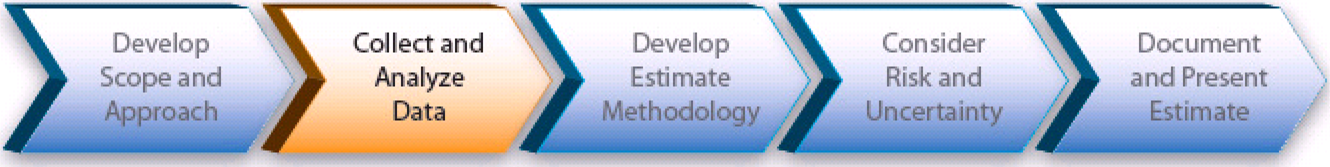
Introduction and Objectives

Welcome to the Collect and Analyze Data lesson. After completing this lesson, you will understand this second of the five major steps of developing a software cost estimate.

Lesson Objectives

After completing this lesson you will be able to:

- Compare various sources of software data.
- Recommend appropriate data normalization steps to support software cost estimates.
- Critique productivity as a commonly-cited software metric.
- Relate the primary software cost drivers to each other and to development cost.



Develop Scope and Approach

Collect and Analyze Data

Develop Estimate Methodology

Consider Risk and Uncertainty

Document and Present Estimate

D

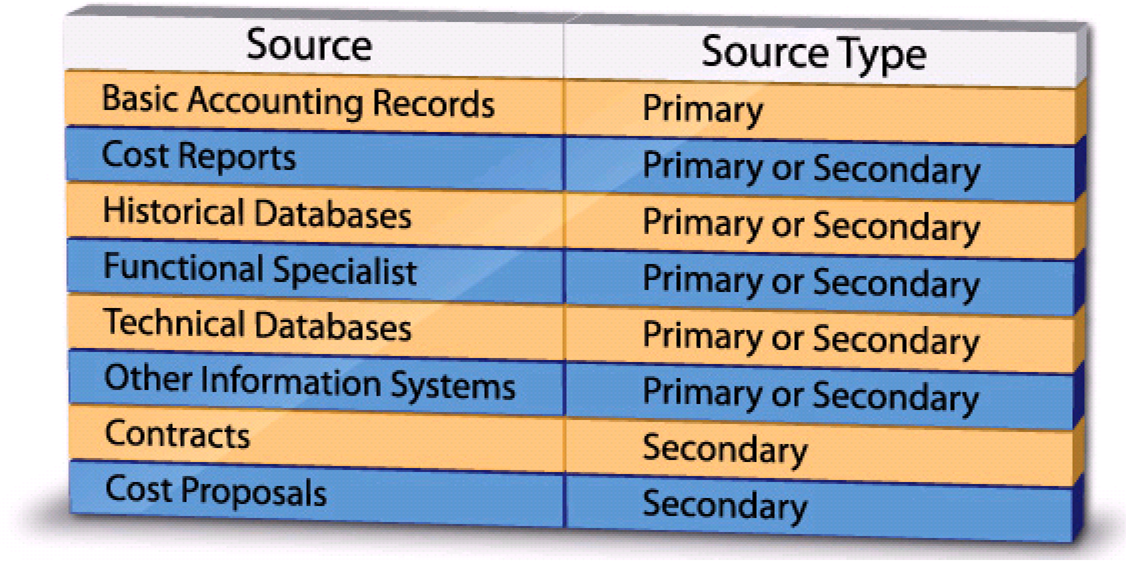
Long Description

Graphic illustrates the steps of the Cost Estimating process. The steps from left to right are:
Develop Scope and Approach, Collect and Analyze Data (highlighted), Develop Estimate Methodology, Consider Risk and Uncertainty, and Document and Present Estimate.

Software Data

The basic estimating methodologies (analogy, parametric, engineering, and extrapolation from actuals) are all data-driven. Credible and timely data inputs are required when using any of these methodologies. If data required for a specific approach are not available, then that estimating methodology cannot be used. Because of this, the estimator must identify the best sources for the method to be used.

The table below shows eight basic sources of data and whether they are considered a primary or secondary source of information. When preparing a cost estimate, estimators should consider all credible data sources; however, whenever feasible, primary sources of data should be given the most consideration.



Source	Source Type
Basic Accounting Records	Primary
Cost Reports	Primary or Secondary
Historical Databases	Primary or Secondary
Functional Specialist	Primary or Secondary
Technical Databases	Primary or Secondary
Other Information Systems	Primary or Secondary
Contracts	Secondary
Cost Proposals	Secondary

D

Popup Text

Analogy

An estimate of costs based on historical data of a similar (analogous) item.

Parametric

A cost estimating methodology using statistical relationships between historical costs and other program variables such as system physical or performance characteristics, contractor output measures, or manpower loading.

Engineering

Derived by summing detailed cost estimates of the individual work packages and adding appropriate burdens. Usually determined by a contractor’s industrial engineers, price analysts, and cost accountants.

Popup Text

Extrapolation from actuals: Extrapolation method requires prototype or preproduction actual cost data on the system considered. Primarily used in estimating the production cost of system hardware, and assumes a relationship (technical, performance) between cost of prototypes and production units.

Primary sources

Primary data are obtained from the original source, and are considered the best in quality and the most reliable.

Secondary sources

Secondary data are derived (possibly “sanitized”) from primary data, and are not obtained directly from the source. Because of this, they may be of lower quality and usefulness.

Long Description

Table with two columns and nine rows. The data are as follows:

Source	Source Type
Basic Accounting Records	Primary
Cost Reports	Primary or Secondary
Historical Databases	Primary or Secondary
Functional Specialist	Primary or Secondary
Technical Databases	Primary or Secondary
Other Information Systems	Primary or Secondary
Contracts	Secondary
Cost Proposals	Secondary

Software Data, Cont.

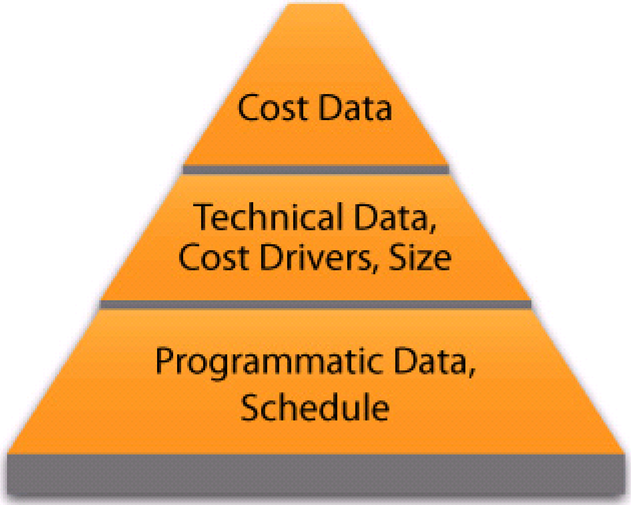
Like risk, software may have broader range of potential data, since most systems contain software.

While there are certain platform considerations that drive differences in software complexity and hence development productivity, software data should not be considered commodity-specific a priori.

In general, the preference is to combine data and adjust for differences appropriately rather than discard data altogether and end up with a data set that is unnecessarily narrow.

While focus is on cost data, do not forget to collect technical data, which will help identify cost drivers and support size estimates, and programmatic data, which will support schedule estimates.

Where possible, collect both initial estimates and final values, in order to capture historical growth.



Cost Data

Technical Data, Cost Drivers, Size

Programmatic Data, Schedule

Questions Managers Should Ask

D

Popup Text

Questions Managers Should Ask

- Does the estimating organization have a method for organizing and retaining information on completed projects (a historical database)?
- Does the database contains a useful set of completed projects?
- Has unpaid overtime, if used, been quantified, so that recorded data provide a valid basis for estimating future effort?
- Are schedule milestones (start and finish dates) described in terms of criteria for initiation or completion, so that work accomplished between milestones is clearly bounded?
- Have any cost models that were used for estimating been used also to provide consistent frameworks for recording historical data (which helps ensure that comparable terms and parameters are used across all projects, and that recorded data are suitable for use in the estimating models)?
- Does the organization have a structured process for capturing effort and cost data from ongoing projects?
- Does the development organization hold postmortems at the completion of projects to ensure that recorded data are valid, and to ensure that events that affected costs or schedules get recorded and described while they are still fresh in people’s minds?
- Does information on completed projects include the life-cycle model used, together with the portion covered by the recorded cost and schedule, and a workflow schematic that illustrates the software process used?
- Does information on completed projects include actual (measured) size, cost, and schedule, and the actual staffing profile?
- Does information on completed projects include an estimate at completion, together with the values for cost model parameters that map the estimate to the actual cost and schedule?

Long Description

Inverted pyramid graphic with the text in the following descending order:

Top tier: Cost Data, Technical Data
Second tier: Programmatic Data, Cost Drivers, Size
Bottom tier: Schedule Estimate

Software Data - Development Data

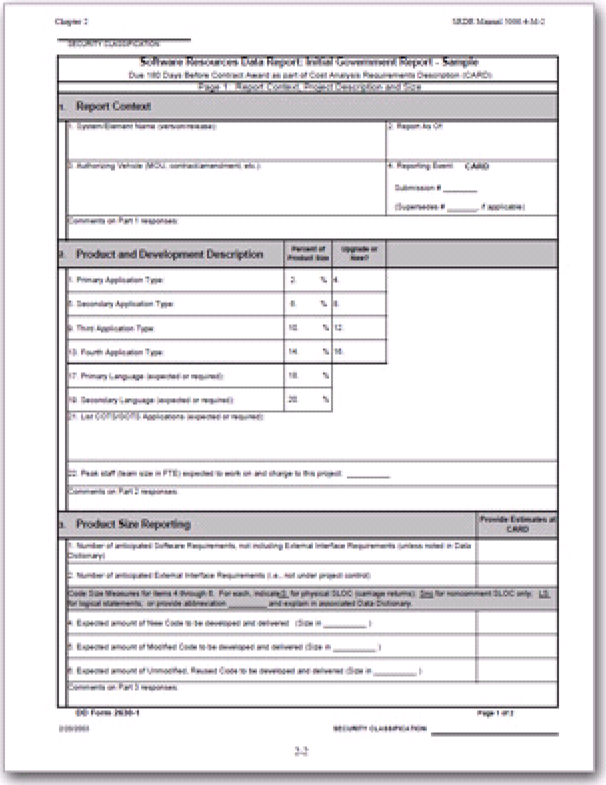
Software Resource Data Reports (SRDRs) are used to collect and report software element data on major Department of Defense (DoD) software-intensive systems, and they represent DoD's only standard centralized approach to software data collection.

Data is required from MAIS programs and MDAPs containing software effort with a projected value greater than $25M (FY 2002 dollars).

The data collected includes the type and size of the software application, the schedule and labor resources (effort) needed for its development, the quality of the delivered software, and other descriptive development data, capturing both the estimated and actual characteristics of new software developments or upgrades.

Both the Government program office and, later on after contract award, the development contractor submit this report. For contractors, this report constitutes a contract data deliverable (CDRL) that formalizes the reporting of software metric and resource data.

Select image to view enlarged version.



Chapter 2 SRDR Manual 5000.4-M-2

SECURITY CLASSIFICATION

Software Resources Data Report: Initial Government Report - Sample

Due 180 Days Before Contract Award as part of Cost Analysis Requirements Description (CARD)

Page 1: Report Context, Project Description and Size

1. Report Context

1. System/Element Name (version/release):	2. Report As Of:
3. Authorizing Vehicle (MOU, contract/amendment, etc.):	4. Reporting Event: CARD Submission # ______ (Supersedes # ______, if applicable)
Comments on Part 1 responses:	

2. Product and Development Description	Percent of Product Size	Upgrade or New?
1. Primary Application Type:	2. %	4.
5. Secondary Application Type:	6. %	8.
9. Third Application Type:	10. %	12.
13. Fourth Application Type:	14. %	16.
17. Primary Language (expected or required):	18. %	
19. Secondary Language (expected or required):	20. %	

21. List COTS/GOTS Applications (expected or required):

22. Peak staff (team size in FTE) expected to work on and charge to this project: ______

Comments on Part 2 responses:

3. Product Size Reporting	Provide Estimates at CARD
1. Number of anticipated Software Requirements, not including External Interface Requirements (unless noted in Data Dictionary)	
2. Number of anticipated External Interface Requirements (i.e., not under project control)	
Code Size Measures for items 4 through 6. For each, indicate S for physical SLOC (carriage returns); Snc for noncomment SLOC only; LS for logical statements; or provide abbreviation ______ and explain in associated Data Dictionary.	
4. Expected amount of New Code to be developed and delivered (Size in ______)	
5. Expected amount of Modified Code to be developed and delivered (Size in ______)	
6. Expected amount of Unmodified, Reused Code to be developed and delivered (Size in ______)	

Comments on Part 3 responses:

DD Form 2630-1 Page 1 of 2

2/20/2003 SECURITY CLASSIFICATION ______

2-2

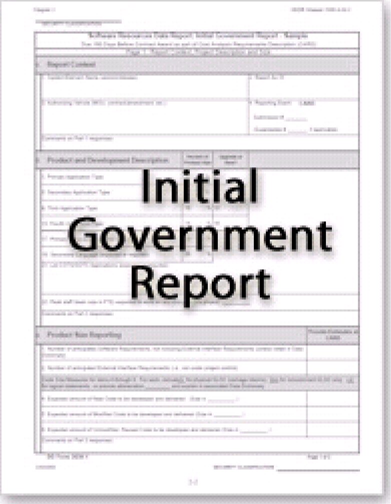
TOC | RESOURCES | PRINT | HELP

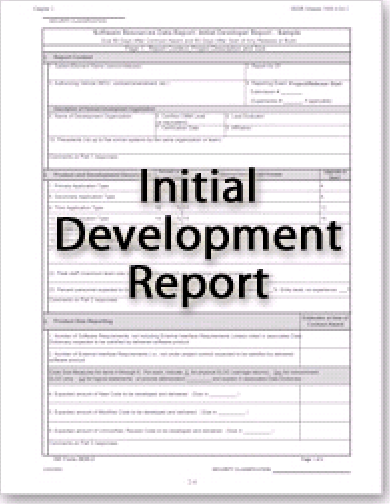
Software Data - Development Data, Cont.

