COUNTING LINES OF CODE, CONFUSIONS, CONCLUSIONS, AND RECOMMENDATIONS

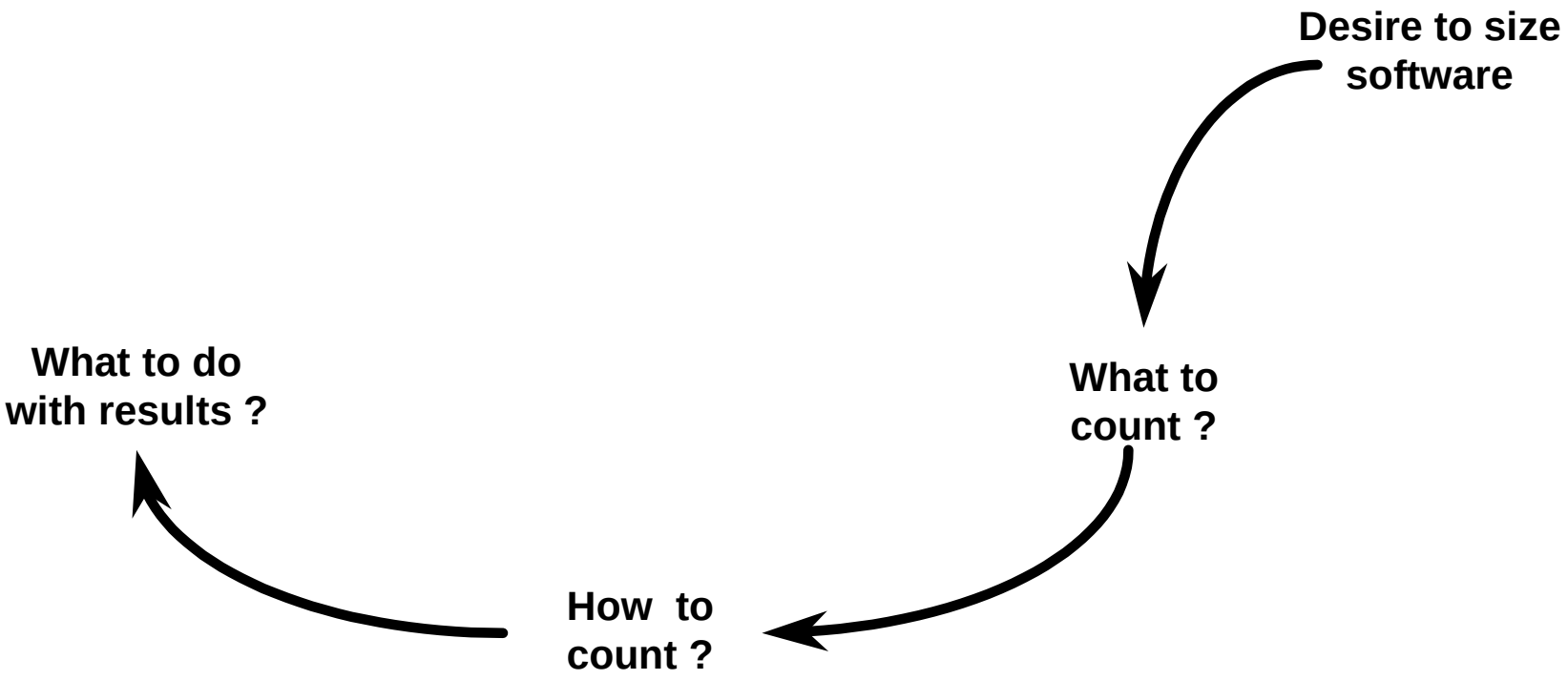
Briefing to the

3rd Annual REVIC User's Group Conference

