



Software Cost Estimation



Jet Propulsion Laborat

Sizing the System

Presented by:

Jairus Hihn

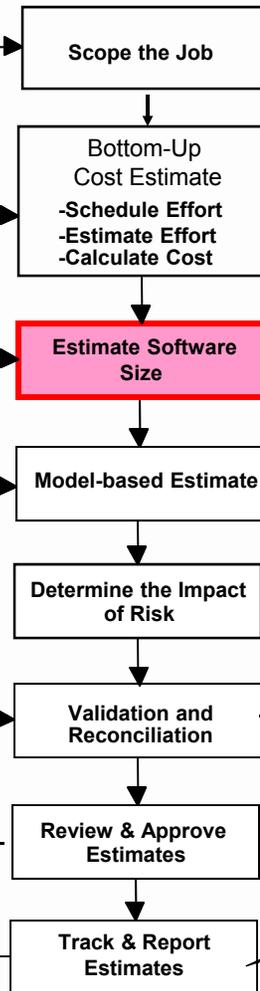
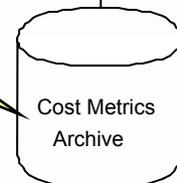
Erik Monson

This work was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. © 2011 California Institute of Technology. Government sponsorship acknowledged.



SW Cost Inputs	<ul style="list-style-type: none"> - Requirements - Architectural Design - Mission/Project Sched. - Implementation Appr. - Mission/Project WBS - SW Implementation and Design Approach
Constraints	<ul style="list-style-type: none"> - Applicable Processes & procedures - Design principles - Std WBS - NASA & OMB Reqs

Save History



When budget is too low
 “Do not look for a silver bullet”
 - DESCOPE

Key elements of Basis of Estimate (BOE)

Follow Through



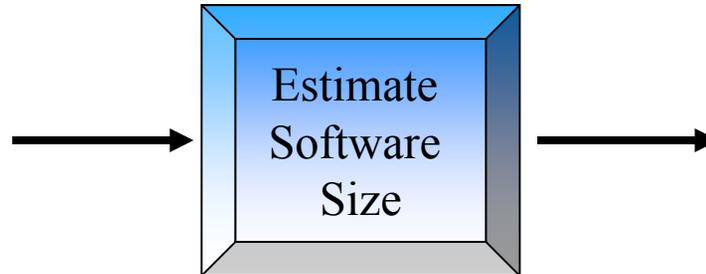
Estimate Software Size



Jet Propulsion Laborat

Inputs

- Functional or Object Decomposition



Outputs

- Software size estimates that distinguish
 - New, Inherited, Modified
- Methods used for size estimation

- The purpose of this step is to estimate the size of the software project
 - Formal cost estimation techniques require software size as an input [Parametric Estimation Handbook, 1999 and NASA Cost Estimation Handbook, 2002]
 - Can be used to generate a bottom-up estimate as shown in handbook
- Size can be estimated in various ways
 - Source Lines of Code (SLOC) or Function Points
 - Interfaces, objects, monitors & responses, widgets
- Size is one of the most difficult and challenging inputs to obtain



Software Size Estimates



- Should be based on the measured sizes for analogous historical software
- Include software reuse assumptions, clearly identifying amount of code reused with no modifications and amount of modified reused code
- Software can be sized using lines of code, work packages, function points
 - Lines of code sizing is institutionally supported
- Use Standard Tools (Some options are)
 - SLiC: Counts 20 languages using multiple size metrics
 - NCSL: counts C (primarily used on flight software)
 - USC CodeCount
 - Diff-SLiC: counts logical differences
 - forthcoming this year



Estimating Software Size Using Source Lines of Code (SLOC)



Jet Propulsion Laborat

Software 'size' is simply a measure of code 'bigness'

The most common way to estimate size is through Source Lines of Code (SLOC)

- Includes any code delivered as a software release
- Many definitions and standards:
 - Raw physical
 - Physical
 - Logical
 -and many others
- SLOC is easy to capture using common counting utilities



Types of SLOC

'Raw Physical' SLOC



Raw Physical* SLOC are the total number of lines in a file

```
1 #include <stdio.h>
2
3 /* This is a one-line comment */
4
5 int main (int argc, char **argv) {
6
7     /* This is a
8        multi-line
9        comment */
10
11     if (argc < 2) {
12         exit(1);
13     }
14
15     fprintf(stderr, "Hello, %s!!\n",
16             argv[1]);
17
18     return 0;
19
20 }
```

Raw Physical SLOC = 20 lines

'Raw Physical' SLOC can be easily counted on UNIX systems using the ``wc -l`` command

This is the easiest and quickest means of counting code, but is of limited use in cost estimation.

