R Counting Standard

University of Southern California

Center for Systems and Software Engineering

May , 2018

Revision Sheet

Date	Version	Revision Description	Author
9/24/2017	1.0	Original Release	1. Rushi Chandurkar 2. Swasti Sharma

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 (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 an assembler or compiler to interpret other elements of the program.

The following lists the R keywords that denote data declaration lines:

Table 1 - R Data Keywords

mode	length	vector	factor	array	matrix	data.frame
ts	list	read	scan	seq	rep	sequence
gl	expand.grid					

1.5. Compiler directive

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

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

Table 2 - Java Compiler Directives

library	import

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. R comment delimiters are “#”. R does not support multiline 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,elseif, switch)
- Iteration statements (for, while, repeat)
- Empty statements (“;”)
- Jump statements (return, next , break)
- 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

Table 3 - 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 line	4	Not included	
Embedded	5	Not included	
Banners	6	Not included	
Empty comments	7	Not included	
Blank lines	8	Not included	Defined in 1.6

Table 4 - 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	do {…} while (…); statement	2	Count once	Braces {…} and semicolon ; used with this statement are 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, braces {…}	4	Count once per pair of braces {…}, except where a closing brace is followed by a semicolon, or an opening brace comes after a keyword “else”	Braces used with R01 and R02 are not counted. Function definition is counted once since it is followed by {…}

R05	Compiler Directive	5	Count once per directive	

3. Examples

3.1. Executable lines

3.1.1. Selection Statement

Table 5 - ESS1 (if, else if, else, and nested if statements)

General Example	Specific Example	SLOC Count

if (<boolean expression>)
  <statements>;
 	
if (<boolean expression>)
  <statements>;
else <statements>;
 	
if (<boolean expression>)
  <statements>;
else if (<boolean expression>)
  <statements>;
  .
  .
  .
else <statements>;
 	
NOTE: complexity is not considered, i.e. multiple “&&” or “||” as part of the expression.
	if (x != 0)
	  print (“non-zero”);
	
	if (x > 0)
	  print (“positive”);
	else
	  print (“negative”);
	
	if (x == 0)
	  print (“zero”);
	else if (x > 0)
	  print (“positive”);
	else {
	  print (“negative”);
	}
	
	if ((x != 0) && (x > 0))
	  print (x);
		1
		1
		
		1
		1
		0
		1
		
		1
		1
		1
		1
		0
		1
		0
		
		1
		1

Table 6 - ESS2 (ifelse)

General Example	Specific Example	SLOC Count
Ifelse(condition,true_case,false_case)	Ifelse(x > 0,true,false);	1

Table 7 - ESS3 (switch and nested switch statements)

General Example	Specific Example	SLOC Count

switch (<expression>,
<case constant 1>={
<statements>;
},
<case constant 2>={
<statements>;
},
{
	AA = 'foo
	switch(AA,
	  foo={
	  # case 'foo' here...
	  print('foo')
	  },
	  bar={
	  # case 'bar' here...
		1
		1
		0
		1
		1
		0
		0
		1

<default statements>;
}
)
	print('bar')
	},
	{
	print('default')
	}
	)
		1
		0
		0
		1
		0
		0

Table 8 - ESS4 (try-catch blocks)

General Example	Specific Example	SLOC Count

result = tryCatch({
expr
}, warning = function(w) {
warning-handler-code
}, error = function(e) {
error-handler-code
}, finally = {
cleanup-code
}
	x <- tryCatch(stop("an error occured!"),
	warning=function(w){
	return(paste("Warning:", conditionMessage(w)));
	},
	error = function(e) {
	return(paste("Error:", conditionMessage(e)));
	},
	finally={
	print("This is try-catch test. check the output.")
	});
	print(x);
		1
		0
		1
		1
		0
		0
		1
		1
		0
		0
		1
		1
		0
		0
		1

3.1.2. Iteration Statement

Table 9 - EIS1 (for loops)

General Example	Specific Example	SLOC Count

for (val in sequence)
{
statement
}
	for (year in 2010:2015){
	print(year)
	}
		1
		1
		0

Table 10 - EIS2 (empty statements)

General Example	Specific Example	SLOC Count
for (val in sequence)	for (year in 2010:2015)	2