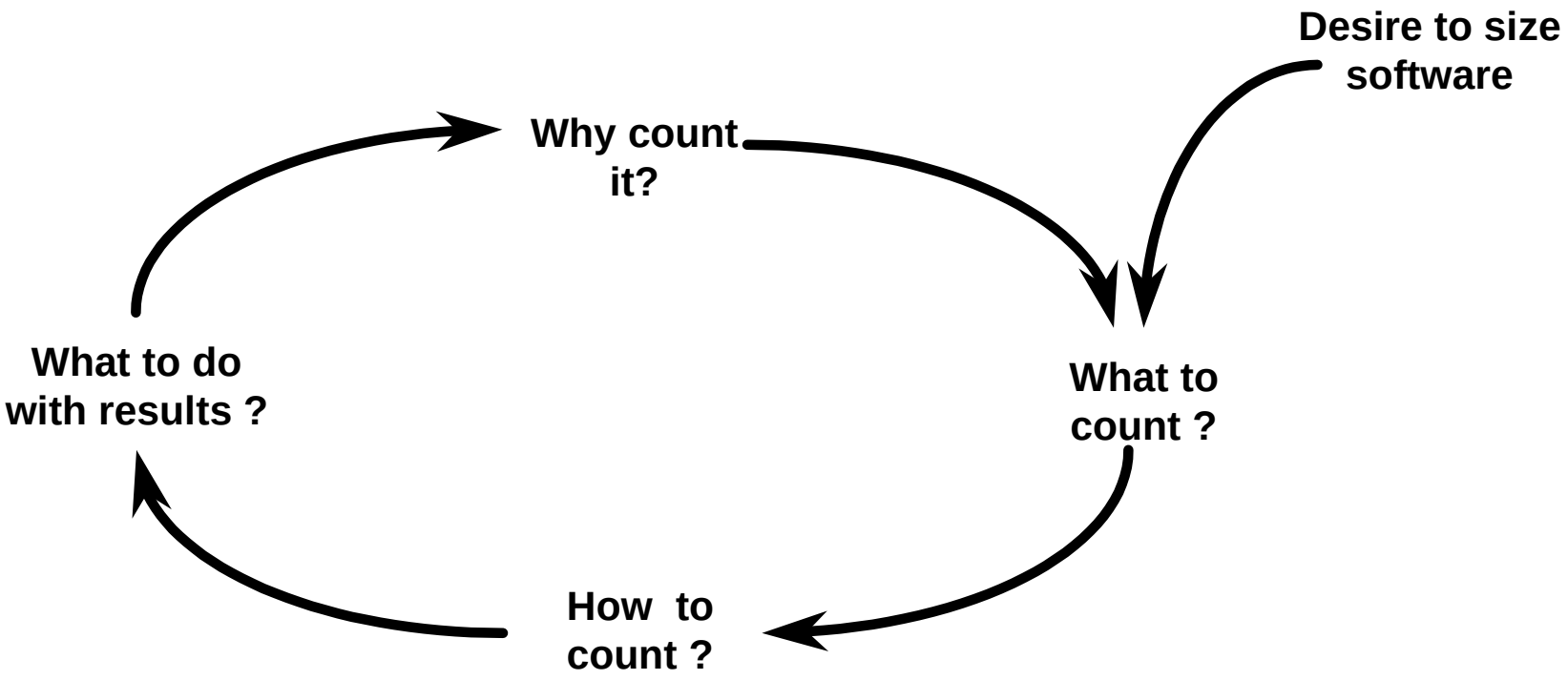
January 10-12, 1990

George E. Kalb
Northrop Grumman Corporation
Electronic Sensors & Systems Sector (ESSS)

