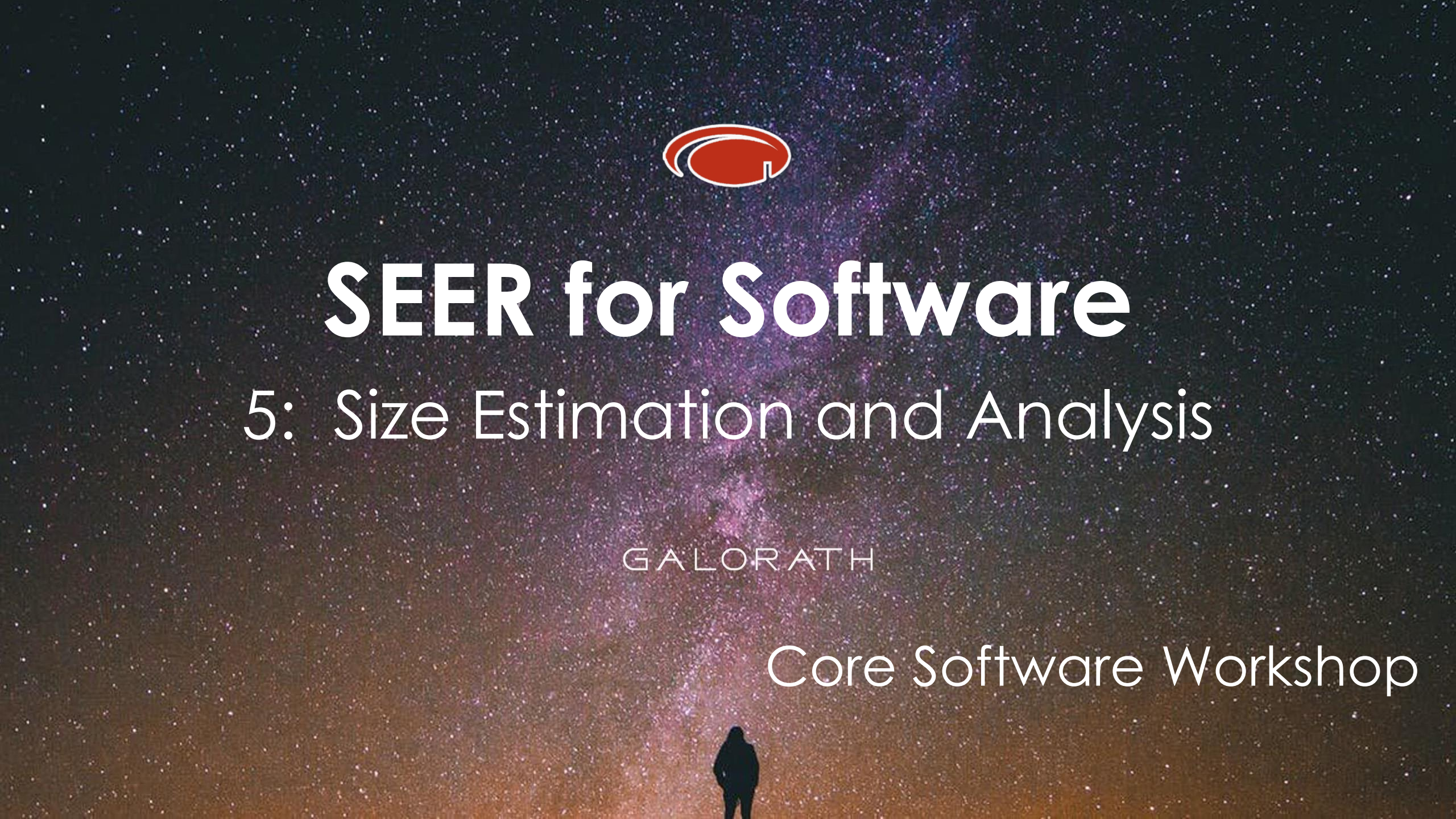


SEER for Software

5: Size Estimation and Analysis

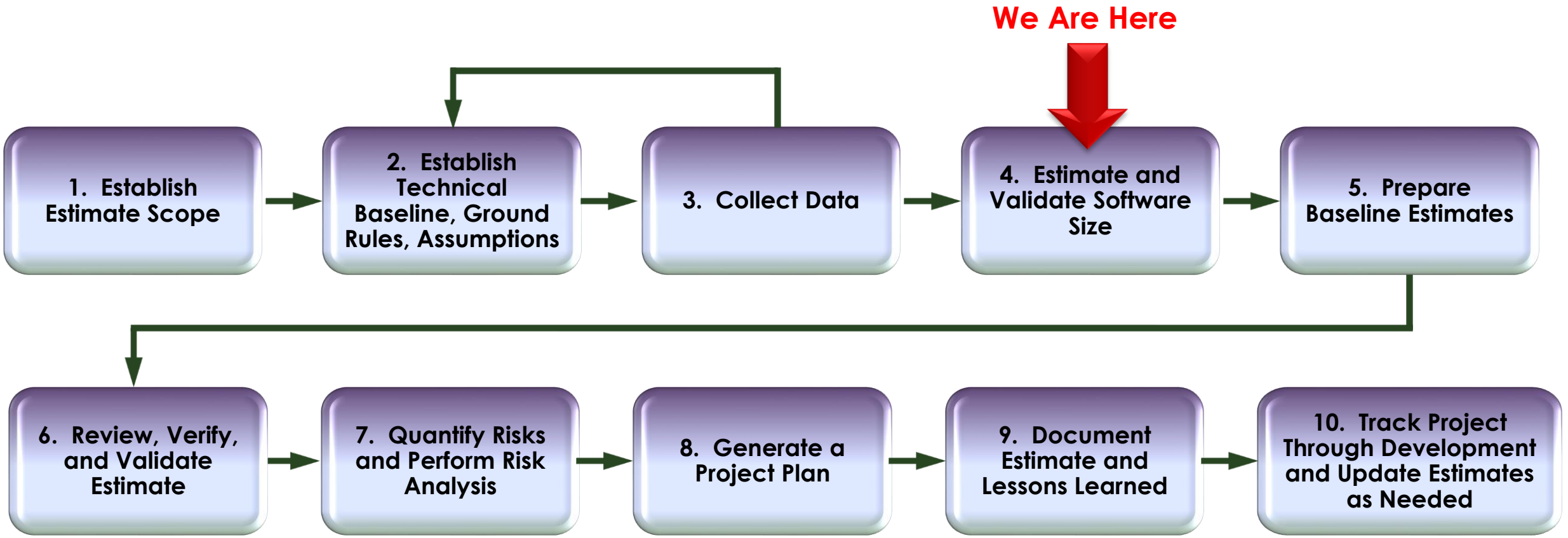
GALORATH

Core Software Workshop

Objective: To understand the critical role of size metrics in software estimation by discussing the following:

- What we mean by software size and why it matters
- The SEER-SEM concept of an "effective effort unit"
- Software size metrics:
  - Function points
  - Lines of code
  - Objects
  - Story points
  - Other metrics and how they can be implemented in SEER-SEM
- COTS integration "sizing"
- Parameters dealing with size
- Redesign, reimplementation, and retest of existing code
- Methods to estimate software size

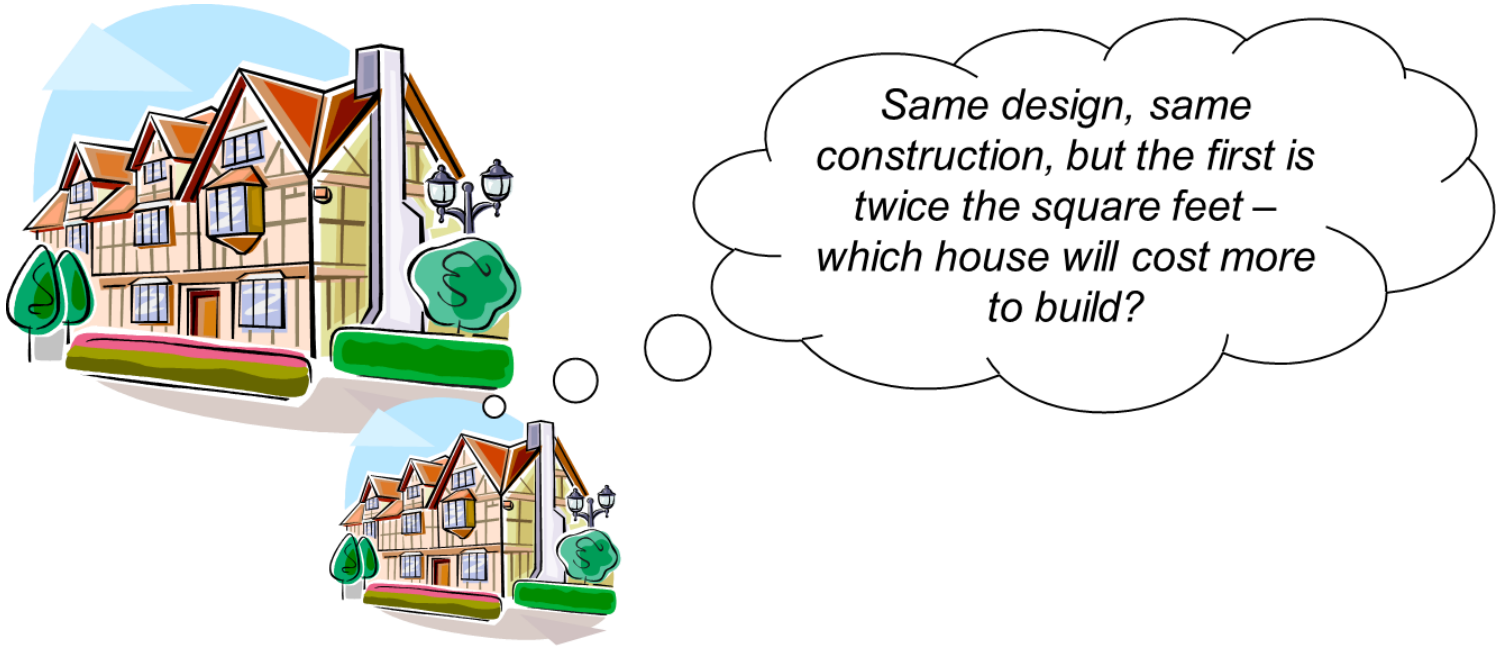
Review: Estimation Process



GALORATH

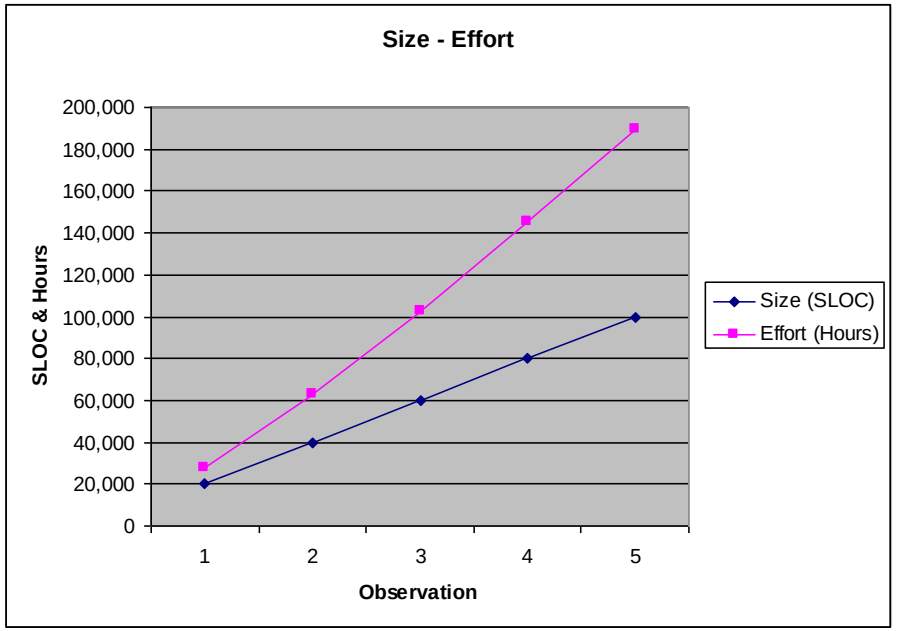
Size DOES Matter!

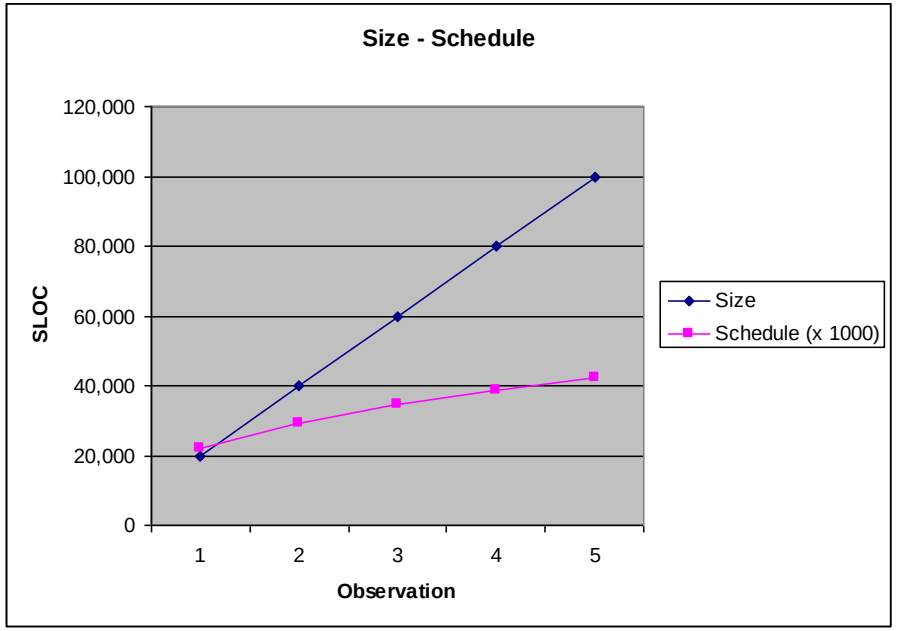
- Size is the most critical driver of cost and schedule on a software project
- All other things equal, the larger the size, the greater the effort and cost and the longer the schedule



Size-Effort Relationship

- Effort & schedule vary in proportion to size (but not linearly!)
- Knowing size allows estimators to determine effort (cost) & schedule
- Better size estimates = better effort & schedule estimates
- Software Size is the main driver of software development effort, cost, and schedule -- use the best available estimate of size, and ***use a range***

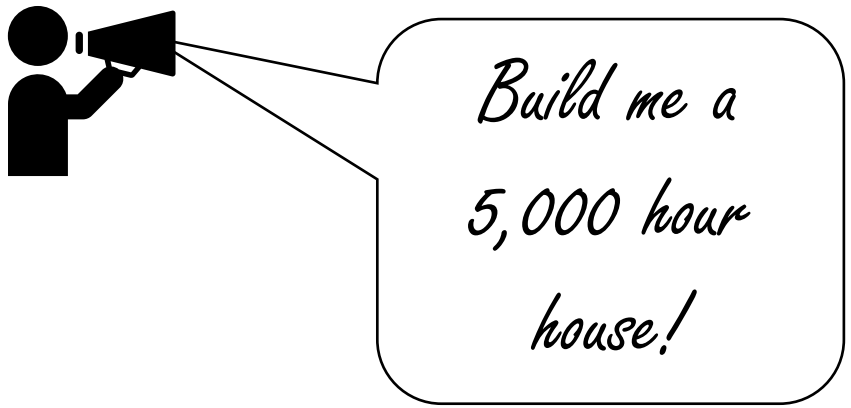




GALORATH

Why Estimate Size?

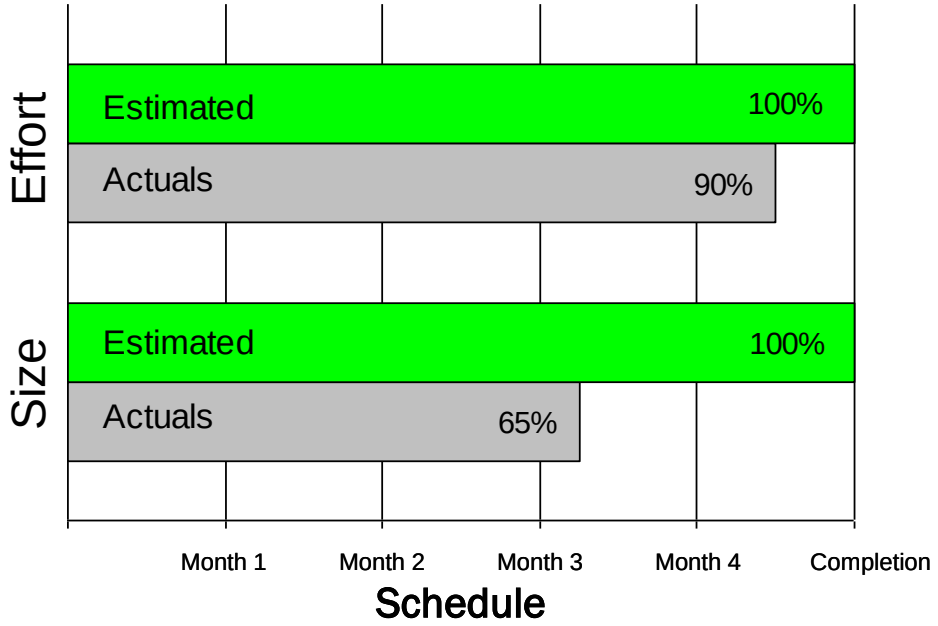
- Size is an essential component of structured repeatable estimation process
- Size can be measured objectively and estimated via a repeatable process
- Size can be used as a basis for comparison with completed development projects
- Size can help track production attributes against estimate
  - Not just resource consumption (i.e. hours)



GALORATH

Tracking Actuals

- How "done" is this project?
- Effort measurement gives the idea that the project is almost complete.
- Size measurement indicates that there is much more work to be done.



You don't want to find out too late in the game that you are going to overrun project cost and schedule

GALORATH

Size metric definition matters as well

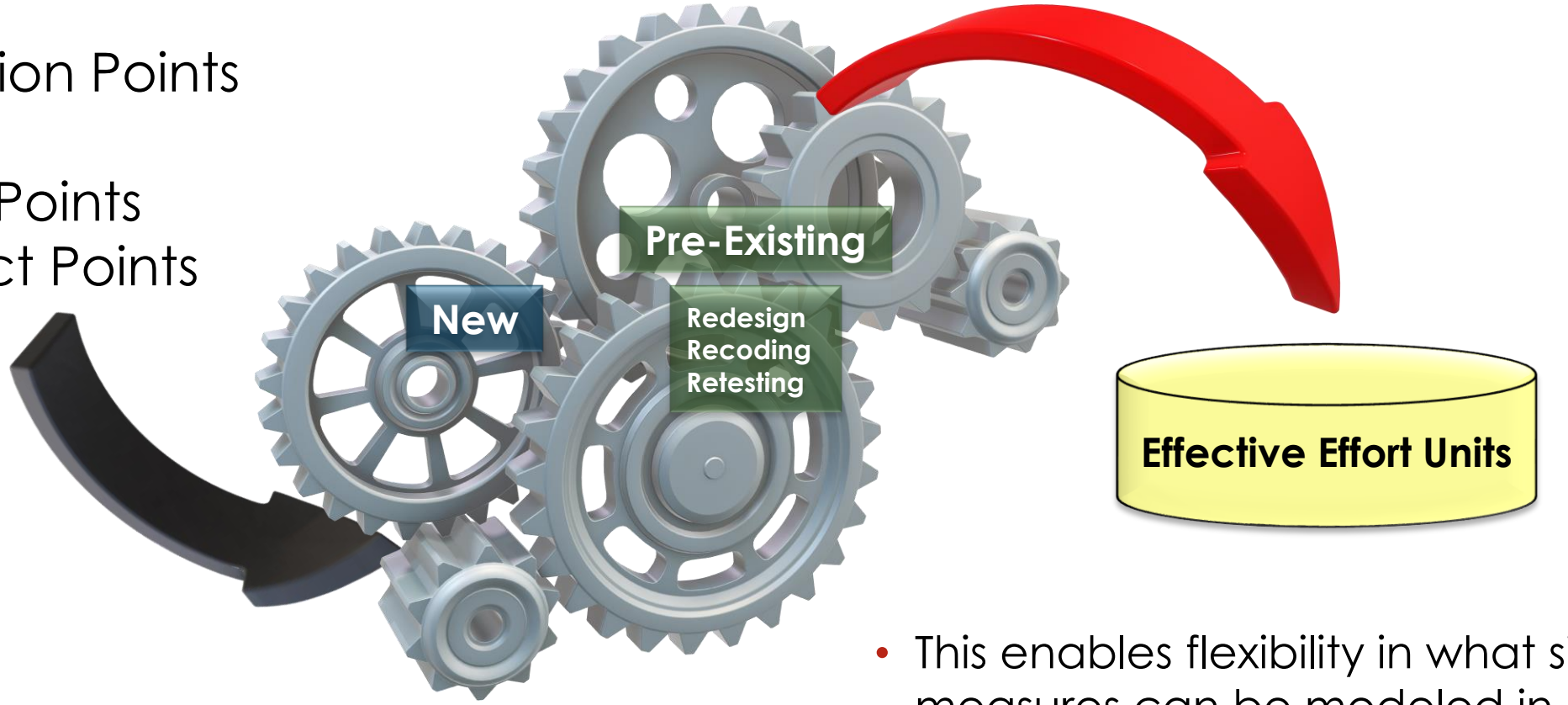
- Do size estimates need to be consistent across the entire application development life cycle?
- Is there a need for an industry standard size measure for benchmarking or productivity metrics?
- Does the size measurement methodology need to be fully documented and repeatable?
- What size measure best fits the application technology or programming language?
- How else can the size measure be used?

