estimate

estimate • analyze • plan • control

Using Source Lines of Code As A Size Input For Estimating Software Effort & Schedule

SEER by GALORATH

Copyright Galorath Incorporated 2017

Many Viable Size Metrics

- SLOC is still viable for many items
  - Sometimes fails due to misunderstanding counting rules
- Functional sizes
- Story points (normalized to SLOC or function points)
- Use Cases
- Components
- Others

Source lines work for estimation in many environments...
Like any sizing metric…
If you use rigorous processes and definitions for counting / estimating them

© 2017 Galorath Incorporated

Simplified Software Equation

$$Effort = a * fxsize^{b}$$

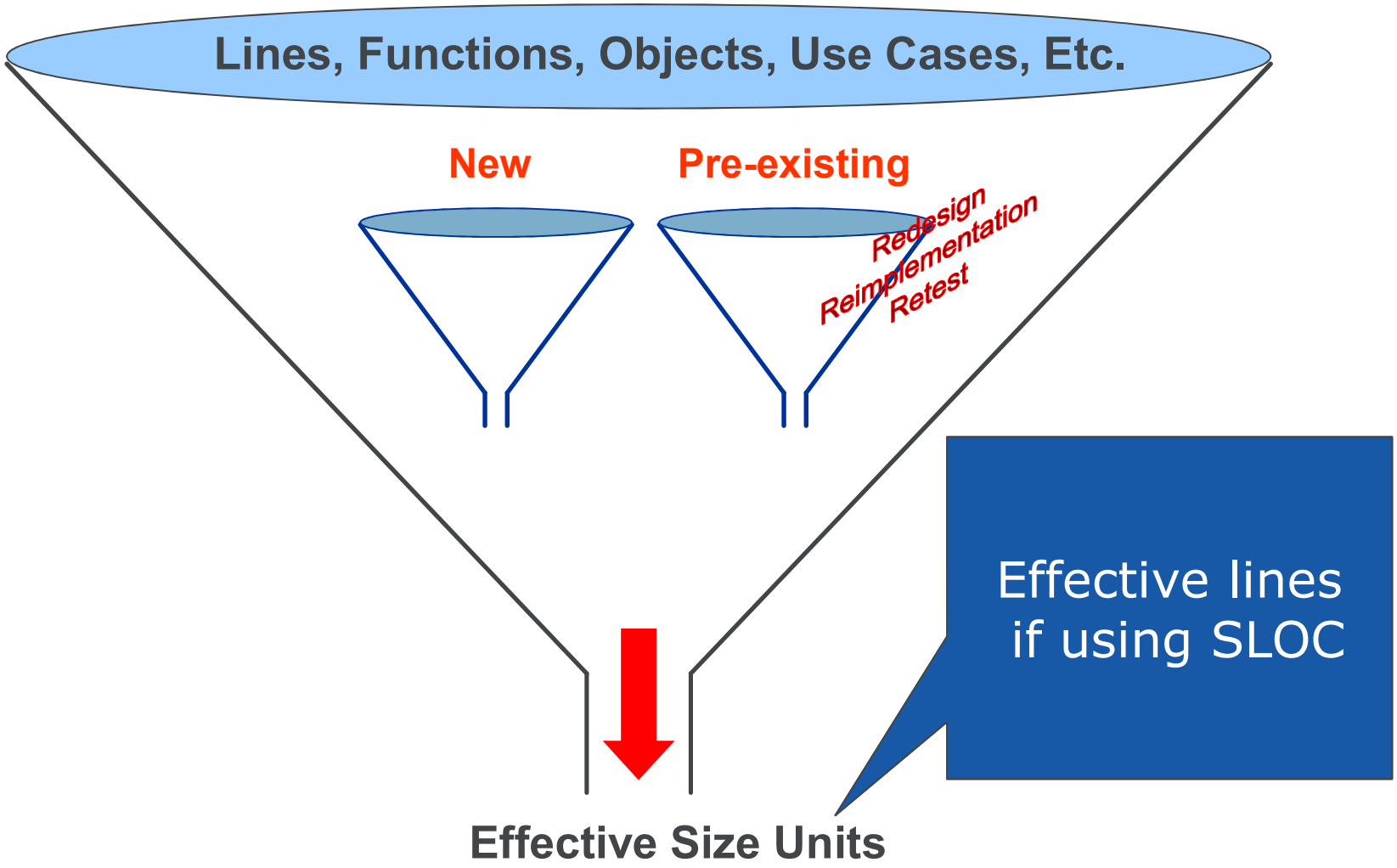
a is the scaler

b is the exponent

Fxsize is the effective size in effort units

© 2017 Galorath Incorporated

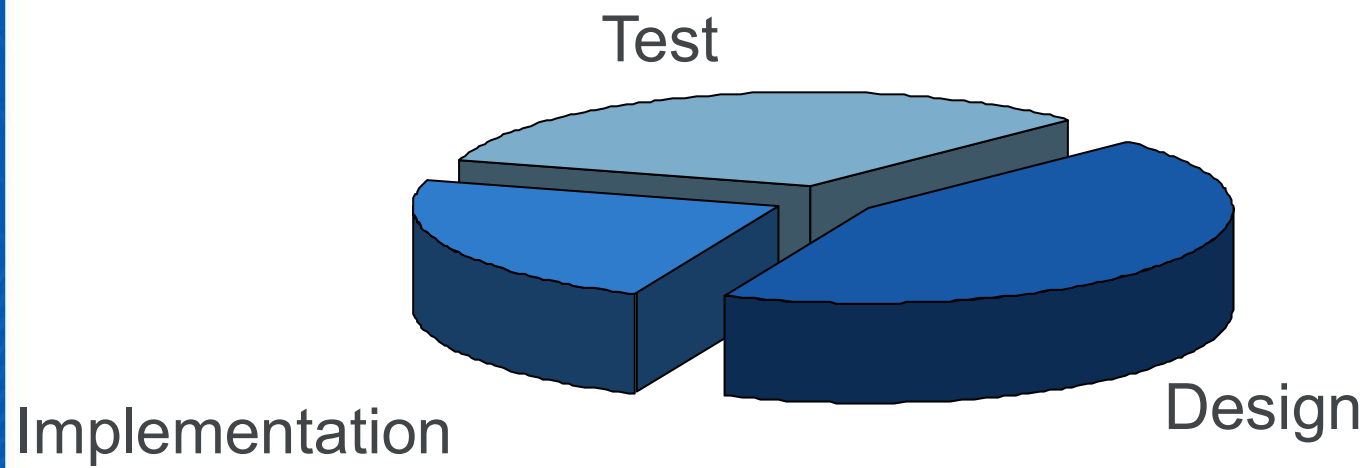
Effective Size Is What Should Be Used In Estimate, Not Total



© 2017 Galorath Incorporated

Using Existing Software Usually Requires Significant Work

- Typical allocation of effort
  - Redesign = 40% of total effort
  - Reimplementation = 25% of total effort
  - Retest = 35% of total effort



© 2017 Galorath Incorporated

Reuse Multiple For Computing Effective Size

Step 1: Set Redesign Factors

Redesign Breakdown

Formula: 0.22*A+0.78*B+0.5*C+0.3*(1-(0.22*A+0.78*B))*(3*D+E)/4

Result Redesign Percentage: 0.00% 0.00% 0.00%

Weight	Redesign Component	Least	Likely	Most	Percentage of the existing software that...
