMPLE	d if statement SPECIFIC EXAMPLE	SLOC COUNT
if (<boolean expression="">)</boolean>	if (x != 0)	1
<statements></statements>	printf("non-zero")	
end	end	
Chu	Chu	
if (<boolean expression="">)</boolean>	if (x == 0)	1
<statements>;</statements>	printf("zero");	1
else if (<boolean expression="">)</boolean>	elsif (x > 0)	1
<statements>;</statements>	printf("positive");	1
	else	0
else	printf("negative");	1
<statements>;</statements>	end	0
end		
if (<boolean expression="">)</boolean>	if (x > 0)	
<statements>;</statements>	printf("positive")	
else	else	0
<statements>;</statements>	printf("not positive")	
end	end	0
NOTE: complexity is not	if ((x != 0) && (x > 0))	1
considered, i.e. multiple "&&" or	printf("%d", x);	1
" " as part of the expression.	end	0
ESS2 - ?: operator		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
Exp1 ? Exp2 : Exp3	x >= 0 ? printf("+") : printf("-")	1
ESS3 – case-when-else-end stat	ement	
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

case (<expression>)</expression>	case (grade)	1
when <constant 1=""></constant>	when "A"	1
<statements></statements>	puts "Top marks!";	1
when <constant 2=""></constant>	when "B"	1
<statements></statements>	puts "Good try.";	1
when <constant 3=""></constant>	when "C"	1
<statements></statements>	puts "You are having difficulty.";	1
else	else	0
<statements></statements>	puts "Seek assistance.";	1
end	end	0

ESS4 – unless statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
unless <expression> [then]</expression>	unless (x != 0)	1
<statements></statements>	printf("non-zero");	1
else	else	0
<statements></statements>	printf("zero");	1
end	end	0
<statements> unless <bool expr=""></bool></statements>	printf("positive") unless x > 0	1
NOTE: complexity is not	if ((x != 0) && (x > 0))	1
considered, i.e. multiple "&&" or	printf("%d", x);	1
" " as part of the expression.	end	0

ITERATION Statement

EIS1 – for loops

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
for variable [, variable] in	for i in 05	1
expression [do]	puts "Value of variable is #{i}";	1
<statements></statements>	end	0
end		

EIS2 – while loops

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
while <boolean expression=""> [do]</boolean>	while \$i < \$num	1
<statements></statements>	puts ("Inside the loop i = \$i");	1
end	\$i += 1;	1
	end	0
<statement[; statement;]="">while <boolean expression=""></boolean></statement[;>	puts \$i += 2 while \$i < 10	2
begin	begin	1

		1.
<statements></statements>	puts ("Inside the loop \$#\$i");	1
end while <boolean expression=""></boolean>	\$i += 1;	
	end while \$i < \$num	1
EIS3 – until loops		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
until <boolean expr=""> [do]</boolean>	until \$i > \$num	1
<statements></statements>	puts("Inside the loop i = #\$i");	1
end	\$i +=1;	1
	end	0
<statement[; statement;]="">until <boolean expression=""></boolean></statement[;>	puts \$1 += 2 until \$i > 10	2
begin	begin	1
<statements></statements>	puts("Inside the loop i = #\$i");	1
end until <boolean expression=""></boolean>	\$i +=1;	1
	end until \$i > \$num	1
EIS4 – each iterator		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
<collection>.each do variable[,</collection>	(05).each do i	1
variable]	puts "Value of variable is #{i}";	1
<statements></statements>	end	0
end		
EIS5 – collect iterator		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
<collection> = <collection>.collect</collection></collection>	b = a.collect	1
<collection> =</collection>	$c = a.collect{ x 10*x}$	2
<collection>.collect{ variable </collection>		
expr}		
CAPI J		
	<u>Jump</u> Statement	
EJS1 – throw statement		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
throw <: labelname>	throw :greeting	1
throw <:labelname> <condition></condition>	throw :greeting if TIME == 0	2
EJS2 – catch statement		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

