Definition Checklist for Source Statement Counts

Definition name: _____ Date: _____

_____ Originator: _____

Measurement unit: **Physical source lines** ☐
Logical source statements ☐

Statement type	**Definition** ☐ **Data array** ☐		**Includes**	**Excludes**
	When a line or statement contains more than one type, classify it as the type with the highest precedence.			
1	Executable	**Order of precedence ->** 1		
2	Nonexecutable			
3	Declarations	2		
4	Compiler directives	3		
5	Comments			
6	On their own lines	4		
7	On lines with source code	5		
8	Banners and nonblank spacers	6		
9	Blank (empty) comments	7		
10	Blank lines	8		
11				
12				

How produced	**Definition** ☐ **Data array** ☐	**Includes**	**Excludes**
1	Programmed		
2	Generated with source code generators		
3	Converted with automated translators		
4	Copied or reused without change		
5	Modified		
6	Removed		
7			
8			

Origin	**Definition** ☐ **Data array** ☐	**Includes**	**Excludes**
1	New work: no prior existence		
2	Prior work: taken or adapted from		
3	A previous version, build, or release		
4	Commercial, off-the-shelf software (COTS), other than libraries		
5	Government furnished software (GFS), other than reuse libraries		
6	Another product		
7	A vendor-supplied language support library (unmodified)		
8	A vendor-supplied operating system or utility (unmodified)		
9	A local or modified language support library or operating system		
10	Other commercial library		
11	A reuse library (software designed for reuse)		
12	Other software component or library		
13			
14			

Usage	**Definition** ☐ **Data array** ☐	**Includes**	**Excludes**
1	In or as part of the primary product		
2	External to or in support of the primary product		
3			

Figure 3-2 Definition Checklist for Source Statement Counts

Definition name: _____

_____

Delivery Definition ☐ Data array ☐	**Includes**	**Excludes**
1 Delivered		
2 Delivered as source		
3 Delivered in compiled or executable form, but not as source		
4 Not delivered		
5 Under configuration control		
6 Not under configuration control		
7		

Functionality Definition ☐ Data array ☐	**Includes**	**Excludes**
1 Operative		
2 Inoperative (dead, bypassed, unused, unreferenced, or unaccessed)		
3 Functional (intentional dead code, reactivated for special purposes)		
4 Nonfunctional (unintentionally present)		
5		
6		

Replications Definition ☐ Data array ☐	**Includes**	**Excludes**
1 Master source statements (originals)		
2 Physical replicates of master statements, stored in the master code		
3 Copies inserted, instantiated, or expanded when compiling or linking		
4 Postproduction replicates—as in distributed, redundant, or reparameterized systems		
5		

Development status Definition ☐ Data array ☐	**Includes**	**Excludes**
Each statement has one and only one status, usually that of its parent unit.		
1 Estimated or planned		
2 Designed		
3 Coded		
4 Unit tests completed		
5 Integrated into components		
6 Test readiness review completed		
7 Software (CSCI) tests completed		
8 System tests completed		
9		
10		
11		

Language Definition ☐ Data array ☐	**Includes**	**Excludes**
List each source language on a separate line.		
1 _____		
2 Job control languages _____		
3 _____		
4 Assembly languages _____		
5 _____		
6 Third generation languages _____		
7 _____		
8 Fourth generation languages _____		
9 _____		
10 Microcode _____		
11		