0.22	Architectural Design Change	0%	0%	0%	... requires architectural design change
0.78	Detailed Design Change	0%	0%	0%	... requires detailed design change
0.5	Reverse Engineering Required	0%	0%	0%	... requires reverse engineering
0.225	Redocumentation Required	0%	0%	0%	... requires redocumentation
0.075	Revalidation Required	0%	0%	0%	... requires revalidation with the new design

Step 2: Set Reimplementation Factors

Reimplementation Breakdown

Formula: .37* A + .11*B +.52*C

Result Reimplementation Percentage: 0.00% 0.00% 0.00%

Weight	Inputs	Least	Likely	Most	Percentage of the existing software that...
0.37	Recoding Required	0%	0%	0%	... requires actual code changes
0.11	Code Review Required	0%	0%	0%	... requires code reviews
0.52	Unit Testing Required	0%	0%	0%	... requires unit testing

Step 3: Set Retest Factors

Retest Breakdown

Formula: .10*A + .04*B + .13*C + .25*D + .36*E + .12*F

Result Retest Percentage: 0.00% 0.00% 0.00%

Weight	Inputs	Least	Likely	Most	Percentage of the existing software that...
0.1	Test Plans Required	0%	0%	0%	... requires test plans to be rewritten
0.04	Test Procedures Required	0%	0%	0%	... requires test procedures to be identified and written
0.13	Test Reports Required	0%	0%	0%	... requires documented test reports
0.25	Test Drivers Required	0%	0%	0%	... requires test drivers and simulators to be rewritten
0.36	Integration Testing	0%	0%	0%	... requires integration testing
0.12	Formal Testing	0%	0%	0%	... requires formal demonstration testing

