ESSS

Automated Collection of Software Sizing Data

Briefing to the
International Society of Parametric Analysts
Southern California Chapter
October 16, 1996

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

George E. Kalb

ESSS

Mr. Kalb is a Fellow Engineer with over fourteen years at Northrop Grumman ESSS and is responsible for advanced software technologies and business development activities. His engineering experience includes Principal Investigator for the Avionics Fault Tolerant Software (AFTS) Program and the associated Ada Technology Insertion Program (ATIP) and has been involved with numerous CR&D contracts at WL/AAAF-3. His management experience includes the highly successful Generic Dis-Assembler Program and numerous Proposal Preparation activities where on-time, on-budget performance is critical. Past engineering activities include AN/APG-68 Operational Flight Program (OFP) software development, J-STARS & SDI Ground Base Radar proposal signal processing software architect, YF-22 & YF-23 processor timing and sizing studies, and lead software engineer for the E-3A Programmable VHSIC Signal Processor Operational Software development.

Mr. Kalb is responsible for conceiving, developing, and successfully distributing a set of source lines of code counting tools that are the foundation for ESSS's software metrics program.

Mr. Kalb is also a faculty member at the G.W.C. Whiting School of Engineering of the Johns Hopkins University since 1988 where he teaches a graduate-level course in Software Engineering and the Software Lifecycle and was the recipient of the 1995 Johns Hopkins University "Excellence in Teaching" award.

THE MANY USES OF SIZING DATA

- **Software Cost Models**
  - cost estimation
  - project post mortem
- **Metrics Programs**
  - historical project data
  - productivity
  - defect rate
- **Development Projects**
  - amount of reuse software
  - estimated / actuals overall project size
- **Software Quality Assurance**
  - amount of commentation in source code product
  - use of language constructs
  - defect rate calculations
- **Re-Engineering Activities**
  - language translations
  - platform translation
- **Compiler Efficiency Analysis**

ESSS

Who?
What?
How?
When?
Maturity of SW?

WHAT TO COUNT?

LANGUAGE-LEVEL

- Deliverable Source Instructions
- Machine Instructions
- Semicolons
- Logical Source Statements
- Data Declarations
- Comments
- Compiler Directives

LOC Definitions
&
Language-Level Counting Rules

PROJECT-LEVEL

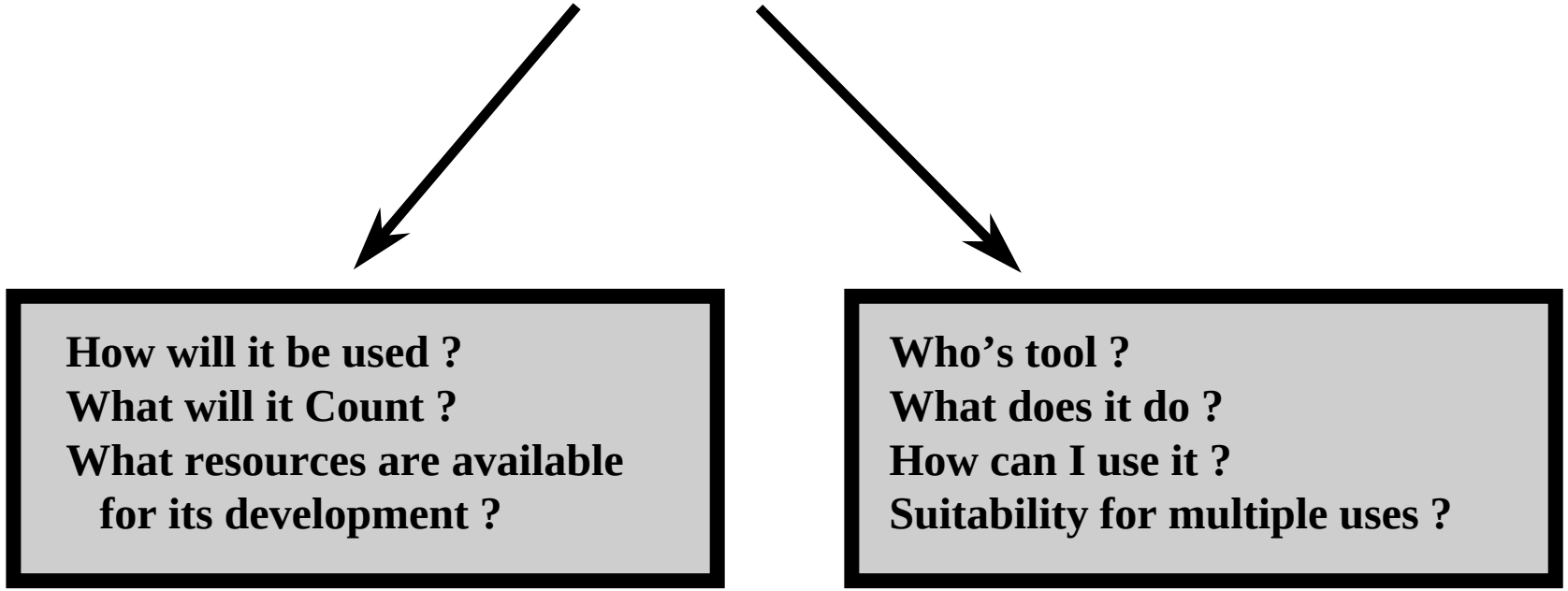
- Include Files
- Generics
- Reuse Code
- Unused Code, Dead Code
- Support Software
- Patch Files, Data Files
- Test Software, Command Files
- Multiple Language Implementations
- Off-The-Shelf Software
- Non-Deliverable Software
- Modified Software