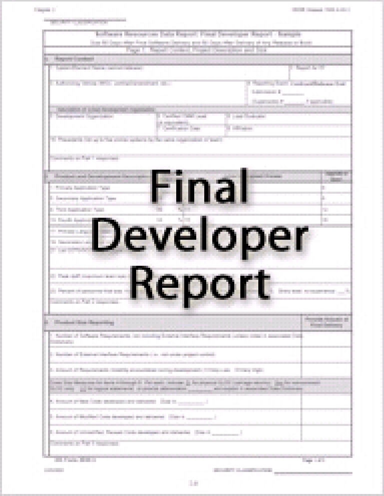
Note that the SRDR forms were previously designated DD 2630, but since they are now tailorable, no DD designation is given.

During each of the phases of software development, numerous activities will be performed in disciplines such as software development, software program management (PM), software configuration management (CM), and software quality assurance (QA). It is important to try to capture the associated effort as comprehensively and with as much granularity as possible.

The DoD 5000.4-M-1, Cost and Software Data Reporting (CSDR) Manual, OSD CAPE, November 4, 2011 provides guidance. Select each graphic below to view examples of the Initial Government Report, Initial Developer Report, and Final Developer Report.







Popup Text

Program Management (PM)

The process whereby a single leader exercises centralized authority and responsibility for planning, organizing, staffing, controlling, and leading the combined efforts of participating/assigned civilian and military personnel and organizations, for the management of a specific defense acquisition program or programs, throughout the system life cycle.

Configuration Management (CM)

The technical and administrative direction and surveillance actions taken to identify and document the functional and physical characteristics of a Configuration Item (CI), to control changes to a CI and its characteristics, and to record and report change processing and implementation status. It provides a complete audit trail of decisions and design modifications.

Quality Assurance (QA)

A planned and systematic pattern of all actions necessary to provide confidence that adequate technical requirements are established, that products and services conform to established technical requirements, and that satisfactory performance is achieved.

Software Data - Cost Data

SRDR data are entered into the Defense Automated Cost Information System (DACIMS) that provides the cost community with instant access to historical cost data needed to develop independent, substantiated estimates. DACIMS is a secure website that allows cost analysts to browse through thousands of Contractor Cost Data Reports (CCDRs) and SRDRs and associated documents via the Internet.

It is the largest repository of DoD cost information. Registration for access to the DACIMS can be obtained through the Defense Cost and Resource Center (DCARC).

SRDRs contain effort data (labor hours) and the corresponding CCDRs have cost data (dollars), which provides a potential cross-check for software development labor rates (dollars per hour).

Mr. Mike Popp, a cost analyst for the Naval Air Systems Command (NAVAIR) routinely posts summary SRDR data in convenient Excel form to one of the DCARC eRooms.

Software Data - Sizing Data

Estimating product size lays the foundation for software cost and schedule estimation.

Size is considered by software project managers to be a major technical performance or productivity indicator, allowing them to track a project during development.

Estimating product size is not simple to do. It is often dependent on the experience of the persons doing the estimating.

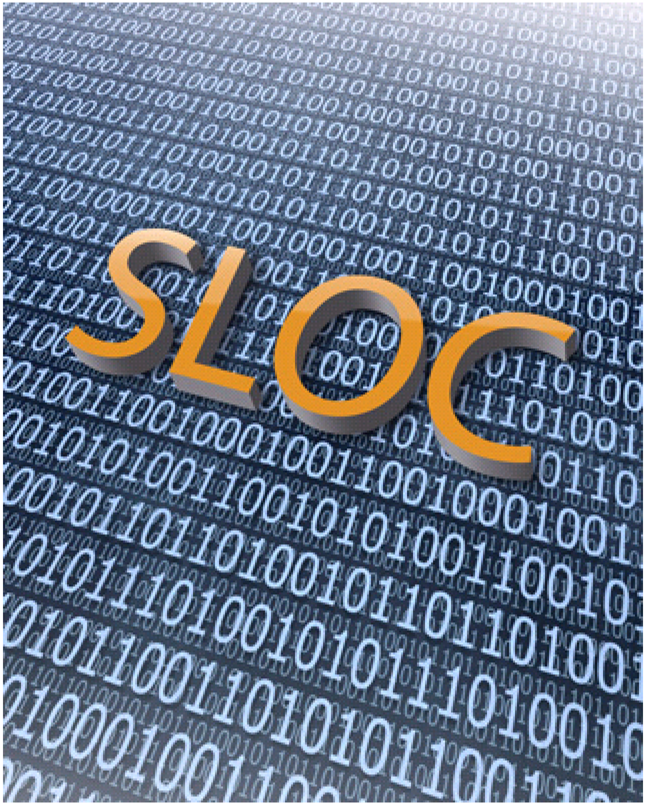
To transcend an expert-based approach, it is crucial to have sizing data in the historical database, for the purpose of generating both sizing estimates and size-driven cost estimating relationships (CERs).

Software Data - Sizing Data, Cont.

The traditional software data sizing measurement is Source Lines Of Code (SLOC). One of the advantages of SLOC is that it can be counted empirically in an automated fashion after the code is released.

Counting conventions vary widely, but there is a movement in the community toward a common standard of the "USC code counter."

SLOC counts can represent either physical SLOC, a simple tally of all non-blank, non-comment lines, or logical SLOC, which tries to arrive at a consistent semantic unit by taking into account language syntax and varying programming styles.

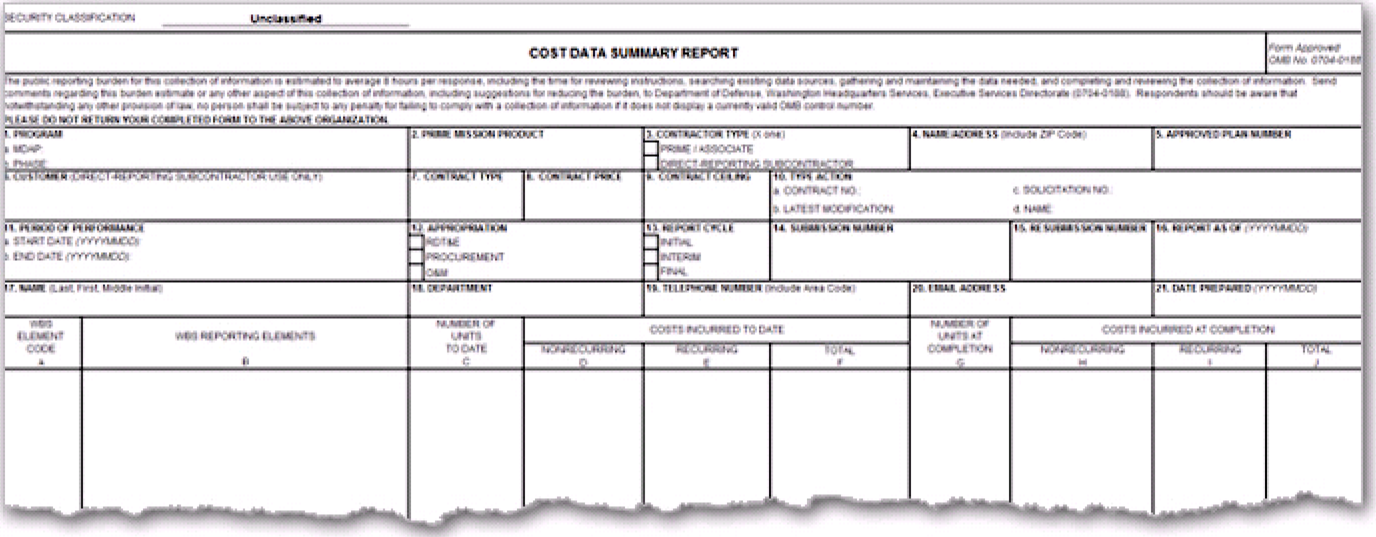


Software Data - Operating and Support (O&S) Data

As previously noted in the CES discussion, O&S cost elements 6.2 Sustaining Support and 6.3 Software Maintenance Support are of particular interest in software.

Collecting data to support Post-Deployment Software Support (PDSS) estimates is a particular challenge, and a joint effort is underway by the three service cost centers to collect software maintenance data from both organic support activities and contractors.

In the area of contractor logistics support (CLS), a DD Form 1921-4 is being considered to capture some data on software support.



SECURITY CLASSIFICATION Unclassified

COST DATA SUMMARY REPORT

Form Approved
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 8 hours per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Executive Services Directorate (0704-0188). Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR COMPLETED FORM TO THE ABOVE ORGANIZATION.

1. PROGRAM
a. MDAP:
b. PHASE:

2. PRIME MISSION PRODUCT

3. CONTRACTOR TYPE (X one)
□ PRIME / ASSOCIATE
□ DIRECT-REPORTING SUBCONTRACTOR

4. NAME/ADDRESS (Include ZIP Code)

5. APPROVED PLAN NUMBER

6. CUSTOMER (DIRECT-REPORTING SUBCONTRACTOR USE ONLY)

7. CONTRACT TYPE

8. CONTRACT PRICE

9. CONTRACT CEILING

10. TYPE ACTION
a. CONTRACT NO.:
b. LATEST MODIFICATION:
c. SOLICITATION NO.:
d. NAME:

11. PERIOD OF PERFORMANCE
a. START DATE (YYYYMMDD):
b. END DATE (YYYYMMDD):

12. APPROPRIATION
□ RDT&E
□ PROCUREMENT
□ O&M

13. REPORT CYCLE
□ INITIAL
□ INTERIM
□ FINAL

14. SUBMISSION NUMBER

15. RESUBMISSION NUMBER

16. REPORT AS OF (YYYYMMDD)

17. NAME (Last, First, Middle Initial)

18. DEPARTMENT

19. TELEPHONE NUMBER (Include Area Code)

20. EMAIL ADDRESS

21. DATE PREPARED (YYYYMMDD)

WBS ELEMENT CODE A	WBS REPORTING ELEMENTS B	NUMBER OF UNITS TO DATE C	COSTS INCURRED TO DATE			NUMBER OF UNITS AT COMPLETION G	COSTS INCURRED AT COMPLETION		
			NONRECURRING D	RECURRING E	TOTAL F		NONRECURRING H	RECURRING I	TOTAL J

Popup Text

Post-Deployment Software Support (PDSS)

Those software support activities that occur after the deployment of the system.

Software Data - Operating and Support (O&S) Data, Cont.

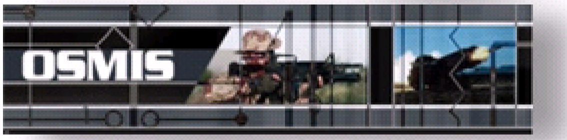
Office of the Secretary of Defense (OSD) undertook the Visibility and Management of Operating and Support Costs (VAMOSC) initiative to facilitate the development of a well-defined, standard presentation of O&S costs by major defense acquisition programs.

VAMOSC data can serve as a basis for decisions concerning affordability, budget development, support concepts, cost tradeoffs, modifications, and retention of current systems.

Unfortunately, in the area of software support, VAMOSC data alone are generally insufficient to support estimating much beyond a simple average.

The Services have implemented VAMOSC through the following programs:

- Army - Operating and Support Management Information System (OSMIS)
- Navy - Navy VAMOSC
- Air Force - Air Force Total Ownership Cost (AFTOC)



Software Data - Rates Data

Because software development is a labor-intensive activity, the traditional estimating approach is to estimate effort first in labor hours or person-months and then "dollarize" that effort by applying the appropriate labor rate.

Rates should be representative of the capability used to produce the effort estimate, and premiums should be included for specialized skills such as experience with programming languages or development platforms or qualifications such as security clearances.

These analyses should be based on historical data for labor rates and associated inflation trends.

Such data may be available via government surveillance organizations such as the Defense Contract Audit Agency (DCAA) and Defense Contract Management Agency (DCMA).

DEFENSE CONTRACT AUDIT AGENCY
DEFENSE CONTRACT MANAGEMENT AGENCY
DEPARTMENT OF DEFENSE

Software Data - Collecting Rates Data

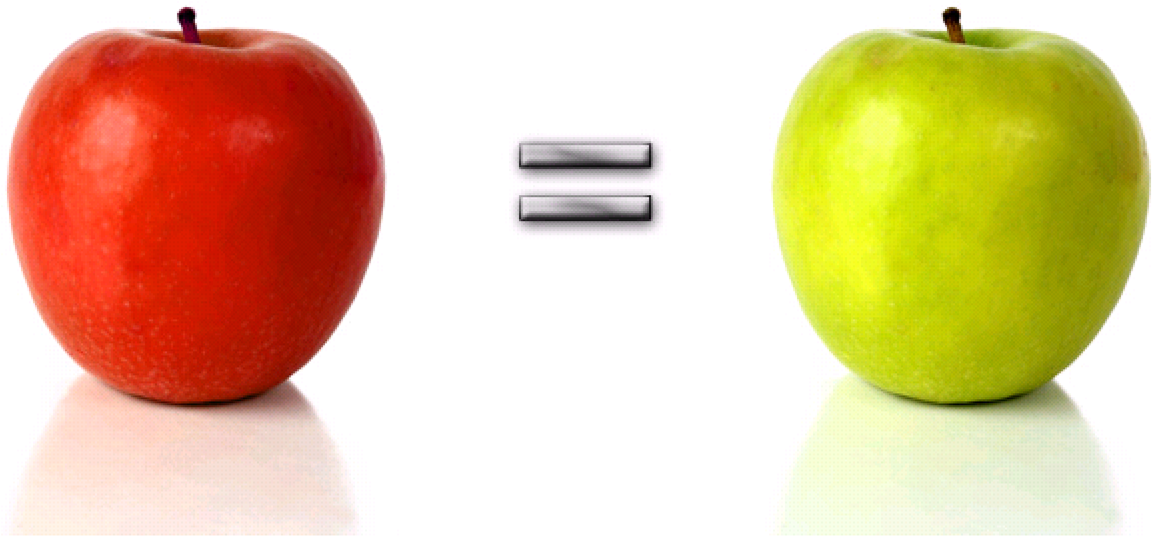
In collecting rates data, it is preferable to have actual rates across a wide range of labor categories, each with clearly-defined qualifications, and several years.