© 2017 Galorath Incorporated

Total vs. Effective Lines/Functions

- **Total software size reflects the total functionality of the program**
  - Newly developed functionality
  - Existing functionality (integrated)
  - COTS functionality
- **"Effective Size" is more directly related to the amount of effort required to develop the software, considering**
  - Redesign, reimplementation and/or retesting of existing code
  - Use of existing design or concept
  - Use of code generators, GUI Builders, etc.
- **Many programs include a combination of new and existing software**
  - New code needs to be designed, coded, tested, integrated
  - Existing code
    - May require some degree of design change, recoding, retesting, reintegration
    - May also require "reverse engineering" to discover how it works

© 2017 Galorath Incorporated

Source line Types & Contents

(Source Software Cost Estimation Metrics Manual for Defense Systems)

- SLOC Count Definitions

Statement Type	Logical In	Logical Out	NCSS In	NCSS Out	Physical In	Physical Out	Total In	Total Out
Executable	✓		✓		✓		✓	
Nonexecutable								
Declarations	✓		✓		✓		✓	
Compiler directives	✓		✓		✓		✓	
Comments		✓		✓	✓		✓	
Blank lines		✓		✓		✓	✓	

Recommend USC code counter for counting existing software sizes

© 2017 Galorath Incorporated

Logical Vs Physical SLOC

Logical SLOC

- Language specific (but very similar & simple for most modern languages)
- Counts number of statements dependent on syntax
- Ignores blank lines & comments

Physical SLOC

- Count line feeds / carriage returns
- Not language specific
- Sensitive to formatting & style
- Often twice logical SLOC

Logical lines work best for effort estimation

© 2017 Galorath Incorporated

Lines of Code Can Be Accurate If Definitions Are Consistent

SEER by GALORATH

Example C++ Program	Physical Carriage Returns	Physical Lines	Logical Lines	Logical Lines Using Language Rules
extern double MessageMonitor(double dfComplexity, double dfSuccessRate);	1	1	1	1
/**	2		comment	
* function: ExampleFunction	3		comment	
*	4		comment	
* purpose: Demonstrate counting of C code	5		comment	
*	6		comment	
* arguments: x [IN]: first argument	7		comment	
* y [IN]: second argument	8		comment	
* bar [IN]: third argument, an array of...	9		comment	
*	10		comment	
* returns: return value	11		comment	
*	12		comment	
**/	13		comment	
double ExampleFunction(double x, double y, int bar) {	14	2	partial	2
	15		blank	
int n = (int) ((x + y) / 2);	16	3	2	3
int SuccessfulAlert = 0;	17	4	3	4
	18		Blank	
if (x < MessageMonitor (y, n))	19	5	Partial	5
	20		Blank	
/* this is a comment */	21		Comment	
	22		Blank	
SuccessfulAlert = bar[n] + 5;	23	6	4	6
	24		Blank	
else	25	7	Partial	
	26		Blank	
while (n > 0) {	27	8	Partial	7
	28		Blank	
SuccessfulAlert += (int) MessageMonitor (x, n);	29	9	5	8
	30		Blank	
n--;	31	10	6	9
	32		Blank	
}	33	11	7	
	34		Blank	
return (++x + SuccessfulAlert + bar(n));	35	12	8	10
}	36	13	9	

A Closer View….Lines of Code Counting Is Easy If You Know The Rules

Line Terminating Semicolons or Closing Curley Braces Make Automated Counting Simple For Many Languages