	antah yang ating a dia	1
h <:labelname> do	catch :greeting do	1
tatements>	puts "Good morning!";	1
	end	0
EJS3 – return statement		21222221117
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
rn <expression></expression>	def test2	1
	i = 100; j = 200; k = 300;	3
	return i,j,k;	1
	end	0
ndition> return	if x < 0 return	2
EJS4 – break statement		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
ık	if (i > 2) then	2
	break;	
	end	
EJS5 – next statement		
GENERAL EXAMPLE		
	if (i > 2) then	1
	next	1
	end	0
EJS6 – redo statement		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
)	if (i > 2) then	1
	redo	1
	end	0
FIS7 - retry statement		
	SPECIFIC FYAMDIF	SLOC COLINIT
	-	
ч	_	
	Cilu	
/ <condition></condition>	for i in 15	1
		2
	I	1
GENERAL EXAMPLE EJS5 – next statement GENERAL EXAMPLE EJS6 – redo statement GENERAL EXAMPLE	if (i > 2) then break; end SPECIFIC EXAMPLE if (i > 2) then next end SPECIFIC EXAMPLE if (i > 2) then redo end SPECIFIC EXAMPLE if (i > 2) then redo end	SLOC COUNT 1 1 0 SLOC COUNT 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1

	end	0
EJS8 – abort statement		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
abort	abort	1
EJS9 – exit statement		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
exit(<result>)</result>	exit(0)	1
Chief in Country	chic(e)	
exit!(<result>)</result>	exit!(0)	1
	. ,	
	Expression Statement	
	EXTRESSION Statement	
EES1 - assignment statement		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
<name> = <value></value></name>	y = 3; x = y;	2
\lane - \value>	y - 3, x - y,	2
<name1> = <name2></name2></name1>	\$num = 10	1
Widnie 1	@cust_name = "name"	1
	@@no of customers = 4	1
	PI = 3.141592	1
EES2 – empty statement (is cou	nted as it is considered to be a place	holder for something)
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
I one or more ":" but not following	while \$i < 10 do	1
one or more ";" but not following another statement	while \$i < 10 do puts "Hello!":	
another statement	while \$i < 10 do puts "Hello!"; \$i += 1;	1 1 1
	puts "Hello!";	1
	puts "Hello!";	1 1
	puts "Hello!"; \$i += 1; ;	1 1
another statement	puts "Hello!"; \$i += 1; ;	1 1
another statement EES3 – function call	puts "Hello!"; \$i += 1; ; end	1 1 0
EES3 – function call GENERAL EXAMPLE	puts "Hello!"; \$i += 1; ; end SPECIFIC EXAMPLE	1 1 0 SLOC COUNT
another statement EES3 – function call	puts "Hello!"; \$i += 1; ; end	1 1 0
EES3 – function call GENERAL EXAMPLE <function_name> (<parameters>)</parameters></function_name>	puts "Hello!"; \$i += 1; ; end SPECIFIC EXAMPLE	1 1 0 SLOC COUNT
EES3 – function call GENERAL EXAMPLE <function_name> (<parameters>) EES4 – function calls (special)</parameters></function_name>	puts "Hello!"; \$i += 1; ; end SPECIFIC EXAMPLE puts "Hello"	1 1 0 SLOC COUNT
EES3 – function call GENERAL EXAMPLE <function_name> (<parameters>) EES4 – function calls (special) GENERAL EXAMPLE</parameters></function_name>	puts "Hello!"; \$i += 1; ; end SPECIFIC EXAMPLE puts "Hello" SPECIFIC EXAMPLE	1 1 0 SLOC COUNT 1 SLOC COUNT
EES3 – function call GENERAL EXAMPLE <function_name> (<parameters>) EES4 – function calls (special)</parameters></function_name>	puts "Hello!"; \$i += 1; ; end SPECIFIC EXAMPLE puts "Hello" SPECIFIC EXAMPLE begin	1 1 0 SLOC COUNT 1 SLOC COUNT 1
EES3 – function call GENERAL EXAMPLE <function_name> (<parameters>) EES4 – function calls (special) GENERAL EXAMPLE</parameters></function_name>	puts "Hello!"; \$i += 1; ; end SPECIFIC EXAMPLE puts "Hello" SPECIFIC EXAMPLE begin puts 'I am before the raise.'	SLOC COUNT SLOC COUNT 1 1 1
EES3 – function call GENERAL EXAMPLE <function_name> (<parameters>) EES4 – function calls (special) GENERAL EXAMPLE</parameters></function_name>	puts "Hello!"; \$i += 1; ; end SPECIFIC EXAMPLE puts "Hello" SPECIFIC EXAMPLE begin puts 'I am before the raise.' raise 'An error has occurred.'	SLOC COUNT SLOC COUNT 1 1 1 1
EES3 – function call GENERAL EXAMPLE <function_name> (<parameters>) EES4 – function calls (special) GENERAL EXAMPLE</parameters></function_name>	puts "Hello!"; \$i += 1; ; end SPECIFIC EXAMPLE puts "Hello" SPECIFIC EXAMPLE begin puts 'I am before the raise.' raise 'An error has occurred.' puts 'I am after the raise.'	1 1 0 SLOC COUNT 1 1 1 1
EES3 – function call GENERAL EXAMPLE <function_name> (<parameters>) EES4 – function calls (special) GENERAL EXAMPLE</parameters></function_name>	puts "Hello!"; \$i += 1; ; end SPECIFIC EXAMPLE puts "Hello" SPECIFIC EXAMPLE begin puts 'I am before the raise.' raise 'An error has occurred.' puts 'I am after the raise.' rescue	1
EES3 – function call GENERAL EXAMPLE <function_name> (<parameters>) EES4 – function calls (special) GENERAL EXAMPLE</parameters></function_name>	puts "Hello!"; \$i += 1; ; end SPECIFIC EXAMPLE puts "Hello" SPECIFIC EXAMPLE begin puts 'I am before the raise.' raise 'An error has occurred.' puts 'I am after the raise.'	1 1 0 SLOC COUNT 1 1 1 1