THE PROCESS OF COUNTING LINES OF CODE

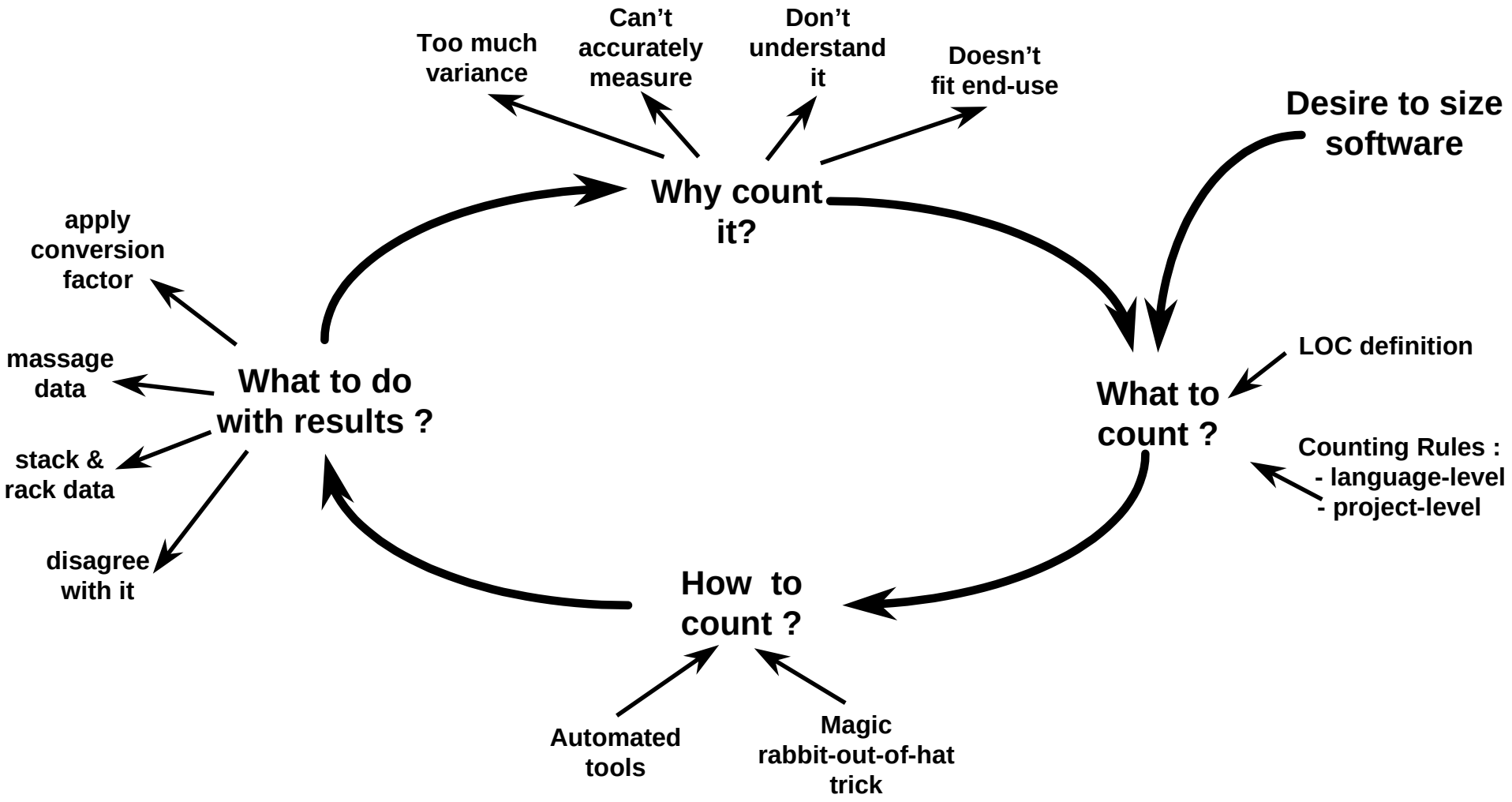


USING A LOC DEFINITION



Each activity will ultimately determine the success of a future activity.

PROBLEMS ASSOCIATED WITH COUNTING LOC



WHY COUNT IT ?

- **SIMPLE : LOC definition widely utilized**
- **PROBLEM : No consistency in LOC definition**
- **RESULT : Orders of magnitude variances**
- **PROBLEM : Over 300 different languages**
  - **MicroCode - Assembly - HOL - 4GL**
- **SOLUTION : Find a LOC definition which is**
  - **suitable for all languages**
  - **suitable for the most utilized languages**
  - **suitable for only one language (Ada)**
  - **not a LOC definition at all**
    - **alternate approach**

WHY COUNT IT ?
ALTERNATIVES TO LOC

- **Function Points , Size of Listings, Size of Executable, Complexity Measures, Number of Components, others …….**

- **Utility of alternative creates another set of problems**
  - **ease of use**
  - **breadth of use**
  - **understandabilty**
  - **reliability**
  - **automatability**
  - **repeatability**

WHY COUNT IT ?

- **CONCLUSIONS : Continue to count LOC**
- **CONFUSION : Define a LOC definition for each programming language individually**
  - **promotes use of conversion factors for comparisons**
- **RECOMMENDATION : Reduce confusion by using a LOC definition which may be applied across a variety of languages**

WHAT TO COUNT ?
PANDORA'S BOX SYNDROME

LANGUAGE-LEVEL

- Physical Lines
- Machine Instructions
- Semicolons
- Source Statements
- Executable Lines
- Data Declarations
- Deliverable Source Instructions

LOC Definitions
&
Language-Level Counting Rules

PROJECT-LEVEL

- Include Files
- Generics
- Reuse Code
- Unused Code, Dead Code
- Deleted Code
- Support Software
- Patch Files, Data Files
- Test Software, Command Files
- Multiple Language Implementations
- Off-The-Shelf Software
- Non-Deliverable Software
- Modified Software