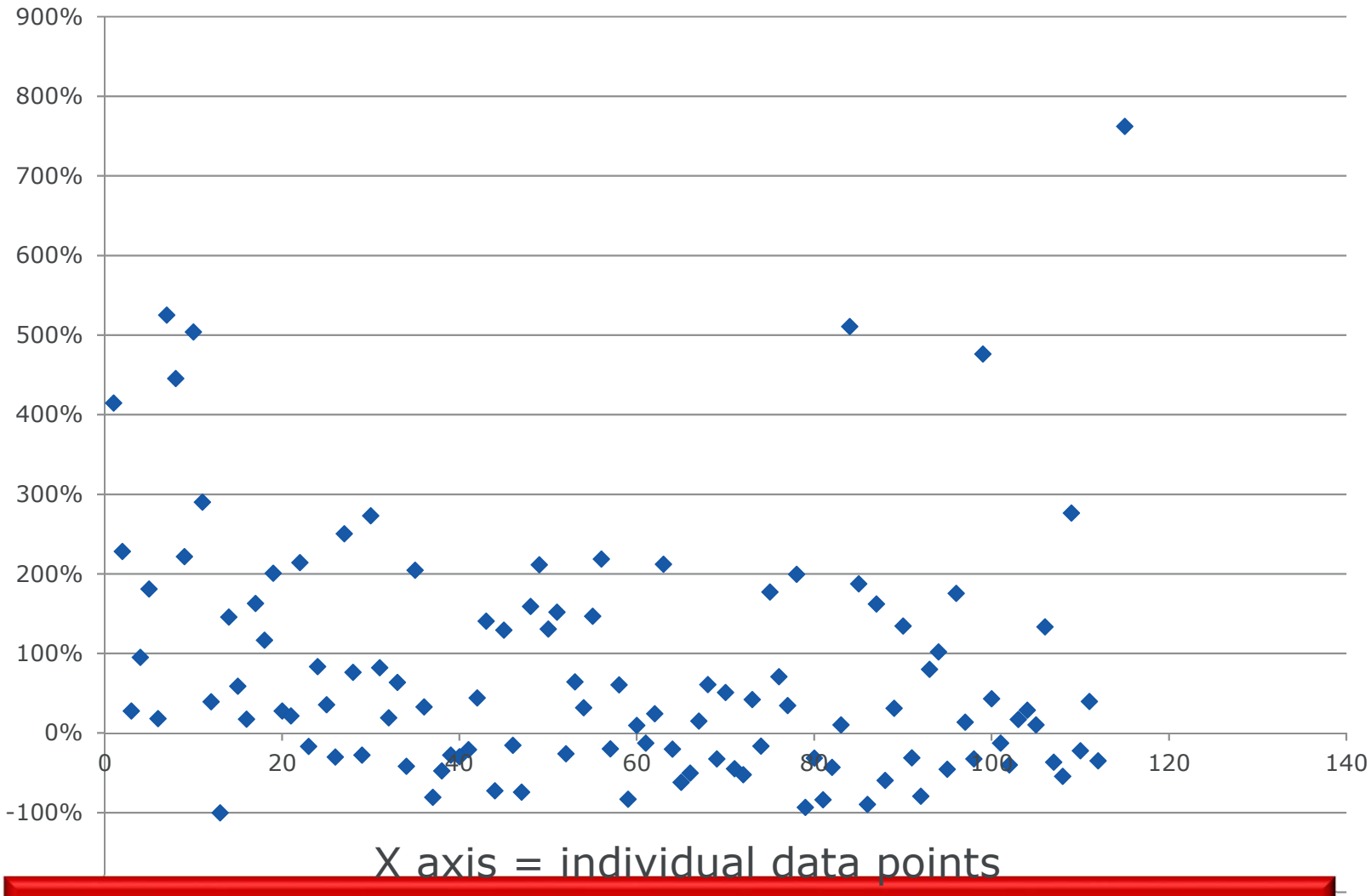
© 2017 Galorath Incorporated

Size Growth

- Whether SLOC or other size measures software usually grows from initial estimates due to
  - Bias
  - Functional growth
- Using a range to describe size can mitigate this

© 2017 Galorath Incorporated

Code Growth Initial New SLOC vs Actual (Note: HUGE outliers removed to make the graph more readable)