		T		
roquiro	require "Week" 1			
require	require week			
include	class Decade	1		
	include Week	1		
	no_of_yrs=10	1		
	def no_of_months	1		
	puts Week::FIRST_DAY	1		
	number = 10*12	1		
	puts number	1		
	end	0		
	end	0		
EDC4 violate to the second	<u>Вьоск</u> Statement			
EBS1 – yield statement	CDECIFIC EVALABLE	CLOC COLINIT		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT		
yield [var1, var2,]	def test1	1		
	yield	1 0		
	end	0		
	def test2	1		
	yield 5	1 1		
	end			
	Cita			
EBS2 – do-end statement				
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT		
<method invocation=""> do</method>	test1 do	1		
<statements></statements>	puts "You are in the block"	1		
end	end	0		
Cita	Cita	-		
EBS3 – {} delimiters				
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT		
<method_invocation> {</method_invocation>	test2 {	1		
<statements></statements>	puts "You are in the block"	1		
}	}	0		
,	,			
EBS4 – BEGIN-END statement				
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT		
BEGIN {	BEGIN {	1		
<statements></statements>	puts "Initializing Ruby Program"	1		
}	}	0		
END {	END {	1		
<statements></statements>	puts "Ending Ruby Program"	1		
}	}	0		

else puts e.message 1 else puts e.backtrace.inspect 1 else puts "Congrats-noerrors!" 1 ensure puts "Congrats-noerrors!" 1 ensure 1 end 0 CLASS AND MODULE Statement CES1 - class statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT end 0 ECS2 - def statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT ECS2 - def statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT ECS3 - undef statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT ECS3 - undef statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT end 0 ECS3 - undef statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT def <method_name>[var = value] end 0 ECS3 - undef statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT undef <method_name> undef hello 1 ECS5 - alias statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT alias <me_method> 1 alias <me_method> 2 cold_glob_var> 2 cold_glob_var> 2 cold_glob_var> 3 cold_glob_var> 4 cold_glob_v</me_method></me_method></me_method></me_method></me_method></me_method></me_method></me_method></me_method></method_name></method_name>					
GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT begin (statements>) begin puts "I'm not raising exception" 1 1 rescue (statements>) rescue Exception => e 1 1 else (statements>) puts e.backtrace inspect 1 1 else (statements>) else (statements) 0 ensure (statements) puts "Congratsnoerrors!" 1 1 end (statements) ensure 1 1 end (statements) puts "Ensuring execution" 1 1 class Calass statement GENERAL EXAMPLE (statements) SPECIFIC EXAMPLE (statements) class Calass_name> (statements) end (statements) 1 end (end (statements)) geno_of_customers = 0 1 1 end (end (statements)) geno_of_customers = 0 1 1 def statement (statements) general Example (statements) statements (statements) statements (statements) end (end (statements)) 1 1 gend (end (statement)) 1 1 gend (end (statement)) 1 1 end (end (statement)) 1 1 end (end (end (statement)) 1 1 end (end (end (statement)) 1 1 end (end (end (end (statement))) 1 1 end (end (end (end (end (end	FBS5 — hegin-rescue-else-ensure-end statement				
statements> puts "I'm not raising exception" 1 rescue	_		SLOC COUNT		
rescue rescue Exception => e 1	begin	begin	1		
Statements puts e.message 1	<statements></statements>	puts "I'm not raising exception"	1		
else puts e.backtrace.inspect 1 else else puts "Congratsnoerrors!" 1 ensure puts "Ensuring execution" 1 end	rescue	rescue Exception => e	1		
else puts "Congrats-noerrors!" 1	<statements></statements>	puts e.message	1		
ensure statements> end puts "Congratsnoerrors!" 1 ensure 1 puts "Ensuring execution" 1 end 0 CLASS AND MODULE Statement ECS1 - class statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT class <class _name=""></class>	else	puts e.backtrace.inspect	1		
end end puts "Ensuring execution" 1 1 end 0 CLASS AND MODULE Statement ECS1 - class statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT class cclass_name> class Customer 1 end 0 ECS2 - def statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT def <method_name>[var = value] <statements> end 0 ECS3 - undef statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT def /method_name>[var = value] end 0 ECS3 - undef statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT undef /method_name> undef hello 1 ECS5 - alias statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT undef /method_name> undef hello 1 ECS\$ - alias statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT alias < new_method> 1 alias < new_method> 1 alias < new_method> 1 alias < new_glob_var> < old_glob_var> ECS5 - super statement</statements></method_name>	<statements></statements>	else	0		