Project-Level Counting Rules

QUEST FOR AUTOMATION

ESSS

- **Automation of collection of software sizing :**
  - decreases subjectivity (guesstimates)
  - improves accuracy and consistency of information
  - reduction in time and effort required to gather data
- **Solution : Let's go make / buy a tool**



PROBLEMS WITH AUTOMATED LOC COUNTING TOOLS

ESSS

- **Developed to satisfy a short-term project specific need**
  - **no documentation**
  - **no visibility into LOC definition used**
  - **use of rule-of-thumb conversion factor(s)**
  - **incorrect counting**
    - **misclassification of language constructs**
  - **produced ambiguous output**
  - **developed using substandard practices**
  - **un-validated, un-maintainable**
  - **not portable**
    - **duplication of effort**
    - **proliferation of poor quality "use at your own risk" LOC counting tools**
- **Variances in validity and interpretation of sizing data across enterprise**

HOW TO BUILD A BETTER LOC COUNTING TOOL

- **LOC definition utilized must :**
  - **span multiple programming language**
    - **3GL HOL (mixed language implementations)**
    - **assembly**
    - **microcode**
    - **test languages**
  - **be easily interpreted**
  - **provide for project utility**
  - **be compatible with software cost model use**
  - **be amenable to automation**

- **Deliverable Source Instruction (DSI) concept utilized**
  - **easy to understand & automate**
  - **may be applied consistently across programming languages**
  - **does not require subclassification of language constructs**
  - **incorporated into many software cost models**

WHY USE DSI's

- **Popular alternatives :**
  - **Function / Feature Points**
  - **Machine-Level instructions**
  - **Semicolons**
  - **Logical Source Statements**
  - **Executable Lines**
  - **Data Declarations**
- **What about other useful information to collect :**
  - **comments**
  - **blank lines**
  - **reserve word usage**
  - **overall size (independent of information content)**

FUNCTION / FEATURE POINTS

ESSS

- **Estimation technique not well integrated across the enterprise (outside of the cost estimation community)**
  - **ease of use**
  - **breadth of use**
  - **understandability**
  - **reliability**
  - **repeatability**
  - **automatability**

- **Solutions**
  - **increase enterprise-wide training**
  - **address customer acceptance of methodology**
  - **adjust metrics programs to become function point based**
  - **provide a gradual cultural replacement of LOC for function point processes**

