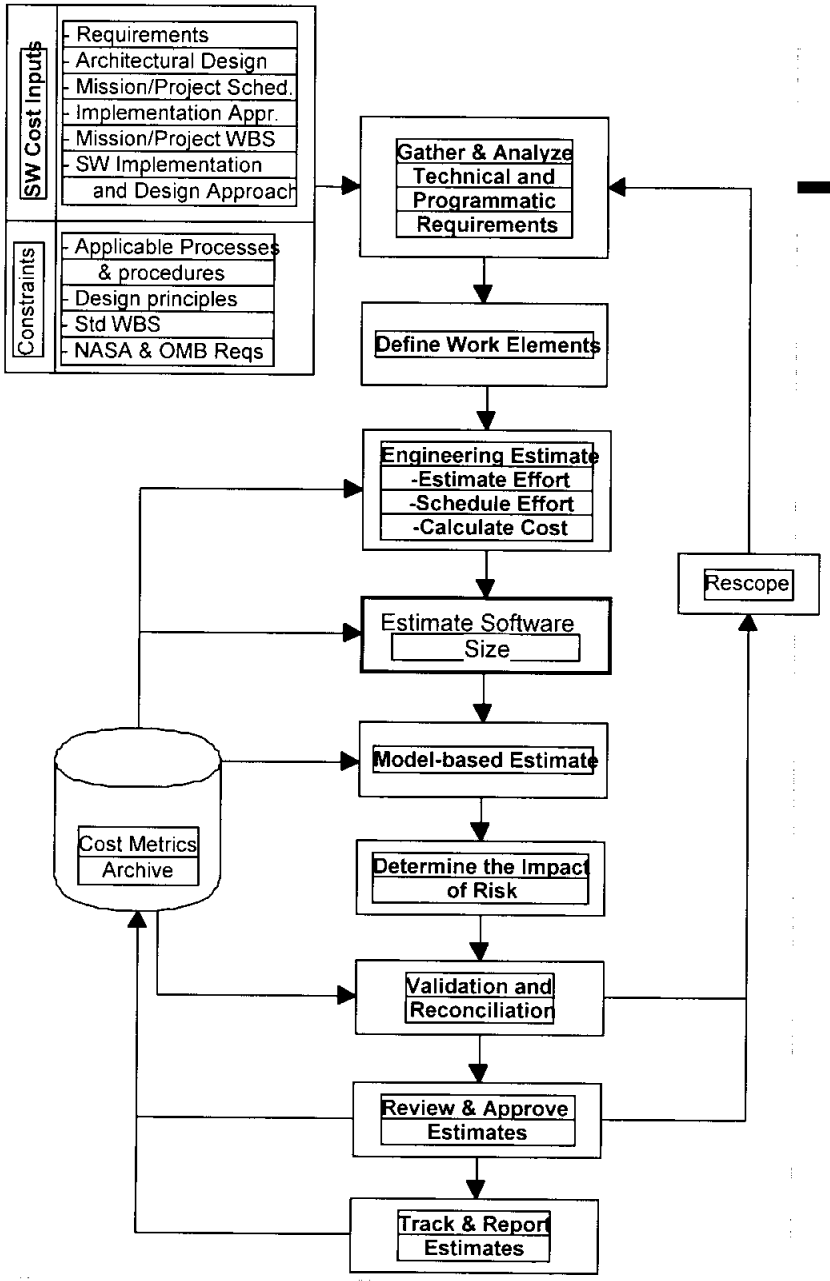
# Sizing the System

Presented by:
Jairus Hihn
JPL Software Quality Improvement Project

International Conference on
Space Mission Challenges for Information Technology
July 13, 2003

# Software Estimation Steps




# Estimate Software Size


- 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 an engineering 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


# Source Lines of Code (SLOC) Physical vs. Logical lines of Code


- Typically either physical lines or logical executable lines are used when counting SLOC
- Comments and blanks should never be included in any lines of code count
- The physical SLOC metric is very simple to count because each line is terminated by the enter key or a hard line break
- Logical executable statements may encompass several physical lines and typically include executable statements, declarations, and compiler directives
  - Preferred input to cost models.