end puts "Ensuring execution" 1 0 CLASS AND MODULE Statement ECS1 - class statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT class cclass_name> class Customer 1 0 0 ECS2 - def statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT ECS2 - def statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT def <method_name>[var = value] of the lib of</method_name>	ensure	puts "Congratsnoerrors!"	1		
ECS1 - class statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT class <class_name> class Customer 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</class_name>	<statements></statements>	ensure	1		
CLASS AND MODULE Statement ECS1 - class statement GENERAL EXAMPLE class Class Customer <pre></pre>	end	puts "Ensuring execution"	1		
ECS1 - class statement GENERAL EXAMPLE class Cclass_name>		end	0		
ECS1 - class statement GENERAL EXAMPLE class Cclass_name>					
GENERAL EXAMPLE class <class_name></class_name>		CLASS AND MODULE Statement			
GENERAL EXAMPLE class <class_name></class_name>	ECC1 _ class statement				
class <class_name></class_name>		SPECIFIC FYAMDI F	SLOC COLINIT		
<pre>end</pre>					
ECS2 – def statement GENERAL EXAMPLE def <method_name>[var = value]</method_name>	_				
ECS2 – def statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT def <method_name>[var = value]</method_name>					
GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT def <method_name>[var = value] def hello 1 <statements> puts "Hello Ruby!" 1 end 0 ECS3 - undef statement GENERAL EXAMPLE GENERAL EXAMPLE undef hello SPECIFIC EXAMPLE SLOC COUNT 1 ECS\$ - alias statement GENERAL EXAMPLE SPECIFIC EXAMPLE SPECIFIC EXAMPLE SLOC COUNT alias <new_method></new_method></statements></method_name>	enu	enu	10		
GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT def <method_name>[var = value] def hello 1 <statements> puts "Hello Ruby!" 1 end 0 ECS3 - undef statement GENERAL EXAMPLE GENERAL EXAMPLE undef hello SPECIFIC EXAMPLE SLOC COUNT 1 ECS\$ - alias statement GENERAL EXAMPLE SPECIFIC EXAMPLE SPECIFIC EXAMPLE SLOC COUNT alias <new_method></new_method></statements></method_name>	FCS2 – def statement				
def <method_name>[var = value]</method_name>		SPECIFIC EXAMPLE	SLOC COUNT		
end end 0 ECS3 - undef statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT undef <method_name> undef hello 1 ECS\$ - alias statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT alias < new_method> 1 <pre></pre></method_name>					
ECS3 - undef statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT undef <method_name> undef hello 1 ECS\$ - alias statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT alias < new_method> 1 alias <new_method> 1 alias <new_glob_var> <old_glob_var> ECS5 - super statement</old_glob_var></new_glob_var></new_method></method_name>	-				
ECS3 - undef statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT undef <method_name> undef hello 1 ECS\$ - alias statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT alias <new_method> 1</new_method></method_name>		1 .			
GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT undef <method_name> undef hello 1 ECS\$ - alias statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT alias <new_method> 1 alias <new_glob_var></new_glob_var> old_glob_var> ECS5 - super statement</new_method></method_name>			1		
GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT undef <method_name> undef hello 1 ECS\$ - alias statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT alias <new_method> 1 alias <new_glob_var></new_glob_var> old_glob_var> ECS5 - super statement</new_method></method_name>	ECS3 – undef statement				
undef <method_name> undef hello 1 ECS\$ - alias statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT alias <new_method> 1 alias <new_glob_var></new_glob_var> cold_glob_var> ECS5 - super statement</new_method></method_name>		SPECIFIC EXAMPLE	SLOC COUNT		
ECS\$ – alias statement GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COUNT alias < new_method> 1					
GENERAL EXAMPLE SPECIFIC EXAMPLE alias < new_method>	under smethod_names	ander nene			
GENERAL EXAMPLE SPECIFIC EXAMPLE alias < new_method>					
alias <new_method> 1</new_method>	ECS\$ – alias statement				
<pre>cold_method></pre>	GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT		
alias <new_glob_var> <old_glob_var> ECS5 – super statement</old_glob_var></new_glob_var>	alias	alias <new_method></new_method>	1		
alias <new_glob_var> <old_glob_var> ECS5 – super statement</old_glob_var></new_glob_var>		<old_method></old_method>			
<old_glob_var> ECS5 – super statement</old_glob_var>			1		
ECS5 – super statement		alias <new_glob_var></new_glob_var>			
		<old_glob_var></old_glob_var>			
GENERAL EXAMPLE SPECIFIC EXAMPLE SLOC COLINE	ECS5 – super statement				
SECOCOSIVI	GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT		
super class Employee < Sample 1	super	class Employee < Sample	1		

