C# Code Count™

Counting Standard

University of Southern California

Center for Systems and Software Engineering

December , 2016

Revision Sheet

Date	Version	Revision Description	Author
5/26/2016	1.0	Original Release	Matthew Swartz
7/8/2016	1.1	Updated complexity table	Matthew Swartz

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 table lists the C# keywords that denote data declaration lines:

Simple Data Types	Compound and User Defined Data Types	Access Specifiers	Type Qualifiers
sbyte	Class	private	const
bool	Struct	protected	static
byte	Enum	public	volatile
char	Object	internal	
double	Interface		
float	Sealed	**Storage Class Specifiers**	**Miscellaneous**
int	Abstract	Extern	explicit
long	Delegate	Static	implicit
short	Event		readonly
decimal	Unsafe		namespace
string	List		override
uint			operator
ulong			virtual
Ushort			void

Data Declaration Data Types

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 C# keywords that denote data declaration lines:

#endregion	#warning	#define
#region	#undef	#endif

#error	#else	#elif
#line	#if	

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.

C# comment delimiters are "//" and "/*". 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.

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, switch)
  - Iteration statements (for, while, do-while)
  - Empty statements (one or more ";")
  - Jump statements (return, goto, break, continue, exit function)
  - 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*", "*for each*", "*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, i.e. *};* 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

EXECUTABLE LINES

SELECTION Statement

ESS1 - if, else if, else and nested if statements

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

if (<boolean expression>)
  <statements>;
 	
if (<boolean expression>)
{
  <statement>;
}
else
{
  <statement>;
}
 	
if (<boolean expression>)
  <statements>;
else if (<boolean expression>)
  <statements>;
.
.
.
else <statements>;
 	
if (<boolean expression>)
{
  <statements>;
}
 	
NOTE: complexity is not considered, i.e. multiple "&&" or "||" as part of the expression.
	if (check == true)
	  Console.Writeline("Flag is true");
	
	if (check == true)
	{
	  Console.Writeline("Flag true");
	}
	else
	{
	  Console.Writeline("Flag false");
	}
	
	if (x == 0)
	  Console.Writeline("zero");
	else if (x > 0)
	  Console.Writeline ("positive");
	else
	  Console.Writeline ("negative");
	
	if ((x != 0) && (x > 0))
	  Console.Writeline("non-zero");
	
	if (check == true)
	{
	  Console.Writeline("Flag is true");
	}
		1
		1
		
		1
		0
		1
		0
		0
		0
		1
		0
		
		1
		1
		1
		1
		0
		1
		
		1
		1
		
		1
		0
		1
		0

ESS2 - ? : OPERATOR

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
Exp1?Exp2:Exp3	S = x != 0.0 ? Math.Sin(x)/x : 1.0;	1

ESS3 - SWITCH ANDNESTED SWITCH STATEMENTS

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

switch (<expression>)
{
  case <constant 1> :
  <statements>;
  break;
  case <constant 2> :
  <statements>;
  break;
  case <constant 3> :
  <statements>;
  break;
  default
  <statements>;
}
	switch (number)
	{
	  case 1:
	  case 11:
	  foo1();
	  break;
	  case 2:
	  foo2();
	  break;
	  case 3:
	  foo3();
	  break;
	  default
	  Console.Writeline(“default”);
	}
		1
		0
		0
		0
		1
		1
		0
		1
		1
		0
		1
		1
		0
		1
		0

ITERATION Statement

EIS1 - FOR

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

for (initialization; condition; increment) *statement*;
 	
NOTE: “for” statement counts as one, no matter how many optional expressions it contains i.e.
for (i = 0, j = 0; i < 5, j < 10; i++, ,j++)
	for (i = 0; i < 10; i++)
	  Console.Writeline(i);
	
	for (i = 0; i < 10; i++)
	{
	  Console.Writeline(i);
	}
		1
		1
		
		1
		0
		1
		0

EIS2 - empty statements(could be used for time delays)

