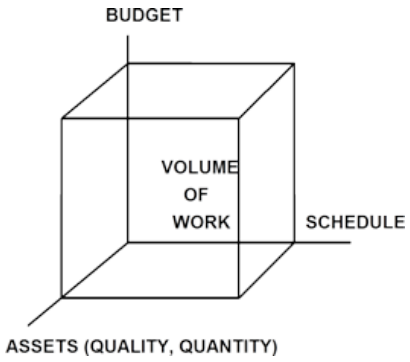
Software Cost, Schedule and Quality

As mentioned earlier, cost and schedule must be aligned to help produce a quality product as shown in the graphic below.

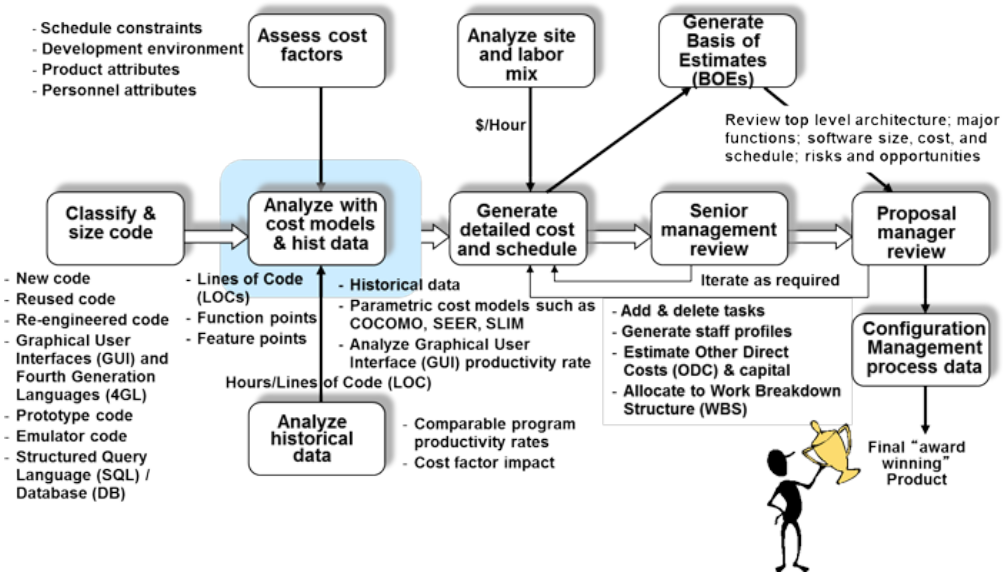


Quick Labor estimate and Rule of Thumb

This module builds on the previous modules and further discusses how to estimate software cost and schedule. Remember the labor and schedule estimates are based on:

- Software Requirements Analysis
- Software Design, Code and Unit Testing
- Software Integration and Test (SWIT)

But the software manager must also take into account and increase the estimate by some percentage to support to Systems Engineering Requirements Analysis and Design; planning and preparation to setup Software Development Environment; support to the Independent Test Teams; Software Configuration Management; management, clerical, and software process (metrics, Software Engineering Process Group (SEPG)); the development system, administration, and maintenance (operating system, tools, and COTS); and software licenses and COTS for the Target System. As we estimate the cost and schedule, remember the software estimation process.



Quick Labor and Rule of Thumb Schedule Estimates

Assumptions made during the size and effort estimation activity must be documented. For example, let's agree on the following assumptions:

- Our standard corporation tools will be used; this is our computer aided software engineering (CASE) environment
- Our development system will support one workstation (or terminal) per software engineer.
- Target hardware, COTS, Government Furnished Equipment (GFE), Customer Furnished Equipment, and reuse software will be available when needed.
- Our system will have normal software reliability; it will not be a man-rated system, one that supports human activities (e.g., NASA shuttle system).
- Our system will not need special security requirements (e.g., multi-level security activities).
- We will not have special documentation or reviews.
- Our system will have an adequate development environment in terms of quality and speed.
- We will assume a reasonable schedule and appropriate budget constraints.

Wouldn't it be great if all systems' assumptions were like these! With these assumptions, we can apply the Quick Labor Estimate which gives you the staff months needed to complete your activity and the Rule of Thumb Schedule estimate which gives you the likelihood of success (or the percent of time) to successfully implement this program in the time frame.

To determine the quick labor estimate, we need to first determine how many hours per month each person is working on average. Let's assume an average of 160 hours / staff month. To get this, an employee works on average:

40 hours/week * 52 weeks/year = 2080 hours/year

Minus 2 weeks' vacation (80 hours) = 2000 hours/year

Minus 10 holidays (80 hours) = 1920 hours/year

Some companies subtract out hours for sick leave or additional leave which further reduces the average number of productive hours/staff month (SM).

Productive hours/staff year = **1920 hours/year**

Productive hours/SM = 1920 hours/12 months = **160 hours/SM**

Note: Companies often use the 5/4/4 accounting cycle over a three month period, that is, the first month has five (5) accounting weeks, and the second and third accounting months have four (4) accounting weeks each. Thus the accounting month and calendar month are not exactly aligned. For example, the first accounting month may run from January 1 to February 4, the second account month is from February 5 to March 4, and the third accounting month is from March 5 to April 1; thus, over a three month period 13 weeks are included. Using this method the number of hours/staff month is more granular and may vary each month. For the purpose of this course, we will generally use 150 or 160 hours/SM.