Use is not recommended

However, it is no longer the de-facto measure for cost estimation due to the advent of logical counting standards.

*A term I made up for the lack of a better description. There is no accepted term for this type of SLOC, therefore we will use this for the purposes of this class. SQI tools also use this terminology.



Types of SLOC

'Physical' SLOC



'Physical' SLOC are the total number of non-blank, non-comment lines

```
1 #include <stdio.h>
2
3 /* This is a one-line comment */
4
5 int main (int argc, char **argv) {
6
7     /* This is a
8        multi-line
9        comment */
10
11     if (argc < 2) {
12         exit(1);
13     }
14
15     fprintf(stderr, "Hello, %s!!\n",
16         argv[1]);
17
18     return 0;
19
20 }
```

Physical SLOC = 9 lines

This is the most widely-accepted approach to counting source lines of code since it is a well-understood standard that is easily implemented.

Since certain languages are more 'compact' than others, it is often difficult to compare Physical SLOC counts of different languages. Most cost estimation tools now use 'Logical SLOC' which helps to normalize out these differences.



Types of SLOC



'Logical' SLOC / Logical Source Statements

Jet Propulsion Laborat

'**Logical**' SLOC captures size using language-specific rules. Logical SLOC are sometimes referred to as '**Logical Source Statements**'

```
1 #include <stdio.h>
2
3 /* This is a one-line comment */
4
5 int main (int argc, char **argv) {
6
7     /* This is a
8        multi-line
9        comment */
10
11     if (argc < 2) {
12         exit(1);
13     }
14
15     fprintf(stderr, "Hello, %s!!\n",
16             argv[1]);
17
18     return 0;
19
20 }
```

Logical SLOC = 6 lines

For example, in C/C++ the following items count as a logical source statement¹:

- Preprocessor Directives
- Terminal Semicolons
- Terminal close-braces

There are slight variations in the standard to handle special cases. Some definitions of logical source statements are more complex (such as USC CodeCount and earlier SLiC rules).

Due to the inherent nature of these logical counting standards, most counters perform best with properly formatted code.

¹ In SLiC v4.0 simplified ruleset



SLOC Standards - A Review



Type	Description	Pros	Cons
Raw Physical	A count of all lines in a source code file	<ul style="list-style-type: none">• Very easy to capture with standard operating system tools	<ul style="list-style-type: none">• Code formatting can cause SLOC counts to vary significantly even for functionally equivalent code
Physical	A count of all non-comment, non-blank lines in source code file	<ul style="list-style-type: none">• Provides better accuracy than Raw Physical• Unambiguous definition of 'comment' and 'blank' lines	<ul style="list-style-type: none">• Generally requires a code counting utility• Differences in code formatting between languages and development teams can cause SLOC counts to vary, to a lesser extent than Raw Physical
Logical	A count of language specific metrics (USC-SEI conventions)	<ul style="list-style-type: none">• Most accurate measure of SLOC - normalizes out many of the counting errors inherent to other counting conventions• Input for many modern software cost estimation models	<ul style="list-style-type: none">• Requires a code counter with support for the language to be counted, or the use of conversion factors (less accurate)• Language-specific standards can be difficult to understand



Handling Special Cases

Physical to Logical Conversion “Rules of Thumb”



- When it is not possible to natively count logical statements (such as when you only have a physical SLOC count) you can derive an **approximation** of logical statements by using the following adjustment levels to physical (non-comment, non-blank line) SLOC counts
- In some programming languages, physical lines and logical statements are nearly the same (as in assembly), but in others significant differences in size estimates (and thus more significant errors in approximation) can result

Language	To Derive Logical SLOC
Assembly	Assume Physical SLOC = Logical SLOC
C	Reduce Physical SLOC by 25%
Perl, SQL	Reduce Physical SLOC by 40%
C++ / Java	Reduce Physical SLOC by 30%



Handling Special Cases

Autogenerated Code



Occasionally, you may run into code that was not hand-coded but rather generated from another program

Unfortunately, since this autogenerated code was not 'worked on', you cannot directly use this for an analogy size reference for cost estimation purposes.

