# Improving Software Cost Estimates Using the Univariate Model

Brian Opaska

OPS Consulting

# Outline

- Background on Univariate (Linear) Method
- Improvement Techniques
  - Code Count Metric
  - Actual SLOC Analysis
  - SLOC Normalization
- Summary

# Outline

- Background on Univariate (Linear) Method
- Improvement Techniques
  - Code Count Metric
  - Actual SLOC Analysis
  - SLOC Normalization
- Summary

# Background (1/2)

- Software effort was traditionally seen as a linear function based on the size of the product, where size could be SLOC or Function Points
- Software effort = Software Product Size x Productivity Factor (unit of effort per size)
- Quick and easy calculation
- Traceable back to actuals

# Background (2/2)

- Productivity factor is usually a composite factor based on historical actuals
  - Single factor can be derived from multiple languages
  - Group of factors based on actuals from specific languages

# Outline

- Background on Univariate (Linear) Method
- Improvement Techniques
  - Code Count Metric
  - Actual SLOC Analysis
  - SLOC Normalization
- Summary

# Univariate Method Improvement Techniques

- Three techniques to improve the consistency and accuracy of the Univariate Method
  - Using a reliable code counting sizing metric
  - Performing thorough analysis of actual SLOC count
  - Normalizing the SLOC

# What to Count? Physical/Logical SLOC Definitions

- SLOC counts are typically used as the sizing metric in the univariate model
- Physical SLOC definition (physical measure):
  - "Counts of physical lines describe size in terms of the physical length of the code as it appears when printed for people to read"[1]
  - "Sets of coded instructions terminated by pressing the enter key of a computer keyboard"[2]
- Logical SLOC definition (instructions):
  - "Counts of logical statements … attempt to characterize size in terms of the number of software instructions, irrespective of their relationship to the physical formats in which they appear"[1]

[1] *Software Size Measurement: A Framework for Counting Source Statements* – Robert Park
[2] *Software Cost Estimation with COCOMO II* – Barry W. Boehm

# Sizing Metric

- What to count?
  - Recommend Physical SLOC to be used as the sizing metric
  - Physical SLOC = Total Lines – Comment Only Lines – Blank Lines
- Why Physical SLOC?
  - Advantages of using Physical SLOC
    - Easy to count
      - Clear beginning and ending points
    - Consistency of counts
      - Physical SLOC consistent across code counting tools
      - For Logical SLOC, CodeCount™ [4] results are approximately 150% higher than the RSM counts [5], and are 110% higher than the LocMetrics [6] numbers[3]
    - Rules for determining when logical statements begin and end are complex and are different for every source language[1]
      - Languages such as Perl, Python, JavaScript



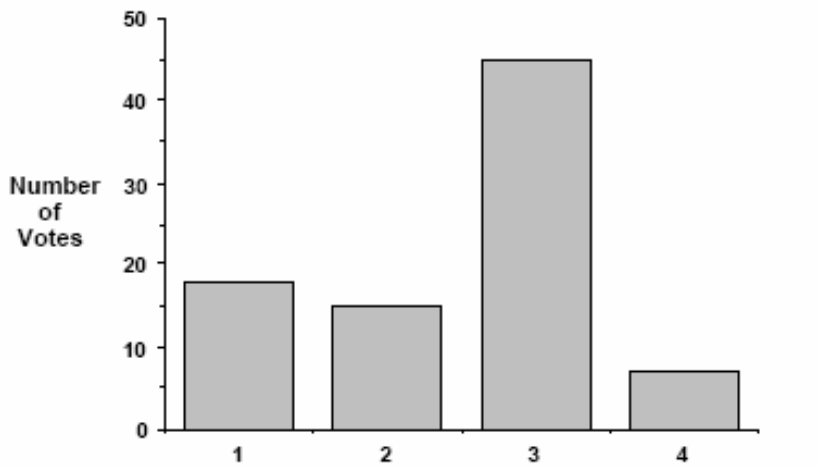
[3] *A SLOC Counting Standard* - Vu Nguyen, Sophia Deeds-Rubin, Thomas Tan, Barry Boehm

[1] *Software Size Measurement: A Framework for Counting Source Statements* – Robert Park

# Logical SLOC Counting Example

- Universal counting standard for Logical SLOC is not available
- How many logical lines of code are in the in the following example?

if A then B else C endif;