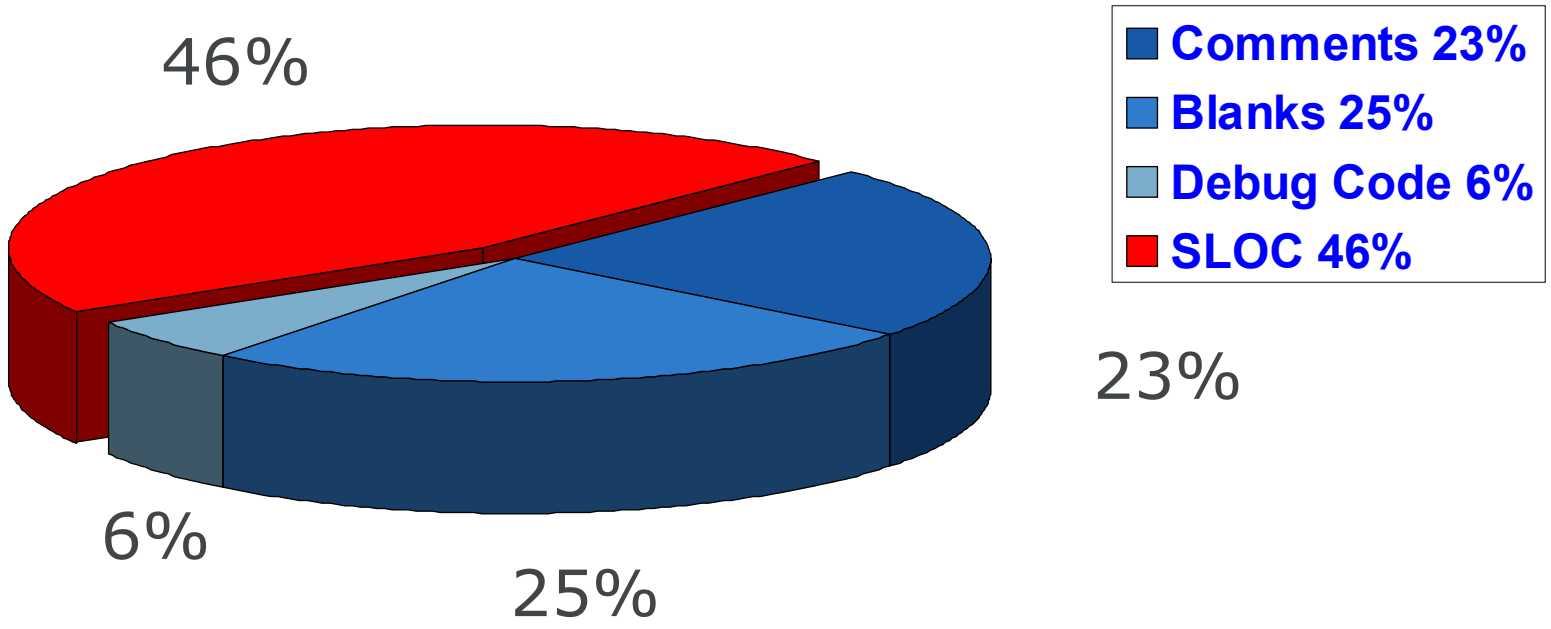


Optimism bias yields initial underestimation (Even after normalization)

© 2017 Galorath Incorporated

Sometimes Size Exaggeration Occurs

Actual Project Example:
Size Claimed = 1.6M Lines
True Size = 736,000 SLOC



Why should we care: Poor sizing is a common reason for poor software estimates

© 2017 Galorath Incorporated

Language & Size--SLOC

- Language used to develop a program will affect program size and effort
  - Different languages require different numbers of lines to achieve the same functionality
  - Different programming styles will result in different program sizes which do the same thing

	Target Language				
Source Language	Gen 3GLs*	4GLs	Ada	Assembly	PL-1/Pascal
General 3GLs		-65%	+22%	+435%	+19%
4GLs	+185%		+248%	+1424%	+238%
Ada	-18%	-71%		+338%	-3%
Assembly	-81%	-93%	-77%		-78%
PL-1/Pascal	-16%	-70%	+3%	+351%	

* General 3GL includes Algol, Basic, FORTRAN, LISP, LOGO, C, C++, COBOL, JOVIAL, MODULA-2, PROLOG and other mixed languages.

© 2017 Galorath Incorporated

Multi Estimate Sizing Process Can Build Confidence and Reduce Risk

New

Pre-existing (rework)

Generated

Glue

Open Source

Multiple size estimates

SLOC Only Example	Least	Likely	most
Analogy			
Sizing Database			
Code Counter			
Estimate by Comparison			
COMPOSITE Size Range Input to SEER-SEM			

Viable size range

Some things require functional size

Most 3rd generation language are similar Ratios allow using different languages to size a new effort

© 201

Where SLOC Goes Wrong

- When code is autogenerated…
  - Count the source not the generated lines
- When code is generated from Graphical interface tool..
  - Use library of effective effort sizes
- When definitions are not known…
  - can convert from various definitions.. E.g. physical to logical ratios
- When people guess at SLOC rather than using a process and clear definitions
- When reuse is not properly accounted for…
- When Open Source or COTS is assumed to be free…
  - use effective size
- Not differentiating between logical and physical SLOC

© 2017 Galorath Incorporated

Bottom Line

- Lines of code are viable for expressing software scope
- Takes care and rigor
- Don’t let people just guess
- Models can provide further refinement of project specifics
- Functional sizes are viable too.. And can have similar underestimating issues
- Size growth must be factored in