PHP CodeCount™ Counting Standard

University of Southern California

Center for Systems and Software Engineering

April, 2010

Revision Sheet

Date	Version	Revision Description	Author
6/22/07	1.0	Original Release	CSSE
11/08/07	1.1	Updated	CSSE
4/08/10	1.2	Updated	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*", "*foreach*", "*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.	

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 (C/C++, Java, PHP) or a carriage return (VB, Assembly), etc. 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.

Each variable is defined with a dollar sign ($) before the variable's name. In addition, like many lines of PHP code, a semicolon is used. Semicolons do not, however, need to be placed at the end of commented lines. Strings, or a combination of characters, are defined with quotation marks around the value, while integers are not.

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

Data Declaration
$
Basic Data Types
boolean
integer
float
string
array
object
resource
NULL

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 PHP directives is presented in the table below:

`define`	`declare`
`include`	`include_once`
`require`	`require_once`

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.

PHP comment delimiters are "**//**","**#**", and "**/*…*/**". A whole comment line may span one or more lines 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, switch)
  - Iteration statements (for, foreach, 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

  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 and nested if statements	if (<boolean expression>)
		  <statements>;
		if (<boolean expression>) :
		  <statements>;
		endif;
		
		if (<boolean expression>)
		  <statements>;
		elseif (<boolean expression>)
		  <statements>;.
		.
		.
		else <statements>;
		
		if (<boolean expression>)
		{
		  <statements>;
		}
		else
		{
		  <statements>;
		}
		
		if (<boolean expression>) :
		  <statements>;
		else:
		  <statements>;
		endif;
			if ($x != 0)
			  echo “non-zero”;
			if ($x != 0):
			  echo “non-zero”;
			endif;
			
			if ($x == 0)
			  echo “zero”;
			elseif ($x > 0)
			  echo “positive”;
			else
			  echo “negative”;
			
			if ($x != 0)
			{
			  echo “non-zero”;
			}
			else
			{
			  echo “zero”;
			}
			
			if ($x != 0):
			  echo “non-zero”;
			else:
			  echo “zero”;
			endif;
				1
				1
				1
				1
				0
				
				1
				1
				1
				1
				0
				1
				
				1
				0
				1
				0
				0
				0
				1
				0
				
				1
				1
				0
				1
				0

ESS2	? operator	Exp1?Exp2:Exp3	x > 0 ? echo “+” : echo “-”;	1

ESS4	try-catch	try
		{
		  // code that could throw
		  // an exception
		}
		catch (exception-declaration)
		{
		  // code that executes when
		  // exception-declaration is thrown
		  // in the try block
		}
			try
			{
			  echo "Calling function";
			  throw Exception(“Error”);
			
			  MyFunc();
			}
			catch (IOException $e)
			{
			  echo “Error: ” . $e;
			}
				0
				0
				1
				1
				0
				1
				0
				1
				0
				1
				0

ESS3	switch and nested switch statements	switch (<expression>)
		{
		  case <constant 1> :
		  <statements>;
		  break;
		  case <constant 2> :
		  <statements>;
		  break;
		  default:
		  <statements>;
		}
		
		switch (<expression>):
		  case <constant 1> :
		  <statements>;
		  break;
		  case <constant 2> :
		  <statements>;
		  break;
		  default:
		  <statements>;
		endswitch;
			switch (number)
			{
			  case 1:
			  foo1();
			  break;
			  case 2:
			  foo2();
			  break;
			  default:
			  echo “invalid case”;
			}
			
			switch (number):
			  case 1:
			  foo1();
			  break;
			  case 2:
			  foo2();
			  break;
			  default:
			  echo “invalid case”;
			endswitch;
				1
				0
				0
				1
				1
				0
				1
				1
				0
				1
				0
				