Contractor-specific rates may also be available in General Services Administration (GSA) schedules and proposals, but make sure to be clear on whether the rates are binding or merely a current best estimate.

Where necessary, a single composite rate, representing the weighted average labor rate of the development team, may suffice.

Data Normalization

The mantra for data normalization is "What's in the number?" While much faith is put into actual data from historical programs, it is important to understand the context in which those data were collected so that we can properly adjust them to be comparable with data from other sources – the proverbial "apples to apples."



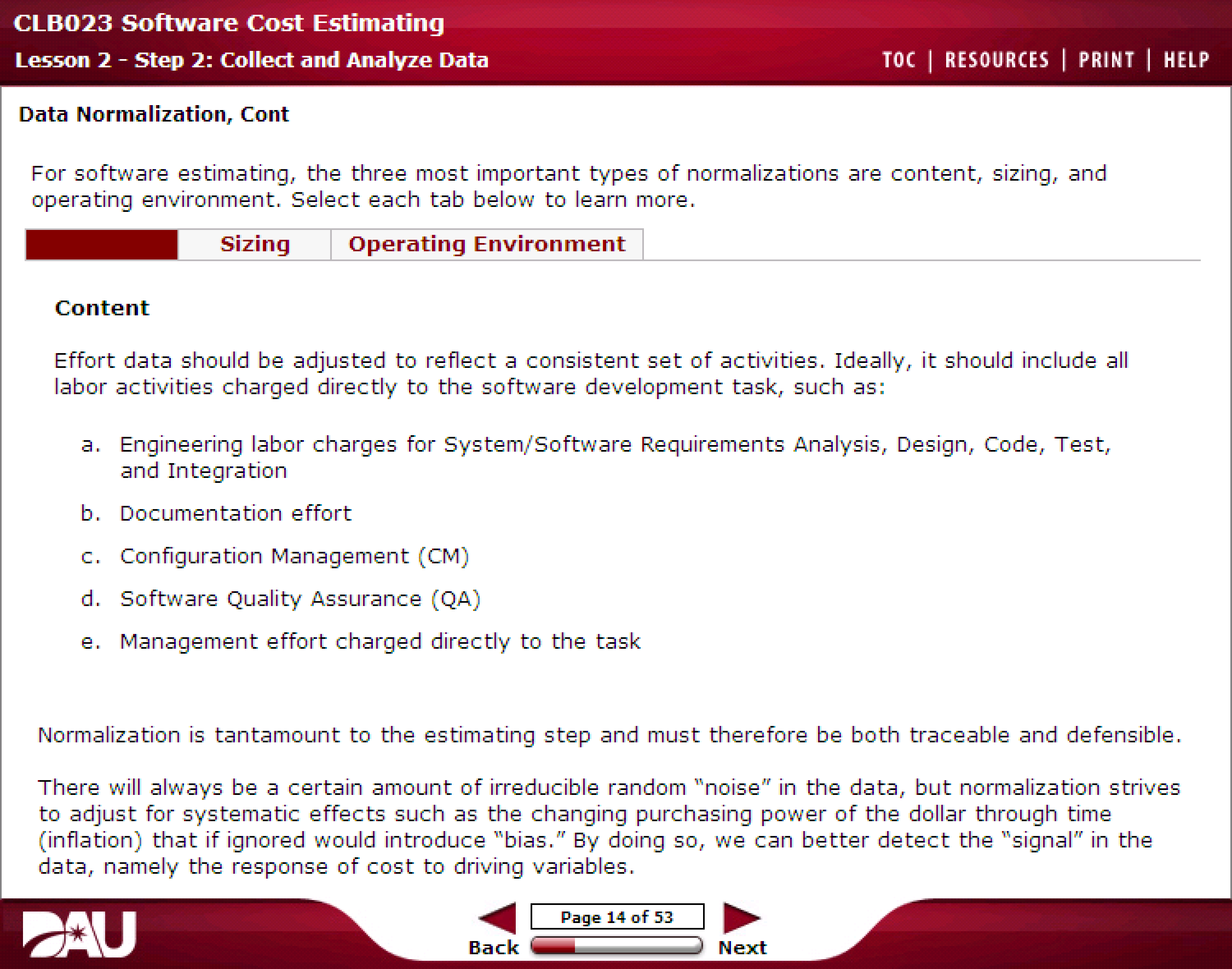
Questions Managers Should Ask

Popup Text

Questions Managers Should Ask

Is the organization’s historical evidence capable of supporting a reliable estimate?

- Are elements included in (and excluded from) the effort, cost, schedule, size, and reuse measures in the database clearly identified?
- Have the data in the historical database been examined to identify inconsistencies, and have anomalies been corrected or explained?
- Does information on completed projects include a work breakdown structure or alternative description of the tasks included in the recorded cost?
- Does information on completed projects include non-labor and management costs?
- Does information on completed projects include a summary or list of significant deliverables (software and documentation) produced by the project, and a summary of any unusual issues that affected cost or schedule?



Data Normalization, Cont

For software estimating, the three most important types of normalizations are content, sizing, and operating environment. Select each tab below to learn more.

Content

Effort data should be adjusted to reflect a consistent set of activities. Ideally, it should include all labor activities charged directly to the software development task, such as:

a. Engineering labor charges for System/Software Requirements Analysis, Design, Code, Test, and Integration

b. Documentation effort

c. Configuration Management (CM)

d. Software Quality Assurance (QA)

e. Management effort charged directly to the task

Normalization is tantamount to the estimating step and must therefore be both traceable and defensible.

There will always be a certain amount of irreducible random "noise" in the data, but normalization strives to adjust for systematic effects such as the changing purchasing power of the dollar through time (inflation) that if ignored would introduce "bias." By doing so, we can better detect the "signal" in the data, namely the response of cost to driving variables.

Popup Text

Content

Effort data should be adjusted to reflect a consistent set of activities. Ideally, it should include all labor activities charged directly to the software development task, such as:

a. Engineering labor charges for System/Software Requirements Analysis, Design, Code, Test, and Integration
b. Documentation effort
c. Configuration Management (CM)
d. Software Quality Assurance (QA)
e. Management effort charged directly to the task

Sizing

Delivered SLOC should be adjusted to reflect degree of reuse to give an Equivalent (New) SLOC number, which should have a more consistent relationship with effort.

Operating Environment

Data are often segregated by platform and/or application to reflect the way that reliability, safety, and other considerations affect the complexity and difficulty of the code.

Data Normalization - Content

Arguably the most important normalization step is to adjust effort or cost numbers to reflect a consistent work scope or "content." One required application of this principle is the segregation of recurring and non-recurring costs. Select each tab below to learn more about each.

Recurring Costs

Non-recurring costs include all the efforts required to develop and qualify a given item, such as requirements definition/allocation, design, analysis, development, and qualification/verification.

Virtually all software development and testing costs prior to initiation of routine system operation are non-recurring.

Popup Text

Non-recurring Costs

Non-recurring costs include all the efforts required to develop and qualify a given item, such as requirements definition/allocation, design, analysis, development, and qualification/verification.

Virtually all software development and testing costs prior to initiation of routine system operation are non-recurring.

Popup Text

Recurring Costs

Recurring costs cover all efforts required to produce end-item hardware, including manufacturing and test, engineering support for production, and spare units or parts. Production support software costs are commonly classified as recurring.

A more germane normalization for software is the segregation of fixed and variable costs.

For example, infrastructure costs such as SILs are largely fixed – you need the lab whether you write one line of code or a million – whereas design/code/test activities do vary with the amount of code being developed.

Examples of adjustment for consistent scope include:

- If the effort data for some historical programs include software requirements but others start at design, then either the requirements effort must be removed from the former or a way must be found to "plus up" the latter to represent the same scope.
- Suppose the systems engineering department compared five similar programs, and found that two included design-to-cost (DTC) Design-to-Cost (DTC) requirements. To normalize the data, the DTC hours must be deleted from those two programs to create a data set with consistent program scope.

Popup Text

Design-to-cost (DTC)

Management concept that historically emphasized cost-effective design (minimizing cost while achieving performance) and targeting an Average Unit Procurement Cost (AUPC). DTC concentrated on the contractors' activities associated with tracking/controlling costs and performing cost-performance analyses/tradeoffs. Cost As an Independent Variable (CAIV) has refocused DTC to consider cost objectives for the total life cycle of the program and to view CAIV with the understanding it may be necessary to trade off performance to stay within cost objectives and constraints. DTC is now those actions that are undertaken to meet cost objectives through explicit design activities. Contractual implementation of DTC should go beyond simply incentivizing the contractor to meet cost commitments—it should also incentivize the contractor to seek out additional cost reduction opportunities.

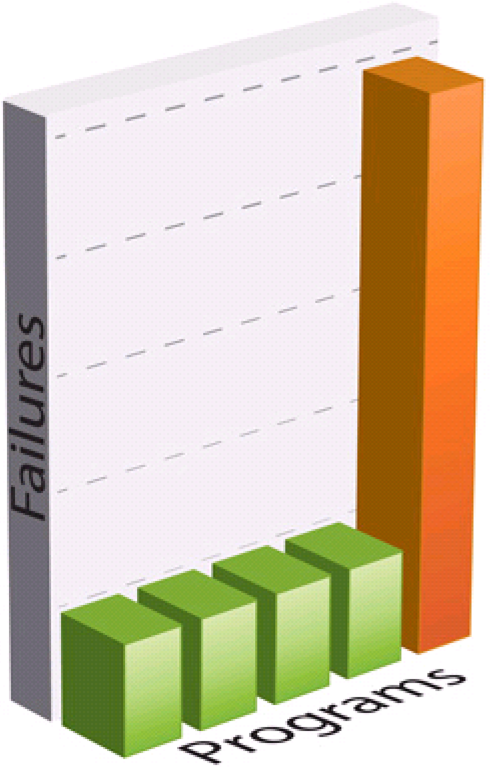
Data Normalization - Historical Cost Data

Historical cost data can also be adjusted for anomalies when it is not reasonable to expect the new project estimates to contain these unusual costs.

For example, development test program data are collected from five similar programs, and it is found that one program experienced a major test failure. A considerable amount of resources were required to find and solve the problem.

If an adjustment is made to this data point, then the analyst must thoroughly document the actions taken to identify the anomalous hours.

If the hours are removed from the base estimate, then a corresponding risk should be added to the program's risk register with a consequence equal to the additional hours and an appropriate probability (apparently 20%).



Data Normalization - Cost

Other than simply keeping dollars ($), thousands ($K), millions ($M), and billions ($B) straight, the most common and important normalization of cost units is adjustment for inflation, called escalation.

Inflation is the general increase in price levels (or conversely decline in purchasing power) over time.

Inflation indices (or index numbers) are used to put costs in a consistent "base-year dollar" for the purposes of estimating.

Inflation is certainly relevant to any software cost data, though for the development, the preference is to estimate effort (labor hours or person-months) first and then apply labor rates, in which case only the rates need to be escalated.

Data Normalization - Sizing Units

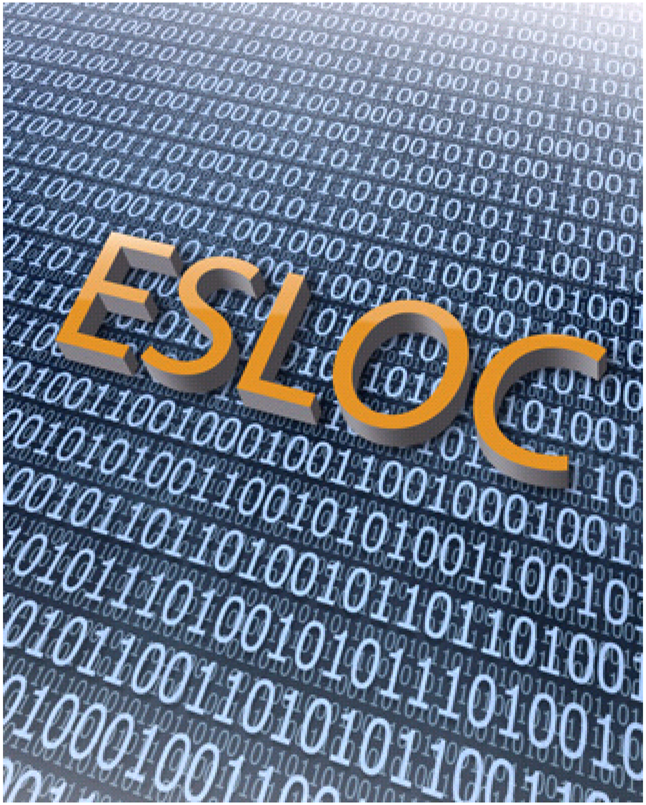
The more relevant normalization for software is adjustment for sizing units.

This is less straightforward than for hardware, where one can simply multiply by 2.54 to convert from inches to centimeters, for example.

The most common sizing unit for software is source lines of code (SLOC), and the most common normalization for SLOC is the conversion to Equivalent SLOC (ESLOC), both of which are addressed later in this lesson.