An example (from real life):

- A hand-written source file (`source`) is **438** lines
- The original source file (`source`) is translated into C source code by a translation utility using its default options. The resulting generated file, `output.c`, is **3,351** lines, over 7 times larger than the code that was written.
- Running the translation utility ***with an optimization flag*** (results in more bloated code but better performance) yields an `output.c` with 17,699 lines, over **40** times larger than the original source code!!!

What we really need is to count the file `source`. If you only have access to `output.c`, you have no way of knowing the exact size of `source`.



Handling Special Cases

Dealing with Autogenerated Code



There may be cases where you cannot access the hand-generated source. Security or intellectual property considerations may restrict your access to the code. In this case:

1. Ask the developer to count the code with the code counting utility of your choosing (one that supports Logical SLOC)
2. If the developer cannot or will not use your specific code counter, ask him/her to use the 'wc -l' command to count all lines in the source files (Raw Physical SLOC). [SQI can then assist you in approximating Logical SLOC from this metric.](#)
3. **If all else fails***, you can use the table below to estimate the size of the hand-generated source code:

Language	To Derive Logical SLOC, Multiply Number of Autocode Lines By:		
	Lowest	Most Likely	Highest
Second-Generation		1	
Third-Generation	0.22	0.25	0.4
Fourth-Generation	0.04	0.06	0.13
Object-Oriented		0.09	0.17

*** Don't let it come to this if you can help it. It is imprecise, but may be your only choice if you've been backed into a corner**



Sizing Considerations



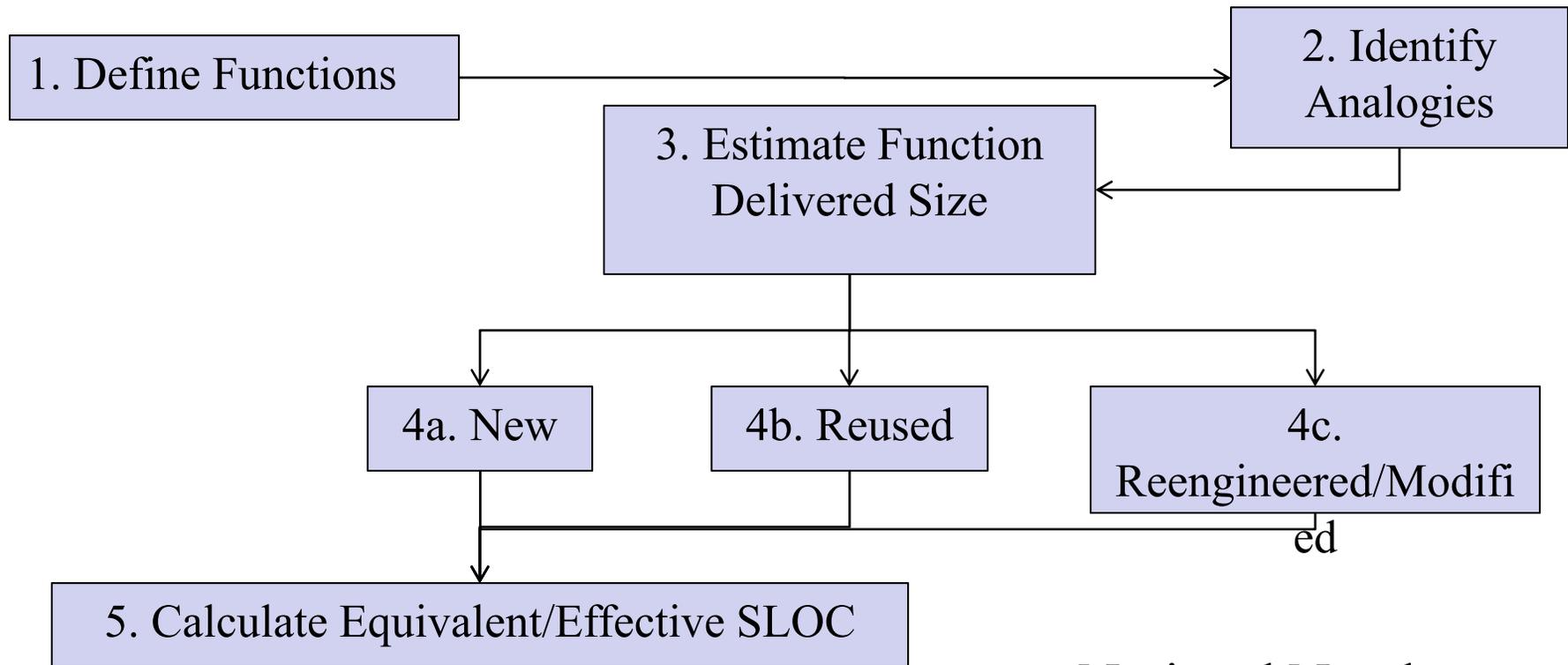
- What you choose to count is important
 - Implementation cost driven mainly by written code
 - Maintenance cost driven by delivered code