*Source: Software Size Measurement: A Framework for Counting Source Statements* – Robert Park

- SEI developed a framework to establish precise definitions for the SLOC metrics
  - User must decide on how to treat many special and language-specific cases

# How to Count?

- How to count?
  - Code counters should be used
    - Counters include CodeMetrics™, CodeCount™, etc.
- Why Code Counters?
  - Produce consistent, accurate SLOC counts
  - Results can be summarized quickly and easily
  - Can detect duplicate source files
  - File structure outputs can be used to further analyze code

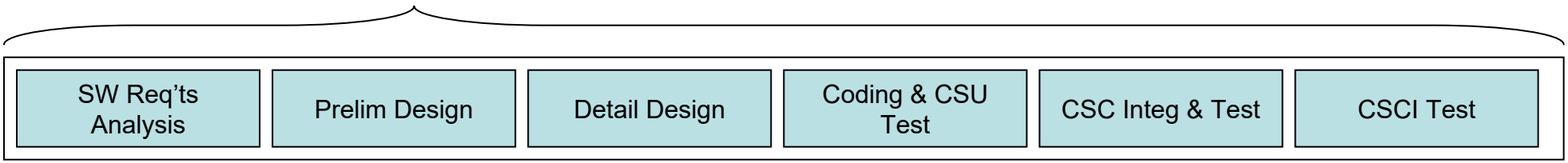
# Actual SLOC Analysis
# Sizing Completed Projects

- Projects generally composed of new code, reused code (with or without modifications) and automatically translated code[2]
- Origin of the code must be determined
  - New development and modified code should be included
  - Duplicate, pre-existing, COTS/GOTS/FOSS, and other prior code that has not been modified should not be included in actual SLOC count
  - Amount of code actually developed would be misstated and would not reflect the code that was actually developed

[2] *Software Cost Estimation with COCOMO II* – Barry W. Boehm

# SLOC Analysis Example

- Given:
  - Total Physical SLOC = 20K comprised of:
    - New Development SLOC = 10K
    - COTS SW SLOC = 5K
    - FOSS SLOC = 5K
  - Development Hours = 10K -> Consistency in collecting over same time period




- Productivity Calculation:
  - Incorrect: 10K hours / 20K SLOC = 0.5 hr/SLOC
  - Correct: 10K hours / (20K – 5K – 5K) SLOC = 1.0 hr/SLOC

**Productivity Factor Could Be Severely Overstated if COTS/GOTS/FOSS Code is Included!**

# SLOC Analysis
# Tips for Sizing Completed Projects

- The following code must be identified and removed
  - Duplicate code
  - Pre-existing code
  - COTS/GOTS/FOSS/libraries
- Use file structure outputs from Code counting tools:
  - 1) Identify files associated with COTS/GOTS/FOSS software and remove associated SLOC from Physical count
  - 2) Examine file structure from one sw release to the next for changes in structure and counts
    - Identify code that may have been added/modified/deleted

# SLOC Normalization (1/3)

- Vast majority of U.S. software projects contain at least 2 programming languages
- Some languages generate more functionality per line of code than others[7]
- Because historical productivity data is typically not available by language, actual SLOC counts should be normalized to a 'standard' language
  - Industry does not track effort by language

[7] *Software Estimation: Demystifying the Black Art* – Steve McConnell

# SLOC Normalization (2/3)

- Normalization should occur to account for productivity variances relating to the development language(s) used
- Resulting productivity factor will be based on normalized SLOC counts

# SLOC Normalization (3/3)

- Example of how SLOC count can vary by language

| # | Assembly | COBOL | Java | Perl |
|---|---|---|---|---|
| 1 | `.MODEL Small` | `IDENTIFICATION DIVISION.` | `class HelloWorld {` | `print "Hello World!\n";` |
| 2 | `.STACK 100h` | `PROGRAM-ID. HELLO.` | `  public static void main(String` | |
| 3 | `.DATA` | `ENVIRONMENT DIVISION.` | `args[]){` | |
| 4 | `  msg db 'Hello, world!$'` | `DATA DIVISION.` | `  System.out.println("Hello World!");` | |
| 5 | `.CODE` | `PROCEDURE DIVISION.` | `  }` | |
| 6 | `  start:` | `  MAIN SECTION.` | `}` | |
| 7 | `  mov ah, 09h` | `  DISPLAY "Hello World!"` | | |
| 8 | `  lea dx, msg` | `STOP RUN.` | | |
| 9 | `  int 21h` | | | |
| 10 | `  mov ax,4C00h` | | | |
| 11 | `  int 21h` | | | |
| 12 | `  end start` | | | |
| | 12 lines of code | 8 lines of code | 5 lines of code | 1 line of code |