ESLOC attempts to capture the varying effort associated with developing new code as compared with reusing code, with or without modification.

The effort required to develop code also varies by programming language, so that either data sets are segregated by language, or a factor is applied to convert size in one language to another.

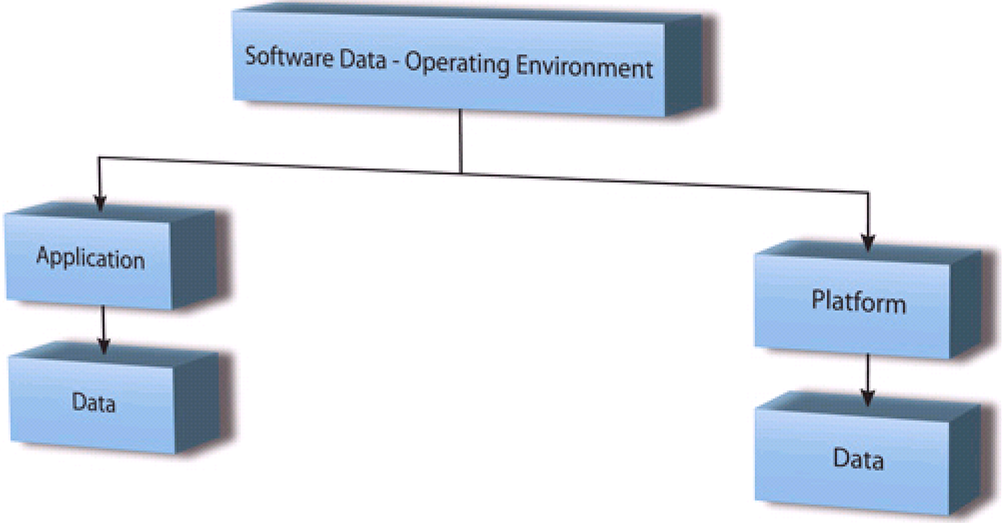


Data Normalization - Operating Environment

Software data are often categorized by Application (the function of the software, such as flight control or mission planning) and Platform (where the software resides, such as Airborne or Ground). The idea is that these are key factors influencing the difficulty and complexity of the code, and therefore the amount of effort required to develop it.

After the categorization is determined, the analyst may segregate the data and analyze each grouping separately. Alternately, the data may be combined and indicator variables used in the subsequent regression analysis to account for any significant differences between groupings.

Operating Environment is also addressed in the cost drivers (complexity) section of this lesson.



Software Data - Operating Environment

Application

Data

Platform

Data

D

Popup Text

Long Description

Flowchart illustrating Software data categorization with Application as on one branch and Platform on the other branch. Each category (Application, Platform) has Data as supporting subordinate components.

Data Analysis

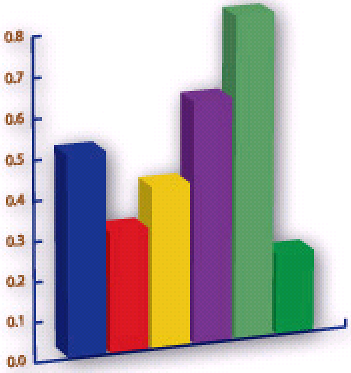
After data collection and normalization, data analysis is conducted to develop estimating relationships.

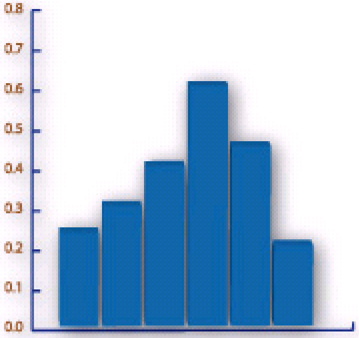
Both software development cost (effort) and schedule (duration) can be related to size and other driving factors, for example. Data analysis is essentially the application of graphical, numerical, and algebraic principles from probability and statistics.

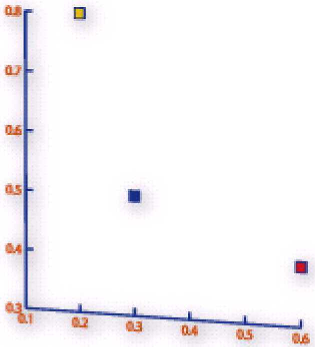
A simple bar chart could be used to show initial estimates and final values as a clear indication of code growth.

A histogram could show the distribution of CSCIs by size in a software database. A two-sample t-test could be conducted to explore differences between two populations (Aircraft vs. Space, or C++ vs. C#).

A scatterplot of effort in hours vs. size in ESLOC could be used to discern a cost estimating relationship, or CER.







Data Analysis - Productivity

One commonly reported software metric is productivity, which attempts to capture the efficiency of designers and programmers in developing code.

It is simply the ratio of code size (usually in SLOC) to effort (in labor hours, days, or months). While this quotient is often called a factor, it is really more of a rate. Its inverse (effort per size), which is often used, makes this clearer.

Popup Text

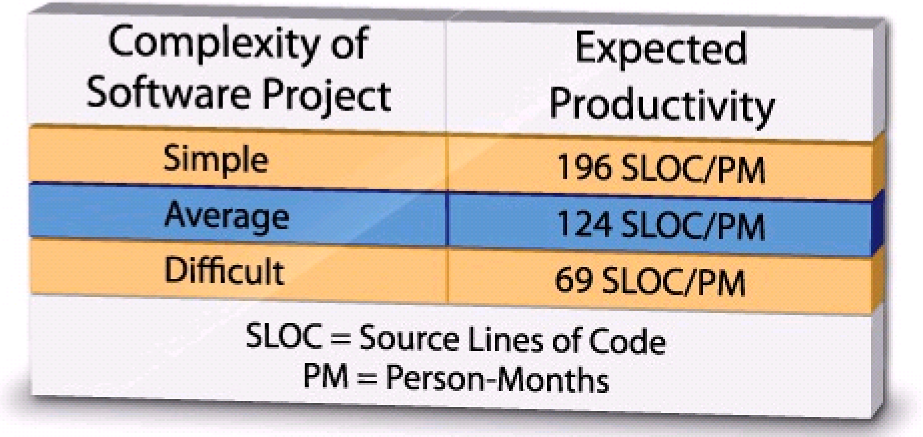
Productivity

The actual rate of output or production per unit of time worked.

Data Analysis - Productivity, Cont.

The table below illustrates notional historical productivity rates based upon the level of complexity of a software project. If these numbers seem low, it is important to note that the programmer is not simply sitting down at the keyboard and typing away, as if writing an email.

There is considerable requirements analysis, design, code, and test effort that must be amortized across each individual line of code. Ideally, these rates would be based on a database of recently completed comparable development efforts.



Complexity of Software Project	Expected Productivity
Simple	196 SLOC/PM
Average	124 SLOC/PM
Difficult	69 SLOC/PM

SLOC = Source Lines of Code
PM = Person-Months

Caution should be taken when using productivity as the primary basis of a software estimate, as it neglects the effects of fixed costs (which can cause larger projects to have a slightly higher apparent productivity) and diseconomies of scale (which can cause larger projects to have a much lower effective productivity).

D

Long Description

Table with data:

Complexity of Software Project	Expected Productivity
Simple	196 SLOC/PM
Average	124 SLOC/PM
Difficult	69 SLOC/PM

SLOC = Source Lines of Code
PM = Person-Months	

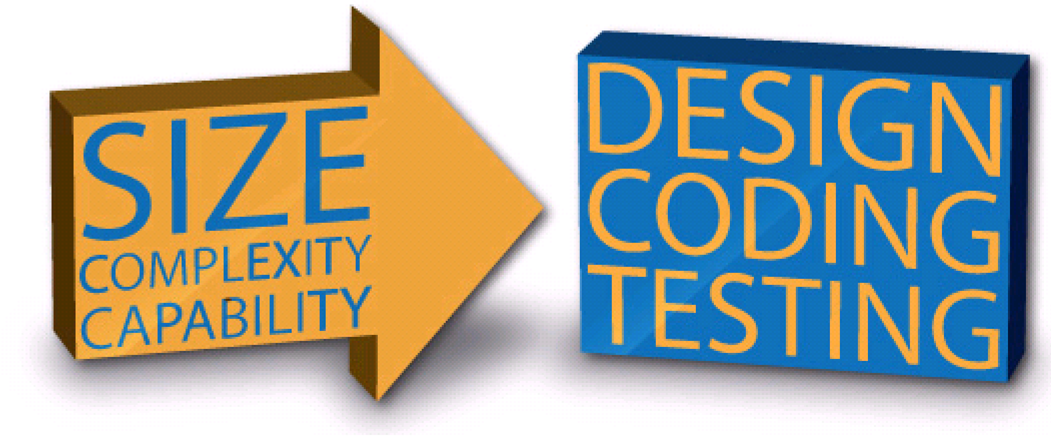
Software Cost Drivers

A primary goal of data analysis is to understand the factors or parameters that drive cost in order to facilitate the development of CERs. The three main categories of software development cost drivers are Size, Complexity and Capability, each of which is discussed in the following pages.

Throughout this discussion it is important to note that the nature of cost drivers is expected to vary by the type of software. Select each tab to learn more about types of software.

COTS | ERP

Developed - The effort for designing, coding, and testing developed software is driven by the size and complexity of the code, and the capability of the development team.



Popup Text

Developed

Developed - The effort for designing, coding, and testing developed software is driven by the size and complexity of the code, and the capability of the development team.

COTS

Commercial off-the-shelf (COTS) - The functionality provided by the COTS package and its sophistication drive purchase and licensing costs, and the ease of use and thoroughness of documentation of its application programming interface (APIs) drive integration costs.

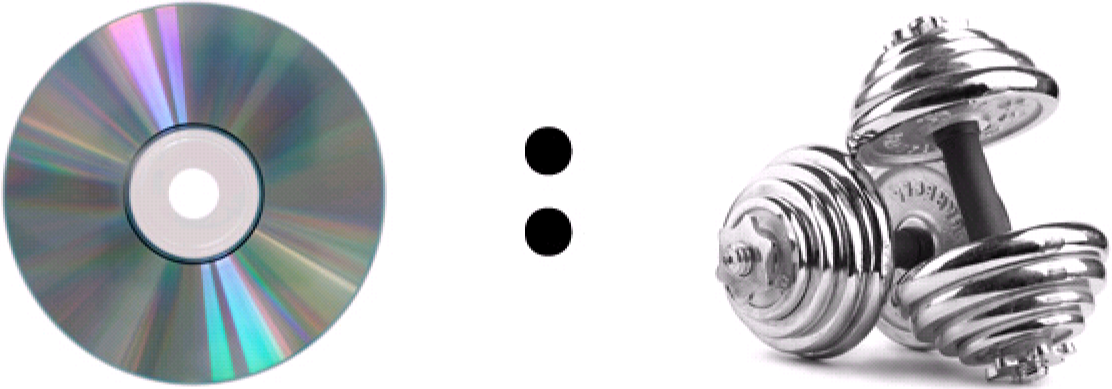
ERP

Enterprise Resource Planning (ERP) - The scope and effort for an ERP implementation are driven by the RICE-FW count, an enumeration of the required Reports, Interfaces, Conversions, Extensions, Forms and Workflows.

Software Cost Drivers - Size

The primary driver for developed software is the size of the code, as an indicator of the overall scope of the effort. Just as a larger ship requires more welding, and a larger IT installation requires running more cable, producing more code generally requires more effort. (The advent of auto-generated code presents a challenge in relating effort to size of the delivered code, and the focus of the discussion here is on human-generated code.)

Software size is roughly analogous to weight, which is commonly used as a cost driver for hardware production estimates (arguably as a proxy for scope of the effort).



Just as miniaturization can drive a reversal in the weight-cost relationship in hardware (i.e., lighter costs more), placing size restrictions on software, such as the memory limitations on a satellite, can cause a similar reversal. To paraphrase Blaise Pascal and Mark Twain, "If I'd had more time, I would've written less code."

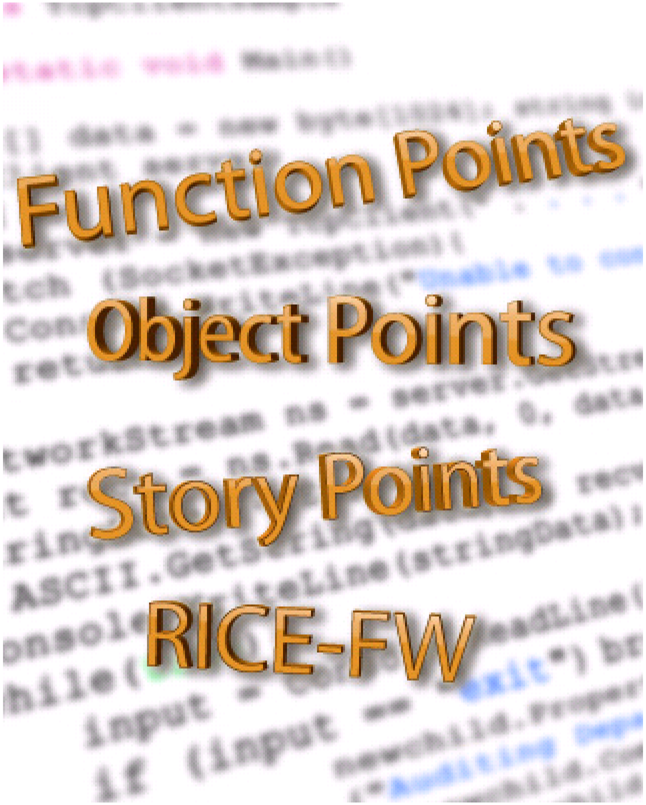
Software Cost Drivers - Size, Cont.