Analogous size data typically provides **delivered** code

- Since relatively few software projects at NASA are written from scratch, your project is likely to have:
 - New Code
 - Inherited or reused code
 - This is code that is incorporated into a projected **as-is**, and must be re-tested
 - Projects tend to overestimate the amount of inherited code and the degree to which modifications would be unnecessary
 - Modified Inherited code
 - Modifying code that is inherited/reused requires effort in addition to testing and may not necessarily result in significant cost savings
- Each type of code requires work; **nothing is free!**



Software Size Estimates



Notional Numbers

For Flight Software:

$$\text{Equivalent SLOC} = \text{New Code} + (0.25)(\text{Reused code}) + (0.80)(\text{Modified Code})$$

For Ground Software:

$$\text{Equivalent SLOC} = \text{New Code} + (0.15)(\text{Reused code}) + (0.65)(\text{Modified Code})$$

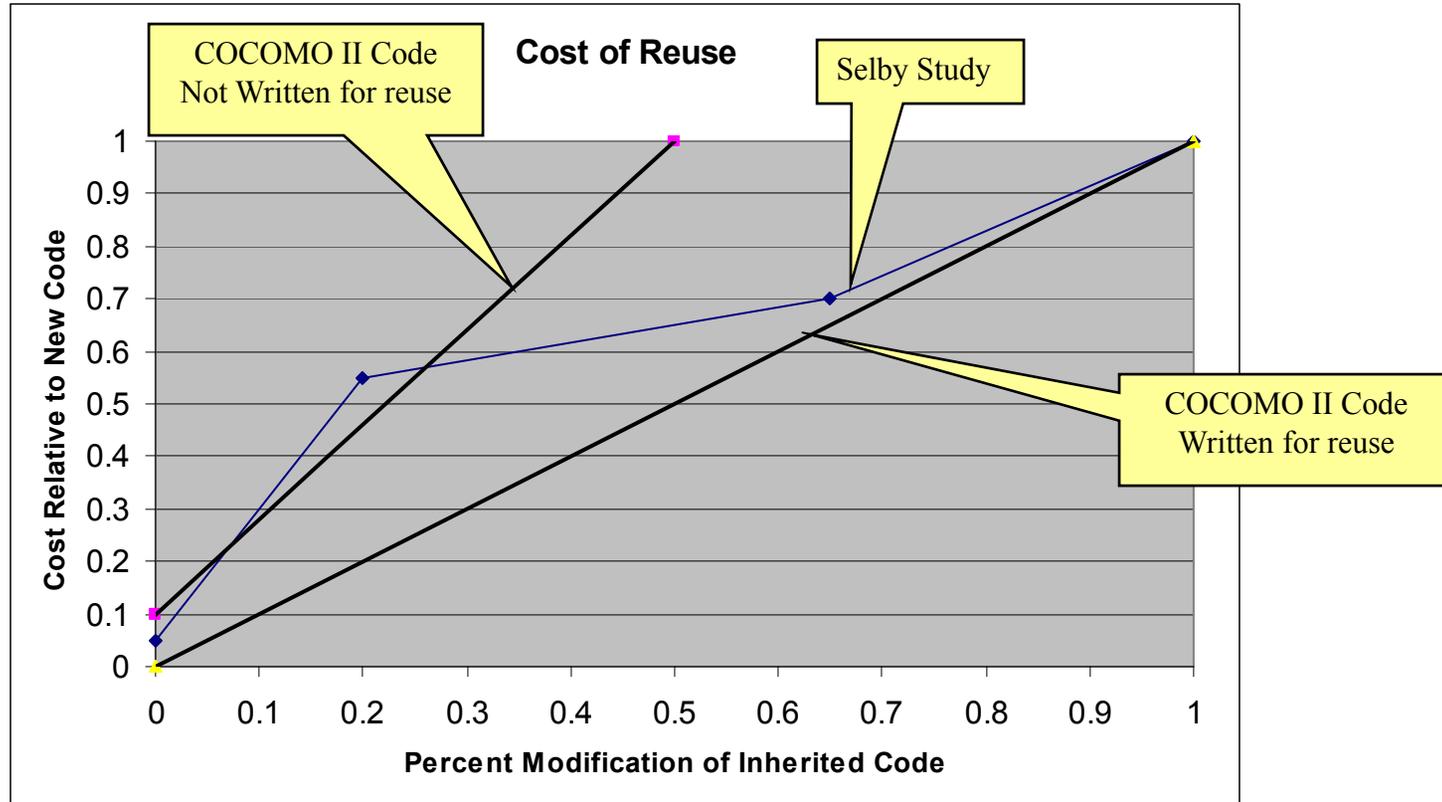


Cost of Inherited Code

(And you thought it was free!)



Jet Propulsion Laboratory



It has been shown that even minor modifications can cost more than **half** of the cost of developing an application from scratch!



Size Estimation Example



C&DH SW Module	Historical Mission Actuals	New Mission		
		New	Modified	Reused
CMD	3292	4000		
TLM	1406		400	1000
DM	1845	1000	1000	
CMD IF	1373	1373		
CMD/TLM BD	1442	1000		
TLM IF	656			656
App	419	500		
MM	2221	2300		
TS	1864	1100	900	
Time	97			97
TM	649			649
FS	59			59
SCU RM	387	400		
Time Sync	344	400		