Software Sizing Measures

- Wide range of different measures used across industry
- No single common standard
- Measures can be either physical or logical in nature
- SEER-SEM offers various ways of measuring and entering size:
  - IFPUG unadjusted function points
  - Source lines of code
  - Specialized COTS size measures
  - Alternative metrics, in which inputs are converted to lines or function points via custom Size Metrics
    - Cosmic function points
    - Story Points
    - Use Case points
    - RICE objects
    - Other user-defined Size Metrics
- Different size measures may be combined within an estimate and even within a single work element

SEER-SEM: Software Size ⇨ Effective Effort Units

- Function Points
- SLOC
- Story Points
- Object Points
- Etc…



- This enables flexibility in what size measures can be modeled in SEER

GALORATH

Total Size Versus Effective Size

- **Total** software size reflects the total functionality of the program
  - Newly developed functionality
  - Existing functionality (integrated)
  - COTS functionality

- **Effective** Software Size is more directly related to the amount of effort required to develop the software, considering
  - Redesign, reimplementation (recoding) 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

Effective Size Math

NEW CODE

SEER-SEM assumes all life cycle activities will be performed at 100%

EXISTING CODE

SEER-SEM calculates "effective size" based on user-provided rework percentages. Results in a "reuse benefit" that reduces the overall effort and schedule estimates

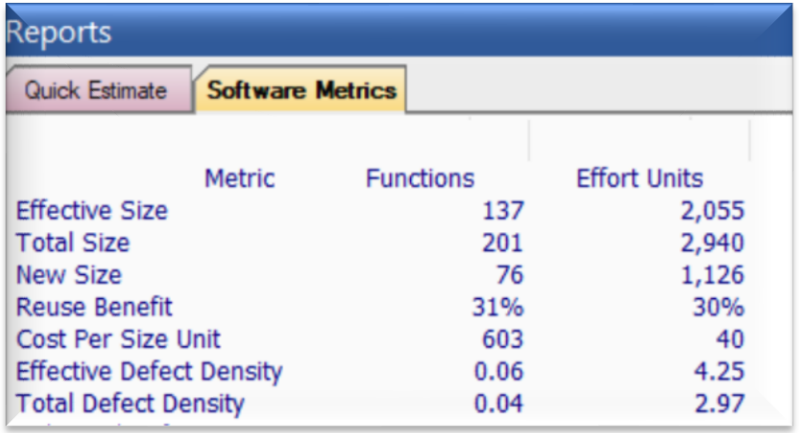
SEER-SEM ESLOC Calculator		Least	Likely	Most
New Code		1,000	2,000	3,000
Design	40%	100%	100%	100%
Implementation	25%	100%	100%	100%
Test	35%	100%	100%	100%
Calculated Values		1,000	2,000	3,000
PERT Mean Weight		1	4	1
New ESLOC			**2,000**	
Existing Code		9,000	10,000	11,000
Redesign	40%	7.00%	19.00%	34.00%
Reimplementation	25%	2.00%	7.00%	9.00%
Retest	35%	5.00%	16.00%	21.00%
Calculated Values		455	1,495	2,552
PERT Mean Weight		1	4	1
Existing ESLOC			**1,498**	
Total ESLOC			**3,498**	
Blue = User Controlled Values				
Red = Model Controlled Values				
Black = Model Calculated Values				

GALORATH

Effective Versus Total Size Measures

- Available in Quick Estimate, Basic Estimate, & Software Metrics reports
- Effective functions/lines
  - Equivalent new size that could be built for the effort described
  - Representation of the effort involved in developing the software
- Total functions/lines
  - Size that will be delivered (new and reused)
  - Representation of the functionality of the software
- "Reuse Benefit" indicates productivity gains

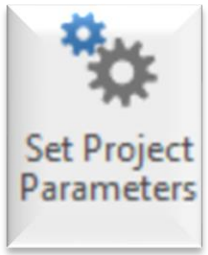
  1 – (effective/total)

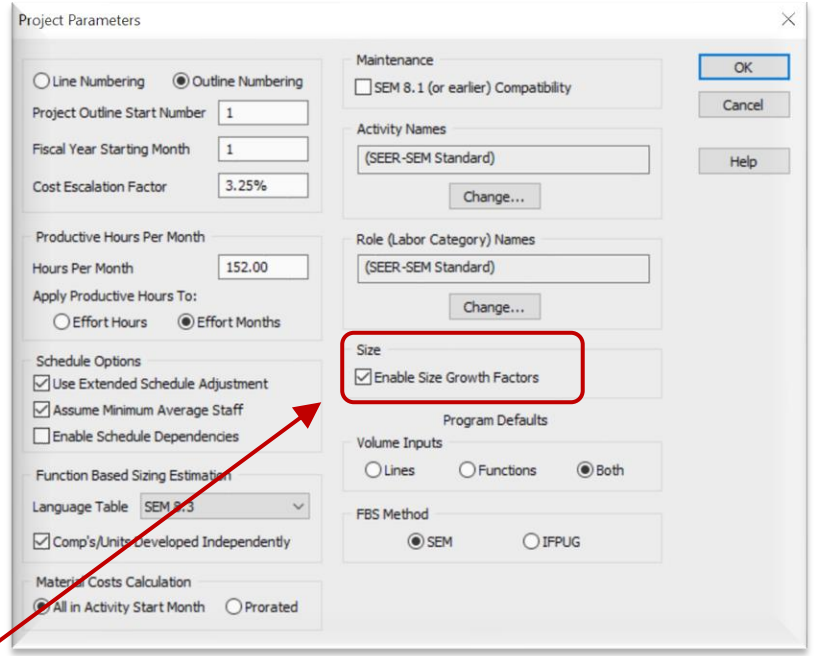


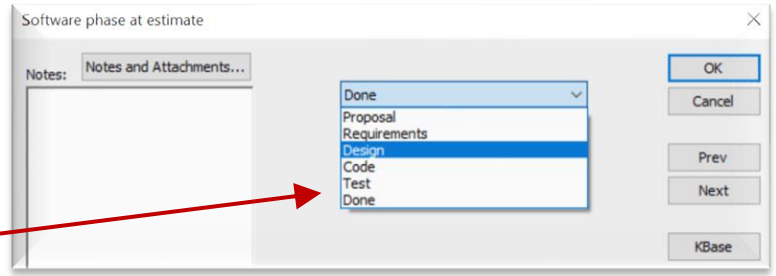
Metric	Functions	Effort Units
Effective Size	137	2,055
Total Size	201	2,940
New Size	76	1,126
Reuse Benefit	31%	30%
Cost Per Size Unit	603	40
Effective Defect Density	0.06	4.25
Total Defect Density	0.04	2.97

Size Growth in Software Estimates

- History demonstrates that software size typically increases from initial estimates
  - Growth in new software (likely)
  - Growth in reused software (unlikely, but possible)
- Reasons for size expansion
  - Optimistic estimates
  - Misunderstood requirements
  - Customer clarifications
  - Scope creep
- SEER-SEM includes size expansion (growth) factors as a parameter input (found under Options Menu |
  - Note that this input parameter applies only to SLOC size inputs – function point scope expansion is handled by Software Phase at Estimate input parameter



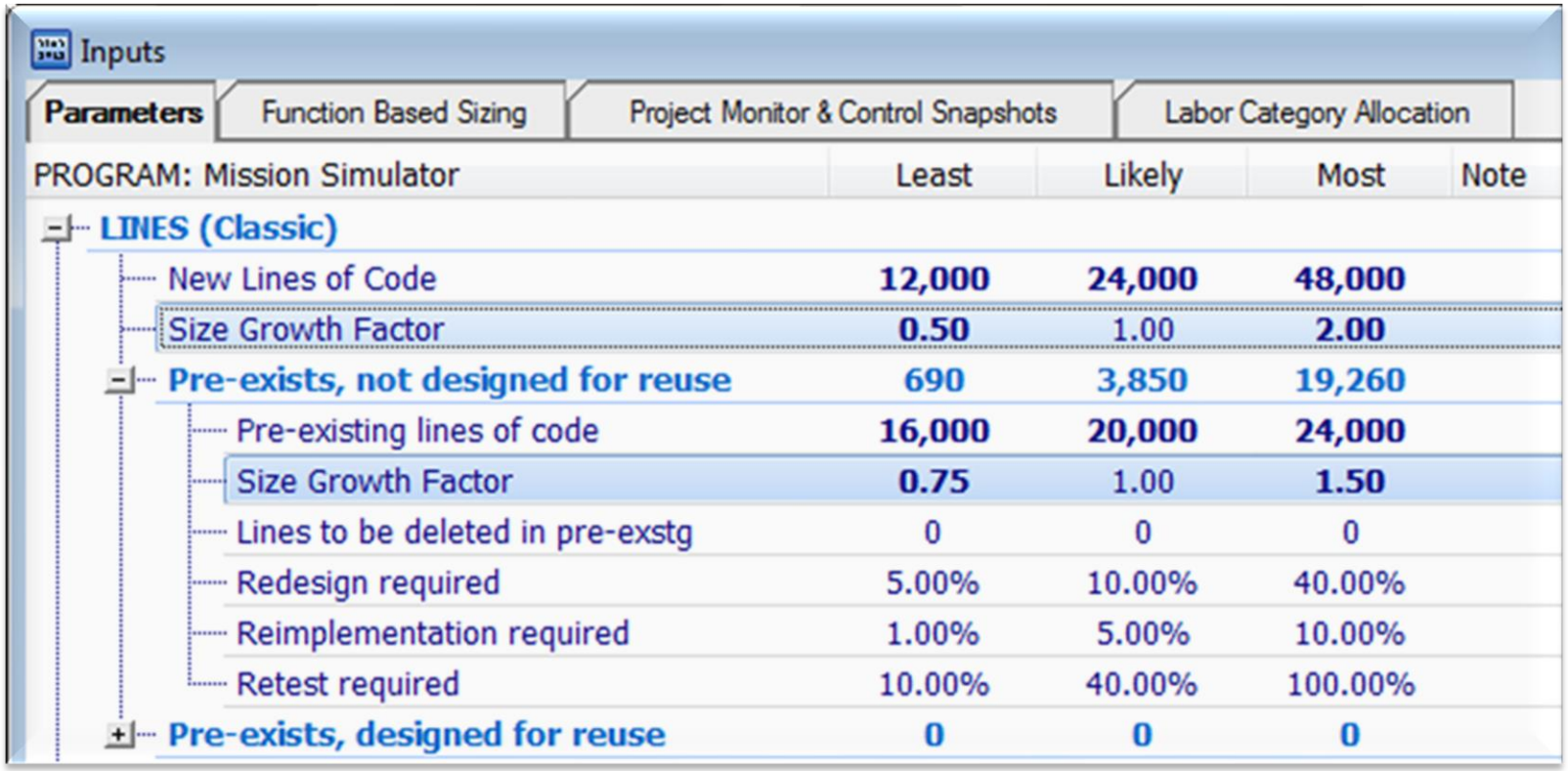




GALORATH

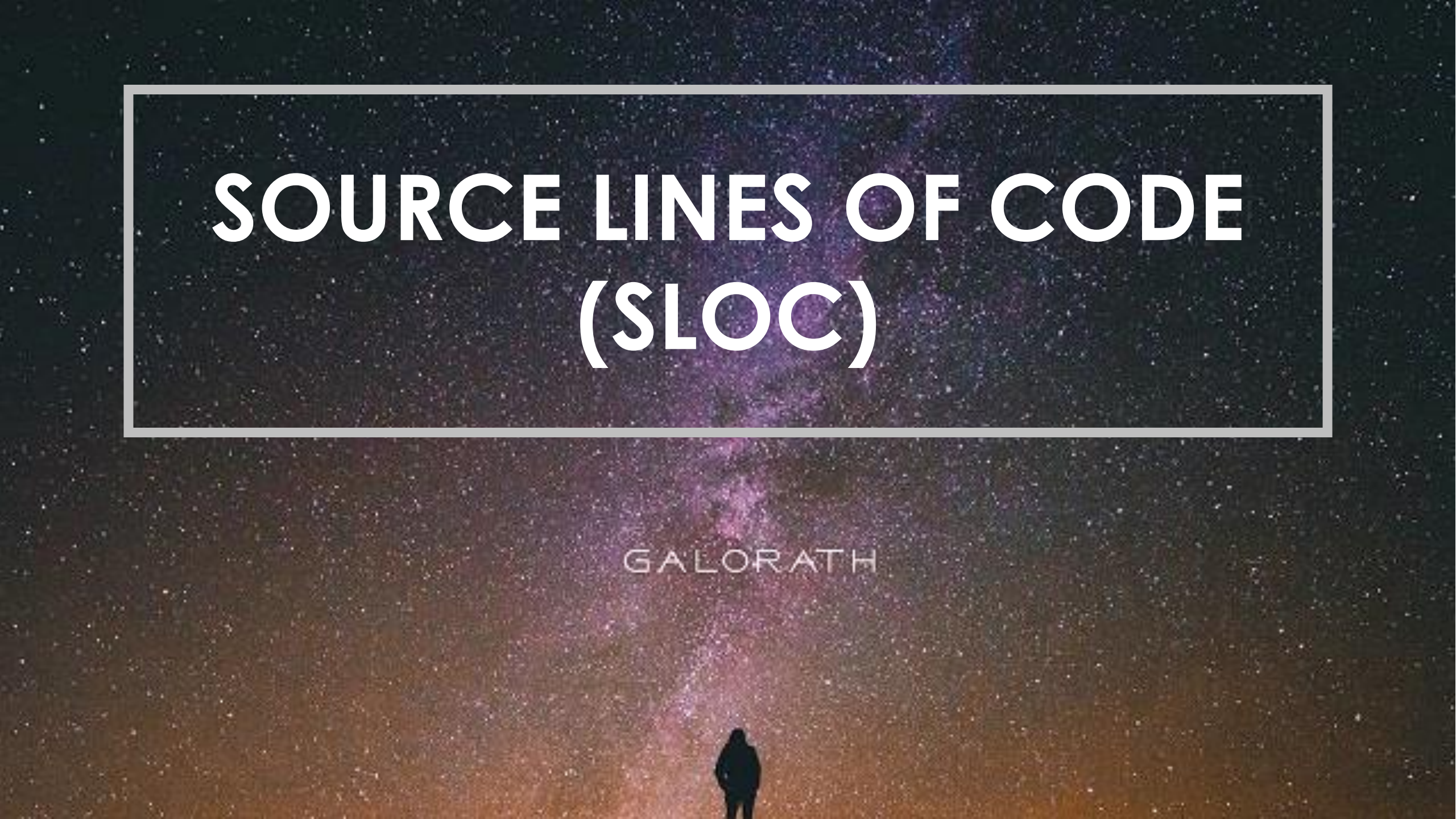
Size Growth Factor

- If enabled, linear size growth factor parameters are visible for all size input categories



PROGRAM: Mission Simulator	Least	Likely	Most	Note
LINES (Classic)				
New Lines of Code	**12,000**	**24,000**	**48,000**	
Size Growth Factor	**0.50**	1.00	**2.00**	
Pre-exists, not designed for reuse	**690**	**3,850**	**19,260**	
Pre-existing lines of code	**16,000**	**20,000**	**24,000**	
Size Growth Factor	**0.75**	1.00	**1.50**	
Lines to be deleted in pre-exstg	0	0	0	
Redesign required	5.00%	10.00%	40.00%	
Reimplementation required	1.00%	5.00%	10.00%	
Retest required	10.00%	40.00%	100.00%	
Pre-exists, designed for reuse	**0**	**0**	**0**	

SOURCE LINES OF CODE (SLOC)



SLOC Definition

- SLOC is probably the oldest and most widely used sizing method.
- Two major types of SLOC measures: physical SLOC and logical SLOC.
  - Specific definitions vary, but the most common definition of physical SLOC is a count of "non-blank, non-comment lines" in the text of the program's source code.
  - Logical SLOC attempts to measure the number of "statements", but specific definitions are tied to specific computer languages.

Logical SLOC Includes

- Executable source lines
- Control (DO WHILE, REPEAT UNTIL, ...)
- Mathematical (i:=a**b/c;)
- Conditional (IF, THEN, ELSE)
- Input/output & formatting
- Data declarations (including data typing & equivalence)
- Deliverable script

Logical SLOC Does *NOT* Include

- Comments & blanks (part of documentation)
- BEGIN statements from BEGIN-END pairs (pair counts as one)
- Continuation separate physical lines for convenience
- Machine/library generated statements (lines instantiated)
- Non-final test code or debug statements (part of testing)

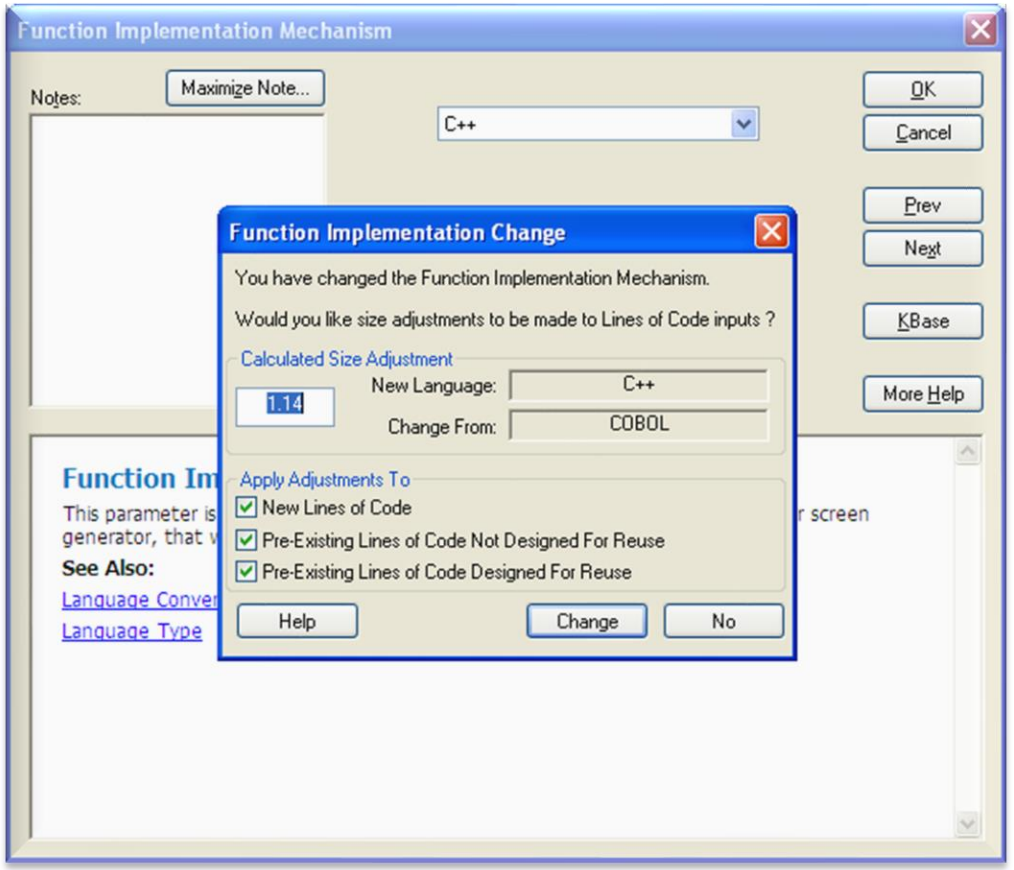
SLOC: Language

- The programming language used to develop a program will impact program size and effort
  - Different languages require different SLOC 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.					