The most common sizing measure is SLOC; however, other potential sizing units include:

- Function Points, which attempt to remain language-independent by focusing on the required functionality of the software
- Object Points, which are tailored for use with object-oriented (OO) programming
- Story Points, which are used in Agile Software Development
- RICE-FW, the sizing measure specific to ERP

Keep in mind that the SRDRs should be considered a primary source of Size and other software data.

Also remember that for new programs, software size is itself an estimate!



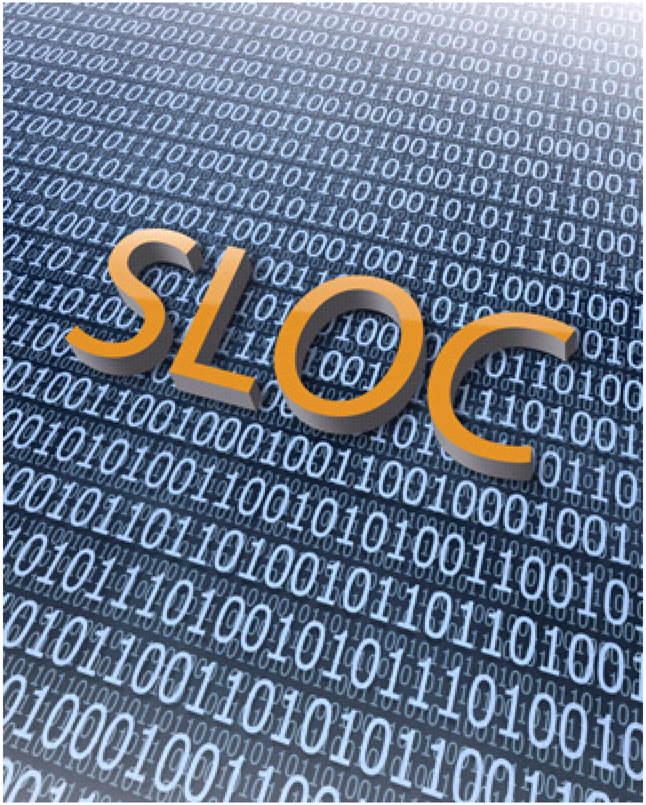
Software Cost Drivers - Size - Source Lines of Code (SLOC)

Source Lines Of Code (SLOC) is exactly what it sounds like.

When each code release is completed, an automated code counter can be run to tally Delivered Source Lines Of Code (DSLOC) for inclusion in the historical database.

Logical SLOC is generally preferred to physical SLOC, though consistency throughout the database is most important.

For new programs, SLOC must be estimated up front, often by analogy to comparable completed efforts.



Popup Text

Source Lines of Code (SLOC)

Number of lines of software code, including executable instructions and data declarations but excluding comments, blanks, and continuation lines. Can be accurately and consistently counted using automated tools once the code is developed.

Delivered Source Lines of Code (DSLOC)

The actual source lines of code (SLOC) delivered as part of a software release. May be physical or logical.

Software Cost Drivers - Size - SLOC, Cont.

In a typical software development CER, as in the one from COCOMO II, the exponent on the size parameter indicates the response of effort to size.

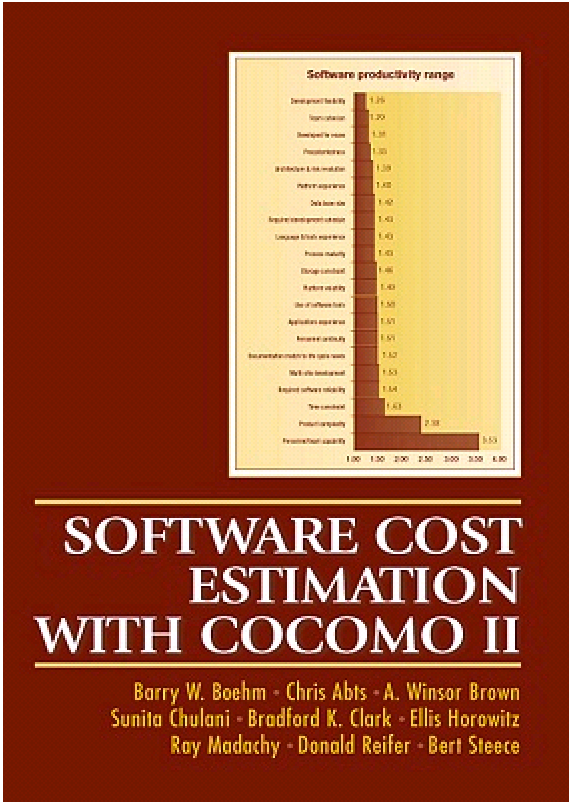
A priori, we might expect a direct linear relationship between size an effort, and this is the implicit assumption of the productivity metric just discussed.

However, if economies of scale are present, effort should increase at a decreasing rate with size, corresponding to an exponent less than 1.

Conversely, if diseconomies of scale are present, effort should increase at an increasing rate with size, corresponding to an exponent greater than 1.

Experience shows that the latter is more common, and our examples will reflect that scenario.

These equations are calibrated to estimate the effort of an entirely new development, writing code from scratch.


Software productivity range
SOFTWARE COST ESTIMATION WITH COCOMO II
Barry W. Boehm · Chris Abts · A. Winsor Brown
Sunita Chulani · Bradford K. Clark · Ellis Horowitz
Ray Madachy · Donald Reifer · Bert Steece

Software Cost Drivers - Size - Equivalent SLOC (ESLOC)

Modern software is typically not written from scratch, but instead its development involves the reuse of code. Therefore, a new sizing metric must be used to reflect the change. Equivalent SLOC (ESLOC) is now used.

It is important to to understand the varying terms used for reused code.

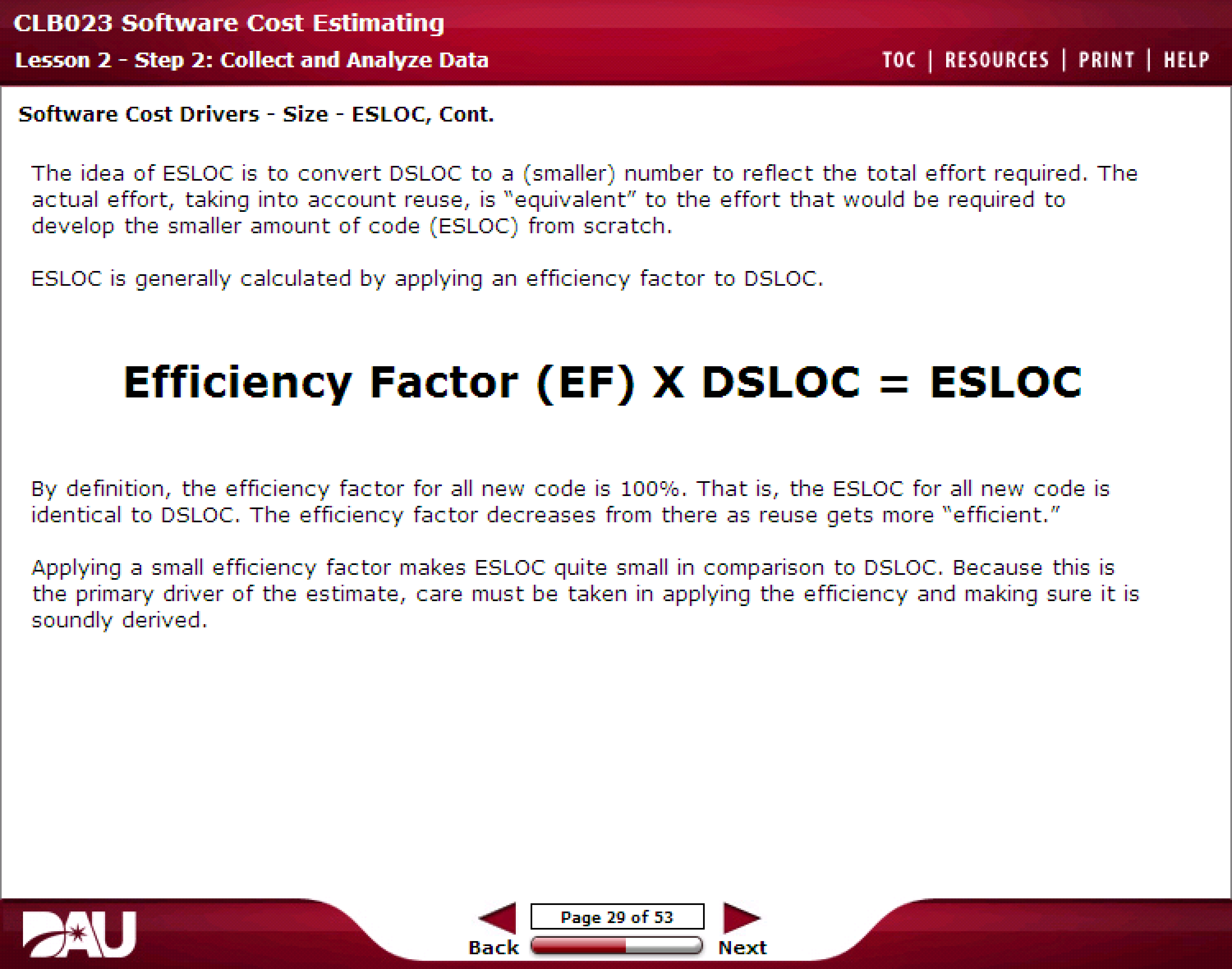
"Reuse" often implies no modification, whereas "modified" usually refers to reused code that was altered to some extent.

"Carryover" is often used to denote code that is being reused from the previous release of the software.

Popup Text

Equivalent Source Lines of Code (ESLOC)

A weighted average of the amounts of code requiring re-design, re-implementation, and re-test.



Software Cost Drivers - Size - ESLOC, Cont.

The idea of ESLOC is to convert DSLOC to a (smaller) number to reflect the total effort required. The actual effort, taking into account reuse, is "equivalent" to the effort that would be required to develop the smaller amount of code (ESLOC) from scratch.

ESLOC is generally calculated by applying an efficiency factor to DSLOC.

$$\text{Efficiency Factor (EF)} \times \text{DSLOC} = \text{ESLOC}$$

By definition, the efficiency factor for all new code is 100%. That is, the ESLOC for all new code is identical to DSLOC. The efficiency factor decreases from there as reuse gets more "efficient."

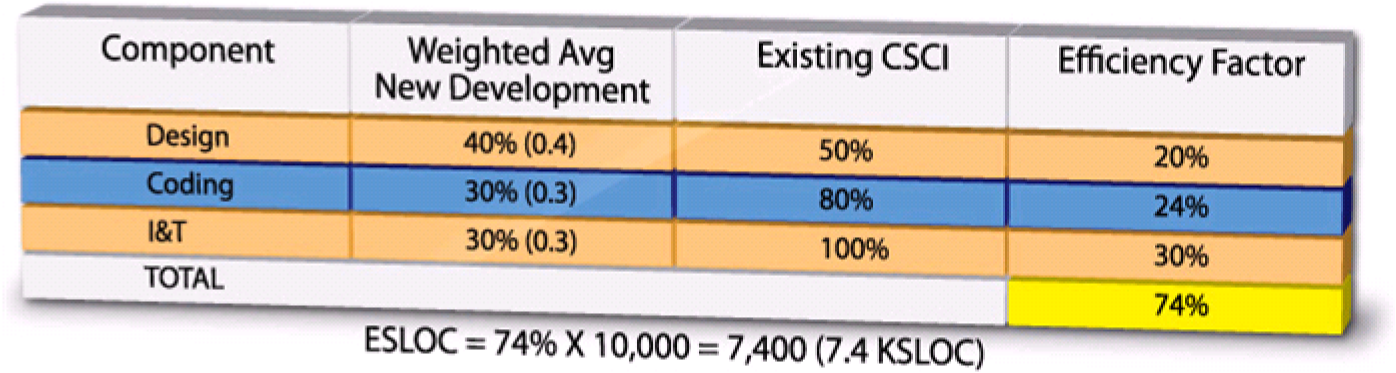
Applying a small efficiency factor makes ESLOC quite small in comparison to DSLOC. Because this is the primary driver of the estimate, care must be taken in applying the efficiency and making sure it is soundly derived.

Software Cost Drivers - Size - ESLOC, Cont.

A standard approach is to compute the efficiency factor as a weighted average of factors for each phase of development. For example, assume that 40% of a new development effort is design, 30% is coding, and 30% is integration and test (I&T).

The weighted averages can be applied to the fraction of existing code that needs to be redesigned, the fraction of existing code that needs to be rewritten, and the degree of retest required, respectively.

The information presented in the graphic assumes the following software development scenario: existing computer software configuration item (CSCI) of 10,000 SLOC (10 KSLOC), 50% redesign, 80% recode, 100% retest.



Component	Weighted Avg New Development	Existing CSCI	Efficiency Factor
Design	40% (0.4)	50%	20%
Coding	30% (0.3)	80%	24%
I&T	30% (0.3)	100%	30%
TOTAL			74%

ESLOC = 74% X 10,000 = 7,400 (7.4 KSLOC)

The effort to reuse this code is 74% of what it would take to write it from scratch, a 26% savings. To compute total ESLOC for a planned release, add the size of the new code to be produced and the ESLOC calculations for each piece of reused code.

D

Popup Text

Degree of retest

Generally all code needs to be retested, even the portion reused without modification.

Long Description

Table with the following data:

Component	Weight Avg. New Development	Existing CSCI	Efficiency Factor
Design	40% (0.4)	50%	20%
Coding	30% (0.3)	80%	24%
I&T	30% (0.3)	100%	30%
Total			74%
ESLOC = 74% X 10,000 = 7,400 (7.4 KSLOC)			

Software Cost Drivers - Size - ProRad ESLOC Calculations

The number of SLOC for the software development effort of each waveform was known to be a driving cost of the estimate.