Equivalent Lines of Code



To simplify the process of cost estimation, we need a way of accounting for the increased scope of incorporating inherited and modified code into a project.

- Standard of practice is that written code is measured by what is called **Equivalent** (Effective) lines of code
 - Equivalent SLOC takes into account the differences in effort required to incorporate new vs. inherited code into a delivered system
 - Equivalent lines of code takes into account the additional effort required to modify reused/adapted code for inclusion into the software product
 - Estimated equivalent size \leq Delivered equivalent size
- Example
 - $\text{EqSLOC} = \text{New} + 0.25 * \text{Reused} + 0.6 * \text{Modified_Inherited}$



Computing Equivalent Lines of Code



Jet Propulsion Laborat

The first step is to identify code heritage:

- **Inherited code** without modifications
- **Modified code**
- **New code**

Any major modifications ($\geq 50\%$) to inherited code should be treated as new code



Computing Equivalent Lines of Code Methods



Jet Propulsion Laborat

- Method #1 (Quick)
 - (a) Treat inherited code with 50% or greater modifications as new code
 - (b) Compute Equivalent SLOC:

For Flight Software:

Equivalent SLOC = New Code + (0.25)(Reused code) + (0.80)(Modified Code)

For Ground Software:

Equivalent SLOC = New Code + (0.15)(Reused code) + (0.65)(Modified Code)



- Method #2
 - Use full algorithm as provided in COCOMO II tool
 - We will explore this option in detail during the model-based estimates lecture