Project-Level Counting Rules

LANGUAGE-LEVEL DEFINITION
PHYSICAL LINES

- **Tally of carriage-returns (card images) within source code regardless of information content**
- **BENEFITS :**
  - **easy to understand & automate**
  - **supports other LOC definitions based on physical lines**
- **CONFUSION :**
  - **low utility as a sizing measure**
  - **yields no visibility into information contents**
- **CONCLUSION : Not a good indicator of software size**

LANGUAGE-LEVEL DEFINITIONS
MACHINE INSTRUCTIONS

- **Tally of native processor specific instructions**
- **BENEFITS : supports memory utilization studies**
- **CONFUSION :**
  - **count assembly directly**
    - **how to handle Pseudo Ops, Data Declarations, Macro Definitions**
    - **processor architecture differences :**
      (RISC vs. CISC ; instruction lengths)
  - **count HOL or MicroCode**
    - **promotes use of conversion factors**
    - **compiler dependencies**
    - **recursion penalized**
- **RECOMMENDATION : Count that which has been directly produced by the development staff**

LANGUAGE-LEVEL DEFINITION
THE ALMIGHTY SEMICOLON

- **Tally of semicolons found within source code**
- **BENEFITS : less susceptibility to coding style ?**
- **CONFUSION : Which ‘;’ definition to count**
  - **all ‘;’s**
  - **non-literal ‘;’s**
  - **terminal ‘;’s**
  - **limited terminal ‘;’s**
  - **essential ‘;’s**
  - **package body ‘;’s**
  - **mixtures of above**
- **promotes use of conversion factors**
- **relationship to non-Ada language**
- **degradation of understandability and automatability**

LANGUAGE-LEVEL DEFINITION
THE ALMIGHTY SEMICOLON

Ada	Jovial (J73)	1750A Assembler	Pascal	Lisp, C, FORTRAN
A,B,C : integer; vs. A : integer; B : integer; C : integer; vs. null; null; null;	Define thenn = ";" IF (a=b) thenn a := 11 ;	LD A ; load it ST A ; store it	IF a = 1 THEN IF b <> 1 THEN c := 1 ELSE ELSE c := 0 ; vs. ; ; ; ; ; ; ;	?
";" is a statement terminator	";" may be overloaded	";" is a comment delimiter	";" is a statement separator	";" is not applicable to LOC counting methods

THE ALMIGHTY SEMICOLON

- **CONFUSION :**
  - **consistency of application**
  - **degree of regulation by Coding Std or SQA**
  - **effects of overloading or replacement characters**
  - **susceptibility to future versions or enhancements to language**
  - **susceptibility to coding standard and style variances**
- **CONCLUSION : Do not define a LOC based upon a language specific statement delimiter or syntax**

LANGUAGE-LEVEL DEFINITION SOURCE STATEMENTS

- **Tally of logical language dependent statements (NCSS)**
- **BENEFITS : less susceptible to coding style than physical lines**
- **CONFUSION :**
  - **temptation to use statement delimiters**
  - **promotes subclassification of statements**
    - **data declarations vs. executable**
    - **JCL & Compiler Directives**
    - **program structors**
  - **promotes weighting of various subclassifications**
  - **definition varies widely across programming languages**

LOGICAL SOURCE STATEMENTS (cont'd)