# Size Metrics
# Physical to Logical Conversion

- For example, in C this requires counting semicolons and sets of open-close braces
- Most commercial cost models require logical executable lines of code as input rather than the physical lines of code, as it is considered to be more accurate and changes less between languages
- In some programming languages, physical lines and logical statements are nearly the same, but in others, significant differences in size estimates can result

| Language | To Derive Logical SLOC |
|---|---|
| Assembly and Fortran | Assume Physical SLOC = Logical SLOC |
| Third-Generation Languages (C, Cobol, Pascal, Ada 83) | Reduce Physical SLOC by 25% |
| Fourth-Generation Languages (e.g., SQL, Perl, Oracle) | Reduce Physical SLOC by 40% |
| Object-oriented Languages (e.g., Ada 95, C++, Java, Python) | Reduce Physical SLOC by 30% |

# Size Estimation Example No. 1

| C&DH SW | Historical Mission | New Mission | | |
|---|---|---|---|---|
| Module | Actuals | 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 | | |

# Size Estimation Example No. 2

| Function / SubFunction | LOC New L | LOC New ML | LOC New H | Size | Ref | % Modified L | % Modified ML | % Modified H | Size | REF | % Re-Used L | % Re-Used ML | % Re-Used H | Size | REF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ***Traj Optimization*** | | | | | | | | | | | | | | | |
| Common Structure | 400 | 800 | 1000 | | – | | | | | | 35% | 60% | 80% | 4102 | E |
| Traj Propagation | | | | | | 5% | 20% | 30% | 1147 | D | 10% | 25% | 35% | 1147 | D |
| " | | | | | | 5% | 15% | 20% | 7436 | C | 10% | 20% | 25% | 7436 | C |
| " | 10% | 20% | 25% | 4528 | A | | | | | | | | | | |
| " | 10% | 20% | 25% | | B | | | | | | | | | | |
| " | | | | | | | | | | | | | | | |
| Traj Partials | | | | | | 5% | 10% | 15% | 3682 | E | 10% | 25% | 40% | 3682 | E |
| " | | | | | | 5% | 10% | 20% | 7436 | C | 0% | 5% | 10% | 7436 | C |
| " | 10% | 25% | 35% | 4528 | A | | | | | | | | | | |
| " | 10% | 25% | 35% | 13448 | B | | | | | | | | | | |

SW Systems Engineer estimated percentage of functionality being developed, modified or re-used