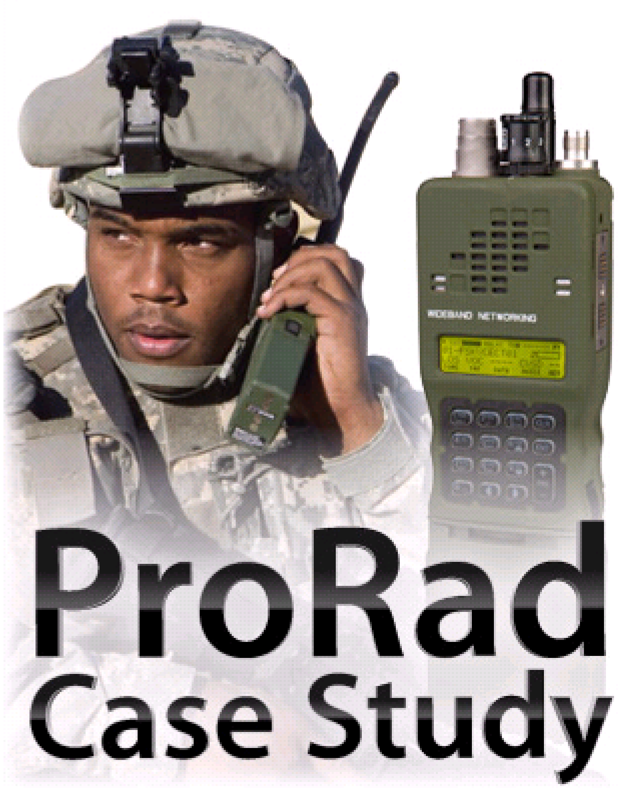
The sizing estimate started in FY1998 with a review of the waveforms to be applied to the Air Force's SpeakEASY Radio.

The Rome Laboratories had developed a list of waveforms and expected SLOC as part of the design for SpeakEASY.

These SLOC counts were reviewed by a team of radio and waveform experts from the Army's PM TRCS, the Navy's Digital Modular Radio Program, and from DARPA to come up with expected SLOC for the ProRad application waveforms.

Similar reviews were conducted with other analogous radio systems.

Click here to view a ProRad waveform SLOC data table.



Software Cost Drivers - Size - ProRad ESLOC Calculations, Cont.

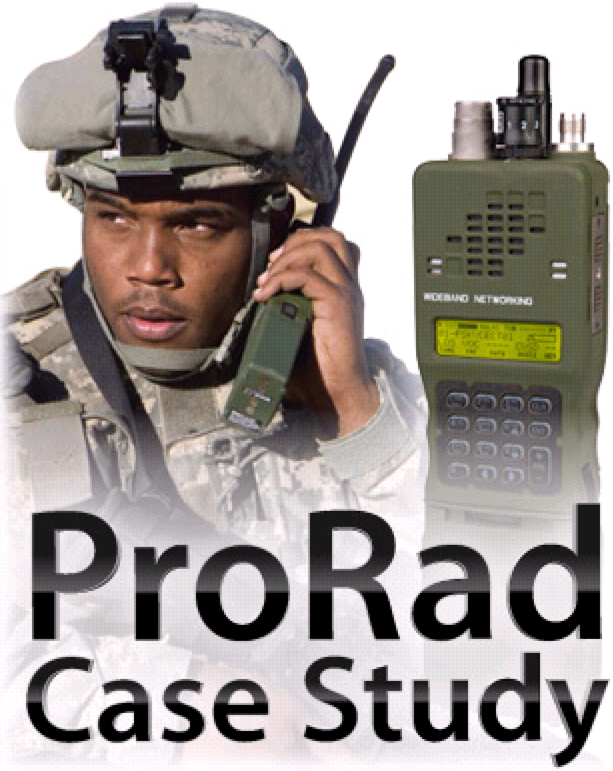
In sum, the following programs were used to help estimate the SLOC for the various ProRad waveform software development efforts:

- Army Systems: Joint Tactical Terminal (JTT); Joint Communications Interface Terminal (JCIT)
- Navy Systems: Digital Modular Radio (DMR); Multi-Function Information Distribution System (MIDS)
- Air Force Systems: Airborne Integrated Terminal Group (AITG); SpeakEASY Radio

In addition, two acquisition strategies were evaluated for each waveform:

1. Develop the entire waveform software from scratch; and
2. Modify the existing waveform software to make it SCA-compliant

Click here to view a ProRad waveform SLOC data table.

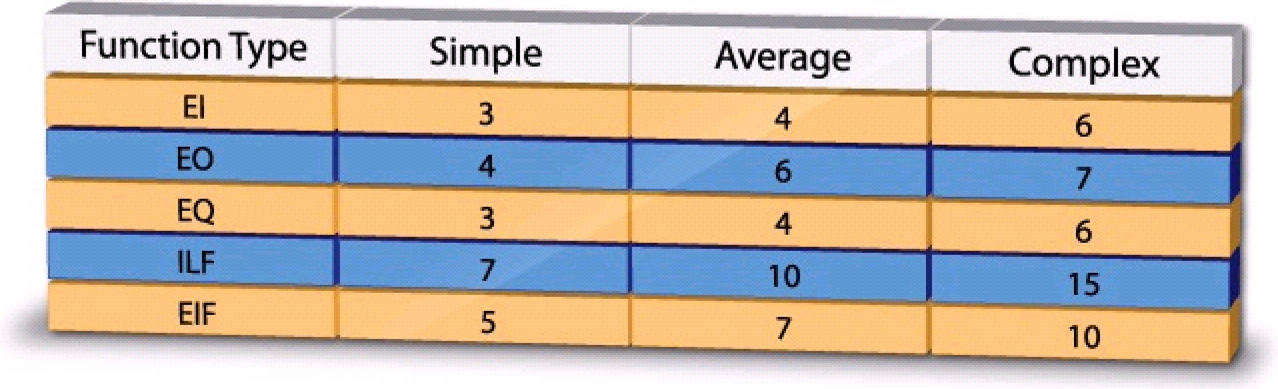


Software Cost Drivers - Size - Function Points

Function points are a size measure that considers the number of functions being developed based on the requirements specification. It is perhaps the most perhaps the most prevalent alternative to SLOC.

The functions counted comprise:

- External Inputs (EI) – User inputs that provide data
- External Outputs (EO) – Output to users such as reports, screens, error messages
- External Inquiries (EQ) – Data sent to other applications
- Internal Logical Files (ILF) – Online input that results in software response
- External Interface Files (EIF) – Machine readable interfaces used to transmit information to another system (disks and tapes)



Function Type	Simple	Average	Complex
EI	3	4	6
EO	4	6	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

D

Long Description

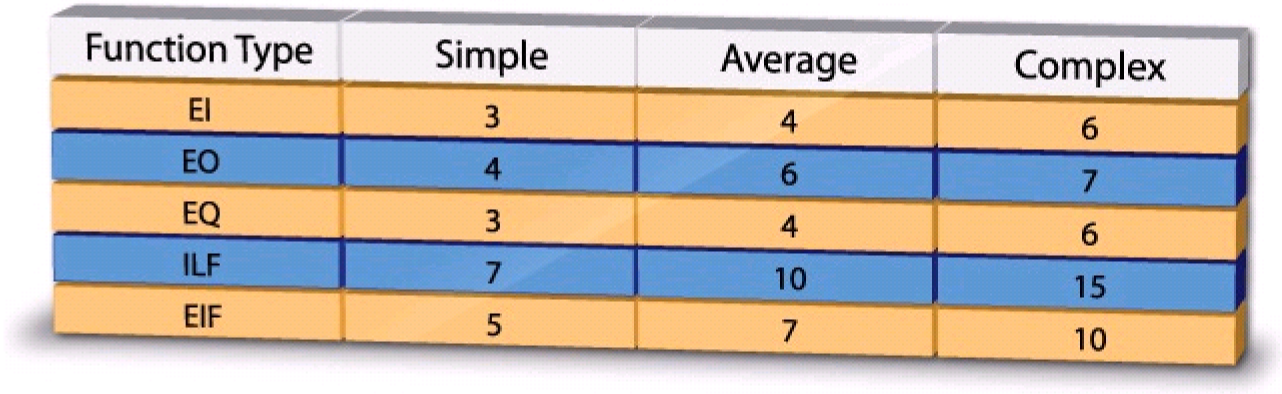
Table with the following data:

Function Type	Simple	Average	Complex
EI	3	4	6
EO	4	6	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

Software Cost Drivers - Size - Function Points, Cont.

After the functions are counted, they are weighted first for complexity to get an unadjusted function point (UFP) count—see table below. The UFP count is then further adjusted to account for 14 additional factors.

There are a number of standards for function point counting, and the International Function Point Users Group (IFPUG) certifies function point counters.



Function Type	Simple	Average	Complex
EI	3	4	6
EO	4	6	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

D

Long Description

Table with the following data:

Function Type	Simple	Average	Complex
EI	3	4	6
EO	4	6	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

Software Cost Drivers - Complexity

After Size, the next broad category of software cost drivers is Complexity, factors relating to the software itself.

The factors may include:

- Quality – requirements for safety or reliability that dictate the degree to which the code must be bug-free.
- Language – the programming language(s) used for development.
- Application – the intended function of the software, which dictates the difficulty of the algorithms needed
- Hardware Limitations – limited memory or processor speed, as on a satellite.
- Number of Modules – drives the degree of integration, standardization, communication, and coordination required.
- Number of Interfaces – both internal interfaces with COTS packages and external interfaces with other systems.

Software Cost Drivers - Complexity - Complexity Factors

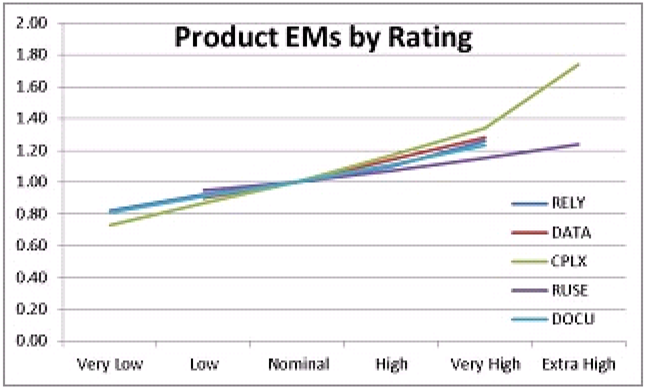
Greater complexity drives up cost, often at increasing rates. While the response may not be exponential in a strict mathematical sense, complexity factors can have a significant impact. COCOMO II divides complexity-related considerations into two groups, product factors and platform factors.

Platform Factors

- RELY Required Software Reliability
- DATA Database Size
- CPLX Product Complexity
- RUSE Developed for Reusability
- DOCU Documentation Match to Life-Cycle Needs

Whether or not you use COCOMO, the response illustrated here is typical.

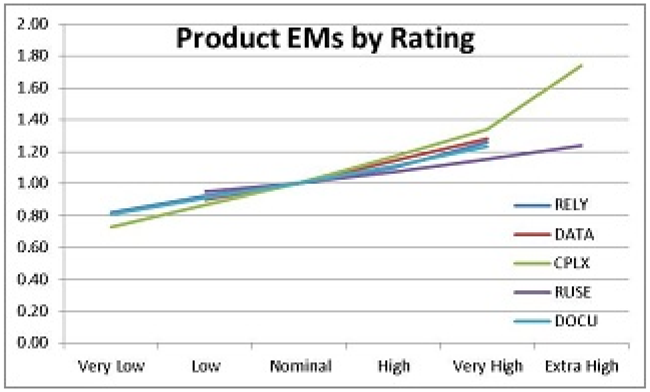
The EMs are Effort Multipliers, and you can see that a single factor can easily drive effort up or down (usually the former) by 20 or 30 percent.



Select the image to enlarge.

Popup Text

Product Factors



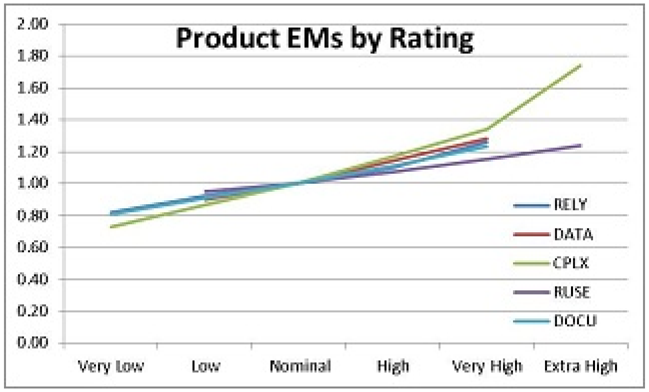
- RELY Required Software Reliability
- DATA Database Size
- CPLX Product Complexity
- RUSE Developed for Reusability
- DOCU Documentation Match to Life-Cycle Needs

Whether or not you use COCOMO, the response illustrated here is typical.

The EMs are Effort Multipliers, and you can see that a single factor can easily drive effort up or down (usually the former) by 20 or 30 percent.

Popup Text

Platform Factors



- RELY Required Software Reliability
- DATA Database Size
- CPLX Product Complexity
- RUSE Developed for Reusability
- DOCU Documentation Match to Life-Cycle Needs

Whether or not you use COCOMO, the response illustrated here is typical.

The EMs are Effort Multipliers, and you can see that a single factor can easily drive effort up or down (usually the former) by 20 or 30 percent.

Software Cost Drivers - Complexity - Quality

Quality requirements for DoD software are usually defined by system safety (if this flight software fails, the plane may crash, and the crew may die) and operational availability (high system reliability, so that it is almost never "down").

High quality calls for well-defined requirements, careful design, thorough testing, and often additional Mission Assurance (MA) or Independent Verification and Validation (IV&V), all driving up effort in comparison to a slapdash development.