	def initialize(fname, lname, pos)	1
	super(fname,Iname)	1
	@position = pos	1
	end	0
	def to s	1
	super + ", #@position"	1
	end	0
	end	0
	Letiu	0
ECS6 - module statement		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
module <module identifier=""></module>	module Trig	1
	PI = 3.141592654	1
	def Trig.sin(x)	1
	nil; # Code for sine of x	1
	end	0
	def Trig.cos(x)	1
	nil; # Code for cosine of x	1
	end	0
	end	0
	OPERATORS AND PSEUDO VARIABLES	
EOP1 – defined? statement		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
defined? [parameter]		
	defined? foo	1
	defined? \$_	1
(parameter = variable,	defined? \$_ defined? puts	
	defined? \$_	1
(parameter = variable,	defined? \$_ defined? puts	1 1
(parameter = variable,	defined? \$_ defined? puts defined? puts(bar)	1 1 1
(parameter = variable,	defined? \$_ defined? puts defined? puts(bar) defined? super	1 1 1 1
(parameter = variable, method_call, super, yield)	defined? \$_ defined? puts defined? puts(bar) defined? super	1 1 1 1
(parameter = variable, method_call, super, yield) EOP2 – nil statement	defined? \$_ defined? puts defined? puts(bar) defined? super defined? yield	1 1 1 1
(parameter = variable, method_call, super, yield) EOP2 - nil statement GENERAL EXAMPLE	defined? \$_ defined? puts defined? puts(bar) defined? super defined? yield	1 1 1 1 1 SLOC COUNT
<pre>(parameter = variable, method_call, super, yield) EOP2 - nil statement GENERAL EXAMPLE <variable> = nil;</variable></pre>	defined? \$_ defined? puts defined? puts(bar) defined? super defined? yield	1 1 1 1
<pre>(parameter = variable, method_call, super, yield) EOP2 - nil statement GENERAL EXAMPLE <variable> = nil; (functions as a variable with a logic</variable></pre>	defined? \$_ defined? puts defined? puts(bar) defined? super defined? yield	1 1 1 1 1 SLOC COUNT
<pre>(parameter = variable, method_call, super, yield) EOP2 - nil statement GENERAL EXAMPLE <variable> = nil;</variable></pre>	defined? \$_ defined? puts defined? puts(bar) defined? super defined? yield	1 1 1 1 1 SLOC COUNT
<pre>(parameter = variable, method_call, super, yield) EOP2 - nil statement GENERAL EXAMPLE <variable> = nil; (functions as a variable with a logic</variable></pre>	defined? \$_ defined? puts defined? puts(bar) defined? super defined? yield	1 1 1 1 1 1 SLOC COUNT
<pre>(parameter = variable, method_call, super, yield) EOP2 - nil statement GENERAL EXAMPLE <variable> = nil; (functions as a variable with a logic</variable></pre>	defined? \$_ defined? puts defined? puts(bar) defined? super defined? yield SPECIFIC EXAMPLE @name = nil;	1 1 1 1 1 1 SLOC COUNT 1
<pre>(parameter = variable, method_call, super, yield) EOP2 - nil statement GENERAL EXAMPLE <variable> = nil; (functions as a variable with a logic value false) nil</variable></pre>	defined? \$_ defined? puts defined? puts(bar) defined? super defined? yield	1 1 1 1 1 1 1 1 1 1 1 1 1 1
(parameter = variable, method_call, super, yield) EOP2 - nil statement GENERAL EXAMPLE <variable> = nil; (functions as a variable with a logic value false)</variable>	defined? \$_ defined? puts defined? puts(bar) defined? super defined? yield SPECIFIC EXAMPLE @name = nil; def Trig.sin(x) nil # Code for sine of x	1 1 1 1 1 1 1 1 1 1 1 1 1 1
<pre>(parameter = variable, method_call, super, yield) EOP2 - nil statement GENERAL EXAMPLE <variable> = nil; (functions as a variable with a logic value false) nil</variable></pre>	defined? \$_ defined? puts defined? puts(bar) defined? super defined? yield SPECIFIC EXAMPLE @name = nil;	1 1 1 1 1 1 1 1 1 1 1 1 1 1