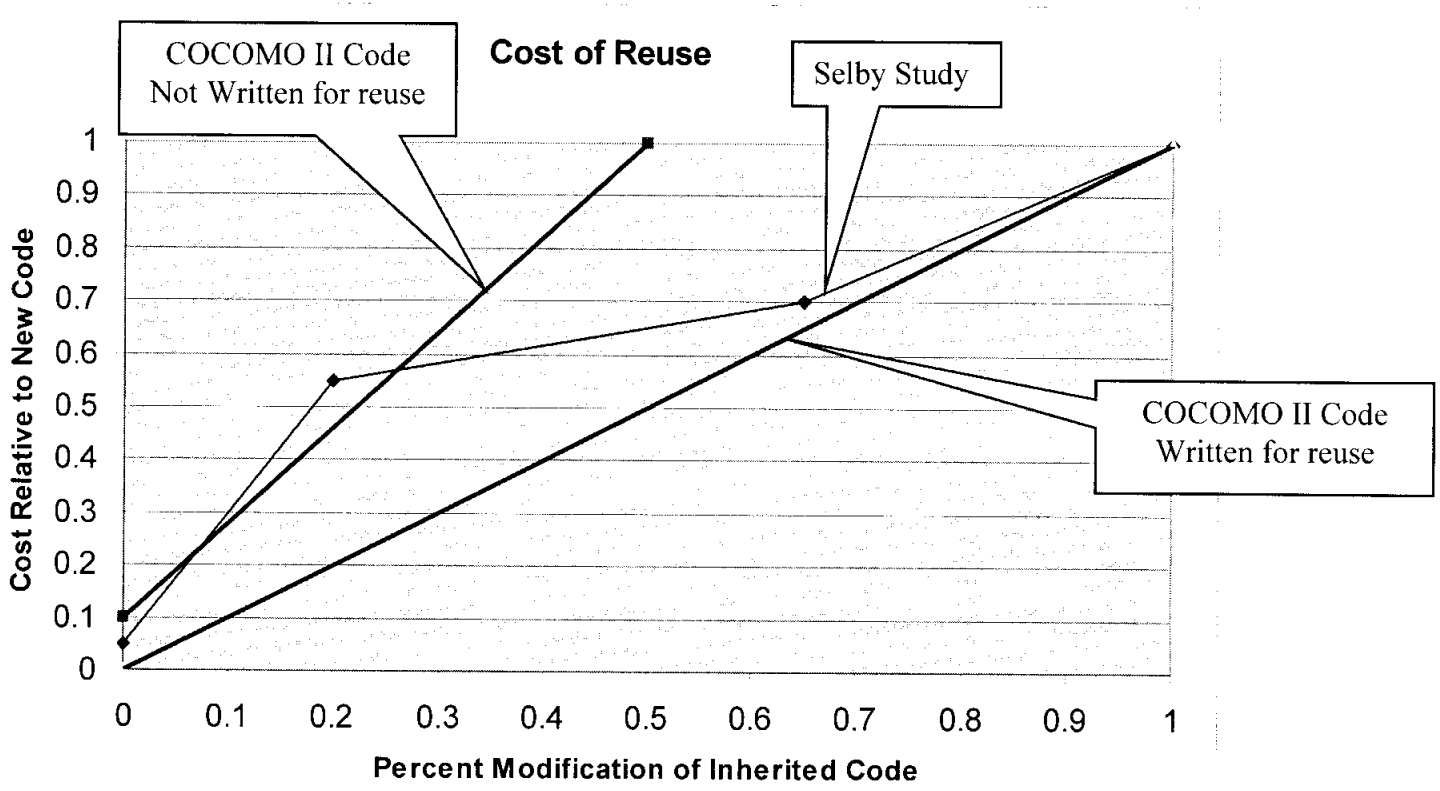
# Written SLOC vs. Delivered SLOC

- Written SLOC vs delivered SLOC
  - Implementation cost driven by written SLOC
  - Maintenance cost driven by delivered SLOC
- Analogous size data typically provides delivered SLOC
- Code comes in three different flavors
  - New
  - Inherited or reused
  - Modified Inherited
- Each type of code requires different amounts of work to make it part of the delivered system and none of it is free

# Written SLOC vs. Delivered SLOC Equivalent Lines of Code

- Standard of practice is that written code 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
  - While inherited code can save some effort, it typically saves much less then people assume
  - Equivalent lines of code takes into account the additional effort required to modify reused/adapted code for inclusion into the software product
- Example
  - EqSLOC = New + 0.25*Reused + 0.6*Modified_Inherited

# Cost of Inherited Code



# Computing Equivalent Lines of Code

- First identify code heritage
  - New code
  - Inherited code with no modifications (Reused)
  - Inherited code with modifications (Modified)
- Any major modifications to inherited code should be treated as if it were new code
- Software development typically consists of evolutionary software design with new code development

# Computing Equivalent Lines of Code Methods

- Two recommended ways to estimate Equivalent SLOC
- Method1
  - (a) Treat inherited code with 50% or greater modifications as new code.
  - (b) Equivalent SLOC = Adapted SLOC*((.24*DM) + (.52*IM) + (.24*ReTest))
    - Where
      - DM is percent design modified
      - IM is percent code modified
      - ReTest is percent that must be completely retested
- Method2
  - Use full algorithm as provided in COCOMO II tool
  - Covered when discuss model based estimates


# Computing Equivalent Lines of Code – Example


- You are inheriting 10 KSLOC in two modules
  - 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 5*(.24*0 + .52*0 + .24*1.0) = 1.25 KSLOC
  - Module 2 is 5*(.24*.5 + .52*1.0 + .24*1.0) = 4.4 KSLOC
  - Module 3 is 5 KSLOC
  - Equivalent Size = 10.65 KSLOC

# Computing Equivalent Lines of Code – Auto-generated Code

- Common sources of auto-generated code
  - fault protection, simulation languages and Labview
- When using delivered code analogies, use the table below to determine the appropriate delivered SLOC
- For example 10 KSLOC of auto-generated C code is equivalent to writing 2.5 KLOC

| | **To Derive Logical SLOC, Multiply Number of Autocode Lines By:** | | |
|---|---|---|---|
| **Language** | 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 |

# 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
    - Eg 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 = (Low + 4ML + High)/6
  - Monte Carlo Simulation (preferred)