Computing Equivalent Lines of Code – Example



- You are inheriting three modules of 5 KSLOC each for a ground software project
 - Module 1 is 5 KSLOC with no modifications
 - Module 2 is 5 KSLOC requiring 30-40% modifications
 - Module 3 is 5 KSLOC requiring 50-60% modifications
- Compute equivalent lines of code
 - Module 1 is pure reuse
 - Module 2 is treated as modified code
 - Module 3 requires extensive modifications and is treated as new code
 - $EKSLOC = 5(.15) + 5(0.65) + 5$
 - Equivalent Size = 9 KSLOC



What to Count



- Want to count EqSloc for software that gets delivered as part of the system
- Includes
 - System code
 - Adaptation of standard multi-mission software
 - Simulators
 - Delivered regression test suites
 - Test bed support software (input-output & analysis)
- Excludes
 - Non-delivered items
 - E.g. Non-delivered unit test scripts



Size Estimation Steps



- Decompose SW taking into account heritage, functionality, and complexity
- Estimate Size Distribution parameters
 - Derive Most Likely (ML) based on analogous functions from completed software systems
 - Adjust estimate for differences between current fn and analogous fn
 - Adjust estimate for heritage and auto-generated code
 - Provide low and high size estimates based on best and worst case scenarios
- Convert to logical lines if needed
 - COCOMO and SEER use logical lines
 - Handbook tables are based on logical lines
- Compute Total SLOC based on
 - PERT Mean computation
 - Mean = $(\text{Low} + 4\text{ML} + \text{High})/6$
 - Monte Carlo Simulation (preferred)



Size Estimation Example Assumptions



	Analogy Reference	New			Reused	% Modified		BOE
		Low	Likely	High		Low	High	
Fn1	12	1	2	5	10	5%	15%	
Fn2	8	2	3	4	5	0%	0%	
Fn3	2	2	4	8				
Fn4	12	8	10	20	2	50%	60%	

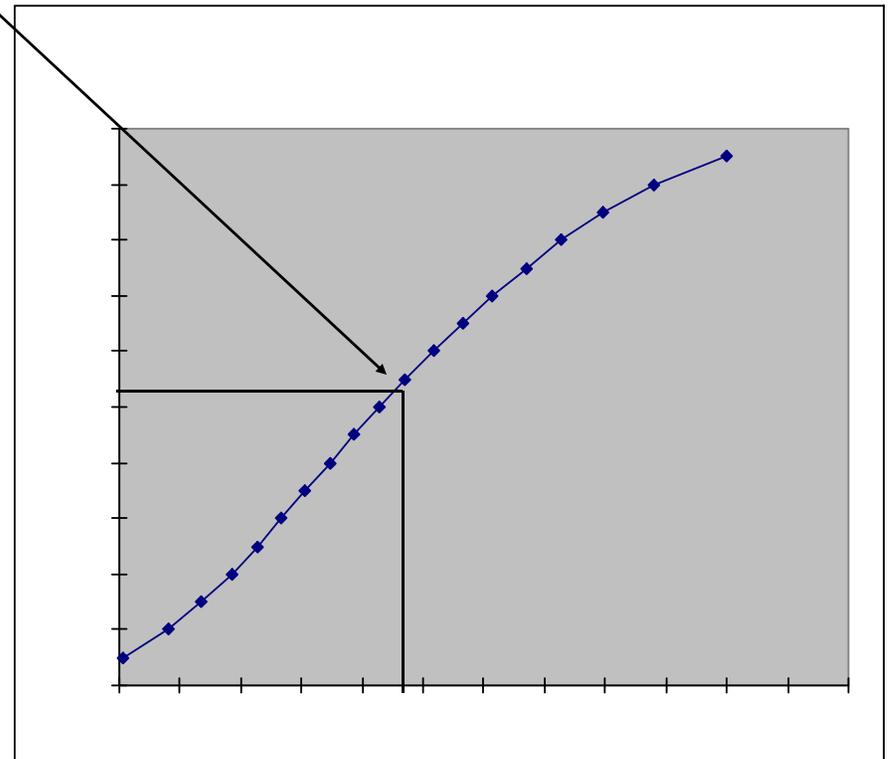
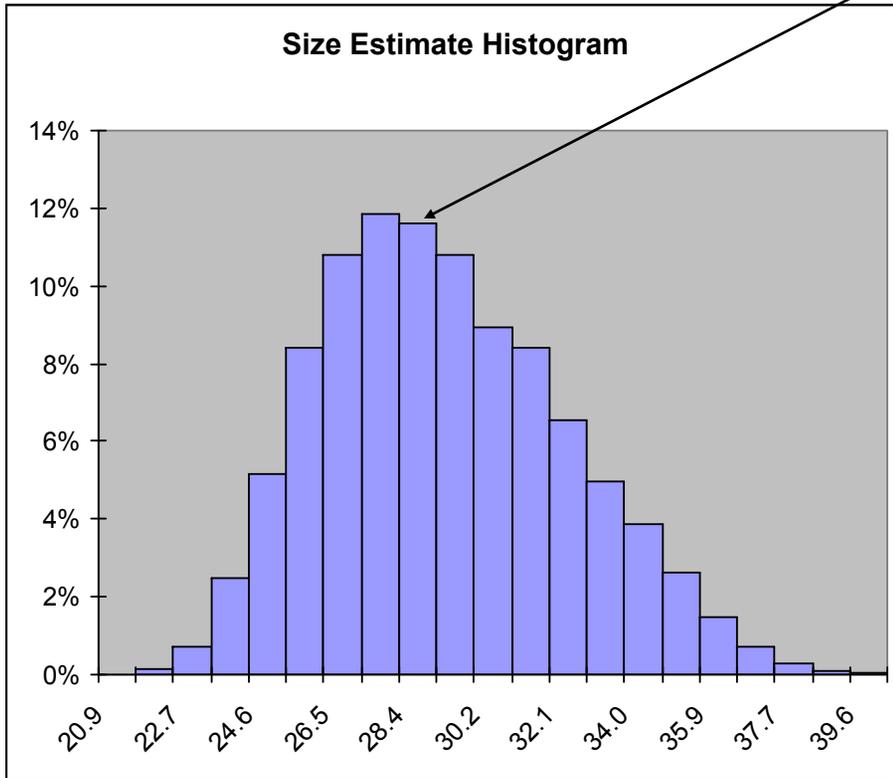
- Basis of Estimate (BOE) should include
 - Analogies supporting Likely and reuse numbers
 - e.g. Fn1 similar to Fn x on DS-1
 - Conditions that drive Low and High estimates and modification ranges
 - e.g. Fn2 Low assumes that the driver sw that comes with the actuator can be used as is, High assumes drivers require extensive high level driver code