Should :

```
DEF TABLE KDTRIG (0 : 256)  S  =
-2048,  -2047,  -2046,  -2042,  -2038,  -2033,  -2026,  -2018,
-2009,  -1998,  -1987,  -1974,  -1960,  -1945,  -1928,  -1911,
-1892,  -1872,  -1851,  -1829,  -1806,  -1782,  -1757,  -1730,
-1703,  -1674,  -1645,  -1615,  -1583,  -1551,  -1517,  -1483,
-1448,  -1412,  -1375,  -1338,  -1299,  -1260,  -1220,  -1179,
-1138,  -1096,  -1653,  -1009,   -965,   -921,   -876,   -830,
 -784,   -737,   -690,   -642,   -595,   -546,   -498,   -449,
 -400,   -350,   -301,   -251,   -201,   -151,   -100,    -50,
    0,     50,    100,    151,    201,    251,    301,    350,
  400,    449,    498,    546,    595,    642,    690,    737,
  784,    830,    876,    921,    965,   1009,   1053,   1096,
 1138,   1179,   1220,   1260,   1299,   1338,   1375,   1412,
 1448,   1448,   1517,   1551,   1583,   1615,   1645,   1674,
 1703,   1730,   1757,   1782,   1806,   1829,   1851,   1872,
 1892,   1911,   1928,   1945,   1960,   1974,   1987,   1998,
 2009,   2018,   2026,   2033,   2038,   2042,   2046,   2047,
 2047,   2047,   2046,   2042,   2038,   2033,   2026,   2018,
 2009,   1998,   1987,   1974,   1960,   1945,   1928,   1911,
 1892,   1872,   1851,   1829,   1806,   1782,   1757,   1730,
 1703,   1674,   1645,   1615,   1583,   1551,   1517,   1483,
 1448,   1412,   1375,   1338,   1299,   1260,   1220,   1179,
 1138,   1096,   1653,   1009,    965,    921,    876,    830,
  784,    737,    690,    642,    595,    546,    498,    449,
  400,    350,    301,    251,    201,    151,    100,     50,
    0,    -50,   -100,   -151,   -201,   -251,   -301,   -350,
 -400,   -449,   -498,   -546,   -595,   -642,   -690,   -737,
 -784,   -830,   -876,   -921,   -965,  -1009,  -1053,  -1096,
-1138,  -1179,  -1220,  -1260,  -1299,  -1338,  -1375,  -1412,
-1448,  -1448,  -1517,  -1551,  -1583,  -1615,  -1645,  -1674,
-1703,  -1730,  -1757,  -1782,  -1806,  -1829,  -1851,  -1872,
-1892,  -1911,  -1928,  -1945,  -1960,  -1974,  -1987,  -1998,
-2009,  -2018,  -2026,  -2033,  -2038,  -2042,  -2046,  -2047,
-2048  ;
```

Be Equivalent To :

```
A := B + 1 ;
```

SOURCE STATEMENTS CONCLUSIONS

- A good indicator of software size
- Automated counting tool may require a sophisticated (compiler-like) parser
  - increased complexity, cost
  - creation of inconsistencies
  - in-house development results in :
    - decreased reliability, portability
    - proliferation of simpler tools

LANGUAGE-LEVEL DEFINITIONS
EXECUTABLE STATEMENTS

- **Tally of any statement which upon compilation produces executable run-time code**
- **BENEFITS : supports sizing of executable image**
- **CONFUSION :**
  - **penalization of highly structured languages, & MicroCode**
  - **can be extremely compiler dependent**
  - **provides misleading data**
    - **in-line, overlays, optimizations, recursion**
  - **definition of executable varies across languages**
  - **difficult to automate**

EXECUTABLE STATEMENTS

BEGIN, END, TASK, FORWARD **<------- Executable Keywords ?**

FORMAT, DEFINE, WRITELN **<------- Executable, sometimes ?**

null; NOOP { } **<------- Executable statement ?**

```
type ABC_TYPE                          <------- Compiler implementation may
  record                                          produce executable code.
      A : integer  := 1;
      B : real     := 1.2;
      C : integer  := 2;
 end_record;
abc : ABC_TYPE := (5, 2.2, 3) ;
```

Microcode Languages **<------- One statement produces multiple executable actions.**

LANGUAGE-LEVEL DEFINITIONS
DATA DECLARATIONS

- **Tally of any statement which reserves memory at compile-time**
- **BENEFITS : Infer storage requirements**
- **CONFUSION : How to handle**
  - **type, label, constant, define, rep. clause, use, implicit declarations**
- **RECOMMENDATION : Useful information, must improve definition to address automation concerns**

LANGUAGE-LEVEL DEFINITIONS
COMPILER DIRECTIVES

- **Tally of statements embedded within the source code that direct the compilation process**
  - **logical vs. physical ?**
- **Not addressed by LOC definition or excluded from count of LOC**
  - **Ada Pragma statements the only exception**
- **Component of the Software Product which requires**
  - **effort to develop & debug**
- **RECOMMENDATION : Include Compilation Directives within LOC count**
  - **maintaining a separate count increases tool complexity**

LANGUAGE-LEVEL DEFINITIONS DELIVERABLE SOURCE INSTR.