# Size Estimation Example Assumptions

| | 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% | |

- 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

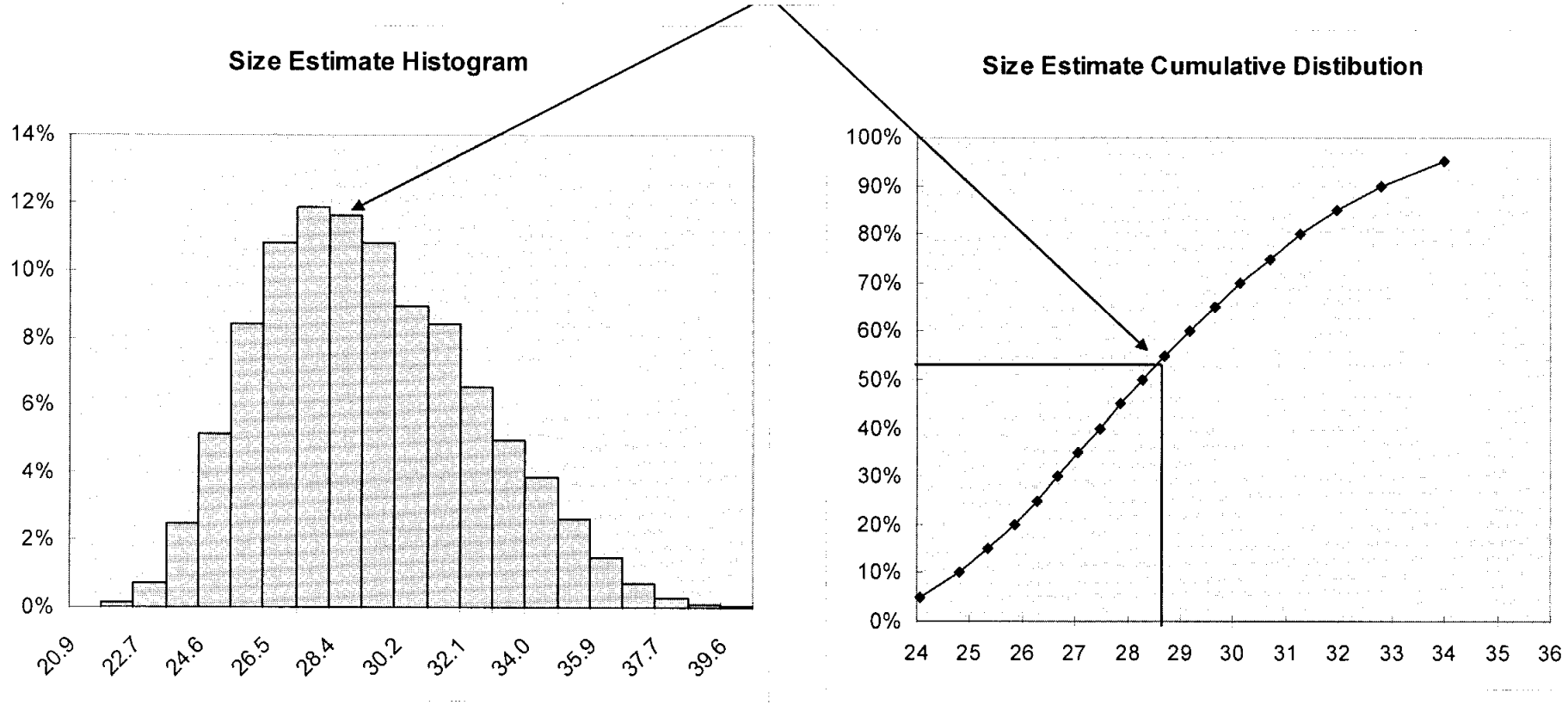
# Size Estimation Example Summary Tables

| | Eq SLOC | New SLOC | Reuse | New SLOC | Eq SLOC |
|---|---|---|---|---|---|
| | Mean | Mean | Mean | PERT Mean | Mode |
| Fn1 | 5.1 | 2.7 | 2.4 | 2.3 | 4.4 |
| Fn2 | 4.2 | 3.0 | 1.2 | 3.0 | 4.2 |
| Fn3 | 4.7 | 4.7 | 0 | 4.3 | 4.0 |
| Fn4 | 14.7 | 12.7 | 2 | 11.3 | 12.0 |
| Total | 28.6 | 23 | 5.6 | 23 | 24.6 |

- New SLOC Means derived from Triangular Distribution
- Reuse Mean for FN1 derived from uniform distribution and conversion factors of .24 for pure reuse, .76 for modifications, if mods > .5 treated as new
- Eq SLOC just sums means
- Other columns provided for comparison

# Size Estimation Example Distributions

Mean Size = 28.6



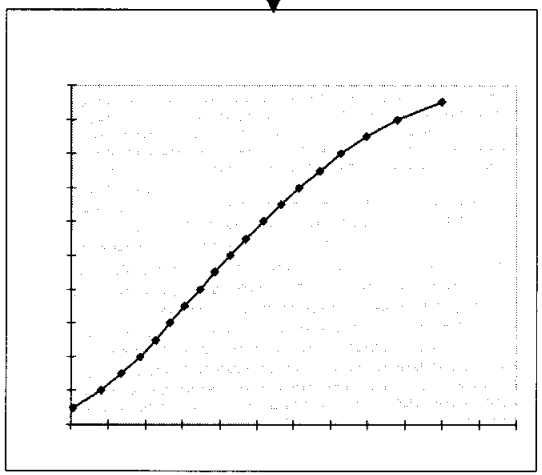
# Alternate Sizing Methods (1)

- Commonly used methods not covered in class but we can provide assistance in using
  - Paired Comparison Matrixes 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
  - Object Points counts classes and methods

# 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 |