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
for (i = 0; i < SOME_VALUE; i++) ;	for (i = 0; i < 10; i++) ;	2

EIS3 - while

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

while (<boolean expression>)
<statement>;
	while (i < 10)
	{
	Console.Writeline("i");
	i++;
	}
		1
		0
		1
		1
		0

EIS4 – do-while

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
do { <statements>; } while (<boolean expression>);	Do { Console.Writeline("i"); } while (x < 5);	0 0 1 1

EIS5 - foreach

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

foreach (type identifier in collection)
{
<statements>;
}
	foreach (int i in array)
	{
	Console.Writeline(i);
	}
		1
		0
		1
		0

JUMP Statement

EJS1 - return

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
return *expression*;	if (i == 0) return;	2

EJS2 – goto, label

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

goto *label*;
.
.
.
label:
	loop1:
	x++;
	if (x < y) goto loop1;
		0
		1
		2

EJS3 - break

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
break;	if (i > 10) break;	2

EJS4 - continue

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

continue;	while (!done)
	{
	ch = getchar();
	if (char == '\n')
	{
	done = true;
	continue;
	}
	}
		1
		0
		1
		1
		0
		1
		1
		0
		0

EJS5 - throw

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
throw exception_obj;	if (s == null) throw new MyException();	1 1

EXPRESSION Statement

EES1 - FUNCTION CALL

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
<function_name> (<parameters>);	read_file (name);	1

EES2 - assignment statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

<name> = <value>;	x = y;
	char name[6] = "file1";
	a = 1; b = 2; c = 3;
		1
		1
		3

EES3 - empty statement(is counted as it is considered to be a placeholder for something to call attention)

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
one or more ";" in succession	;	1 per each

BLOCK Statement

EBS1 - block = related statements treated as a unit

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

/* start of block */
{
<definitions>
<statement>
}
/* end of block */
	/* start of block */
	{
	i = 0;
	Console.Writeline (i);
	}
	/* end of block */
		0
		0
		1
		1
		1
		0

DECLARATION OR DATA LINES

DDL1 - function prototype, variable declaration,struct declaration

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
<type> <name> (< parameter_list>);	void foo (int param);	1
	double amount, price;	1
<type> <name>;	int index;	1
	struct S	0
struct <name>	{	0
{	int x;	1
<type> <name>;	int y;	1
<type> <name>;	};	1
}		
	struct	0
struct	{	0
{	int x;	1
<type> <name>;	int y;	1
<type> <name>;	} S;	2
} <name>;		
	void main()	1
<type> <name> (< parameter_list>)	{	0
	Console.Writeline (“Hello”);	1
{	}	0
…		
}		

COMPILER DIRECTIVES

CDL1 - directive types

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
#define <name> <value>	#define MAX_SIZE 100	1
#define <symbol>	#undef DEBUG	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	Pointer
IEEEremainder	atan2	log10	++	else if	==	#endregion	=	
getExponent	acosh	log1p	--	switch	!=	#warning		
nexttoward	asinh	log2	>>	else	<=	#define		
nearbyint	atanh	log	<<	case	>=	#region		
remainder	acos		%	if	&&	#undef		
nextafter	asin		^		||	#endif		
toRadians	atan		+		<	#error		
toDegrees	cosh		-		>	#else		
copysign	sinh		*		!	#elif		
scalbln	tanh		/			#line		
llround	cos					#if		
scalbn	sin							
tgamma	tan							
lgamma								
lround								
llrint								
remquo								
random								
signum								
frexp								
ldexp								
expm1								
hypot								
floor								
trunc								
round								
lrint								
scalb								
modf								
exp2								
sqrt								
cbrt								
erfc								
ceil								
fmod								
rint								
fdim								

fmax								
fmin								
fabs								
ceil								
exp								
pow								
erf								
nan								
abs								
fma								
max								
min								

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 included 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 C#, the following table lists the conditional keywords used to compute cyclomatic complexity.

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 added at the case statements
for/foreach	+1	for/foreach adds a decision at loop start
while	+1	while adds a decision at loop start
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