Figure 3-2 Definition Checklist for Source Statement Counts, Page 2

Definition name: _____		Includes	Excludes
Clarifications (general)	**Listed elements are assigned to statement type –>**		
1 Nulls, continues, and no-ops			
2 Empty statements (e.g., ";;" and lone semicolons on separate lines)			
3 Statements that instantiate generics			
4 Begin…end and {…} pairs used as executable statements			
5 Begin…end and {…} pairs that delimit (sub)program bodies			
6 Logical expressions used as test conditions			
7 Expression evaluations used as subprogram arguments			
8 End symbols that terminate executable statements			
9 End symbols that terminate declarations or (sub)program bodies			
10 Then, else, and otherwise symbols			
11 Elseif statements			
12 Keywords like procedure division, interface, and implementation			
13 Labels (branching destinations) on lines by themselves			
14			
15			
16			
Clarifications (language specific)			
Ada			
1 End symbols that terminate declarations or (sub)program bodies			
2 Block statements (e.g., begin…end)			
3 With and use clauses			
4 When (the keyword preceding executable statements)			
5 Exception (the keyword, used as a frame header)			
6 Pragmas			
7			
8			
9			
Assembly			
1 Macro calls			
2 Macro expansions			
3			
4			
5			
6			
C and C++			
1 Null statement (e.g., ";" by itself to indicate an empty body)			
2 Expression statements (expressions terminated by semicolons)			
3 Expressions separated by semicolons, as in a "for" statement			
4 Block statements (e.g., {…} with no terminating semicolon)			
5 "{", "}", or "};" on a line by itself when part of a declaration			
6 "{" or "}" on line by itself when part of an executable statement			
7 Conditionally compiled statements (#if, #ifdef, #ifndef)			
8 Preprocessor statements other than #if, #ifdef, and #ifndef			
9			
10			
11			
12			

Figure 3-2 Definition Checklist for Source Statement Counts, Page 3

Definition name: _____		Includes	Excludes
CMS-2	**Listed elements are assigned to**		
1 Keywords like SYS-PROC and SYS-DD	**statement type –>**		
2			
3			
4			
5			
6			
7			
8			
9			
COBOL			
1 “PROCEDURE DIVISION”, “END DECLARATIVES”, etc.			
2			
3			
4			
5			
6			
7			
8			
9			
FORTRAN			
1 END statements			
2 Format statements			
3 Entry statements			
4			
5			
6			
7			
8			
JOVIAL			
1			
2			
3			
4			
5			
6			
7			
8			
Pascal			
1 Executable statements not terminated by semicolons			
2 Keywords like INTERFACE and IMPLEMENTATION			
3 FORWARD declarations			
4			
5			
6			
7			
8			
9			

Figure 3-2 Definition Checklist for Source Statement Counts, Page 4

Definition name: _____		Includes	Excludes
	Listed elements are assigned to statement type –>		
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			

Summary of Statement Types

Executable statements

Executable statements cause runtime actions. They may be simple statements such as assignments, goto's, procedure calls, macro calls, returns, breaks, exits, stops, continues, nulls, no-ops, empty statements, and FORTRAN's END. Or they may be structured or compound statements, such as conditional statements, repetitive statements, and "with" statements. Languages like Ada, C, C++, and Pascal have block statements [begin…end and {…}] that are classified as executable when used where other executable statements would be permitted. C and C++ define expressions as executable statements when they terminate with a semicolon, and C++ has a <declaration> statement that is executable.

Declarations

Declarations are nonexecutable program elements that affect an assembler's or compiler's interpretation of other program elements. They are used to name, define, and initialize; to specify internal and external interfaces; to assign ranges for bounds checking; and to identify and bound modules and sections of code. Examples include declarations of names, numbers, constants, objects, types, subtypes, programs, subprograms, tasks, exceptions, packages, generics, macros, and deferred constants. Declarations also include renaming declarations, use clauses, and declarations that instantiate generics. Mandatory begin…end and {…} symbols that delimit bodies of programs and subprograms are integral parts of program and subprogram declarations. Language superstructure elements that establish boundaries for different sections of source code are also declarations. Examples include terms such as PROCEDURE DIVISION, DATA DIVISION, DECLARATIVES, END DECLARATIVES, INTERFACE, IMPLEMENTATION, SYS-PROC, and SYS-DD. Declarations, in general, are never required by language specifications to initiate runtime actions, although some languages permit compilers to implement them that way.

Compiler Directives

Compiler directives instruct compilers, preprocessors, or translators (but not runtime systems) to perform special actions. Some, such as Ada's pragma and COBOL's COPY, REPLACE, and USE, are integral parts of the source language. In other languages like C and C++, special symbols like # are used along with standardized keywords to direct preprocessor or compiler actions. Still other languages rely on nonstandardized methods supplied by compiler vendors. In these languages, directives are often designated by special symbols such as #, $, and {$}.

Figure 3-2 Definition Checklist for Source Statement Counts, Page 5