- **Physical lines of code exclusive of comments & blank lines**
- **BENEFITS : may be applied across languages,**
  - **easy to understand, automate**
  - **does not require sub-classification of source code**
  - **less susceptible to language enhancements**
  - **incorporated into many software code models**
- **CONFUSION : very dependent on coding style**
- **RECOMMENDATION : Use a source code formatter to normalize input**

LANGUAGE-LEVEL DEFINITION SUMMARY

- **Survey indicates most organizations have automated counting tools developed in-house**
  - **which LOC definition utilized**
  - **consistency of use**
  - **development quality and tool reliability**
- **RECOMMENDATION : Continue use of DSI definition**
  - **limit the sub-classifications of components**
  - **count exactly that produced by the S/W staff**
  - **develop / iterate LOC definitions in light of automation problems**

LANGUAGE-LEVEL DEFINITIONS SUMMARY

- **CONCLUSIONS : It is generally accepted that for S/W sizing :**
  - **do not count blank lines**
  - **do not count comments**
- **RECOMMENDATION : Do not include them within LOC count, maintain separate count on a physical line basis**

WHAT IS A COMMENT ?

- **Rules of use is highly language dependent**
  - column oriented fields
  - one vs. two character delimiters
    - exception handling
  - termination via EOLN or companion delimiter
  - multiple delimiter designations
    - mutual exclusion rules
  - overloading of delimiters
  - replacement characters
  - competition with string delimiters

WHAT IS A COMMENT
SUB-CLASSIFICATIONS

(* ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ *)	**<------- A comment**
(* ~ ~ ~ *) (* ~ ~ ~ ~ ~ ~ ~ ~ ~ *)	**<------- One comment or two ?**
code (* ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ *)	**<------- An embedded comment ?**
(* ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~	**<------- One comment or many ?**
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~	**<------- One commented source statement ?**
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ *)	**<------- # sentences = # comments**
(* *)	**<------- A "real" comment ?**
(* code *)	**<------- A "real" comment ?**
(* ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~	**<------- One, two, or three comments ?**
I'm a blank line	
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ *)	

PROJECT-LEVEL DEFINITIONS

- **What to count is often dependent on end-use of sizing data**
  - **count all code**
  - **count all delivered code**
  - **count all new code**
  - **count deleted code**
- **RECOMMENDATION : A LOC counting automated tool should be constructed to be independent of project-level concerns**

PROJECT-LEVEL DEFINITIONS

- **CONFUSION : What level of visibility should automated tools address :**
  - **file-by-file**
  - **module-by-module**
  - **statement-by-statement**
  - **line-by-line**

  - difficulty to automate?
  - **multi-language implementations**
  - **sub-classifications**
  - **project-level issues**

HOW TO COUNT ?

- **Automated Tools are preferred**
  - **development environment**
  - **level of complexity**
  - **cohesion**
  - **versatility**
  - **internal algorithm dependencies**
- **CONCLUSION : LOC Definitions have fallen short of providing a foundation for consistency of automation**

WHAT TO DO WITH THE ANSWER ?

- **Does the LOC definition support the variety of end-users ?**
  - **project size**
  - **cost & schedule modeling**
  - **productivity analysis**
- **Does the LOC definition depend on ‘when’ the measurement was taken ?**
  - **compilation**
  - **unit-level test**
  - **integration**
  - **FQT / PCA**

OVERALL RECOMMENDATIONS

- **Promote use of DSI**
  - spans multiple language
  - ease of automation
  - understandability
- **Future LOC definitions must**
  - address automation issues
    - tool complexity & reliability
  - support end-use applications
  - minimize use of conversion factors

OVERALL RECOMMENDATIONS

- **Encourage automated tools to**
  - **report use of comments (whole & embedded)**
  - **include compiler directives in LOC count**
  - **limit sub-classifications of code**
  - **de-couple from language specific attributes, project-level issues**
    - **minimize delimiter dependencies**
    - **file-by-file visibility**

SOURCE OF INFORMATION CONSULTED

- **Robert E. Park, Software Engineering Institute**
- **Donald J. Reifer, Reifer Consultants Inc.**
- **Karen E. Sivley, Magnavox Electronics Systems Company**
- **“Establishing Enterprise Software Productivity and Quality Programs”, Capers Jones, Software Productivity Research, Inc. 1986-1987.**
- **“Standard for Software Productivity Metrics”, IEEE Computer Society, Aug. 1990.**
- **Software Engineering A Practitioner’s Approach, Roger S. Pressman, McGraw-Hill, 1987.**
- **Software Engineering Economics, Barry W. Boehm, Prentice-Hall, 1981.**
- **Software Metrics : Establishing a Company-Wide Program, Robert G. Grady & Deborah L. Caswell, Prentice-Hall, 1987.**

Demonstration of Software Product CodeCount™ following today's presentations