				1
				0
				1
				1
				0
				1
				1
				0
				1
				0

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

EIS1	for	for (initialization; condition; increment)
		<*statements*>;
		
		for (initialization; condition; increment):
		<*statements*>;
		endfor;
			for (i = 0; i < 10; i++)
			  echo $i . “</br>”;
			
			for (i = 0; i < 10; i++)
			{
			  echo $i . “</br>”;
			}
			
			for (i = 0; i < 10; i++):
			  echo $i . “</br>”;
			endfor;
				1
				1
				
				1
				0
				1
				0
				
				1
				1
				0

EIS2	empty statements (could be used for time delays)	for ($i = 0; $i < SOME_VALUE; $i++) ;	for ($i = 0; $i < 10; $i++) ;	2

EIS3	while	while (<boolean expression>)
		<*statements*>;
		
		while (<boolean expression>):
		<*statements*>;
		endwhile;
			while ($i < 10)
			{
			  echo $i . “</br>”;
			  $i++;
			}
			
			while ($i < 10):
			  echo $i . “</br>”;
			  $i++;
			endwhile;
				1
				0
				1
				1
				0
				
				1
				1
				1
				0

EIS4	do-while	do
		{
		  <*statements*>;
		} while (<*boolean expression*>);
			do
			{
			  echo $i;
			  $i++;
			 } while ($i > 0);
				0
				0
				1
				1
				1

EIS5	foreach	foreach (array_expression as $value)
		  <*statements*>;
		
		foreach (array_expression as $value):
		  <*statements*>;
		endforeach;
		
		foreach (array_expression as $key => $value)
		  <*statements*>;
			$arr = array(1, 2, 3, 4);
			foreach ($arr as &$value) {
			  $value = $value * 2;
			}
			
			foreach ($arr as &$value):
			  $value = $value * 2;
			endforeach;
			
			$employeeAges;
			$employeeAges["Lisa"] = "28";
			$employeeAges["Grace"] = "34";
			
			foreach($employeeAges as $key => $value) {
			  echo "Name: $key, Age: $value
			";
			}
				1
				1
				1
				0
				
				1
				1
				0
				
				1
				1
				1
				
				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	return	return *expression*;	if ($i == 0) return;	2

EJS2	goto, label	goto *label*;
		.
		.
		label:
			loop1:
			  $x++;
			  if ($x < $y) goto loop1;
				0
				1
				2

EJS3	break	break;	if ($i > 10) break;	2
EJS4	exit function	If (condition) exit;	if ($x < 0) exit("Exit!");	2

EJS5	continue	continue;	while (list($key, $value) = each($arr)) {
			  if (!($key % 2)) {
			  continue;
			  }
			  do_something_odd($value);
			}
				1
				
				1
				1
				0
				1
				0

EXPRESSION STATEMENTS

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT
EES1	function call	<function_name> (<parameters>);	read_file ($name);	1
EES2	assignment statement	<name> = <value>;	$x = $y; $var = 'Joe'; $a = 1; $b = 2; $c = 3;	1 1 3

DECLARATION (DATA) LINES				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**

DDL1	function prototype	*functionname*($*var1,$var2,...,$varX*) {
		<statements>
		<statements>
		<statements>
		}
			function prod($a,$b) {
			$hello = "Hello World!";
			$a_number = 4;
			$anotherNumber = 8;
			}
				1
				1
				1
				1
				0

	variable declaration	$<name>;	$hello;	1

COMPILER DIRECTIVES				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**
CDL1	directive types	include <library_name>	include(test.php);	1
		include_once<library_name>	include_once(foo.php);	1
		require<library_name>	require(testfile.php);	1
		require_once<library_name>	require_once(filename.php);	1
		bool define (string $name, mixed $value [, bool $case_insensitive])	define("CONSTANT", "Hello");	1

		declare (directive)
		statement
			declare(ticks=2) {
			for ($x = 1; $x < 50; ++$x) {
			echo similar_text(md5($x), md5($x*$x)), "
			;";
			}
			}
				1
				1
				1
				0
				0