Size Estimation Example Distributions



Mean Size = 28.6





Alternate Sizing Methods (1)



Jet Propulsion Laborat

- Commonly used methods not covered in class but we can provide assistance in using
 - Paired Comparison Matrices is a way to more rigorously capture expert judgment
 - Method based on rank ordering modules and providing relative size ratios (eg Mod1 is 1.5 times bigger the Mod 2)
 - Can be easily implemented in Excel (e.g MONTE)
 - SEER-SEM is an available commercial tool
 - Function Points counts inputs, outputs, files
 - Method based on counting input, outputs, data items, based on a user-oriented high-level software design
 - IFPUG provides standards and training (<http://www.ifpug.org>)
 - Approach can be adapted around counting inputs and outputs from design documents or detailed requirements documents
 - Difficulty here is consistency
 - The JPL Flight Software Cost model used in Team X contains a sizing tool for flight systems based on mission & system characteristics



JPL Code Counter



Jet Propulsion Laborat

Source Lines Counter (SLiC)

SQI has written a code counter to measure Logical SLOC metrics (and more) for the most common languages in use on lab. This tool is freely available to anyone on lab and has several advantages:

- Counts source lines written in nearly two dozen different languages (and counting) in widespread use at JPL
- Supports the three most common counting standards
 - Raw Physical SLOC
 - Physical SLOC
 - Logical Source Statements (a.k.a. Logical SLOC)
- Easy to use - just tell it where your code resides and it will search for and count supported source code.
 - Searches are highly customizable; you can create powerful search rules or simply specify specific files/folders to count.
- Flexible output formats allow SLiC to fit into automated scripts.
- If you participate in SQI data collection activities, SLiC can automatically upload SLOC metrics to the SQI repository if you so specify, eliminating the need to manually enter these data in the repository.
- Runs on Linux, Mac OS X and Windows (through the Cygwin toolkit)
 - Solaris versions available on request
- **Coming soon to v5.0: Compare two file trees and report source lines added, deleted, or modified**