Size in Different Languages

- In general, SEER will assume your SLOC estimate takes into account the programming language being used in development
  - This means that the Function Implementation Mechanism input parameter does not impact the estimate output if SLOC is used
  - However, if you decide to change the Function Implementation Mechanism setting, SEER will suggest a conversion factor to revise the SLOC estimate
    - If your SLOC estimate already takes the different programming language into account, then reject the suggested size adjustment
    - If you have not already made the conversion, go ahead and select *Change* to enable the size adjustment



GALORATH

SLOC: Strengths and Weaknesses

Strengths

- ✓ Easy to estimate
- ✓ Relatively easy to measure the actual size, either by physically counting code or through an automated code counter
- ✓ There is usually plenty of historical data available in the form of legacy applications
- ✓ Most estimating tools accept SLOC as a sizing input for estimating purposes

Weaknesses

- ✘ No industry standards defining SLOC; counting methodologies can differ significantly between – and even within – organizations
- ✘ Measuring code volume provides no incentive for developers to reduce the number of SLOC in a product
- ✘ Irrelevant to today's GUI, COTS, object-oriented, and code-generated development
- ✘ SLOC measures components, not completed products
- ✘ Studies have concluded that code volume is not an accurate predictor of effort

SLOC: Productivity Paradox

Same software project, 3 different languages

	A	B	C
SLOC Productivity	556	350	333
$ per SLOC	$30	$47	$50
Language	Assembly	Ada 83	C++
Total SLOC	10,000	3,500	2,500
Total Effort (staff-months)	18.0	10.0	7.5
Total Cost	$300K	$166K	$125K

"Best" metrics

Highest effort and cost

GALORATH

SLOC: Examples

- Programming language: COBOL

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. HELLOWORLD.
000300
000400*
000500 ENVIRONMENT DIVISION.
000600 CONFIGURATION SECTION.
000700 SOURCE-COMPUTER. RM-COBOL.
000800 OBJECT-COMPUTER. RM-COBOL.
000900
001000 DATA DIVISION.
001100 FILE SECTION.
001200
100000 PROCEDURE DIVISION.
100100
100200 MAIN-LOGIC SECTION.
100300 BEGIN.
100400 DISPLAY " " LINE 1 POSITION 1 ERASE EOS.
100500 DISPLAY "Hello world!" LINE 15 POSITION 10.
100600 STOP RUN.
100700 MAIN-LOGIC-EXIT.
100800 EXIT.
```

17 SLOC

- Programming language: C

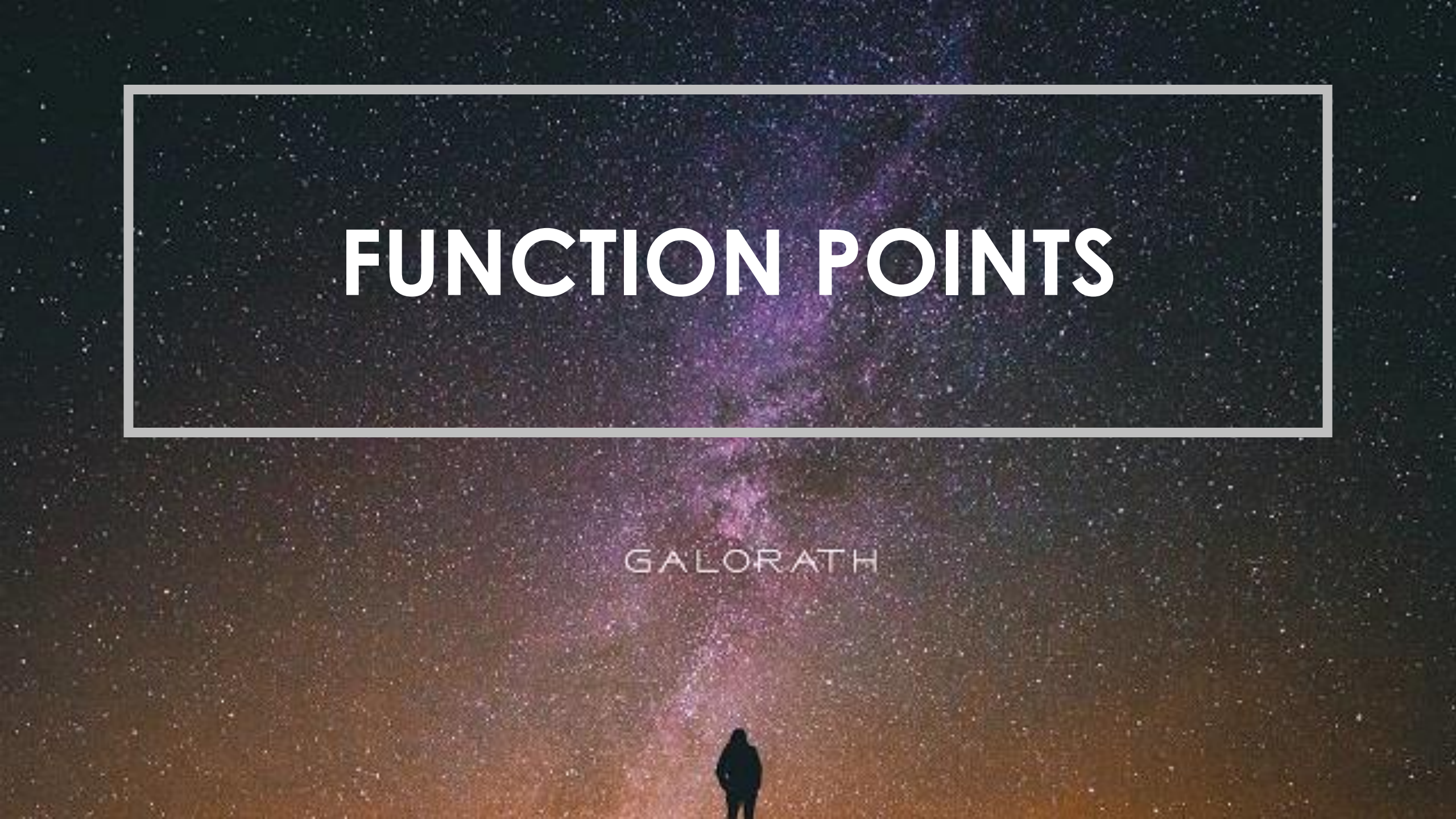
```c
#include <stdio.h>