MACHINE-LEVEL INSTRUCTIONS

ESSS

- **Tally of native processor specific instructions**
- **Confusions :**
  - **count words of memory in executable image**
    - **decoupled from source code product**
  - **count assembly language instructions**
    - **how to handle Pseudo Ops, data declarations, macro definitions, etc.**
    - **processor architecture differences**
      (e.g., CISC vs. RISC, instruction word lengths, etc.)
  - **count HOL**
    - **promotes use of conversion factors**
    - **compiler dependencies**
    - **recursion penalization**
- **Count that which has been directly produced by the development staff**

SEMICOLONS

- **Tally of semicolons found within source code**
- **Confusions**
  - **which “;” definition to use ?**
    - **all “;”s**
    - **non-literal “;”s**
    - **terminal “;”s**
    - **limited terminal “;”s**
    - **essential “;”s**
    - **package body “;”s**
    - **mixtures of above**
- **Leads to degradation of understandability and automatability**

SEMICOLONS (cont'd)

Ada	Jovial (J73)	Mil-Std-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

LOGICAL SOURCE STATEMENTS

ESSS

- **Tally of logical programming language dependent statements (NCSS)**
- **Confusions :**
  - **automation promotes use of statement delimiters**
  - **promotes subclassification of statements**
    - **executable vs. data declarations**
    - **JCL & compiler directives**
    - **program structors**
  - **promotes weighting of various subclassifications**
  - **definition varies widely across programming languages**
- **A good indicator of software size, however**
  - **leads to degradation of understandability and automatability**
  - **inconsistencies across span of 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  ;
```

ESSS

Be Equivalent To :

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

EXECUTABLE LINES

ESSS

- **Tally of any statement which upon compilation produces executable run-time code**
- **Confusions :**
  - **penalization of highly structured languages, parallel languages**
  - **is extremely compiler dependent**
  - **provides misleading data :**
    - **in-line, overlays, optimizations, recursion**
  - **definition of executable varies across programming languages**
  - **difficult to automate**
- **A fair indicator of software size, however**
  - **leads to degradation of understandability and automatability**
  - **inconsistencies across span of programming languages**

EXECUTABLE LINES (cont'd)

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

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

null; NOOP { } <------- Executable Statement ?