Where to get SLiC



```

Terminal — bash — 79x37
confutils.js Java_S 39.5 kB 42 183 1266 1573
cvsclean.js Java_S 3.4 kB 25 75 71 120
deplister.c C 2.0 kB 31 9 21 60
mkdist.php PHP 10.9 kB 53 258 300 420
registersyslog.php PHP 1.0 kB 2 19 28 45

Folder TOTAL C 2.0 kB 31 9 21 60
Folder TOTAL PHP 11.9 kB 55 277 328 465
Folder TOTAL Java_S 50.0 kB 105 412 1516 1956
Folder GRAND TOTAL --- 64.0 kB 191 698 1865 2481

SOURCE CODE TOTALS FOR ALL FILES COUNTED UNDER:
.
-----
LANG FILE SIZE COM- LOG PHY RAW
-----
TOTAL C 23.8 MB 128.0k 370.8k 574.8k 785.8k
TOTAL C++ 22.2 kB 121 611 682 936
TOTAL Tcl/Tk 51.4 kB 330 1.1k 1.5k 1.7k
TOTAL PHP 461.6 kB 2.8k 7.1k 10.9k 15.6k
TOTAL Java_S 50.0 kB 105 412 1.5k 2.0k
TOTAL Perl 911.0 B 5 61 71 84
TOTAL SQL 1.2 kB 0 40 40 44
TOTAL SH 373.6 kB 1.7k 10.3k 10.3k 13.4k
TOTAL LEX 53.0 kB 483 1.2k 1.5k 2.3k
TOTAL YACC 115.3 kB 360 2.1k 2.4k 3.2k
TOTAL MAKE 22.7 kB 147 504 504 817
TOTAL XML 160.5 kB 105 3.4k 4.5k 4.6k
GRAND TOTAL --- 25.1 MB 134.2k 397.5k 608.7k 830.5k

mac07281151556:php-5.2.6 emonson$

```

Example SLiC session showing code totals by language

- SLiC v4.0 (new as of 9/2009) is currently being distributed in binary executable format
 - Linux, Mac OSX, and Windows versions are currently supported
 - Full documentation included

- Users outside JPL can obtain SLiC from the SW PAL on the NEN

- We provide limited support for external users

If you have any questions or comments regarding SLiC, please call [Kevin Smith](mailto:kevin.a.smith@jpl.nasa.gov) at:

Extension (818)354-9437 or kevin.a.smith@jpl.nasa.gov

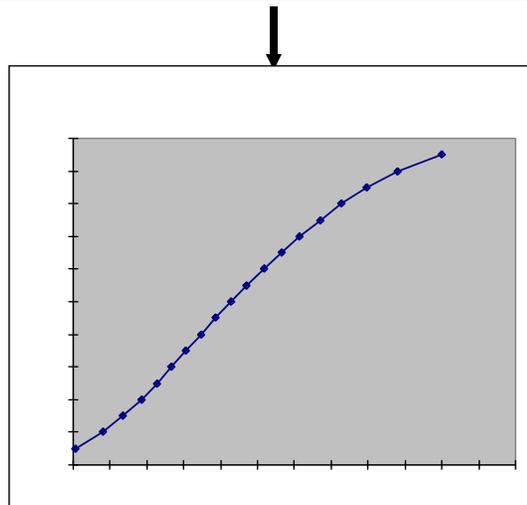


Wrap Up



- The main output of this step is
 - a matrix of size estimates by software module
 - supporting assumptions as a BOE
 - size distribution and summary statistics

	New			Reused	% Modified		BOE
	Low	Likely	High		Low	High	
Fn1	1	2	5	10	5%	15%	
Fn2	2	3	4	5	0%	0%	
Fn3	2	4	8				
Fn4	8	10	20	2	50%	60%	



	Eq SLOC	New SLOC	Reuse
	Mean	Mean	Mean
Fn1	5.1	2.7	2.4
Fn2	4.2	3.0	1.2
Fn3	4.7	4.7	0
Fn4	14.7	12.7	2
Total	28.6	23	5.6