Quality is primarily related to the COCOMO EM for Reliability (RELY).

High-quality software has the side benefit of being eminently more maintainable.

Typically there will not be many bugs to find in the first place, and those that do arise are relatively easy to fix.

Software Cost Drivers - Complexity - Quality, Cont.

As with many complex development efforts, software is often prey to major defects uncovered during integration and testing (I&T), requiring significant rework.

To avoid this situation, a sound development approach includes breaking things down into manageable components, defining standards and interfaces, and verification and validation of each step throughout the process.

Tight schedules and staffing constraints often force a departure from these best practices.

The result may be that the code does not work because of improperly defined interfaces or coders working overtime and simply making mistakes.

Software quality measures are often collected to assist in project management.

Typical measures include defects identified, defects fixed, defect density, defect rates, and McCabe's cyclomatic complexity.

Software Cost Drivers - Complexity - Programming Language

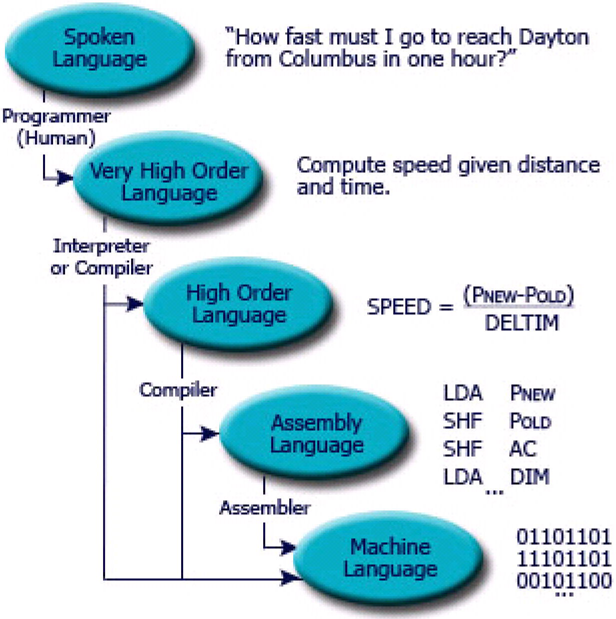
Computers operate based on sets of instructions contained in programs. These programs can be written in various languages. Within the same release, different CSCIs may be written in different languages. The figure to the right shows the various levels of programming languages. Select each oval to read more.

The lowest level represents the binary, or machine, instructions that a computer executes.

The next level represents assembly code; one assembly line of code (LOC) usually generates one machine instruction.

Next are the Higher-Order Languages (HOLs), closer to "normal English." Compared to assembly language, HOLs are easier to read and write, but they also consume more of a computer's resources. Third-generation languages such as FORTRAN, COBOL, ADA, and C are examples of HOLs.

Recently, there has been a trend toward Very High Order Languages (VHOLs), which more closely resemble human languages. VHOLs allow a person with little or no programming background to program a computer.



NOTE

Popup Text

Higher-Order Language (HOL)

A programming language that requires little knowledge of the computer on which a program will run, allows symbolic naming of operations and addresses, provides features designed to facilitate expression of data structures and program logic, and usually results in several machine language instructions for each program statement. Examples include Ada, BASIC, C, C++, COBOL, FORTRAN, PASCAL, and ALGOL. Also called Third Generation Language (3GL).

Note

As previously noted, data should be segregated by language, or different languages should be normalized to a common standard.

Lower-order languages may be more machine efficient, but they result in lower development productivity because they are more onerous to code.

Software Cost Drivers - Complexity - Application

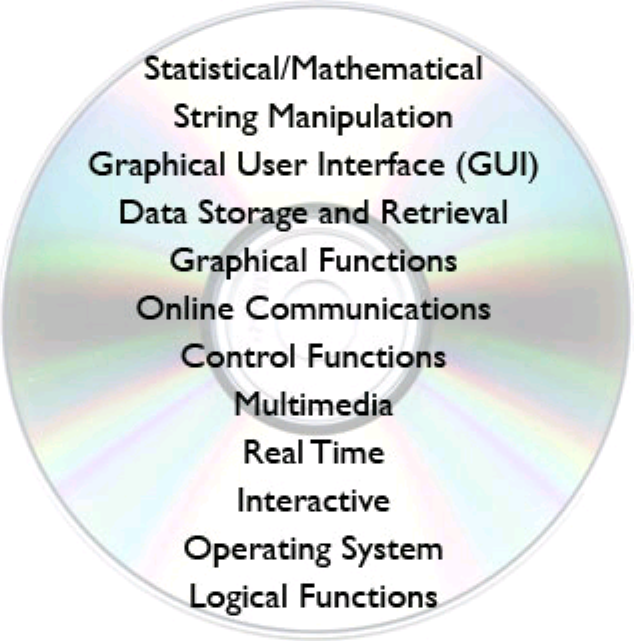
As previously noted, the correlated characteristics of Operating Environment, Platform, and Application drive the effort needed for software development.

Application is primarily related to the COCOMO EM for Product Complexity (CPLX), while Platform is related to all three Platform Factors.

For example, the hardware on which software will run can affect the way software is written. On weapons or space flight systems where space, weight, and power are at a premium, computer storage may be limited, forcing the developer to write code that is much more efficient than for a ground system with unlimited storage space.

Application refers to the use of the software, which may vary by CSCI. It reflects the degree of sophistication required in the component algorithms and relates back to the previous discussion of Quality.

Operating systems with the requirement for reliability and strict timing require more careful design and development and significantly more testing than simple math operations.



Statistical/Mathematical
String Manipulation
Graphical User Interface (GUI)
Data Storage and Retrieval
Graphical Functions
Online Communications
Control Functions
Multimedia
Real Time
Interactive
Operating System
Logical Functions

Select each of the titles on the graphic above to read more.

D

Long Description

A CD disk with interactive (clickable) listed topics overlaid. The topics and their descriptions are as follows:

Statistical/Mathematical - Simple math calculations, statistical routines, calculator functionality; processing time is not important.

String Manipulation - Text based manipulation, sorting, formatting, text-based input and output functionality, text processing, parsing, and sorting.

Graphical User Interface (GUI) - Interactive user interface; response time not critical; input sheets for an application, toolbar functionality, etc.

Data Storage and Retrieval - Reading and writing data to file or database, database management, database access control and security.

Graphical Functions - Data plotting, graphics creating and access, creation and manipulation of line charts, pie charts, etc.

Control Functions - Hardware control functions.

Multimedia - Information processing in a variety of formats (audio, video and text); response time very important but not critical.

Real Time - Machine to machine process communication with critical response time, protocol requirements strict, heavy interaction with hardware.

Interactive - Real time functionality with graphical capability; response must be immediate and visual, critical response time, strict protocol requirements, heavy interaction with hardware.

Popup Text

Operating System - Task and memory functions, response time critical, heavy interaction with hardware, strict timing and high reliability requirements.

Logical Functions - Algorithms containing complex mathematical logic such as smart air protection systems.

Software Cost Drivers - Complexity - ProRad Complexity

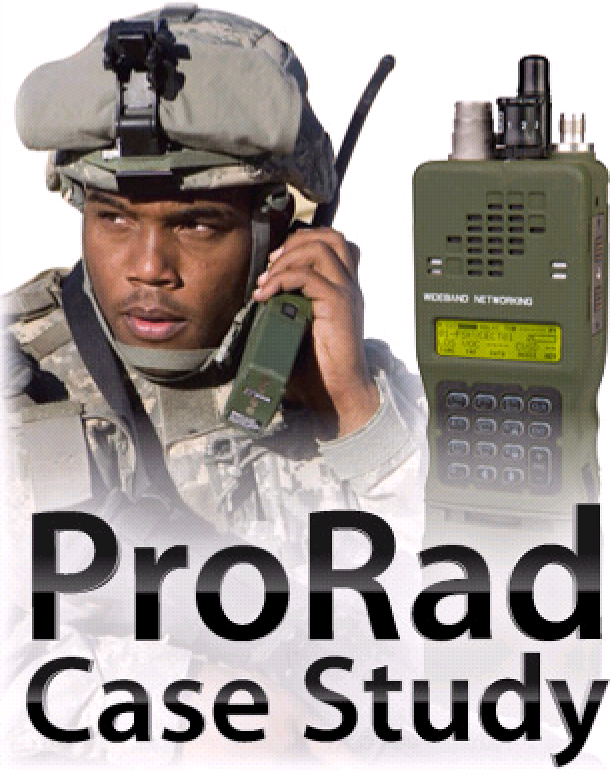
ProRad is a C4ISR system (specifically Communications) embedded into multiple platforms (ground, handheld, etc.).

Because ProRad will be crucial to communications in theater, it has fairly high reliability requirements.

While a software failure is not an immediate safety concern, inability to communicate can certainly lead to loss of life on the battefield.

Data to support the ProRad estimate should be drawn from similar programs.

For the waveforms and the operating software of the ProRad, all new software code was to be developed in a higher order language, such as C++.



Software Cost Drivers - Capability

The third main group of cost drivers is Capability, encompassing both the skill, experience, and expertise of the development team and the tools at their disposal.

Computer-Aided Software Engineering, or CASE, tools should automate the software development process, bringing integrity and efficiency.

Relevant experience on the part of the individual programmers includes experience coding in a particular language, experience developing a certain type of application, and experience in a particular development environment.

Software Cost Drivers - Capability, Cont.

It can be a challenge to assess the collective capability of a sizeable development team, and even more of a challenge to estimate what the capability of a development team will be early in acquisition, before a development contractor has even been selected.

Take care to avoid the "Lake Wobegon Effect," wherein all development teams are claimed to be above average.

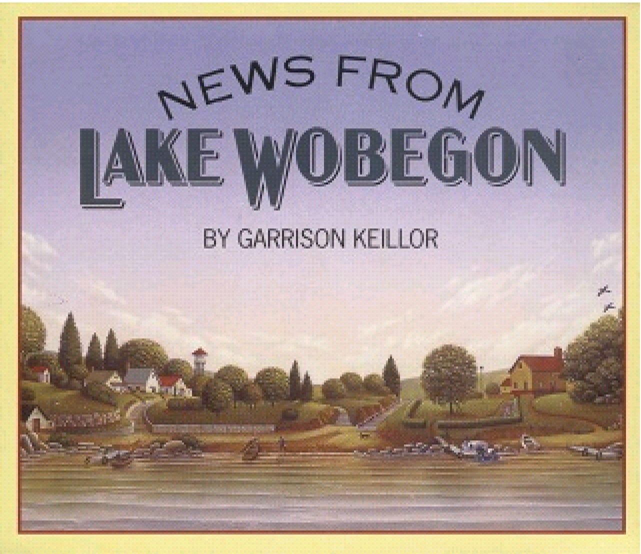
On the contrary, large projects tend to experience regression to the mean.

As you would expect, greater capability drives down cost.

Unfortunately, the effect is asymmetric, with low capability impacting productivity disproportionately more than high capability.

A mnemonic for this is "Bad programmers hurt you more than good programmers help you."

This is illustrated on the next page.



Software Cost Drivers - Capability - Capability Factors

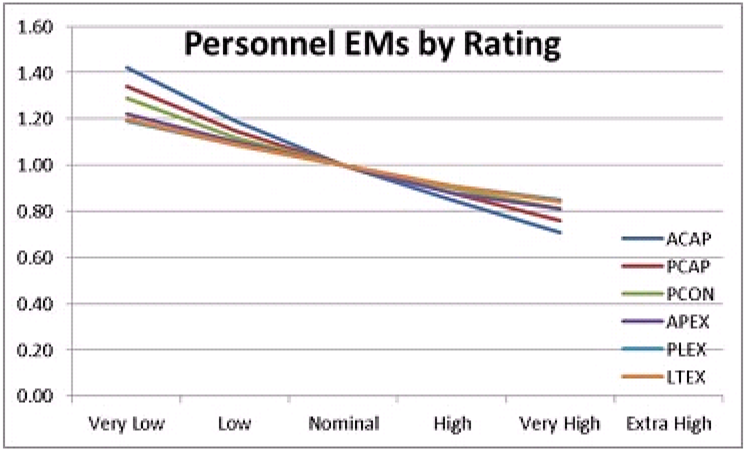
COCOMO II divides capability-related considerations into two groups: personnel factors and product factors. Select each tab to learn more.

Product Factors

- ACAP Analyst Capability
- PCAP Programmer Capability
- PCON Personnel Continuity
- APEX Applications Experience
- PLEX Platform Experience
- LTEX Language and Tool Experience

Whether or not you use COCOMO, the response illustrated here is typical.

Note that the effect is reversed from Complexity.

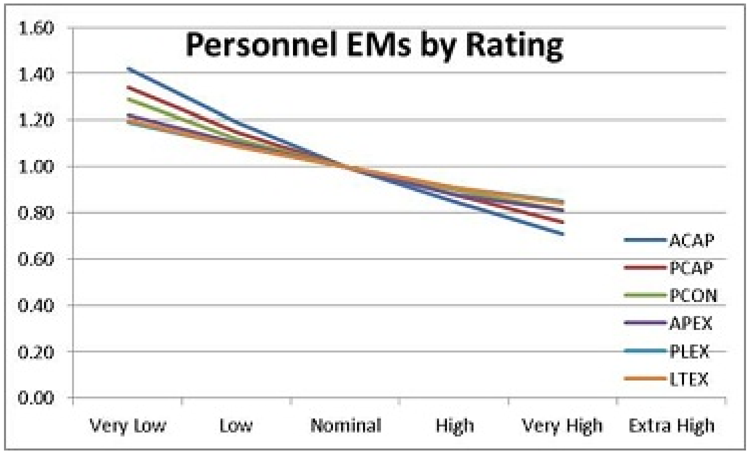


Select the image to enlarge.