Table 11 - EIS3 (while loops)

General Example	Specific Example	SLOC Count
while (<boolean expression>) <statement>;	i <- 1 while (i < 6) {	1 1

	print(i) i = i+1 }	1 1 0

Table 12 - EIS4 (using repeat)

General Example	Specific Example	SLOC Count

repeat{
<statements>;
if(<boolean condition>){
break
}
}
	i <- 5
	repeat{
	print(i)
	if(i == 0){
	break
	}
	i = i -1}
		1
		1
		1
		0
		1
		0
		1

3.1.3. Jump Statement

Table 13 - EJS1 (return statements)

General Example	Specific Example	SLOC Count
return expression	return i	1

Table 14 – EJS2 (break statements)

General Example	Specific Example	SLOC Count
break;	if (I > 10) break	2

Table 15 – EJS3 (exit function)

General Example	Specific Example	SLOC Count
on.exit (<expression>);	require(graphics) opar <- par(mai = c(1,1,1,1)) on.exit(par(opar))	1 1 2

Table 16 – EJS4(next statement)

General Example	Specific Example	SLOC Count

next	v <- LETTERS[1:6]
	for (i in v) {
	if (i == "D") {
	next
	}
	print(i)
	}
		1
		1
		1
		1
		0
		1
		0

3.1.4. Expression Statement

Table 17 - EES1 (function call)

General Example	Specific Example	SLOC Count
<function_name> (<parameters>);	read_file (name)	1

Table 18 - EES2 (assignment statements)

General Example	Specific Example	SLOC Count
<name> <- <value>	X <- 1	1

Table 19 - EES3 (empty statement)

General Example	Specific Example	SLOC Count
one or more ";" in succession	; ;;;	1 3

3.1.5. Block Statement

Table 20 - EBS1 (blocks)

General Example	Specific Example	SLOC Count

/* start of block */
{
<definitions>
<statement>
}
/* end of block */
	/* start of block */
	{
	i = 0
	print(i)
	}
	/* end of block */
		0
		0
		1
		1
		0
		0

3.1.6. Declaration or Data Lines

Table 21 - DDL1 (function prototype, variable declaration)

General Example	Specific Example	SLOC Count
<name> (< parameter_list>); <name>	myFirstFun<-function(n){n*n } amount <- 123	1 1

3.1.7. Compiler Directives

Table 22 - CDL1 (directive type)

General Example	Specific Example	SLOC Count
library <package_name>;	import::from(broom, tidy)	1

import <package_name>;	library(foreach)	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.

Table 23 - Complexity Keywords List

Math Functions	Trig	Log	Calculations	Conditionals	Logic	Pre-processor	Assignment	Pointer
dev.cur	Cos	Log	++	else if	==	Library	=	
dev.set	Sin	Log10	--	switch	!=	Import	<-	
dev.off	Tan	Log1p	//	while	<=			
split.screen	Acos	Log2	%	else	>=			
erase.screen	Asin	logb	^	For	&&			
layout.show	Atan		+	if	||			
plot	Atan2		-	elseif	<			
sunflowerplot	cospi		*		>			
pie	Sinpi				!			
boxplot	tanpi				&			
stripchart					|			
coplot								
interaction.plot								
dotchart								
fourfoldplot								
assocplot								
mosaicplot								
pairs								
plot.ts								
ts.plot								
hist								
barplot								
qqnorm								
qqplot								
contour								
filled.contour								
image								
persp								
stars								
symbols								
termplot								
points								
AIC								

summary								
dotplot								
densityplot								
barchart								
rect								
Polygon								
segments								
axis								
box								
histogram								

5. Appendix - Differences between UCC and UCC-G

Below are the observations from our testing of UCC-G and comparing the results to UCC v2014.11.

- UCC counts != operation as an assignment operation. UCC-G counts it as a logical operation.
- UCC counts == operation as two assignment operations. UCC-G counts it as a logical operation.
- UCC counts << operation as two logical operations. UCC-G counts it as a left shift calculation operation.
- It appears that UCC is reporting CC2 cyclomatic complexity metrics. UCC-G counts and reports CC1 cyclomatic complexity metrics. UCC-G follows cyclomatic complexity report format of UCC v2015.12 (released by USC).