COMPILER DIRECTIVES

CDL1 – directive type

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
\$LOAD_PATH	\$LOAD_PATH << '.'	1
require_relative	require_relative Trig	1

4. Complexity

Complexity measures the occurrences of different keywords in code baseline. Below table identifies the categories and their respective keywords that are counted as part of the complexity metrics.

Math Functions	Trig	Log	Calculations	Conditionals	Logic	Pre-processor	Assignment
exp	atan2	log	%	if	&&	\$LOAD_PATH	=
frexp	cos	log10	+	elsif	II	require_relative	
ldexp	sin		-	when	==		
rand	tan		*	for	!=		
sqrt			**	while	<=>		
srand			/	unless	!		
			>>	until	and		
			<<		not		
			~		or		
			&		>		
					<		
					>=		
					<=		
					===		

Table 2 - Complexity Keywords List

5. 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.

Table 3 – Cyclomatic Complexity Risk Evaluation

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 Ruby the following table lists the conditional keywords used to compute cyclomatic complexity.

Table 4 – Cyclomatic Complexity Counts

Statement	CC Count	Rationale
if	+1	if adds a decision
else if	+1	else if adds a decision
else	0	Decision is at the if statement
switch case	+1 per case	Each case adds a decision – not the

		switch
switch default	0	Decision is at the case statements
for	+1	for adds a decision at loop start
while	+1	while adds a decision at loop start or at end of do loop
do	0	Decision is at while statement – no decision at unconditional loop
try	0	Decision is at catch statement
catch	+1	catch adds a decision
ternary ? :	+1	Ternary ? adds a decision – : is similar to default or else