Popup Text

Personnel Factors

- ACAP Analyst Capability
- PCAP Programmer Capability
- PCON Personnel Continuity
- APEX Applications Experience
- PLEX Platform Experience
- LTEX Language and Tool Experience

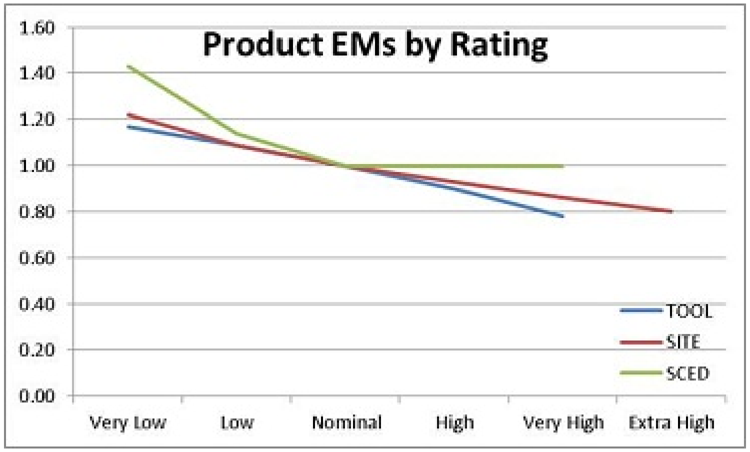


Whether or not you use COCOMO, the response illustrated here is typical.

Note that the effect is reversed from Complexity.

Product Factors

- TOOL Use of Software Tools
- SITE Multisite Development
- SCED Required Development Schedule



Software Cost Drivers - Capability - Capability Maturity Model - Integrated (CCMI)

In 1987, the Software Engineering Institute (SEI) at Carnegie Mellon University developed a methodology for assessing an organization's software processes. This became the framework for the Capability Maturity Model (CMM).

The CMM was initially developed for the Government to evaluate an organization's ability to perform software development and maintenance work on Government contracts. In 2001, SEI replaced CMM-SW with a suite of CMM Integration (CMMI) models.

The CMMI model for systems engineering and software (SE/SW) has five levels of software process maturity. These characteristics are typically demonstrated by organizations at that level.

Software Engineering Institute

Carnegie Mellon

Popup Text

Capability Maturity Model (CMM)

Originally developed by DoD’s Software Engineering Institute (SEI), the Software CMM (SW-CMM) was extensively used for disciplined software process improvement efforts. While references to it are still encountered, a more comprehensive and integrated process model—the Capability Maturity Model Integration (CMMI)—has replaced the SW-CMM. The SW-CMM was retired effective Dec. 31, 2005, and all SW-CMM ratings expired Dec. 31, 2007.

Capability Maturity Model Integration (CMMI)

Derived from the now-retired Software Capability Maturity Model (SW-CMM), the CMMI integrates a number of disciplines into a unified model useful for process improvement. Three domain variations (so-called “CMMI constellations”) of the CMMI exist: one for development organizations (CMMI-DEV), one for acquisition organizations (CMMI-ACQ), and one for service-type organizations (CMMI-SVC). All the models share a common set of core processes with additional processes added as appropriate for the domain. While the CMMI models can provide ratings on a numerical scale (5 being the highest), DoD’s preference is to use them primarily in a process improvement role, de-emphasizing numerical ratings. The Software Engineering Institute (SEI) manages the three CMMI product suites.

Software Cost Drivers - Capability - CMMI, Cont.

Organizations that have implemented software process improvements resulting from CMM and CMMI evaluations have generally achieved many benefits, including significant cost savings and significant return on investment (ROI).

In addition, many Government buying activities want contractors to be certified at a particular level before considering them for contract award.

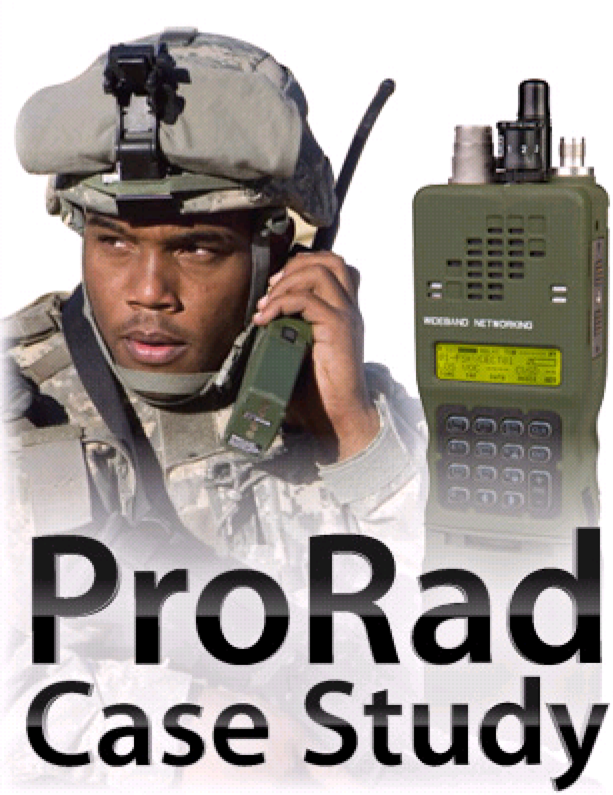
As with most assessments, the certification does not magically make an organization more productive, but rather it is the application of principles and processes that both lead to greater efficiency and support the CMMI rating, which makes CMMI level a reasonable indicator of productivity.

Software Cost Drivers - Capability - ProRad Capability

The Joint ProRad Program Office assumed that software would be developed by "programmers with nominal to high skill levels and experience, using object-oriented skills and practices."

We should be worried that this assumption creates a risk that developers will not be able to staff such an above-average team for this significant effort.

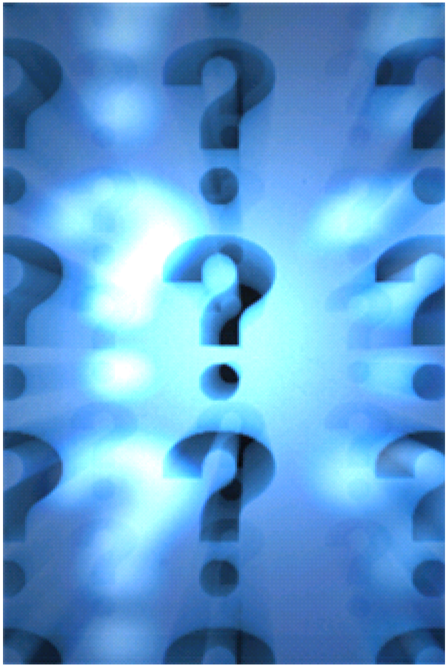
No assumptions were made about the maturity of the vendors' software development processes, such as a particular CMMI rating.



Knowledge Review

Wee Software Coding, Ltd., is a small business with a $30M subcontract for software development on an ACAT I program.

TRUE or FALSE: Because of the relatively small dollar value of the subcontract, they do not have to submit an SRDR.



□ TRUE

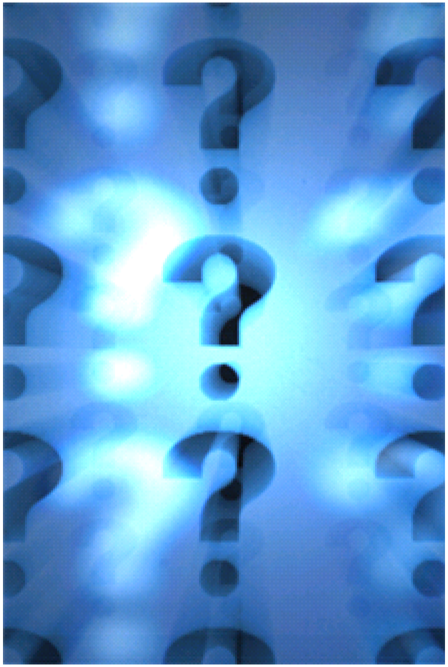
☑ FALSE

Check Answer

The answer is **FALSE**. Any software development effort of $20M for an ACAT I program requires an SRDR submission.

Knowledge Review

The Super-Duper Coding Group produced a 10K SLOC CSCI using 5,000 labor hours. Which of the following would you expect for their productivity in developing a 20K SLOC CSCI?



- □ Less than 10,000 labor hours
- □ 10,000 labor hours
- □ More than 10,000 labor hours
- ☑ Less than 2 SLOC/hour
- □ More than 2 SLOC/hour

Check Answer

Less than 2 SLOC/hour would be expected for the productivity in developing a 20K SLOC CSCI. Demonstrated productivity is 2 SLOC/hour, but productivity is generally expected to decline for larger efforts due to diseconomies of scale.

Knowledge Review

For which of the following factors would a rating of Very High translate to less development effort than a rating of Nominal?

☐ Required Software Reliability

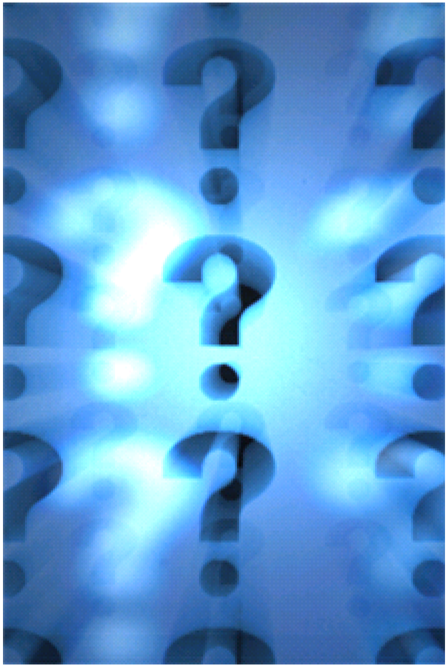
☐ Database Size

☐ Product Complexity

☐ Platform Volatility

☑ Personnel Continuity

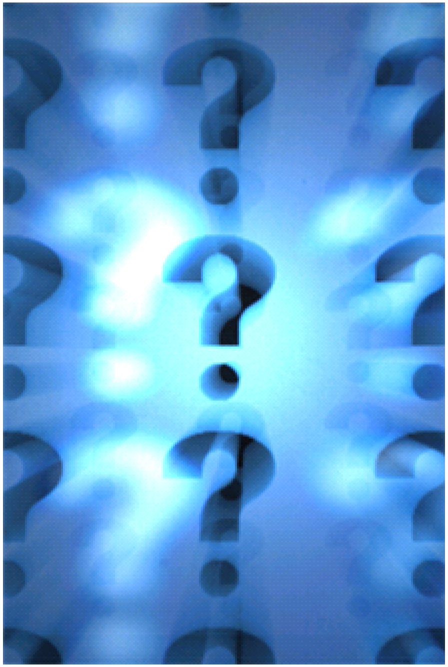
Check Answer



Very High rating in **Personnel Continuity** would translate to less development effort than a rating of Nominal. The first four factors are all related to Complexity, which drives up development effort. Higher Personnel Continuity, which is in the Capability family, would drive down development effort.

Knowledge Review

A software development effort is to produce two CSCIs of 8K SLOC. The first is to be written from scratch. The second is based on an existing CSCI of the same size, and the effort will require 60% recode and 80% retest. What is the combined ESLOC for the two CSCIs?



□ 8,000 SLOC

☑ 11,360 SLOC

□ 13,600 SLOC

□ 16,000 SLOC

□ 16,000 SLOC

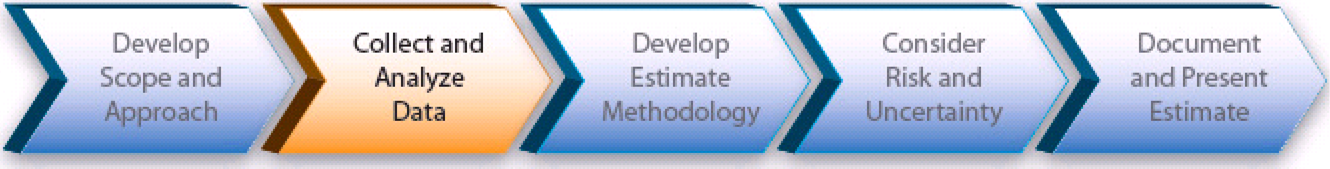
Check Answer

11,360 SLOC is the combined ESLOC for the two CSCIs. For the first CSCI, 8K DSLOC translates into 8K ESLOC, because it's 100% new. The efficiency factor for the second CSCI (assuming 40%/30%/30% Design/Code/Test) is 0.4 * 0% + 0.3 * 60% + 0.3 * 80% = 0.42, which when multiplied by 8K SLOC gives 3,360 SLOC. The sum of the two gives the correct answer above.

Summary

This completes the Collect and Analyze Data lesson. In this lesson you learned:

- Software Resource Data Reports (SRDRs) and other sources of actual effort, sizing, and other data from completed programs are crucial to support reliable software estimates.
- Software data must be normalized in order to be made comparable, including for a consistent set of activities (effort data) and to account for differences in code counters and programming languages (sizing data).
- While development productivity (lines of code per hour, for example) is a useful and commonly cited software measure, it is not the best way to estimate effort.
- Size is the primary driver for developed software, along with Complexity of the code and Capability of the development team and its tools.


Develop Scope and Approach
Collect and Analyze Data
Develop Estimate Methodology
Consider Risk and Uncertainty
Document and Present Estimate

D

Long Description

Graphic illustrates the steps of the Cost Estimating process. The steps from left to right are:
Develop Scope and Approach, Collect and Analyze Data (highlighted), Develop Estimate Methodology, Consider Risk and Uncertainty, and Document and Present Estimate.

Lesson Completion

You have completed the content for this lesson.

To continue, select another lesson from the Table of Contents on the left.

If you have closed or hidden the Table of Contents, click the Show TOC button at the top in the Atlas navigation bar.