int main(void) {

   printf("Hello World");
}
```

4 SLOC

FUNCTION POINTS



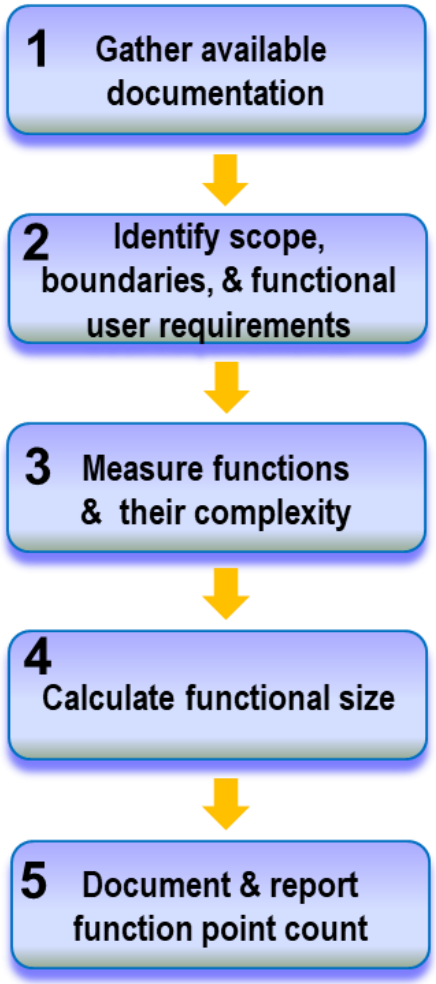
Function Points: Definition

- Function point methodology developed by Allan Albrecht at IBM in the late 70s to address shortcomings in line of code metrics
- Function points are a standardized metric that describes a unit of work product suitable for quantifying software that is based on what the end-user requests and receives
- Function points are a logical size measure, versus a physical size measure like SLOC or Objects
- Function points are comprised of inputs, outputs, inquiries, internal data, and external interface data
- Technology, platform, and language independent

- ✓ Function point standards are established and managed by International Function Point Users Group (IFPUG)
- ✓ Function points are accepted as a standard size measure by ISO (ISO 20926:2009)
- ✓ The Certified Function Point Specialist (CFPS) professional certification program recognizes trained experts

Function Points: Methodology

FPA provides a consistent, documentable, repeatable measurement methodology



1. Gather available documentation: Collect all documentation and resources available at the time of measurement. Documentation should describe the functionality delivered by the software included in the scope of the count. Counting resources may vary depending on the phase in the lifecycle phase at which the analysis is being performed.

2. Identify scope, boundaries, & functional user requirements: The business purpose will determine what requirements should be included in the FPA and will determine the logical boundaries between application(s) and the end users.

3. Measure functions & their complexity: The functional size is based on the measurement of two function types and their complexity:
   - Data functions: Provide functionality to the user to meet internal and external data requirements; Internal Logical Files (ILF) & External Interface Files (EIF)
   - Transactional functions: Provide functionality to the user to process data by an application; External Inputs (EI), External Outputs (EO), & External Inquiries (EQ)

4. Calculate functional size: The total functional size is calculated based on different formulas depending on the purpose and scope of the function point count.

5. Document & report function point count: A fully documented FPA will facilitate traceability, usability and maintainability.

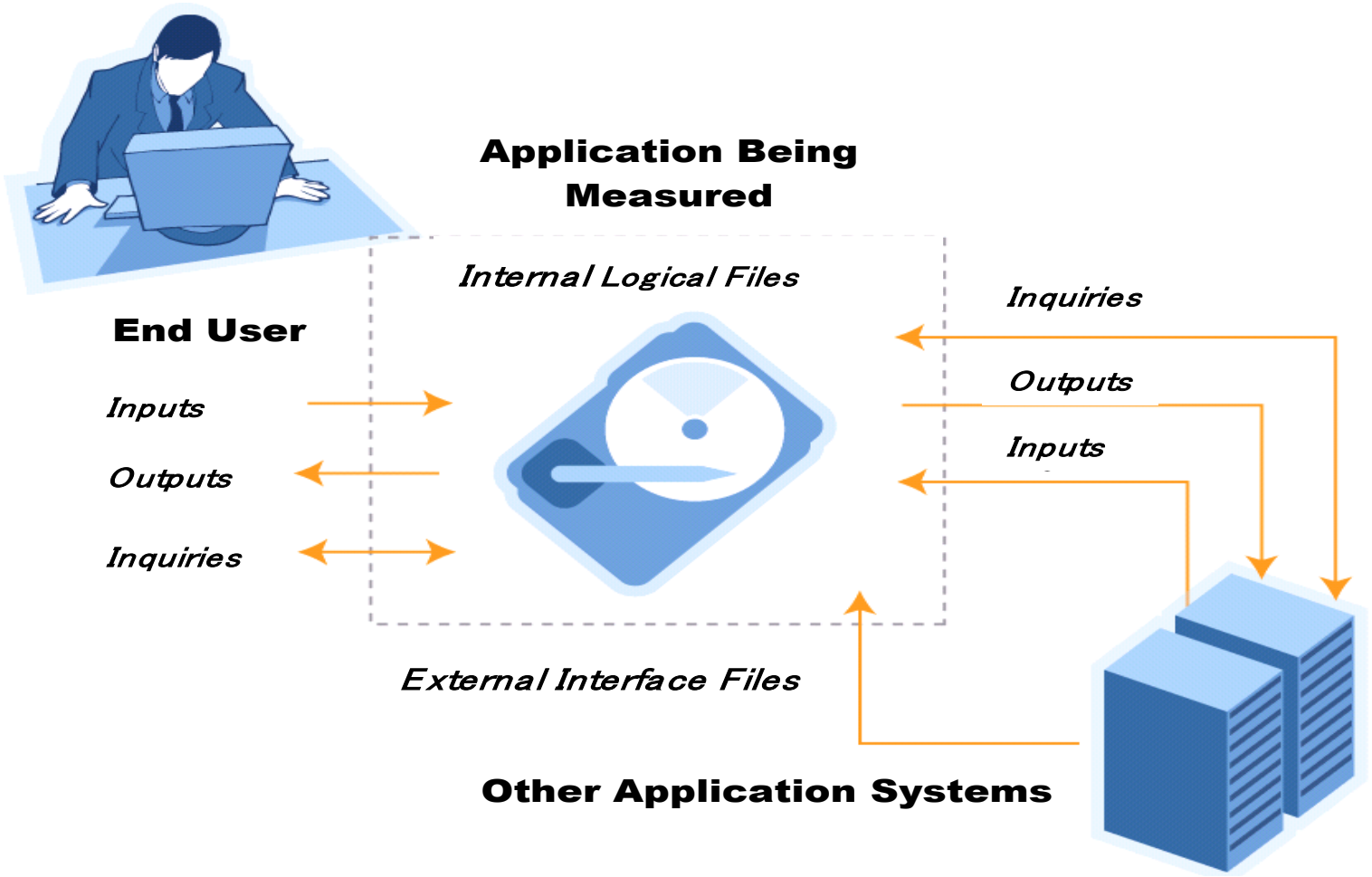
Standard Function Points Matrix

- IFPUG's matrix identifies how many function points each function type receives based on complexity
- This matrix is the foundation of the repeatable, documentable methodology

	Low	Average	High
Internal Logical File	7	10	15
External Interface File	5	7	10
External Input	3	4	6
External Output	4	5	7
External Inquiry	3	4	6

Function Points: Methodology

Function points look at the logical size of the application from the customer's perspective



Function Point Counting Resources

Trained analysts measure function points based on functionality derived from the following typical resources:

- *User/analyst interviews*
- *Requirements documents*
- *Design documents*
- *Data dictionaries*
- *Use cases*
- *User guides*
- *Screen captures*
- *Actual software*
- *Entity-relationship models*
- *Semantic object models*

Function Points: Strengths and Weaknesses

Strengths

- ✓ Standards are established and managed by International Function Point Users Group (IFPUG)
- ✓ Function points accepted as a standard size measure by the International Organization for Standardization (ISO 20926:2003)
- ✓ Certified Function Point Specialist (CPFS) professional certification program recognizes trained experts
- ✓ Available from early requirements stage and applicable for full life-cycle analysis
- ✓ Technology, platform, language independent
- ✓ Fully documentable and repeatable
- ✓ Assists with requirements management; functional size traceable throughout entire life cycle
- ✓ Generates logical, straightforward metrics, as illustrated on the next slide

Weaknesses

- ✘ Accurate counting requires in-depth knowledge of standards
- ✘ Counting is primarily a manual process
- ✘ Some variations exist that aren't standardized (Mark II, 3D, full, feature points, object points, etc.)

Function Points: Productivity Paradox Corrected

Same software project, 3 different languages