Increased Functionality Per LOC

# Sample Normalization Table

**Sample FP to SLOC Conversions**

| Language | Caper Jones[8] | QSM | David Consulting |
|---|---|---|---|
| C | 128 | 148 | 225 |
| C++ | 53 | 60 | 80 |
| COBOL | 107 | 73 | 175 |
| Java | 53 | 60 | 80 |
| PERL | 21 | 60 | 50 |

Example of normalization table based on Java using FP to SLOC Conversion table for relationships

**Ratios of Equivalent Java SLOC**

| Language | Caper Jones[8] | Statements Relative to Java |
|---|---|---|
| C | 128 | 2.42 to 1 |
| C++ | 53 | 1 to 1 |
| COBOL | 107 | 2.02 to 1 |
| Java | 53 | 1 to 1 |
| PERL | 21 | 1 to 2.52 |

C Conversion: 128 / 53 = 2.42

**Note:**

- Code should be normalized to most prominent language
- More research should occur in the future to develop a better conversion table

[8] *Backfiring (UFP to SLOC Conversion) Table* – T Capers Jones, 1996

# SLOC Normalization Example Developing Productivity Factor

**Productivity Factor Calculation:**

| Language | Physical SLOC | Statements Relative to Java | Java Equiv SLOC |
|---|---|---|---|
| C | 10,000 | 2.42 to 1 | 4,141 |
| COBOL | 5,000 | 2.02 to 1 | 2,477 |
| Java | 5,000 | 1 to 1 | 5,000 |
| Total | 20,000 | | 11,617 |

Unnormalized Productivity Factor: 10,000 hours / 20,000 SLOC = 0.5 hr/SLOC

Normalized Productivity Factor: 10,000 hours / 11,617 SLOC = 0.86 hr/SLOC

Actuals Need to be Normalized!

# SLOC Normalization Future Estimates

- Future software estimates need to include a normalization step to be consistent with the basis used to develop the factor
- Process should include:
  - (1) Obtain SLOC estimate
  - (2) Normalize SLOC
  - (3) Multiply Normalized SLOC by Productivity Factor

# SLOC Normalization
# Future Estimates Example

**Estimated SLOC to be Developed**

| Language | SLOC Developed | Java Equiv SLOC |
|---|---|---|
| C | 10,000 | 4,141 |
| COBOL | 10,000 | 4,953 |
| Total | 20,000 | 9,094 |

Java Normalized Productivity Factor = 0.86

Total Effort = 9,094 Java SLOC x 0.86 hr per Java SLOC
= 7,821 Development Hours

Note: Units estimated software product size should be consistent with units that were used to develop the factor (Physical SLOC/Logical SLOC)

# Outline

- Background on Univariate (Linear) Method
- Improvement Techniques
  - Code Count Metric
  - Actual SLOC Analysis
  - SLOC Normalization
- Summary

# Summary

- Use Code Counters for accurate counts
- Use Physical SLOC as the sizing metric
- Perform thorough analysis of actuals
- Normalize SLOC
  - Developing productivity factor
  - Determining future estimates

# References


- [1] *Software Size Measurement: A Framework for Counting Source Statements* – Robert Park, Carnegie Mellon University, 1996
- [2] *Software Cost Estimation with COCOMO II* – Barry W. Boehm, et al. Prentice Hall PTR, 2000
- [3] *A SLOC Counting Standard* - Vu Nguyen, Sophia Deeds-Rubin, Thomas Tan, Barry Boehm, Center for Systems and Software Engineering, University of Southern California 2007
- [4] CodeCount™, USC's Center for Systems and Software Engineering. http://csse.usc.edu
- [5] RSM, M Squared Technologies™, http://msquaredtechnologies.com/index.htm
- [6] LocMetrics, http://www.locmetrics.com
- [7] *Software Estimation: Demystifying the Black Art* – Steve McConnell, Microsoft Press, 2006
- [8] Backfiring (UFP to SLOC Conversion) Table – T Capers Jones, 1996

# Questions