```
type ABC_TYPE
  record
       A : integer  := 1;
       B : real     := 1.2;
       C : integer  := 2;
 end_record;
abc : ABC_TYPE := (5, 2.2, 3) ;
```

<------- Compiler implementation may produce executable code

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

DATA DECLARATIONS

ESSS

- **Tally of any statement which upon compilation reserves memory**
- **Confusions :**
  - **how to handle :**
    - **type**
    - **label**
    - **constant**
    - **define**
    - **representation clauses**
    - **implicit declarations**
- **A fair indicator of software size when coupled to executable lines, however**
  - **leads to degradation of understandability and automatability**
  - **inconsistencies across span of programming languages**

COMPILER DIRECTIVES

- **Tally of statements embedded within the source code that direct the compilation process**
- **Confusions :**
  - **not addressed by LOC definition or excluded from LOC count**
  - **syntax & composition varies widely across compiler vendors and programming language**
  - **logical vs. physical counting rules**
  - **component of the software product that requires effort to develop, test, and document**
  - **maintaining a separate count leads to subclassification of LOC definition**
- **Include compilation directives within count of LOC & software sizing process**

COMMENTS IN SOURCE CODE

- **Generally accepted that for software sizing :**
  - **do not count comments**
  - **do not count blank lines**
- **However, maintain & report usage as part of LOC counting process**
- **Counting rules are highly programming language dependent :**
  - **column oriented fields**
  - **one vs. two character delimiters**
  - **termination via EOLN or companion delimiter**
  - **multiple delimiters sets per languages**
  - **overloading of delimiters**
  - **replacement characters**
  - **competition with string delimiters**
  - **nesting of comments & comment delimiters**

SUBCLASSIFICATION OF COMMENTS

ESSS

(* ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ *)	**<------- 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 ?**
(* ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ***< blank line >*** **~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ *)**	**<------- One, two, or three comments ?**

PROJECT-LEVEL COUNTING RULES

- **What to count is often dependent on end-use of sizing data**
  - count all source code
  - count all new source code
  - count reuse and off-the-shelf software
  - count deleted code
  - count support software
- **The process of collecting sizing information should not be coupled to project-level concerns**
  - concentrate on automation of language-level collection processes
  - project-level concerns addressed as a post process

ESSS LOC COUNTING TOOLSET

ESSS

- **Set of automated LOC counting tools (CodeCount™)**
- **Programming languages analyzed :**
  - **Ada (Mil-Std-1815A and Ada95)**
  - **Jovial J73 (Mil-Std-1589B)**
  - **FORTRAN**
  - **ANSI-C and C++**
  - **Pascal**
  - **two proprietary microcode languages**
- **Share common operational requirements**
  - **set up of input files**
  - **operation / execution**
  - **information content & format of output produced**
- **Developed and successfully used for over a decade**
- **Satisfies variety of end-user requirements (p.#3)**
- **Basis of metrics data collection process**

BENEFITS OF LOC COUNTING TOOLS REALIZED

ESSS

- **Elimination of language-level counting discrepancies**
- **Elimination of need to use ‘conversion factors’**
- **Provides “apples-to-apples” LOC comparisons**
  - **across multiple software development projects**
  - **across multiple programming languages**
- **Portability issue achieved**
  - **VAX (VMS & Ultrix)**
  - **SGI (UNIX)**
  - **SUN (Sun OS & Solaris)**
  - **HP (HP-UX)**
  - **PC (Windows NT)**
- **Significant reduction in data gathering effort**
- **Increased accuracy and consistency in sizing information**

LESSONS LEARNED

- **Develop/Acquire software sizing tools that consistently applies the same LOC definition across programming languages** ***ESSS***
- **Proliferate the same data gathering toolset across the enterprise**
- **Differentiate language-level counting rules (automated in tool) from project-level counting rules (policy & procedures)**
  - **decouple operation of LOC counting from project-level issues**
- **Subclassification of language constructs lead to :**
  - **difficulty in automation**
  - **decrease in understanding, confusion upon interpretation**
  - **misclassification**
  - **lower enterprise-wide utility**
- **Programming Languages will be revised**
  - **automated LOC counting tools must be enhancable**

LESSONS LEARNED

- **Limit complexity of software sizing tools**
  - **reduce parsing complexity via restricting analysis to source code that successfully compiles**
  - **decouple parsing / counting algorithms from coding standards & style requirements**
  - **decouple parsing / counting algorithms from tools that “prepocess” the source code input**
  - **decouple operation of the tool from project-level issues**
    - **what code to count**
    - **when to count the code**
    - **what is the efficiency or quality of the code**
  - **limit amount of LOC subclassifications to detect and report upon separately**
- **Software sizing tool should be :**
  - **easy to use**
  - **portable**
  - **support multiple end-user requirements**

SOURCE OF INFORMATION CONSULTED

ESSS

- **Robert E. Park, Software Engineering Institute**
- **Donald J. Reifer, Reifer Consultants Inc.**
- **“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.**
- **“The Pursuit of Accurate Source Lines of Code Sizing”, George E. Kalb, Proceedings of NAECON’88, (May 1988), vol. II, pp. 698-700.**
- **“Lines of Code Counting Tools”, George E. Kalb, Proceedings of the Second REVIC User’s Group Conference, (Jan. 1990), pp. 225-232.**
- **“Counting Lines of Code, Confusions, Conclusions, and Recommendations”, George E. Kalb, Proceedings of the Third REVIC User’s Group Conference, (Feb. 1991), pp. R-1 to R-28.**

Additional Slides for Backup

VARIANCES BETWEEN ";" LOC DEFINITIONS

- **500 Ada Source Files**
  - Physical Lines 197,754
  - Blank Lines 43,761
  - Comment Whole 107,520
    - Embedded 718
  - DSIs 46,473
  - All ";" 34,388
  - Terminal ";" 29,615
  - Limited Terminal ";" 25,469
- **41 Pascal Source Files**
  - Physical Lines 53,185
  - Blank Lines 8,270
  - Comment Whole 12,113
    - Embedded 4,977
  - DSIs 32,802
  - ";" Statement Separators 20,272