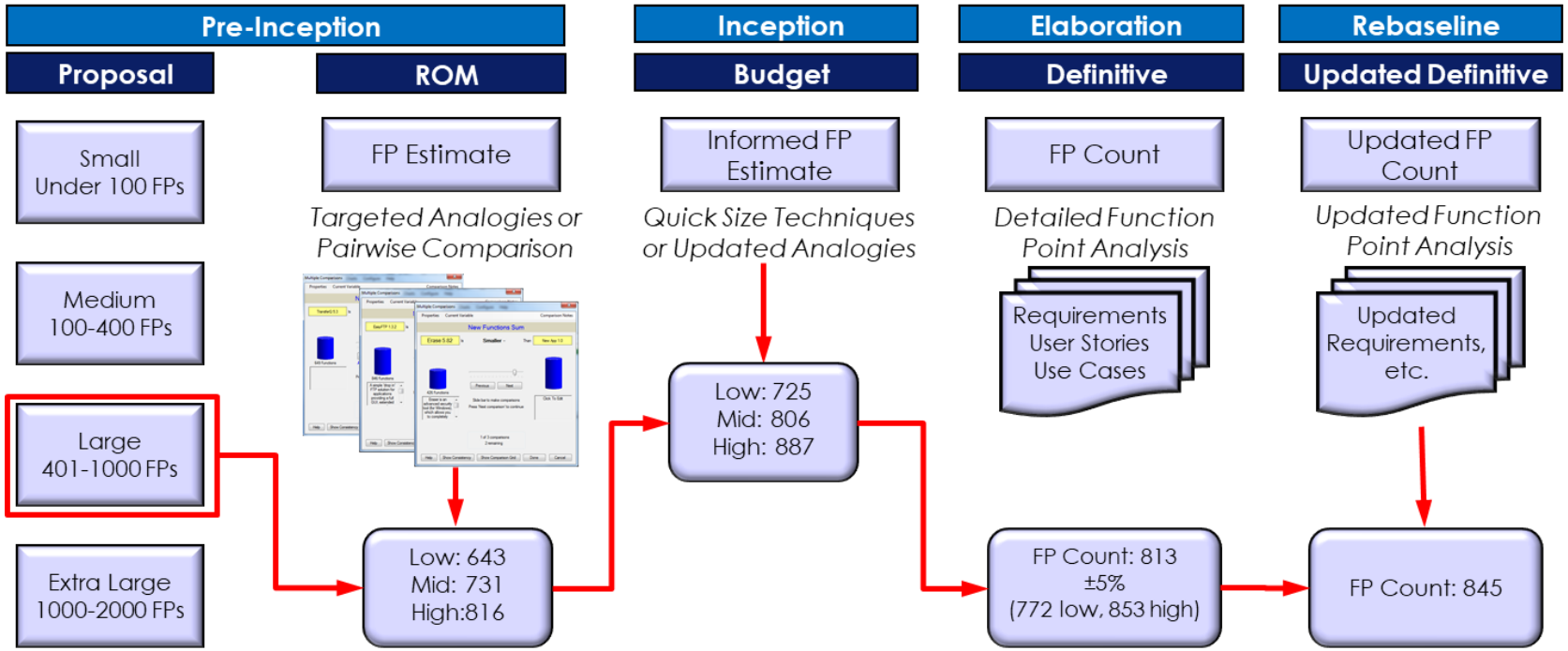
	A	B	C
FP Productivity	2.78	5.00	6.67
$ per FP	$6,000	$3,320	$2,500
Language	Assembly	Ada 83	C++
Total Function Points	50	50	50
Total Effort (staff-months)	18.0	10.0	7.5
Total Cost	$300K	$166K	$125K

Best metrics

Lowest effort and cost

Function Points Through the Development Life Cycle

- Applying FP range estimation techniques to early estimates will provide a quantifiable foundation and will establish a consistent estimation methodology across the entire project lifecycle



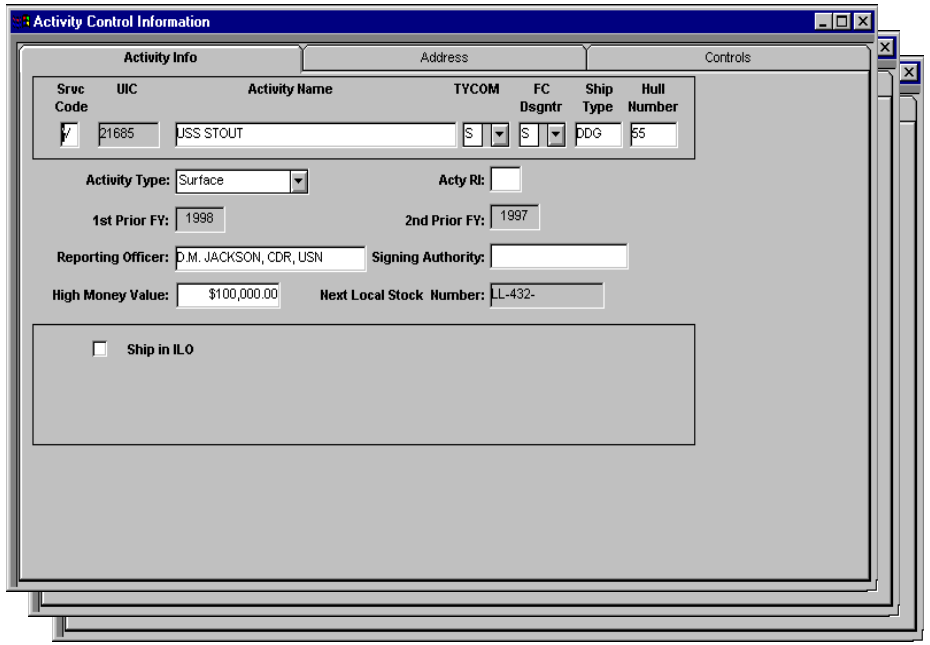
- Using a consistent sizing metric throughout the project lifecycle is an industry best practice; it provides traceability as the estimate is updated as more complete information and requirements become available
- Because it is linked directly to requirements (or the lack thereof), applying this consistent methodology throughout the life cycle enables improved communication with the business customer as to what has changed and why

Function Points Example #1

Requirement	Function Type	Complexity	Quantity	Function Points
The system shall store employee information	Internal logical file	Average	1	10
The system shall permit the user to add, change, and delete employee data	Input	Low	3	9
The system shall permit the user to search for an employee	Inquiry	Low	1	3
The system shall produce a monthly employee summary report	Output	Average	1	5

27 FP

Function Points Example #2



Function	Function Type	Complexity	Quantity	FPs
Activity control Info	ILF	Low	1	7
Enter activity control info	EI	Ave	1	4
Retrieve activity control info	EQ	Ave	1	4
Modify activity control info	EI	Ave	1	4

19 FP

SITE>ACTIVITY CONTROLS>OWN ACTIVITY>ACTIVITY CONTROL INFO

On the Activity Info tab

1. Enter or modify the mandatory fields:

   Srvc Code, Activity Name, TYCOM, FC Dsgntr, Ship Type, Hull Number, Activity Type, Acty RI, Reporting Officer, Signing Authority, High Money Value.

2. Check **Ship in ILO**, if ship is undergoing an Integrated Logistics Overhaul. (This option is not yet available.)

   Additional options display for selection.

On the Address tab

1. Enter or modify the mandatory fields:

   Address, City, State, Zip.

On the Controls tab

1. Enter or modify mandatory fields:

   Fiscal Year, Transfer Tape, Supply Status.

   The next Transmittal Number for the current fiscal year is filled by default.

   The ILO Sequence Numbers is used to maintain ship/ILO tape sequence numbers.

2. Select required Interfaces.

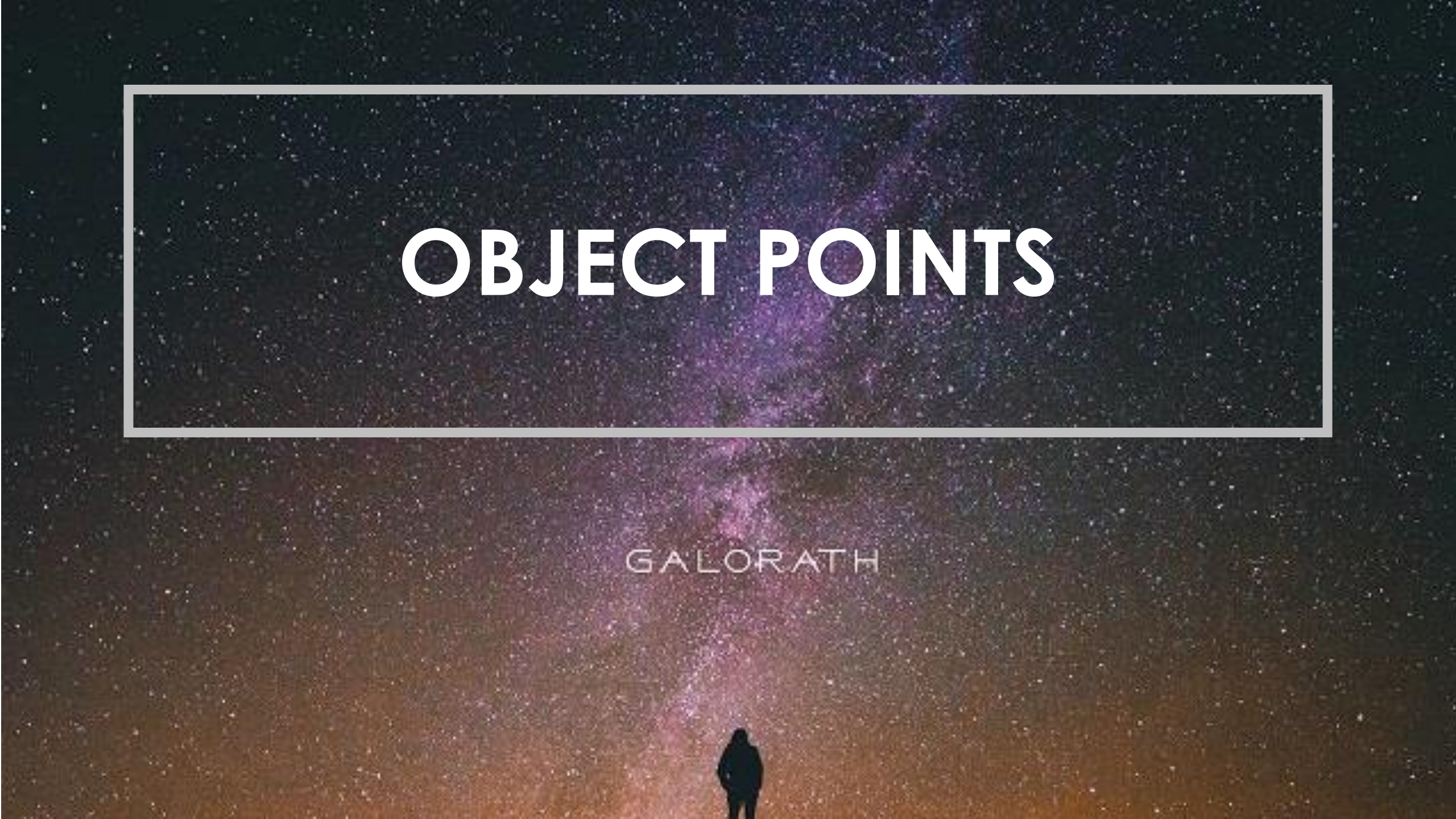
3. The Force Inventory Drawdown allows you to enter a Sequence Number and the Last Date Uplined information.

To complete the process

1. Click **Apply**.
2. Click **Close Window**.

GALORATH

OBJECT POINTS



Object Points: Definition

- Object-oriented (OO) design and development methodologies lend themselves to object-oriented size measurement
- Methodology typically involves counting the number of classes and methods (services) and determining the complexity of each identified object
  - Complexity might be driven by number of attributes, descendant classes, classes referenced, etc.
  - Weighted object points assigned based on complexity

Object Points: Strengths and Weaknesses

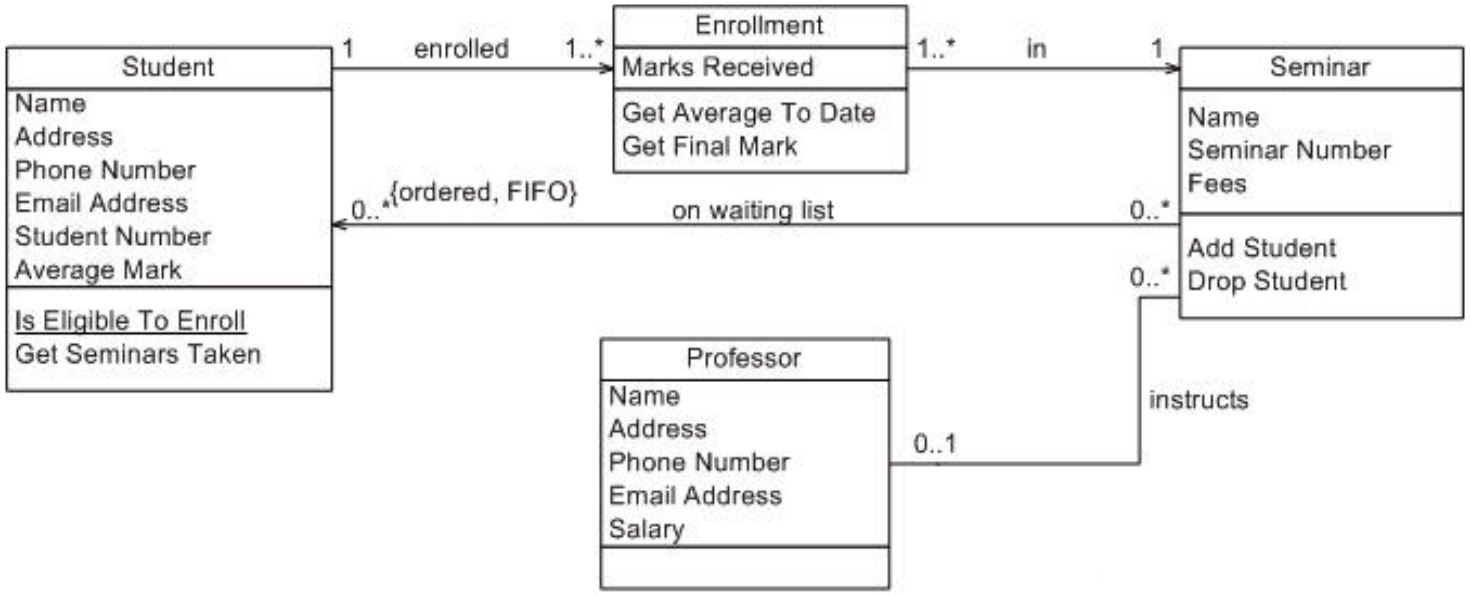
Strengths

- ✓ Suitable for estimating size of OO systems
- ✓ Closely related to the artifacts produced by developers, which can be counted directly to compare estimates to actuals
- ✓ Less experience required than counting function points

Weaknesses

- ✘ No single industry standard to drive estimation process repeatability
- ✘ Estimation techniques can easily vary across projects and even within projects
- ✘ Variation can cause size measures to become very complex to estimate and track
- ✘ Not all estimation tools accept object points as an input

Object Points Example #1

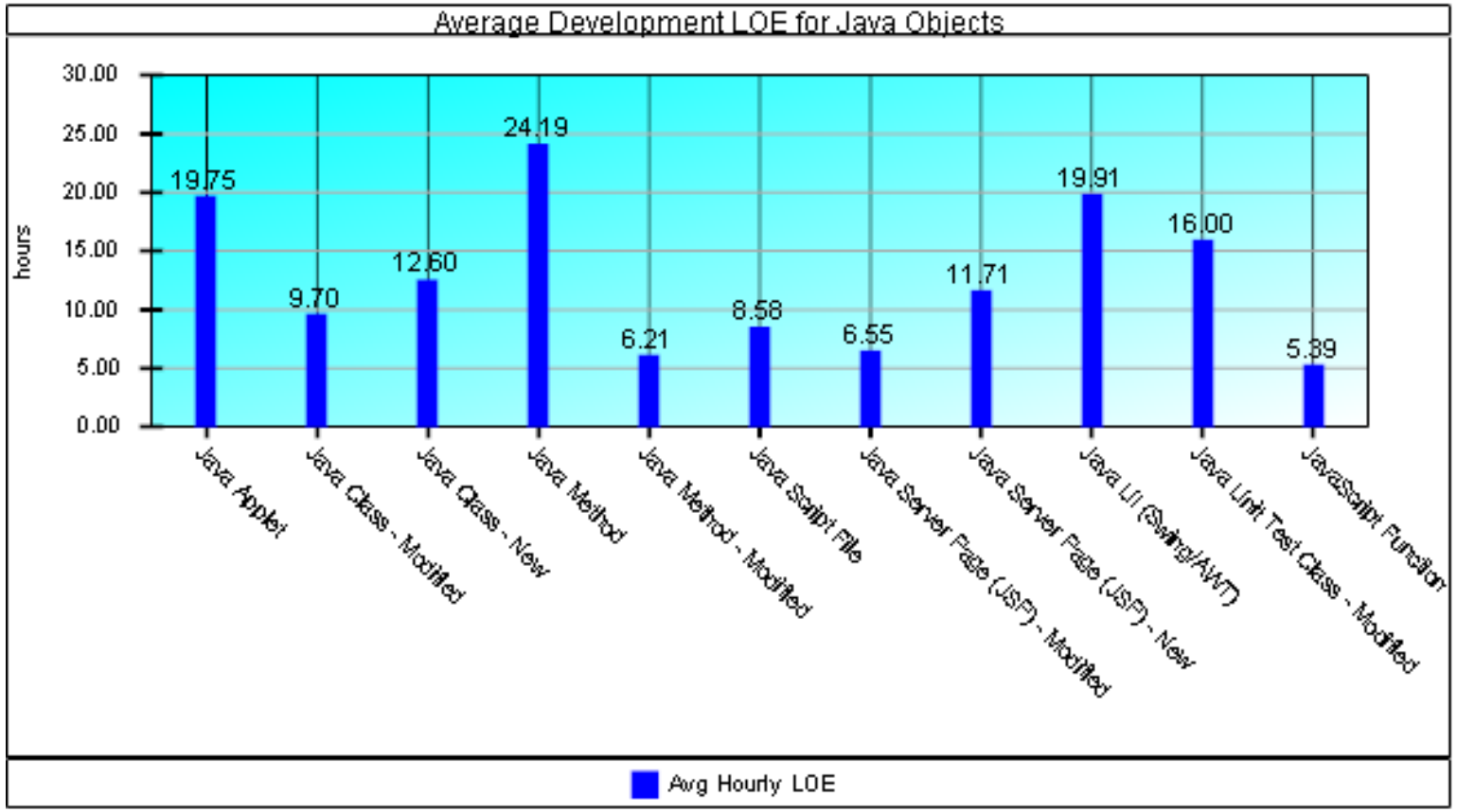


- Class enrollment application
  - 4 low complexity internal classes (7 points each)
  - 2 low complexity input methods (3 points each)
  - 2 low complexity output methods (4 points each)
  - 2 low complexity inquiry methods (3 points each)

48 Object Points

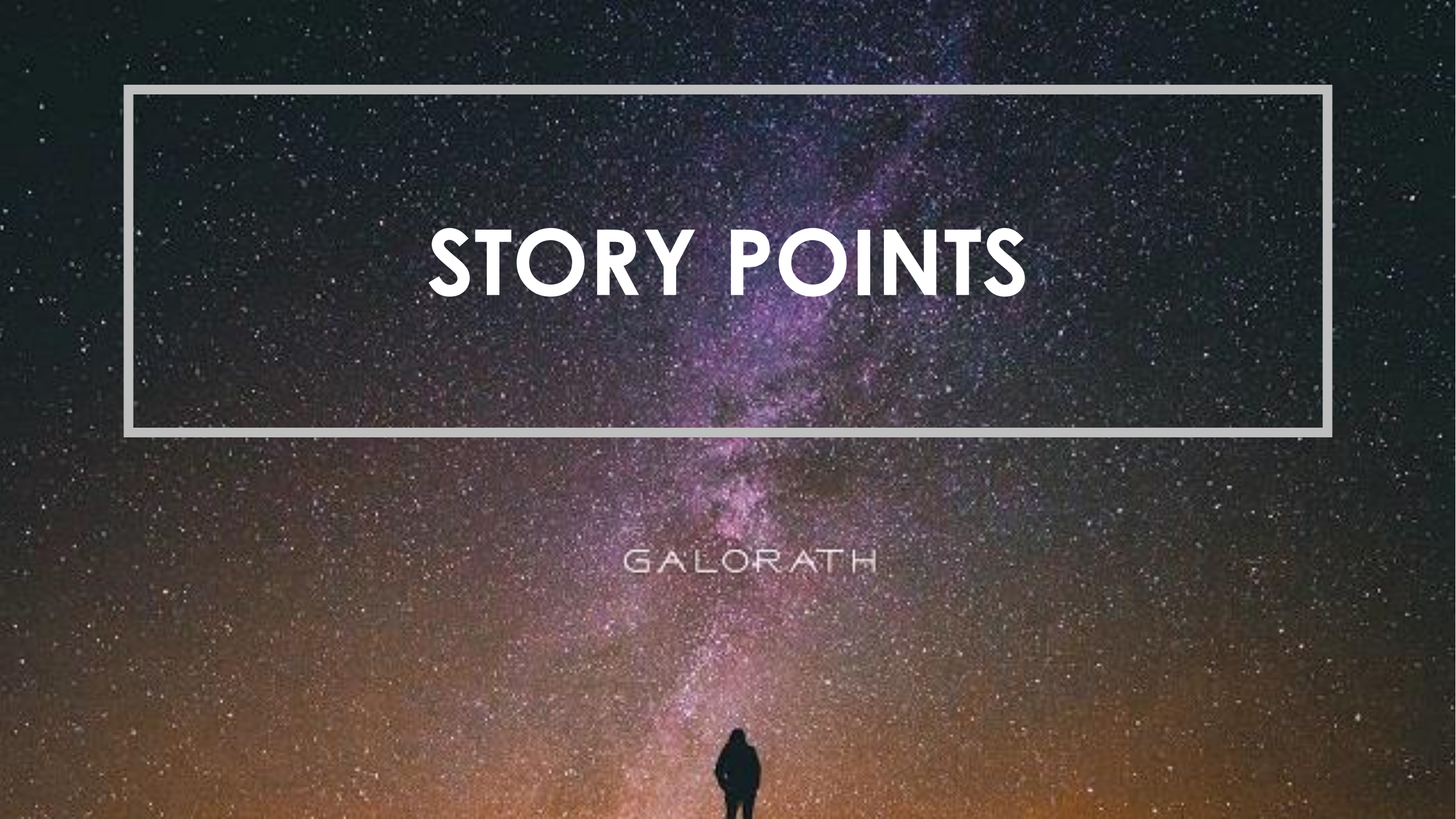
Object Points Example #2

- Using well defined object definitions, project teams use historical project or organizational data to build effort estimates



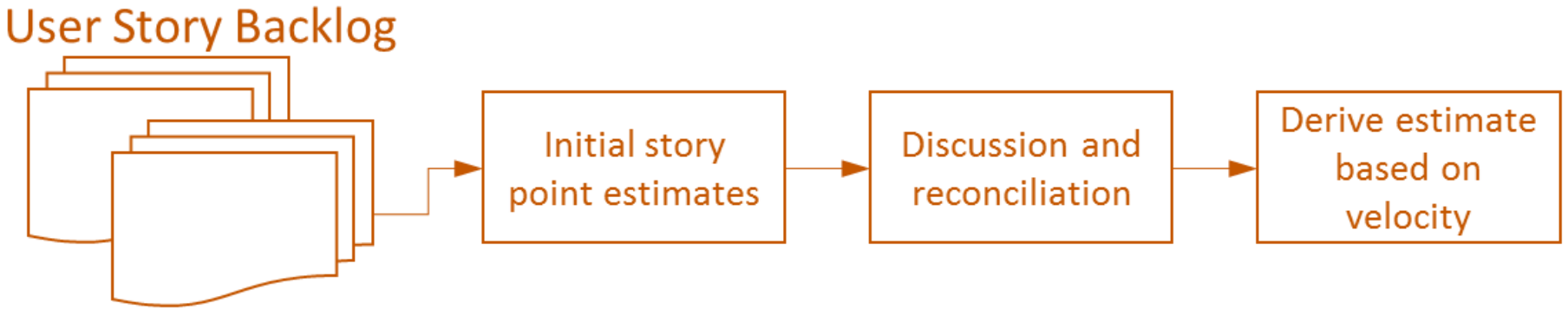
GALORATH

STORY POINTS



Story Points: Definition

- The first rule about story points…is that there is no standard definition of a story point
- Estimation techniques on Agile software development projects vary widely, but one of the most common methods is story point estimation



- User stories are short descriptions of a desired function or feature written from an end-user perspective
  - Typically expressed as "As a user of this system, I want X feature so that I can accomplish Y"
  - Serves as the basis (requirements) for what software will be built

Story Points: Methodology

- Story points help teams assess the relative difficulty of the work they need to accomplish
  - Typically starts with a "reference" story that serves as the basis of comparison to all other stories
  - No "standards" for story points and they are determined on a team-by-team basis
- Planning poker provides opportunity for everyone on the product team to have input to size estimates based on their roles, perspectives, and experiences
  - Teams often apply Fibonacci or other numerical sequences
  - Discussion and iteration helps the team consider potential risk areas and develop a collective agreement about what stories to include in a sprint
- Estimating story points and applying velocity metrics can reduce biases and natural tendencies that typically occur when estimating level of effort in hours

Story Points: Strengths and Weaknesses

Strengths

- ✓ Experienced teams can use story points very effectively
- ✓ The immediacy of feedback of data from the previous sprint during retrospectives provides the opportunity for lessons learned to be applied right away
- ✓ The relative nature of story points allows teams to tailor and calibrate the size unit to their own situation
- ✓ Collaboration and iteration with all stakeholders, including the customer/product owner encourages communication and expectation management

Weaknesses

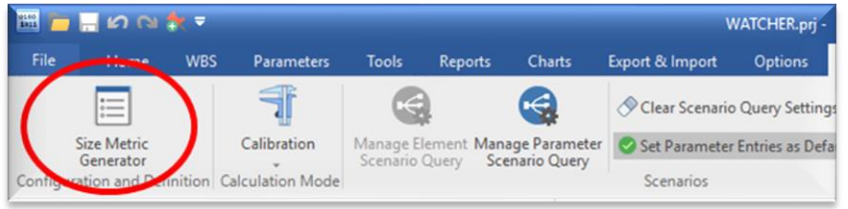
- ✘ Zero standards or consistency across organizations or even within organizations
- ✘ Impossible to generate estimates to establish initial project budgets
- ✘ New scrum teams take some time to dial in story point estimates
- ✘ Impossible to perform organizational portfolio management with consistent metrics across projects
- ✘ Very difficult to communicate sizing assumptions to stakeholders

CUSTOM SIZE METRICS

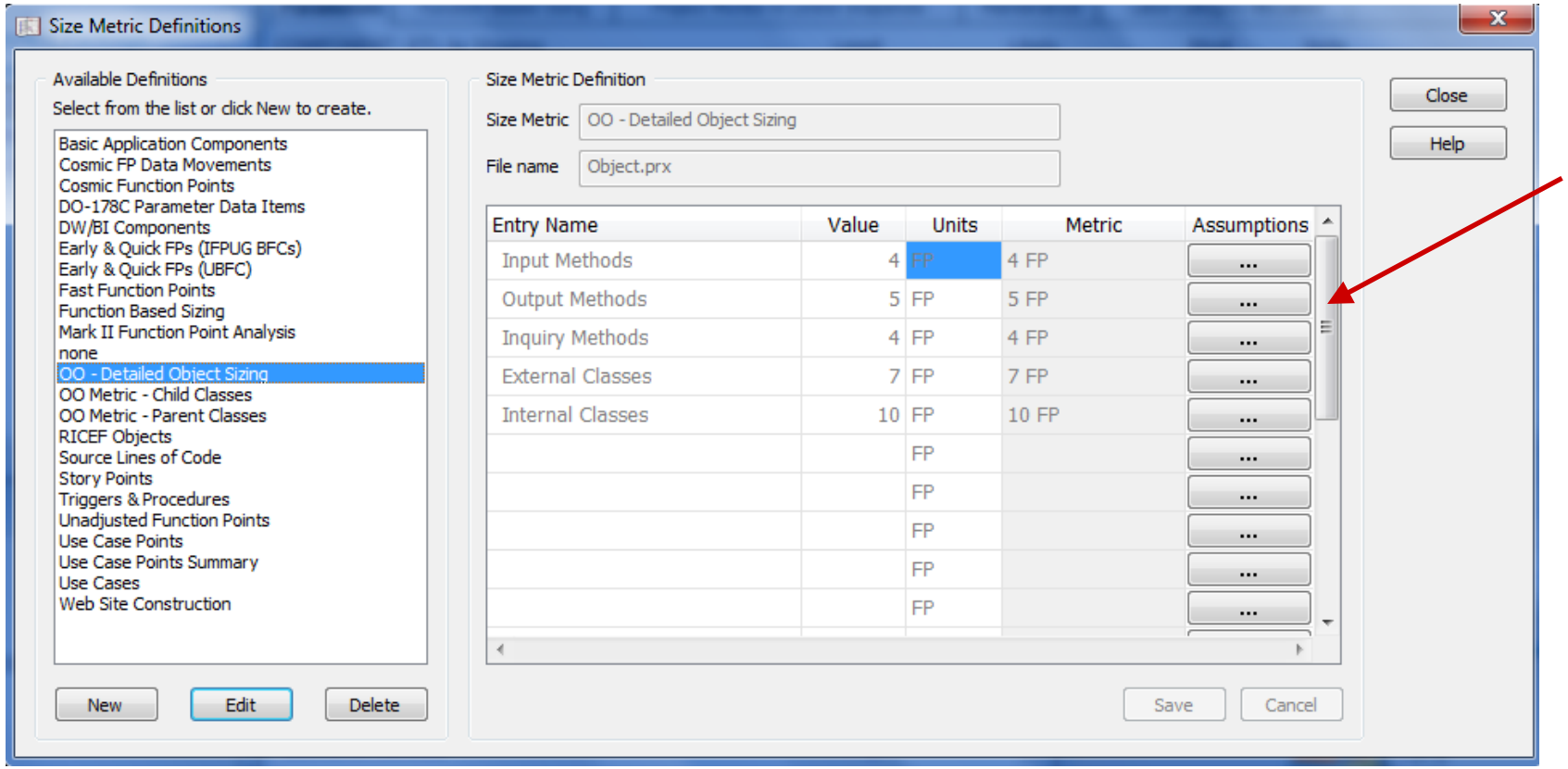
Custom Size Metrics

- SEER-SEM provides the ability to create custom sizing metrics that can be defined by any organization
  - Allow the users in the organization to count what they know
  - Provides a consistent, standardized conversion into metrics (SLOC or UPF) which are understood by SEER-SEM
  - Can be used in place of or in addition to standard SEER size metrics (SLOC and FP)
- Size Metrics are basically conversion tables:
  - Converted to source lines, unadjusted Function Points or Hours
- Example: Choose Size Metric Description…OO-Detailed Object Sizing
  - Enter size inputs in number of:
    - Input Services
    - Output Services
    - Inquiry Services
    - External Classes
    - Internal Classes
  - Object specifications are expressed as Unadjusted Function Points, and added to any other size entered elsewhere

Viewing Size Metric Definitions



- Access this screen through *Advanced* | *Size Metric Definitions*
- Shows how the selected size metric is converted into SLOC, Unadjusted Function Points or Hours

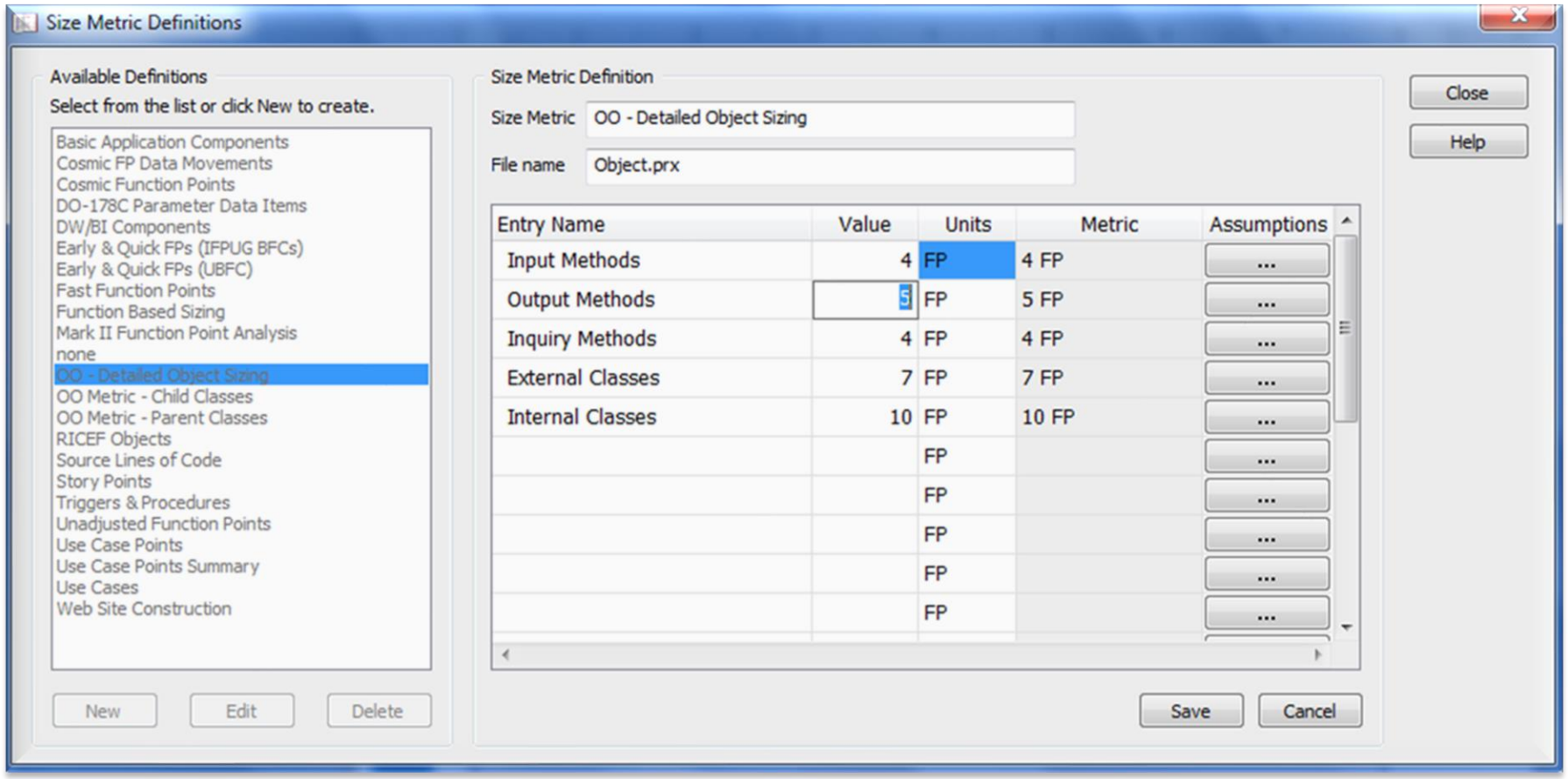


Define assumptions for Size Metrics converted to hours

GALORATH

Editing Size Metric Definitions

- For Size Metrics based on Function Points or SLOC, simply click on the Edit button and make the adjustments
- Don't forget to click Save when done

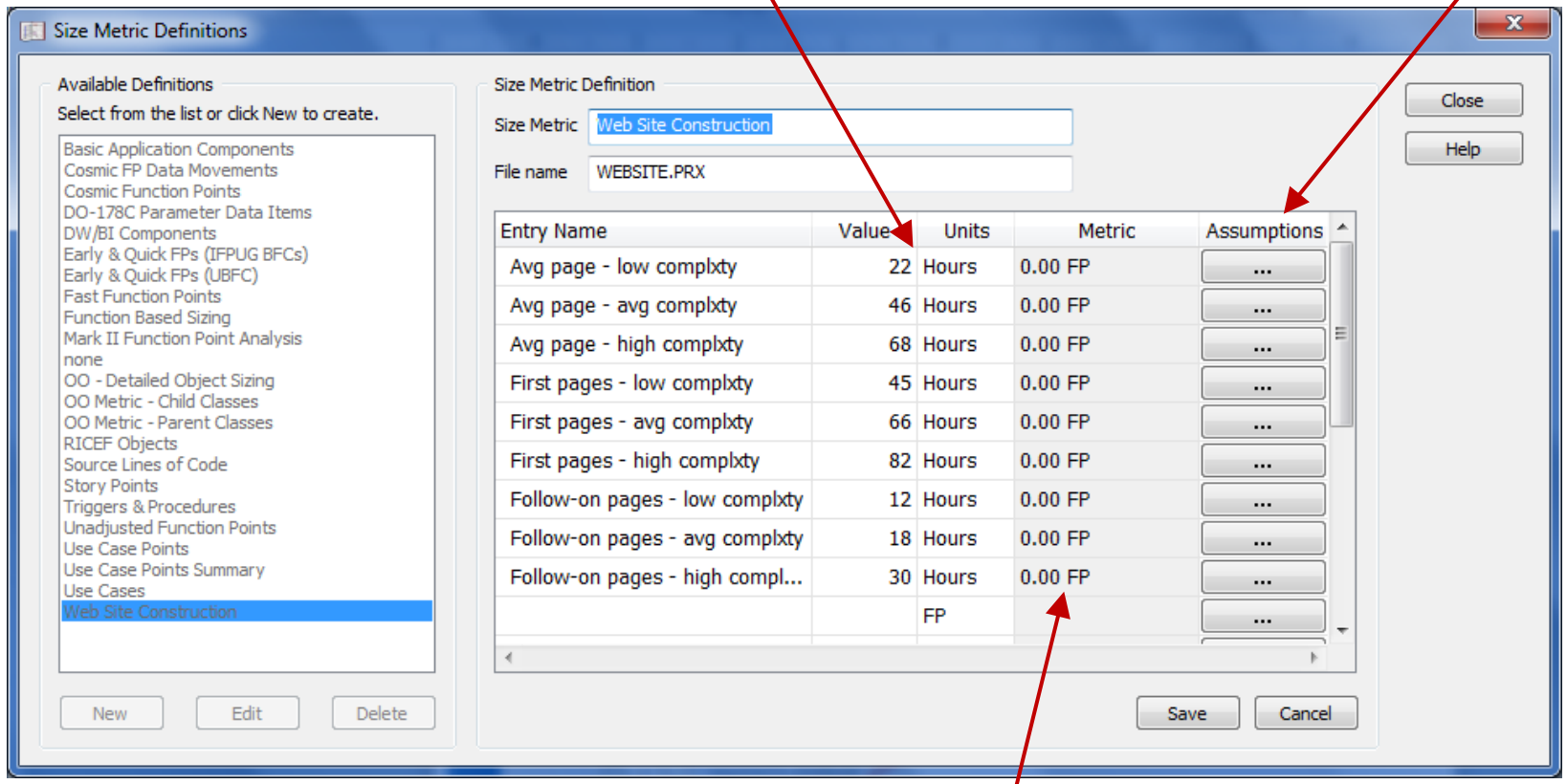


GALORATH

Editing Effort-Based Size Metrics

Define the assumptions (see next slide)

Enter the expected hours for each item

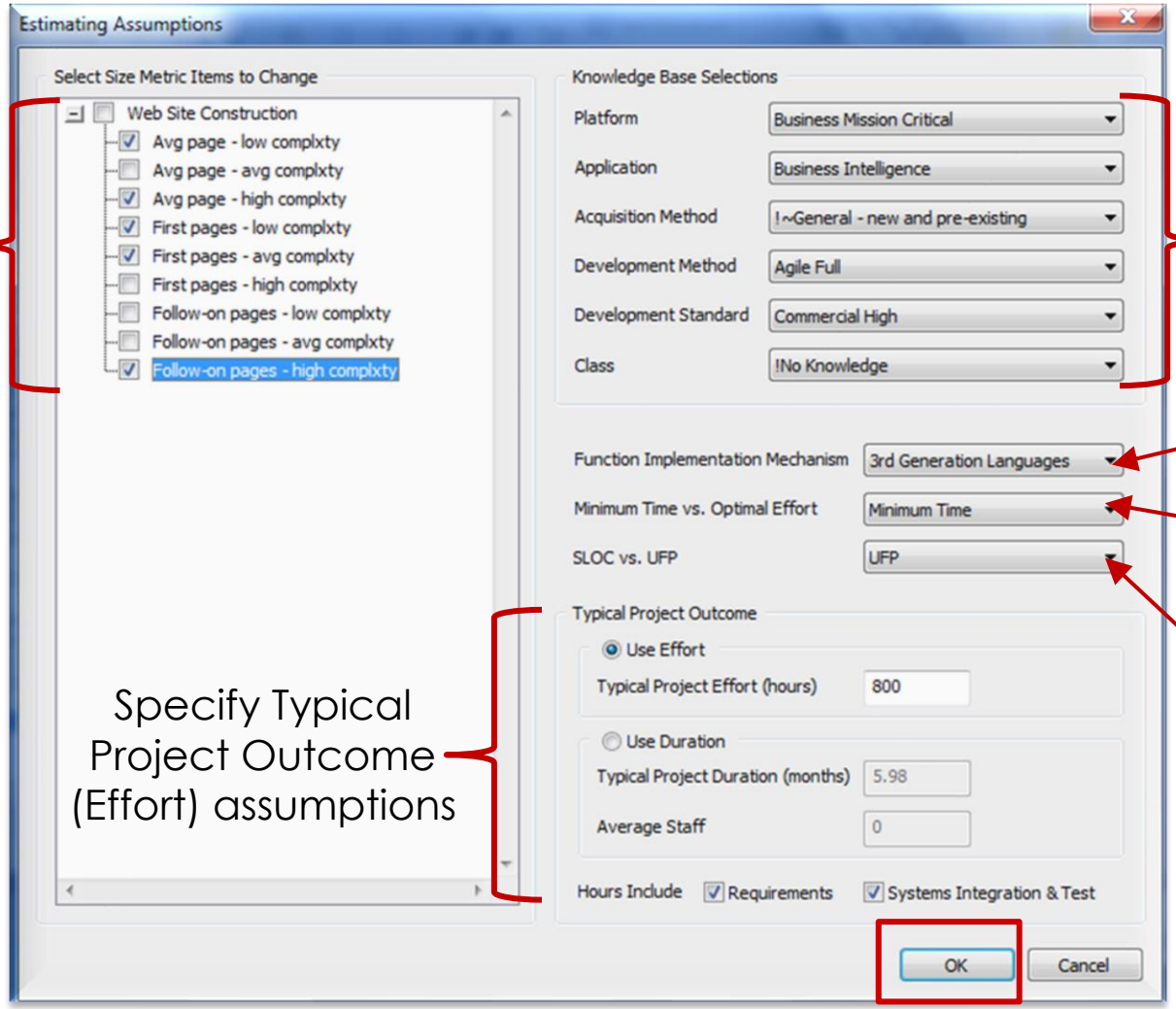


SEER will automatically calculate the FPs or SLOC required to produce the specified hours

GALORATH

Assumptions for Effort-Based Size Metrics

Select the Size Metric items for which the assumptions will apply



Specify Typical Project Outcome (Effort) assumptions

Select the Knowledge Bases for the calculation

Select the Function Implementation Mechanism

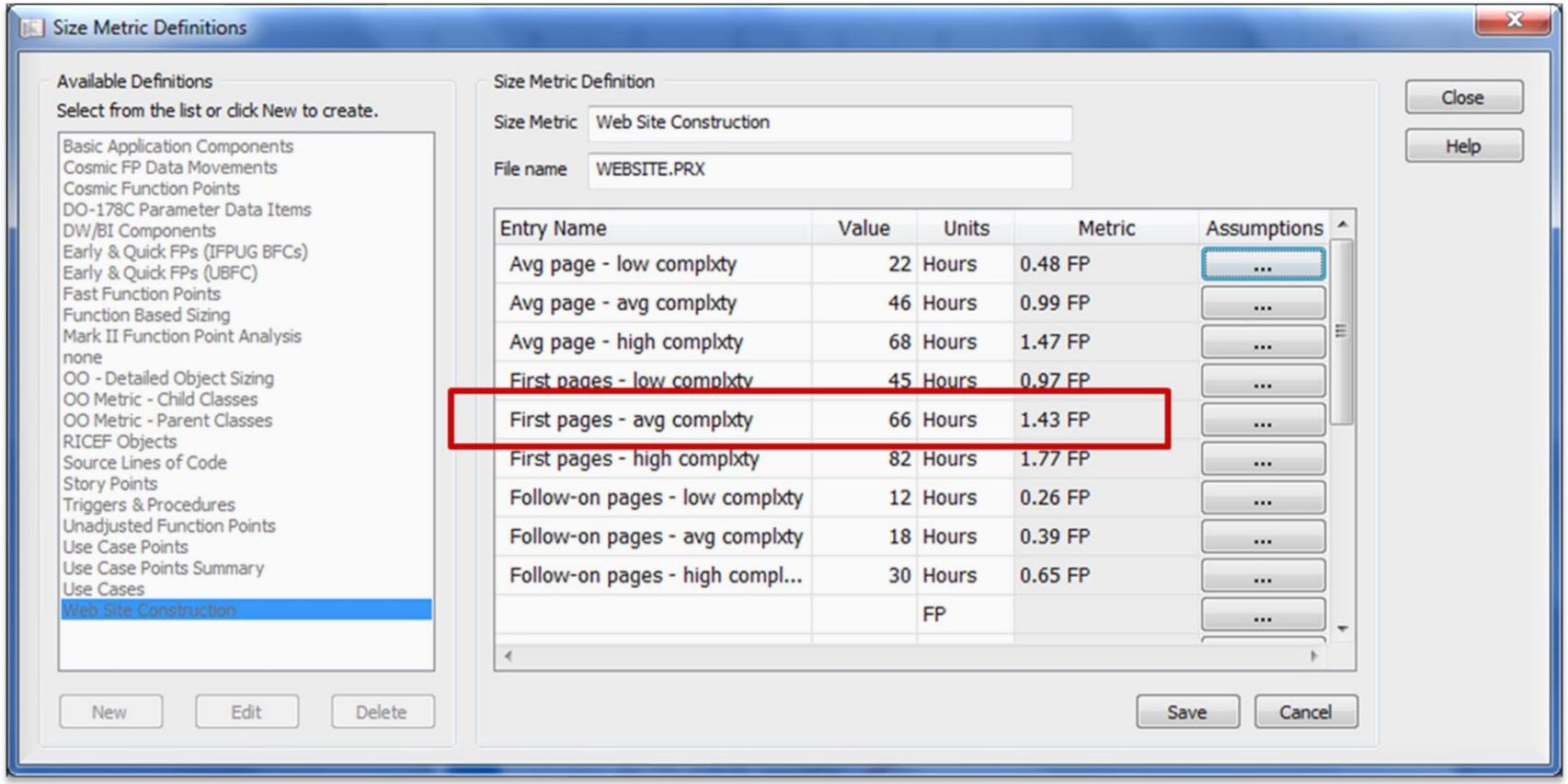
Minimum time vs Optimal effort switch

SLOC or Function Points

Press OK to calculate

Editing Effort-Based Size Metrics

- One "First page – Avg Complexity" item takes 66 hours to build
- Based on the assumptions specified, one "First page – Avg Complexity" is equivalent to 1.43 Function Points

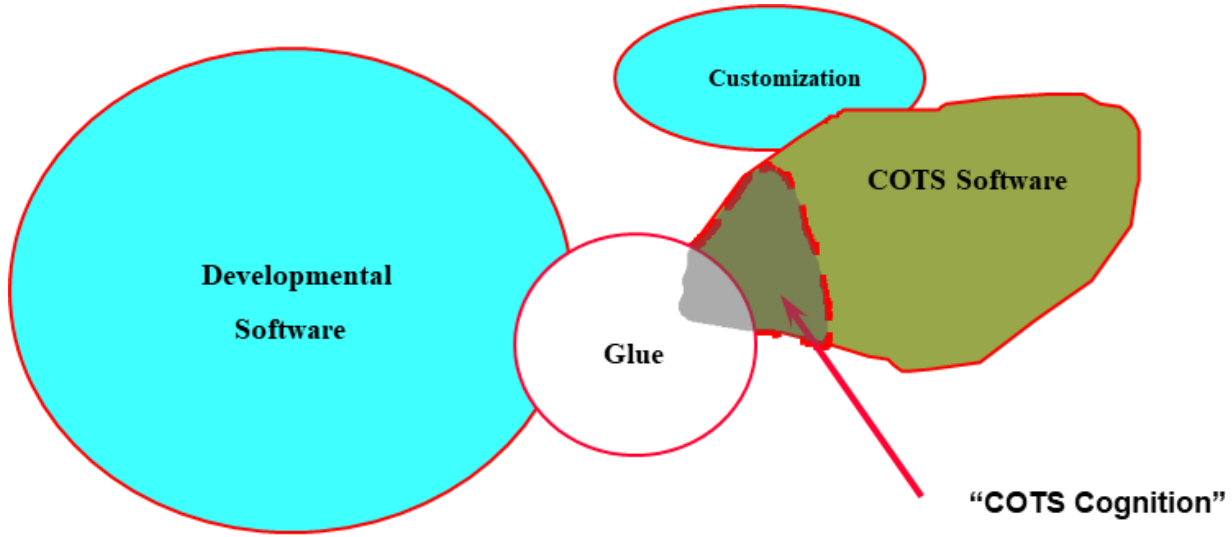


GALORATH

COTS INTEGRATION SIZING

Key Components Of A Software Project That Uses COTS

- Developmental Software:
  - Functionality developed specifically for the project at hand
  - May include customization of COTS
- "Glue" Code:
  - Code written to bind COTS to developmental software
  - Development effort must be captured

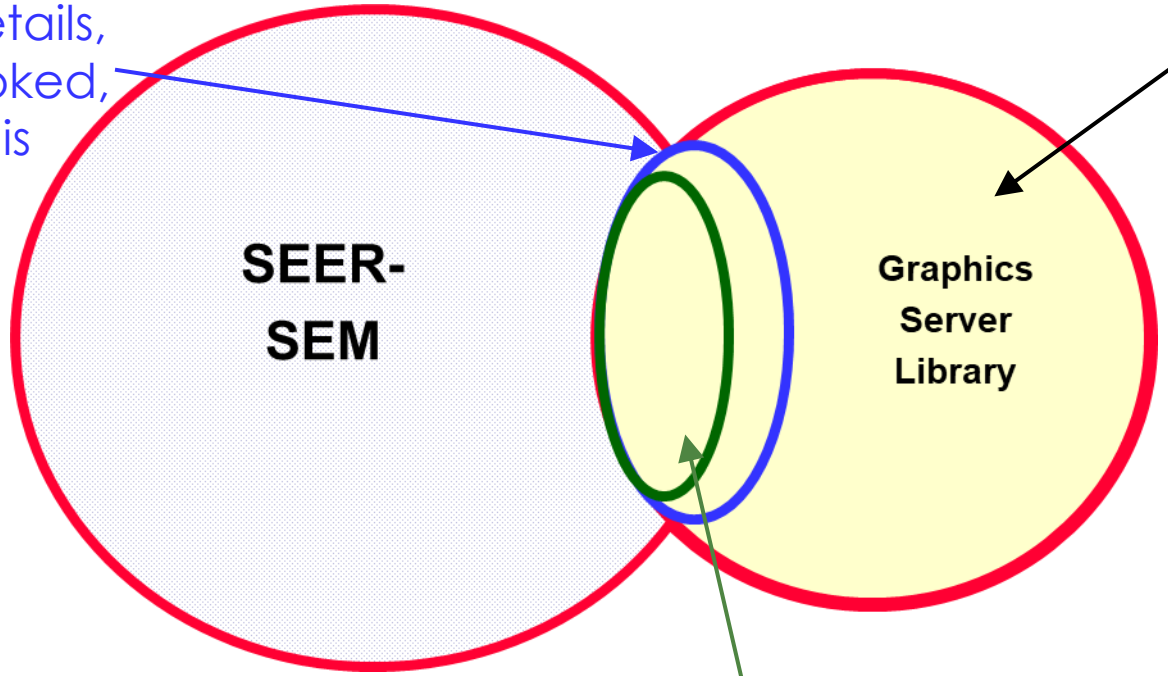


- COTS Software:
  - Purchased functionality
  - Direct Cost component of COTS integration
- COTS Cognition:
  - Required functionality within the COTS software that must be understood
  - Effort component of COTS integration

COTS Cognition: Graphics Server

UNINVOLVED
Not necessary to know
(does not contribute to size)

COGNITION
Architectural and implementation details, not necessarily invoked, but knowledge is needed

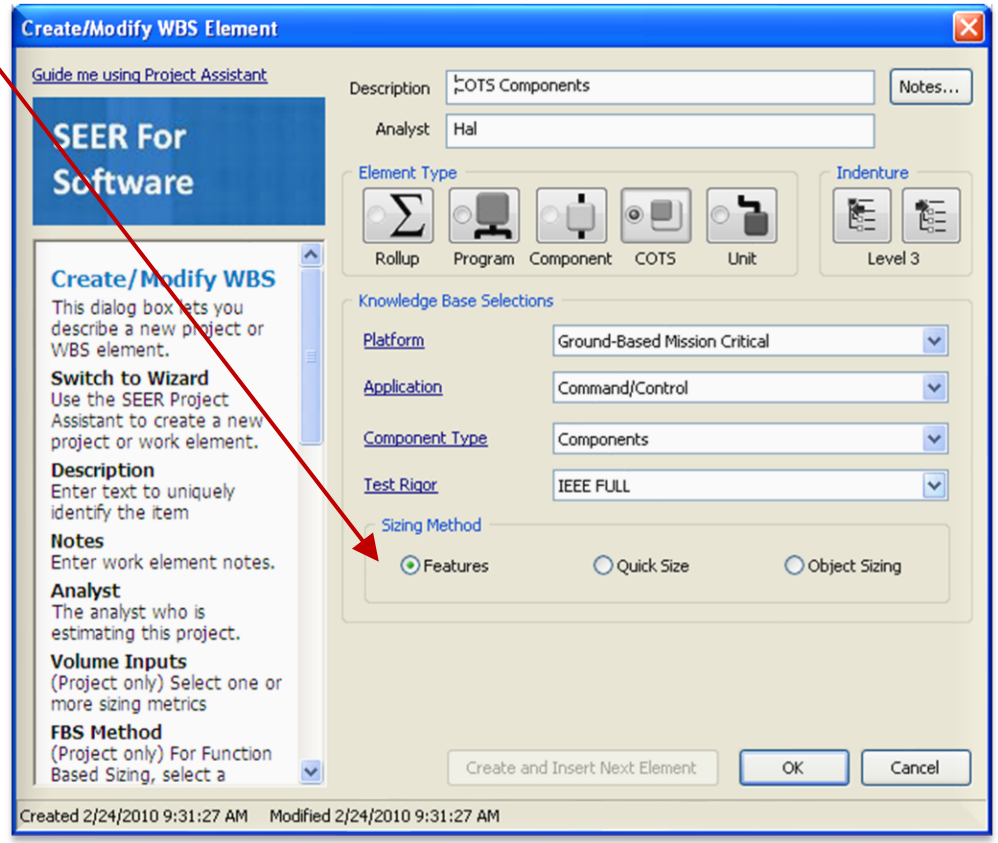


INSTANTIATED
Graphics Server calls used by SEER-SEM

GALORATH

Scope Estimates for COTS Cognition

- COTS Cognition does not have "size" in the same sense that developmental software does
- SEER-SEM offers three methods of scoping integration effort



GALORATH

Scoping COTS Cognition Using Quick Size

- Application Type
  - Commercial application or application type being integrated
  - If specific application is not listed, choose one that is similar in size/functionality to the application being integrated
- Percentage Required
  - Portion of COTS component functionality that the integrating developers are required to learn
  - Reflects effort to acquire a working knowledge of this functional portion

Scoping COTS Cognition Using Features and Objects

- Sizing with Features is closely related to the traditional function point method
- Measures have been translated so that they are closely related to features found in COTS components
  - Unique Functions (Inputs, Outputs, Queries, etc.)
  - Data Tables Referenced
  - Data Tables Configured
- Use Objects if COTS components are implemented as objects
- Object sizing metric is completely compatible with IFPUG-standard function points and object counting guidelines
  - Input Services
  - Output Services
  - Inquiry Services
  - External Classes
  - Internal Classes

SEER-SEM SIZE PARAMETERS

SEER-SEM Size Characterizations

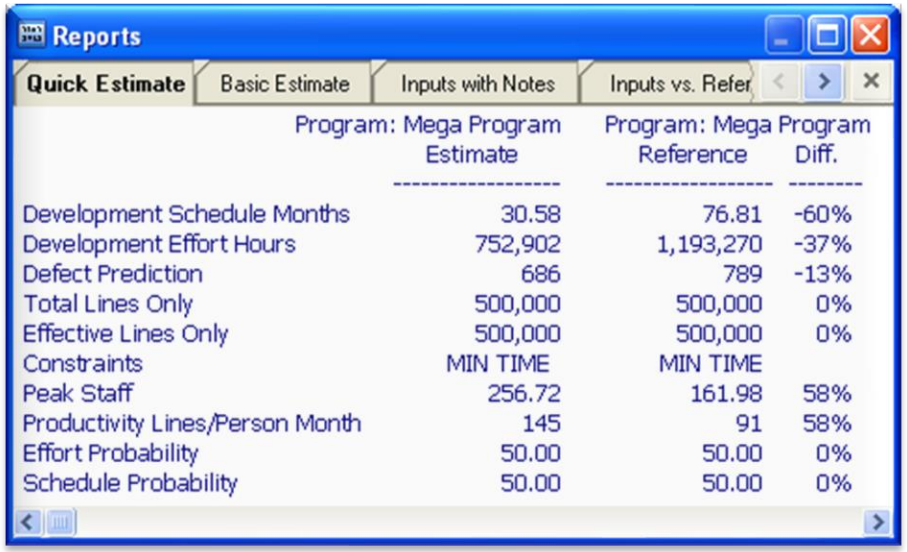
- New (Lines, Functions, Size Metrics)
  - Developed completely within the current project
- Pre-existing
  - Inherited from prior projects or prior builds in same project
  - May or may not be designed for reuse (more on this later)
- Deleted
  - Functionality/size to be deleted from pre-existing software
- Redesign, Reimplementation, and Retest
  - Effort required to modify pre-existing software
  - Expressed as a percentage of effort relative to new code
- Size Growth Factor
  - Enable in Project Parameter selection
  - Apply historical growth experience

SEER Parameters Intended for Function Point Inputs

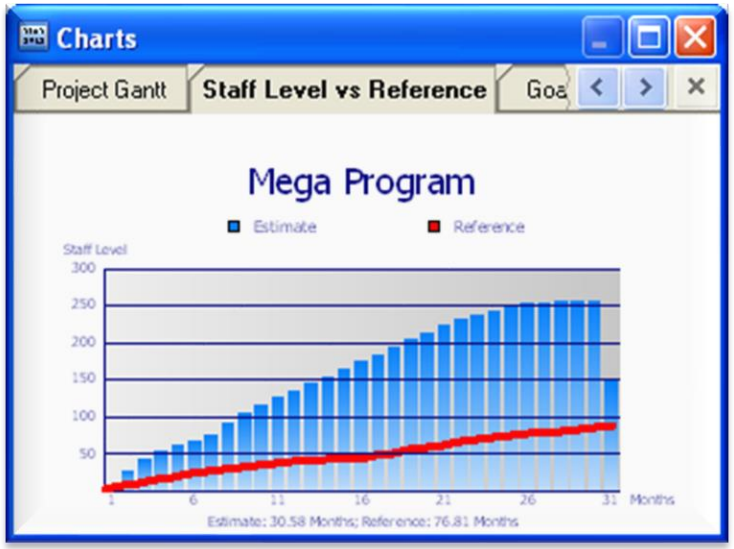
- Function Implementation Mechanism
  - For Line-based size measures, will suggest a conversion factor from one language to another
    - Number of lines will change depending on selected language
  - For Function-based size measures, language, code generator, screen generator impacts effort required
  - Unlike SLOC, the number of function points will not change based on language, but…
  - Effort per function point is highly dependent on language used to implement
- Software Phase at Estimate
  - Effort required to achieve given functionality is determined by the phase in the software development life cycle when the function point count is conducted
  - Early estimates allow for more functionality growth
    - Function point methodology quantifies "scope creep" -- additional functionality not specified in original requirements, but becomes identified as scope is being clarified and functions are defined

Programs Included in Size Example

- Reference : One program, 500 KSLOC (mischaracterized as a single program element)
  - Long schedule
  - Large team, low productivity, high cost
  - Estimate: Ten programs included in size, totaling 500 KSLOC

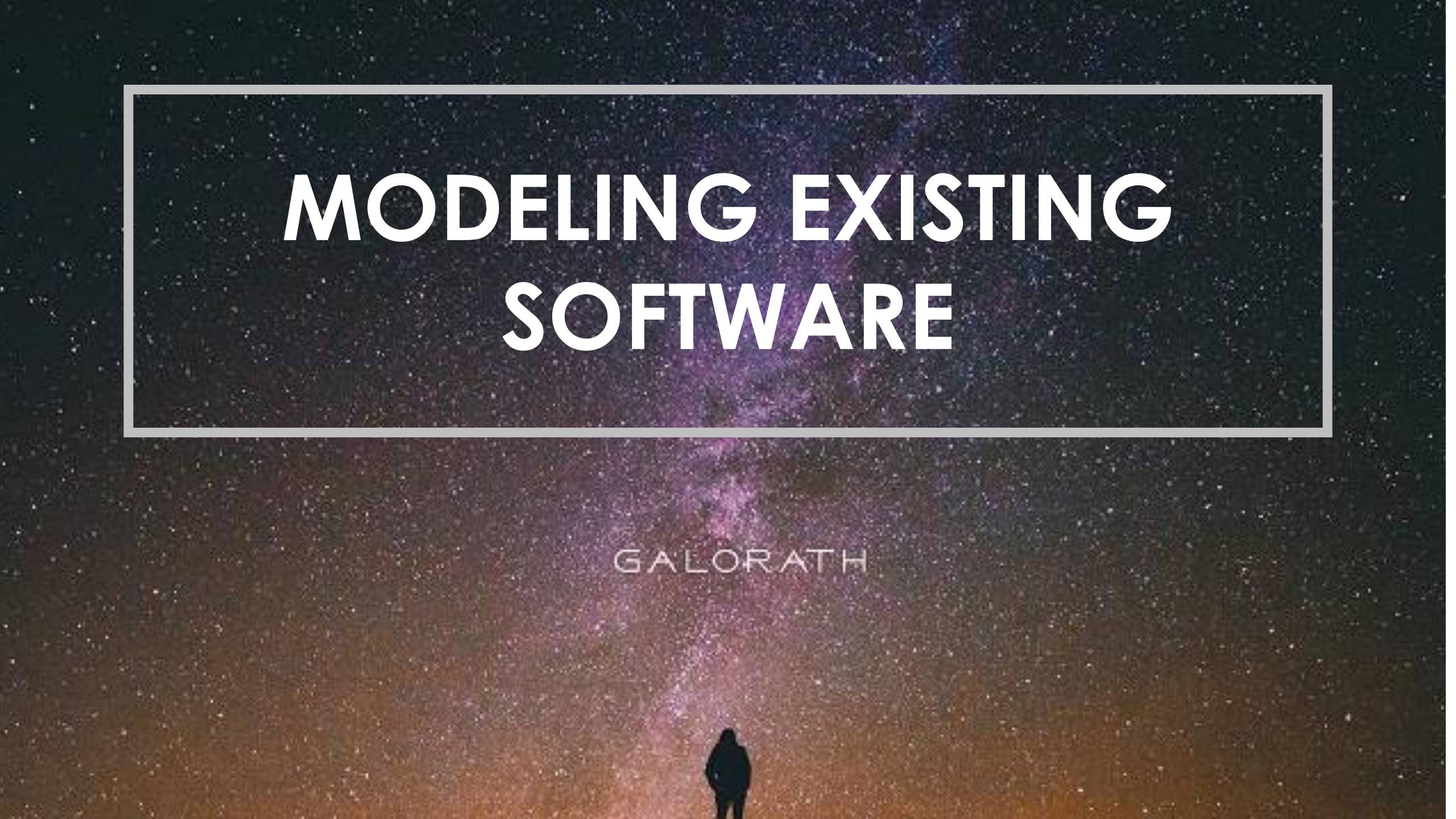


	Program: Mega Program Estimate	Program: Mega Program Reference	Diff.
Development Schedule Months	30.58	76.81	-60%
Development Effort Hours	752,902	1,193,270	-37%
Defect Prediction	686	789	-13%
Total Lines Only	500,000	500,000	0%
Effective Lines Only	500,000	500,000	0%
Constraints	MIN TIME	MIN TIME	
Peak Staff	256.72	161.98	58%
Productivity Lines/Person Month	145	91	58%
Effort Probability	50.00	50.00	0%
Schedule Probability	50.00	50.00	0%



- When more detail is available, all ten programs should be represented as separate program work elements, each with its own size estimate

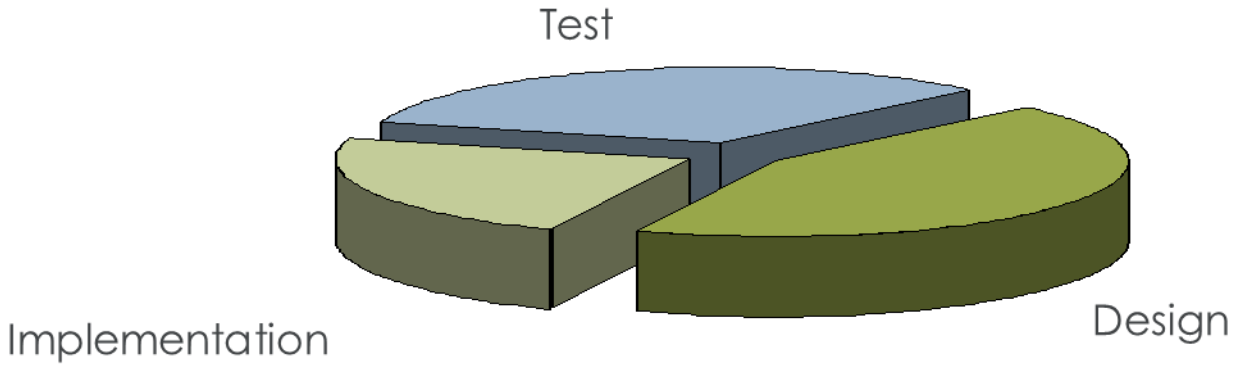
MODELING EXISTING SOFTWARE



Software Designed For Reuse…or Not

- Designed for reuse
  - Reusability requirements specified during original development
  - Significant extra coding and documentation effort during original development to insure reusability
  - Minor to moderate effort (mostly retest) required when reused
- Not designed for reuse
  - Developed without specific extra effort to ensure reusability
  - Sources include any legacy code not specifically designed for reuse
    - Other projects & programs
    - Prior builds of the current program
  - Moderate to major effort required
    - Redesign
    - Reimplementation
    - Retest

Effort on Existing Code



- In general, software development can be divided into three basic activities:
  - Design
  - Implementation (coding)
  - Test
- In adapting or integrating existing software, some or all (or possibly *more* than all) of these activities must be revisited or repeated
- Percentages of effort (redesign, reimplementation, retest) are applied to the *total* size of an existing program to derive the *effective* size which will be used to estimate effort, schedule, etc.
- Input percentages of detailed rework activities are user adjustable
- Underlying weights are fixed
  - Redesign = 40% of total effort
  - Reimplementation = 25% of total effort
  - Retest = 35% of total effort

Rework Percentage Worksheet

SEER-SEM Rework Percentage Calculation

Compute redesign, reimplementation and retest percentages based on detailed rework factors.

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

Step 4: Copy Data Into SEER-SEM

A) Depending on the size category, select the cells below in the red border and copy to the clipboard

B) Return to SEER-SEM

C) When in SEER-SEM, Select Tools / Run Commands From Clipboard or Paste in the Parameters view

- Located in Tools folder where SEER is installed
- Detailed activity breakouts for each area
- Use this to tailor rework estimates to override K-Base defaults
- Allows you to dial-in estimated rework on existing software

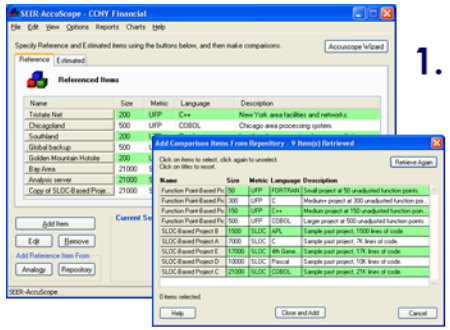
GALORATH

STUCTURED CASE STUDY #2

Sizing Study

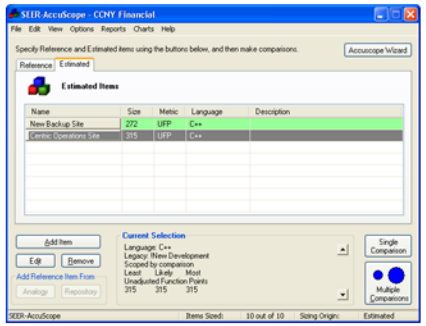
SIZING BY COMPARISON

Four Step Sizing Process

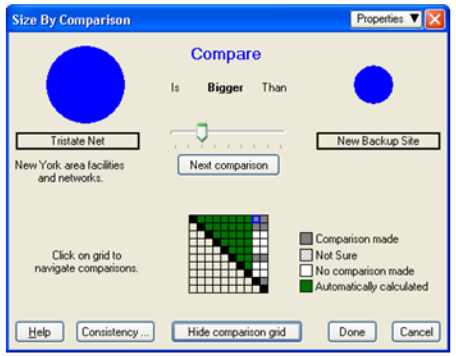


1. Choose reference items for comparison

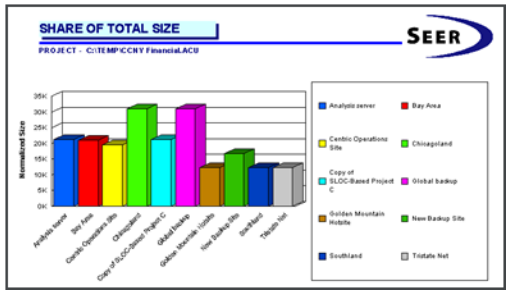
2. Identify Items to size



3. Perform various expert judgment comparisons

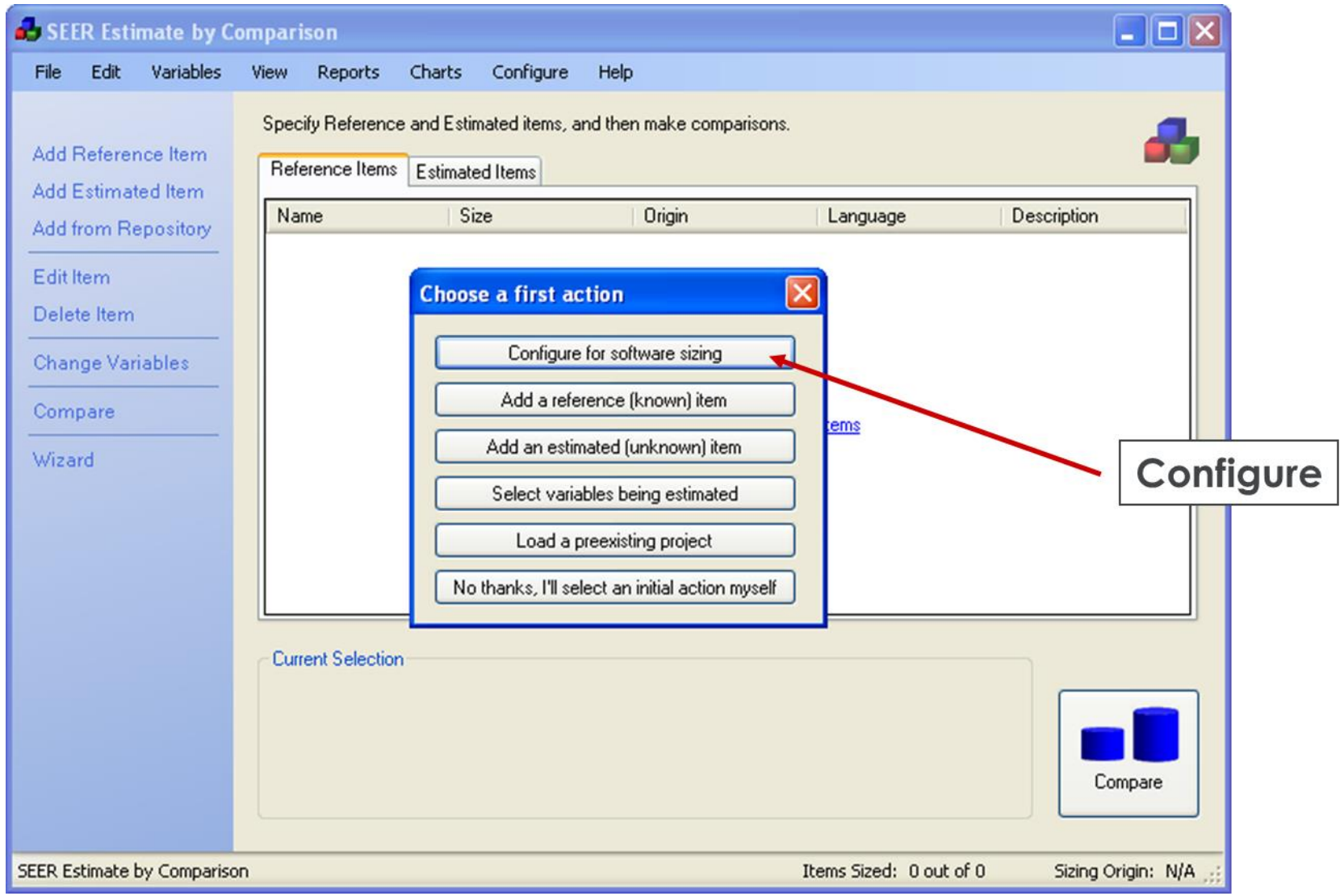


4. View / Use results

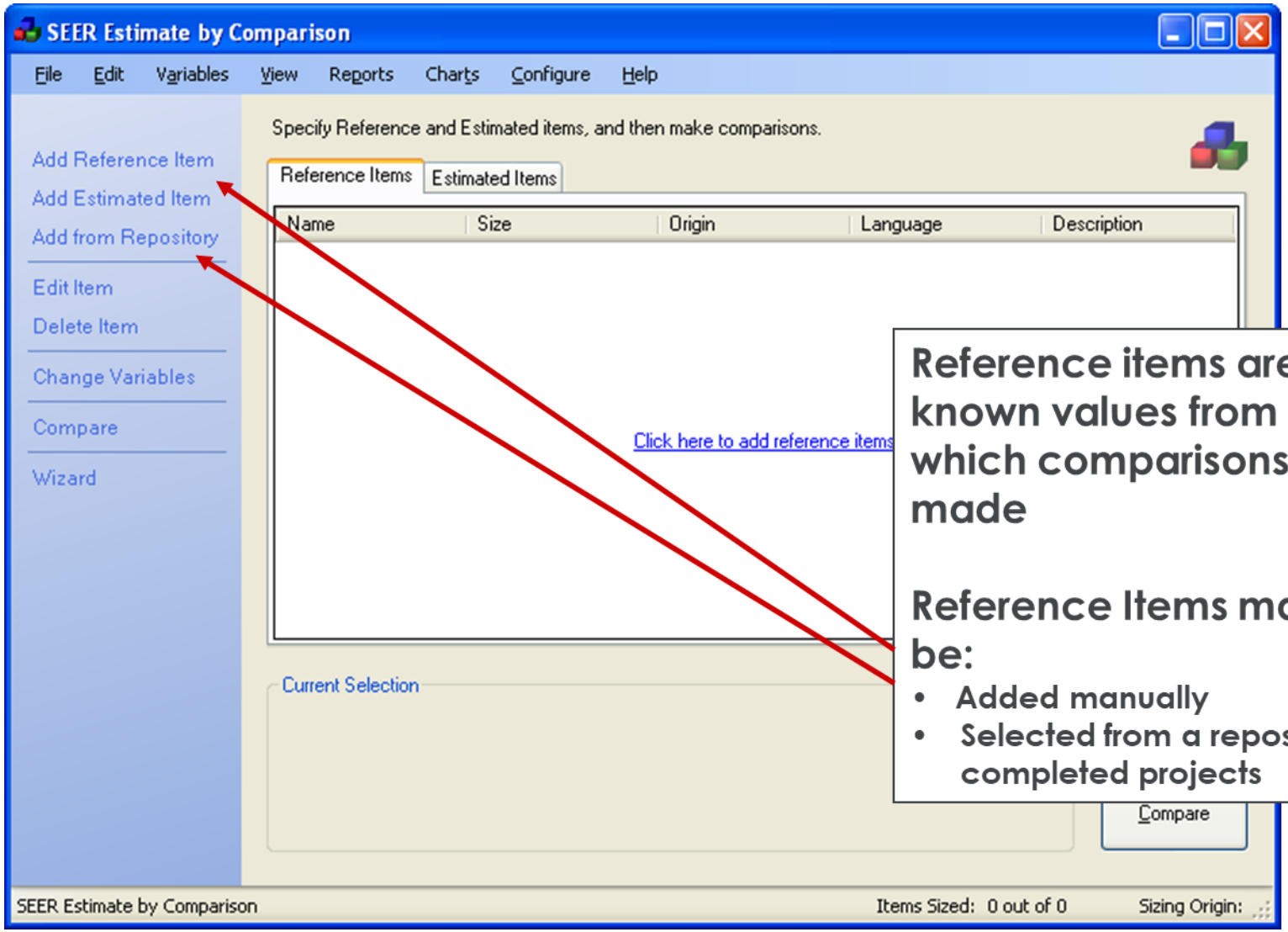


GALORATH

What do you want to do?



Step 1: Choose Reference Items



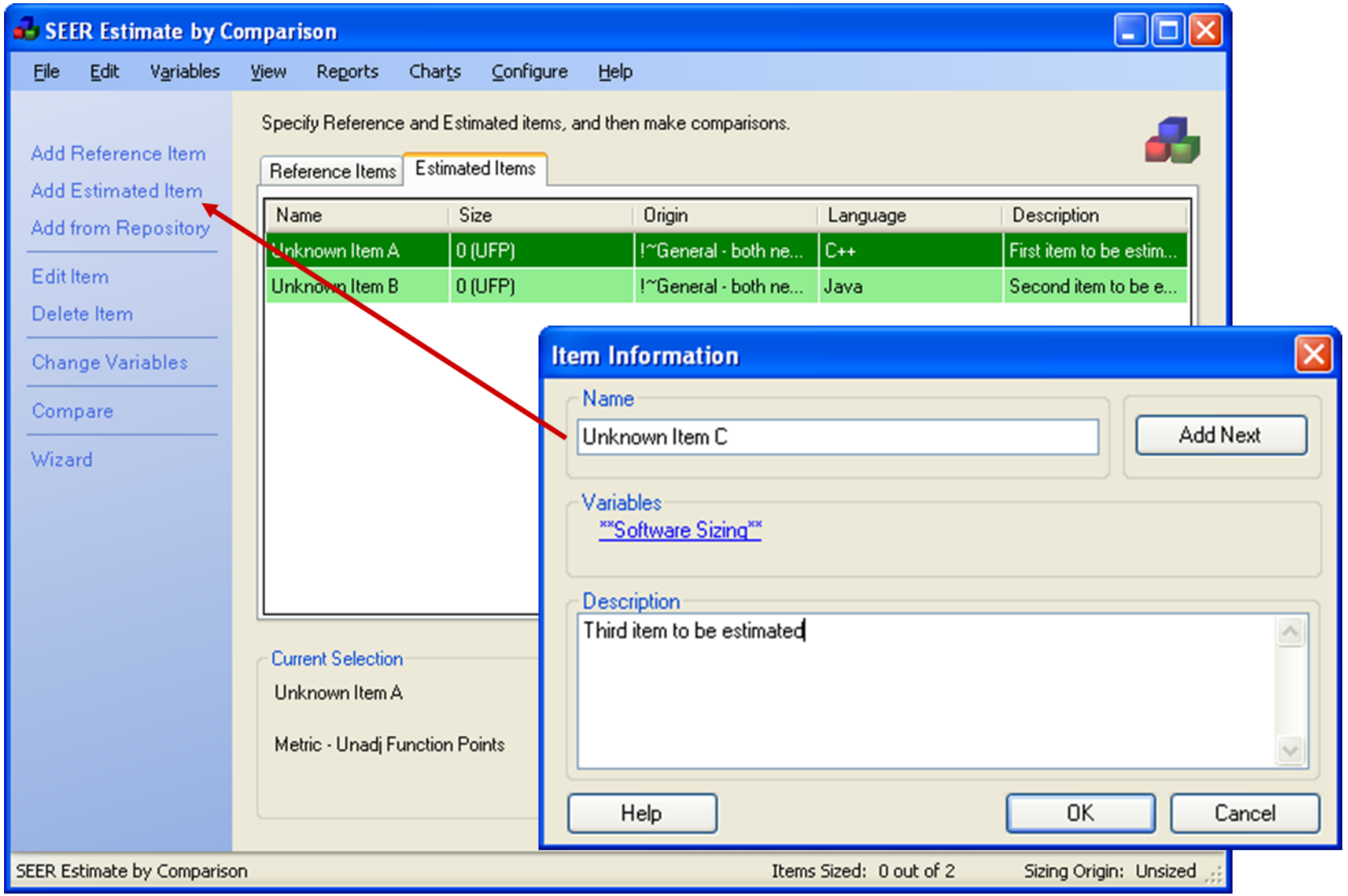
Reference items are known values from which comparisons are made

Reference Items may be:

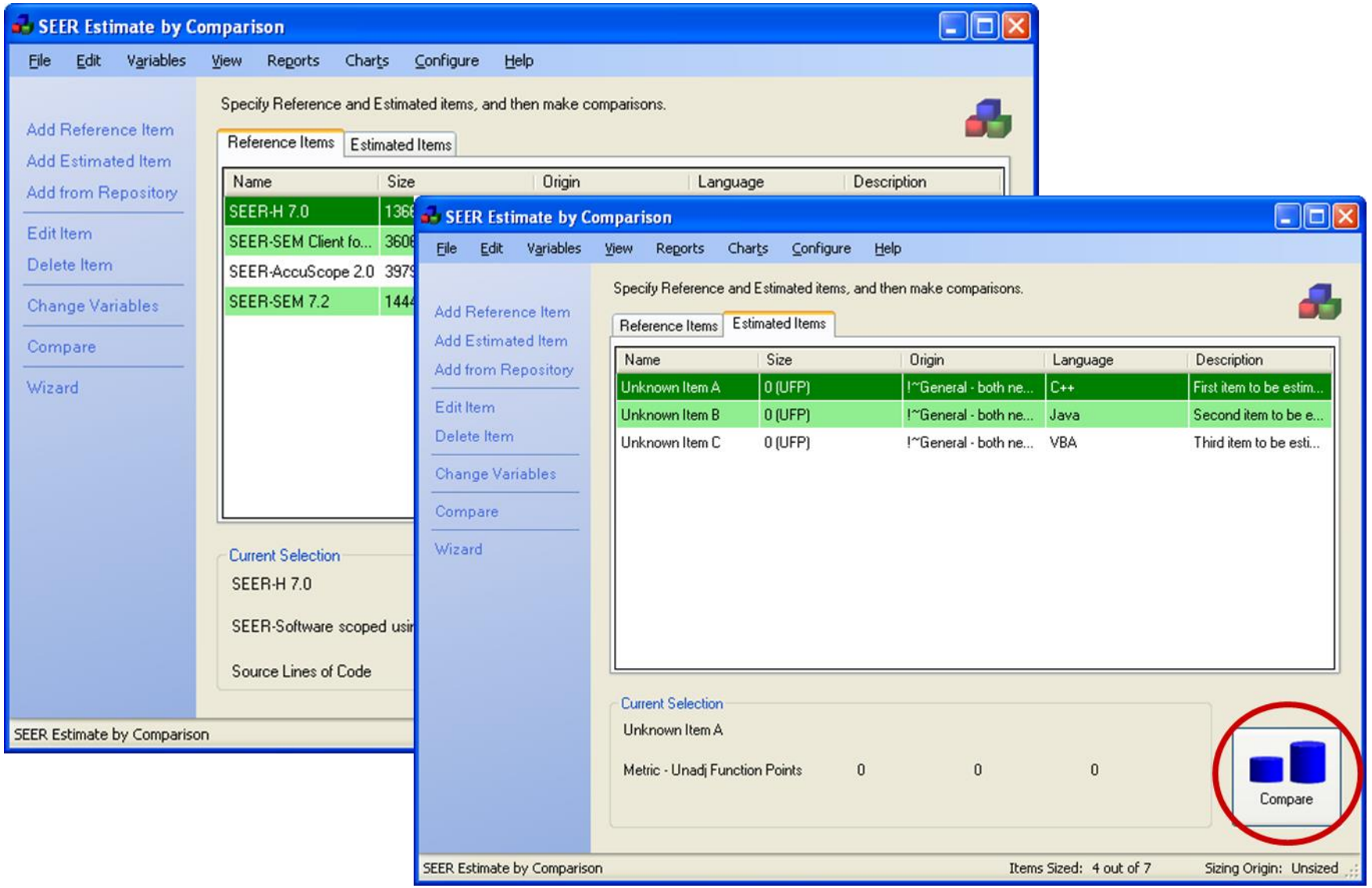
- **Added manually**
- **Selected from a repository of completed projects**

GALORATH

Step 2: Identify Items To Estimate



Ready For Comparisons

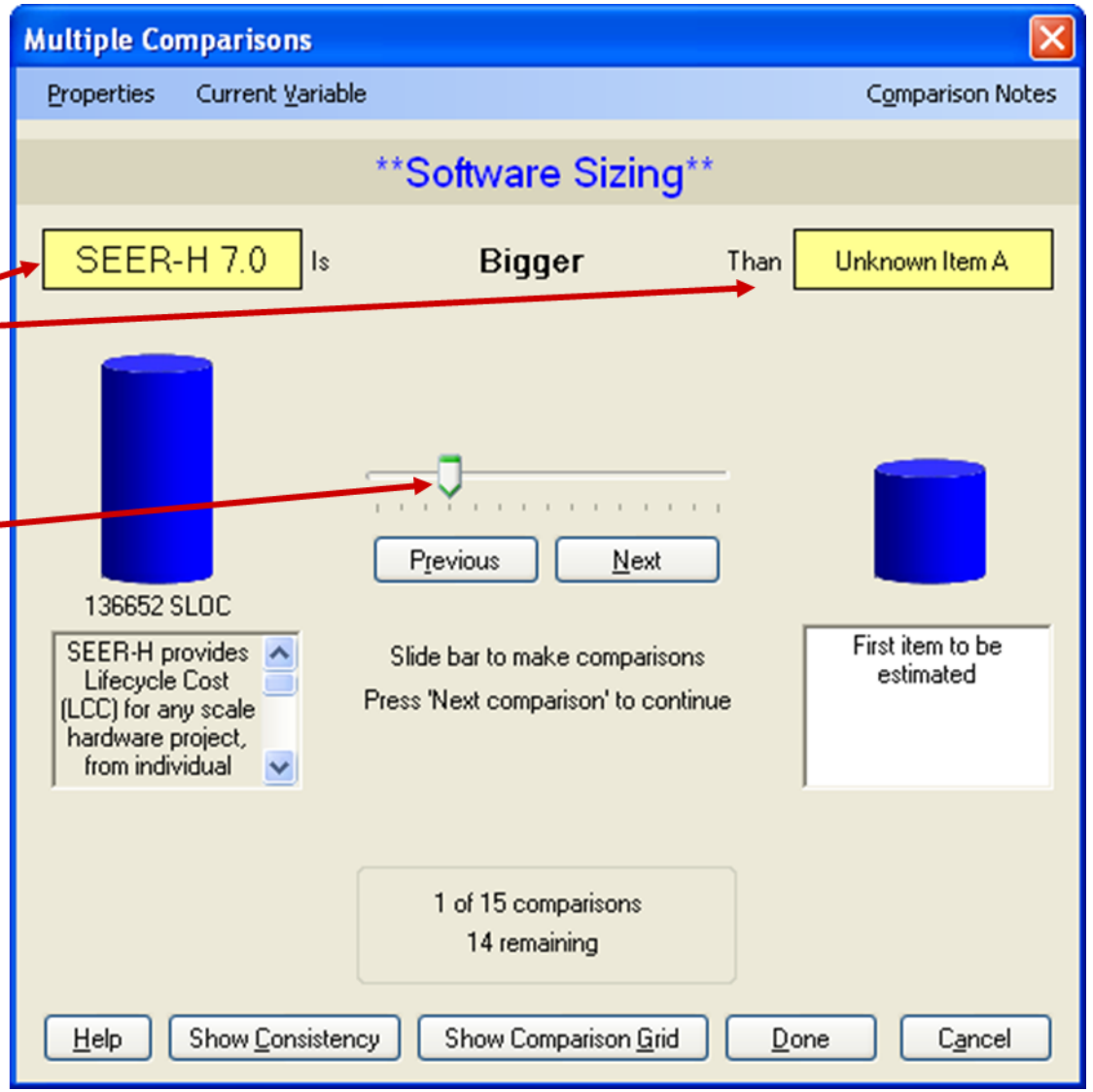


GALORATH

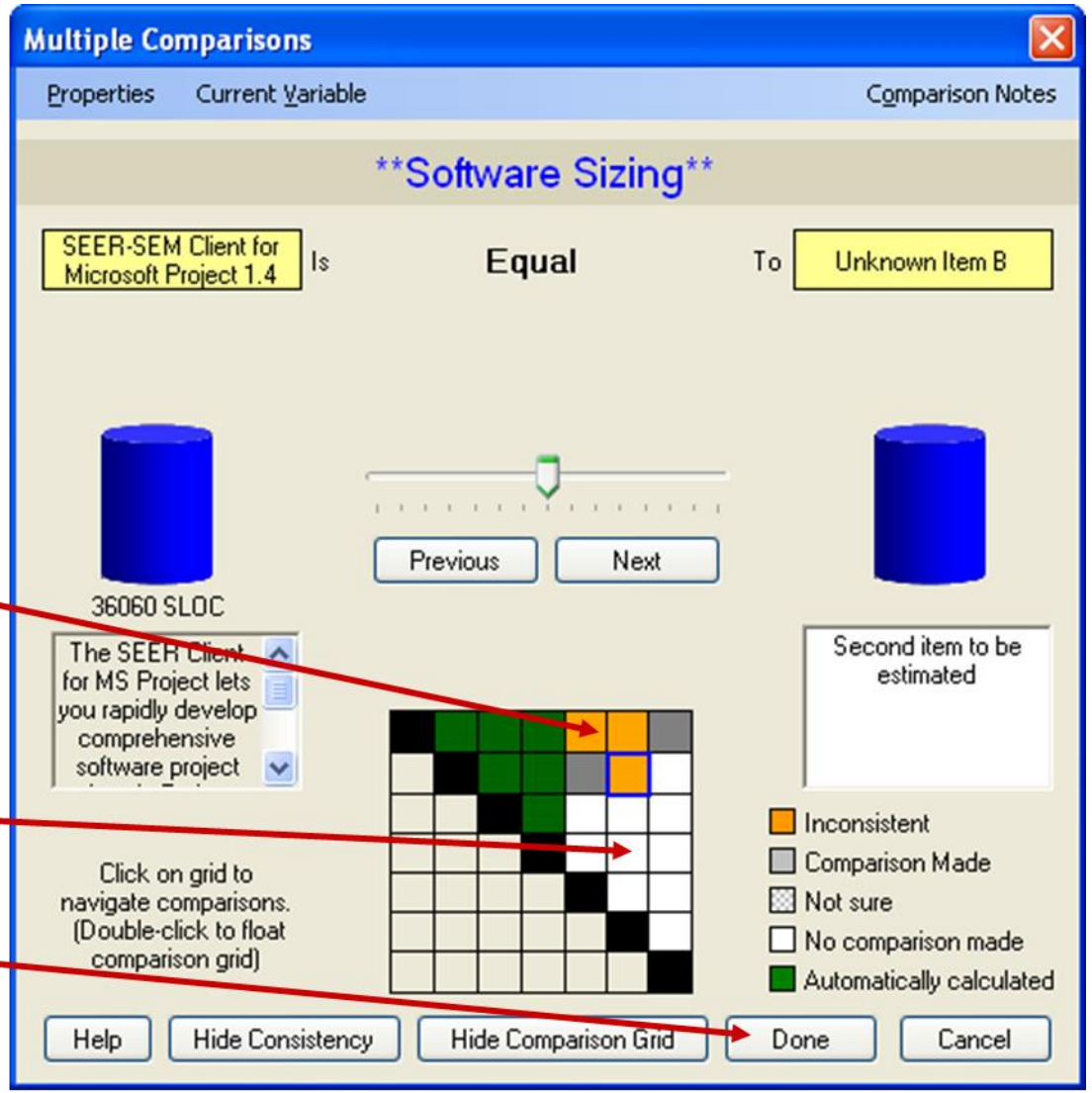
Step 3a: Compare Items

Your own judgment establishes the relative size between reference and unknown items

Use slider to exercise judgment



Step 3b: Check for Consistency & Completeness



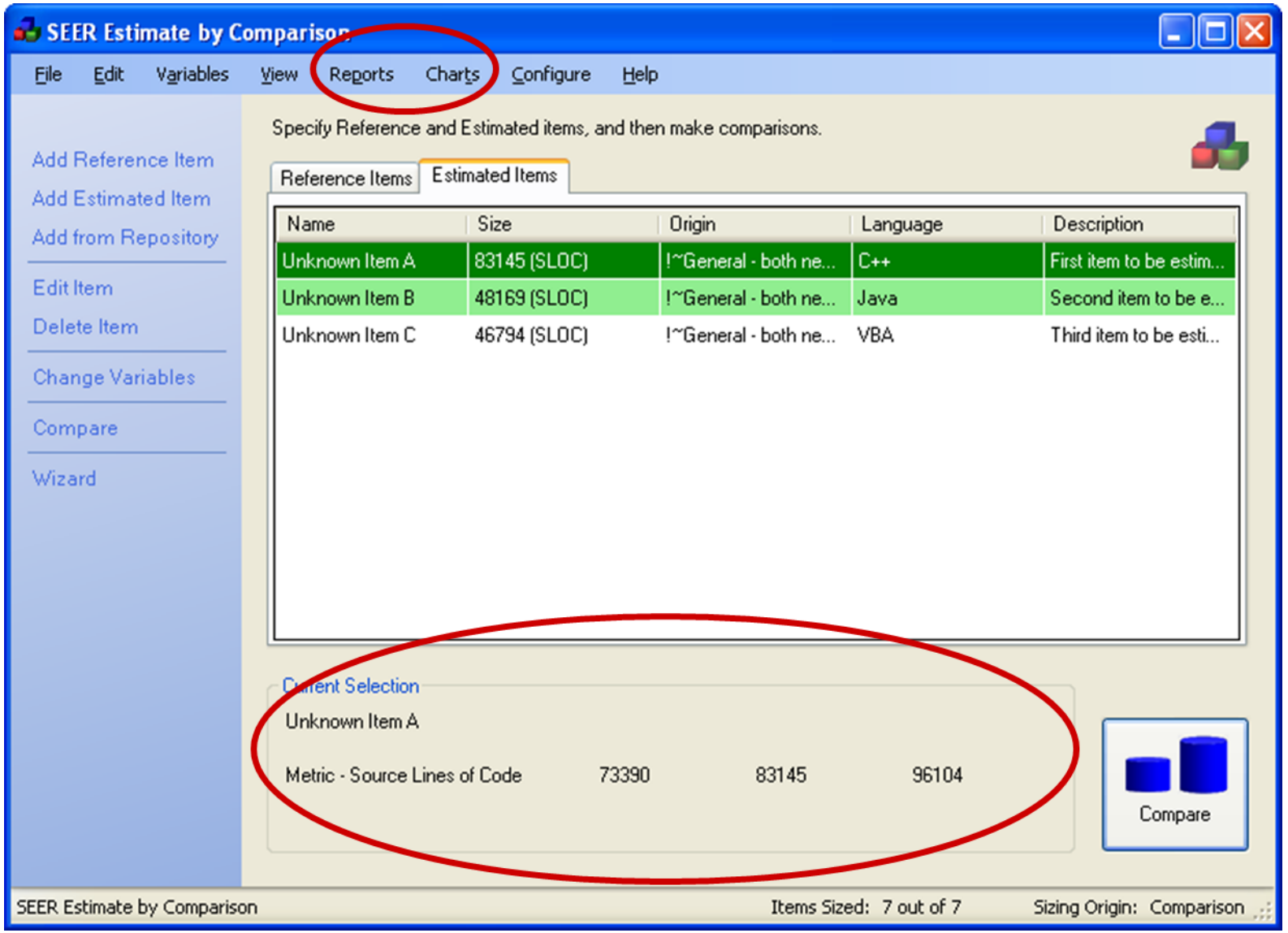
Inconsistent comparisons

Incomplete work

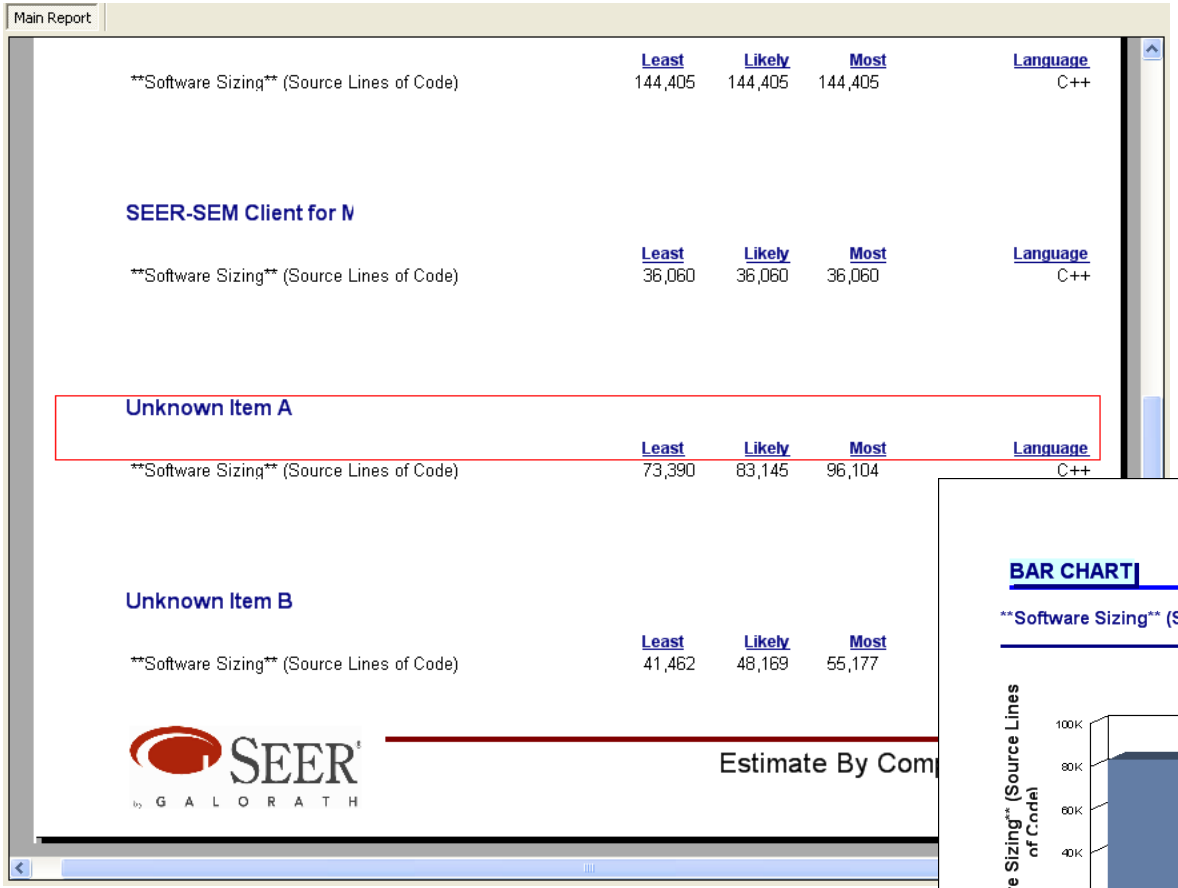
Click Done when all comparisons are complete

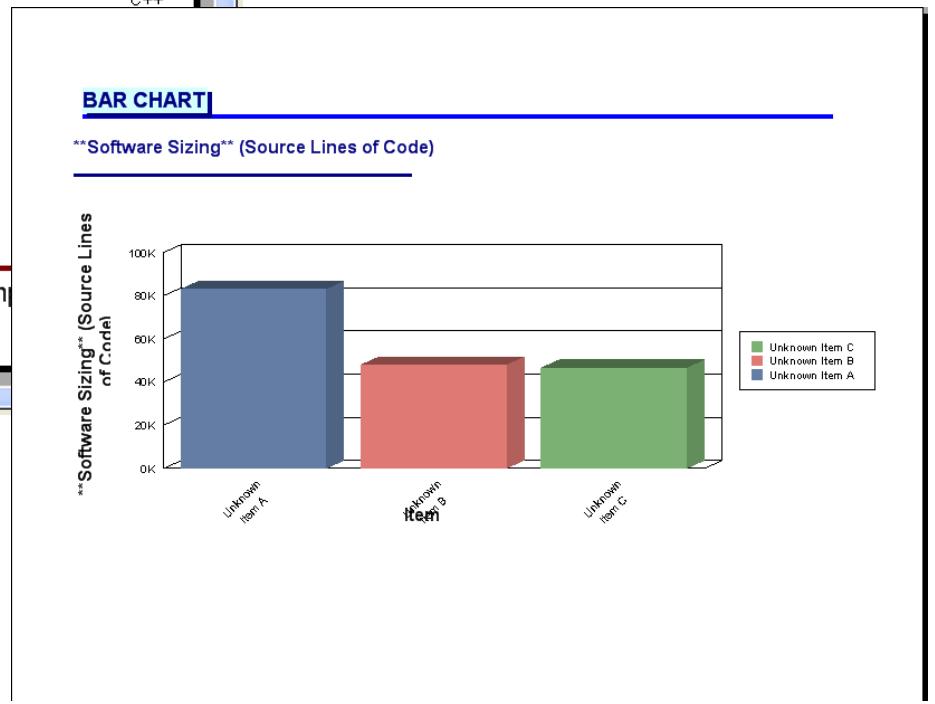
GALORATH

Step 4: View Results



Reports & Charts

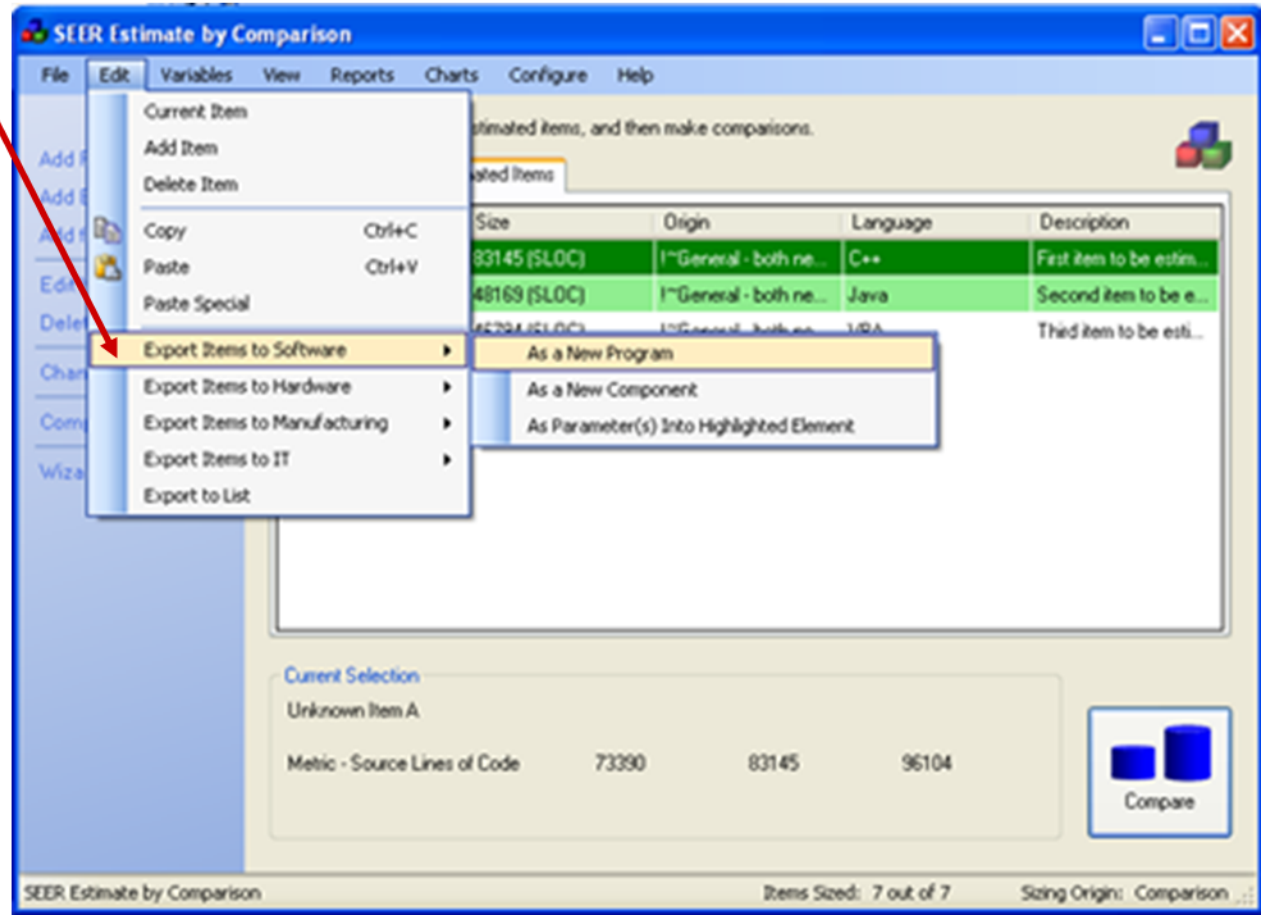




GALORATH

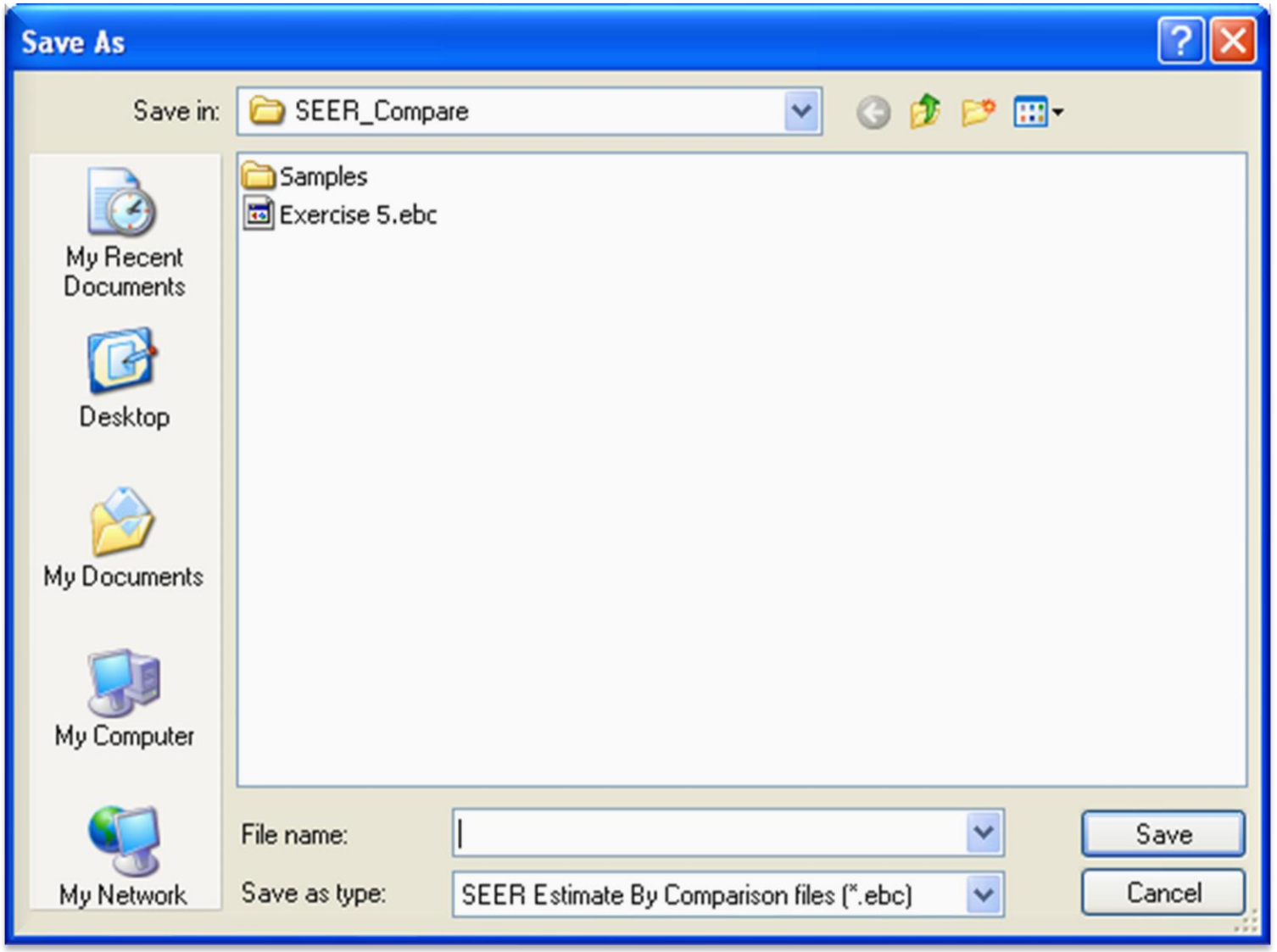
Exporting Items

Reference and Estimated items can be copied from SEER Sizing and pasted into SEER-SEM as Programs, Components, or size parameter entries



GALORATH

Save for Future Reference



GALORATH

GUIDED EXERCISE #6

SEER Sizing By Comparison