Recall the definition for productivity in terms of LOC:

LOCs = Productivity * unit of time or

Productivity = LOC/unit of time and in this case LOC/hour

The **Quick Labor Estimate** or Effort in Staff Months (SM) is defined as:

Effort = Number of LOC estimated /(Productivity * (Productive hours/SM))

= Number of LOC estimated / ((LOC/hour) * (160 hours/SM)) = Effort in SM

The **Rule of Thumb Schedule Estimate**, a widely used formula, is referred to as the Time for Development (T_{Dev}) or Time for Labor (T_L) and is defined as:

Formula	Percentile
$T_L = 2.0 * (SM)^{(1/3)}$	10%
$T_L = 2.5 * (SM)^{(1/3)}$	50% (Moderate Risk)
$T_L = 3.0 * (SM)^{(1/3)}$	90%

That is you take the cube root of the resulting Quick Labor SM estimate and multiply it by a constant to give you the chance of success to successfully implement this program in this time frame. The general idea that schedule is a cube-root function of effort is universally accepted by estimation experts, but...

Beware the Impossible Region: When the constant becomes too small (below 1.875) and therefore the time to develop becomes too short, there is a 0% chance of successfully implementing the program.

$T_L = 1.875 * (SM)^{(1/3)}$	0%

Note: the Rule of Thumb Basic Schedule Equation is not intended for estimation of small projects or late phases of larger projects and another estimation tool should be used for these circumstances.

Example:

You are tasked to estimate the effort needed and the time to develop a software system that is estimated to have 20,000 LOC. You are able to assume moderate risk and the typical productivity for your team (or company) is 15 LOC/day.

Effort = Number of LOC estimated / (Productivity * (Productive hours/SM))

20,000 LOC / (15 LOC/day *160 hours/SM)

Since the productivity is shown in terms of LOC/day, you must convert it to LOC/hour by dividing by 8 to ensure equal terms.

Thus the equation becomes:

Effort = 20,000 LOC / ((15 LOC/day) / (8 hours/day) *160 hours/SM) = 66.67 SM = ~ 67 SM

$T_L = 2.5 * (SM)^{(1/3)} = 2.5 * (66.7)^{(1/3)} = 2.5 * 4.05 = 10.1 \sim 10$ months

That is this 20,000 LOC project will take approximately 67 SM. Fifty percent of the time, 67 SM can be successfully completed in the 10 month period. If you want to lower the level of risk, increase the constant. That is:

$T_L = 3 * (SM)^{(1/3)} = 3 * (66.7)^{(1/3)} = 3 * 4.05 = 12.2 = \sim 12$ months

Now 90% of the time, 67 SM can be successfully completed in the 12 month period.

Remember this is just an estimate and there are multiple assumptions. The Quick Labor and Rule of Thumb Schedule Estimates should only be used as single data points. To determine the best estimates, use these results with other estimating techniques, then analyze and justify the results.

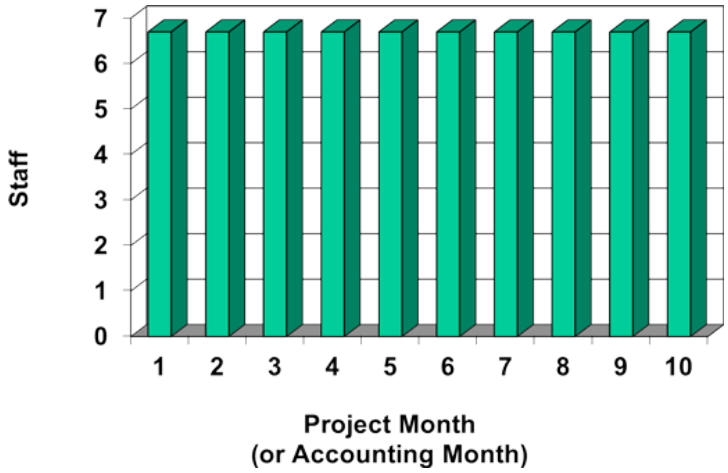
Staffing Tips

Level Loaded Staffing versus Ramp Up and Down Staffing

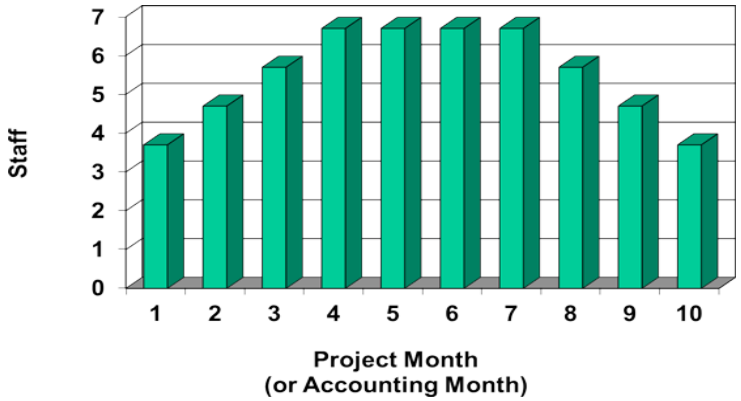
In our example the effort estimate is 67 staff months (SM) and schedule estimate is 10 calendar months. Full Time Equivalent (FTE) is a method used to measure a staff member's participation on a project. Thus,

67 SM / 10 months = 6.7 staff or 6.7 FTE

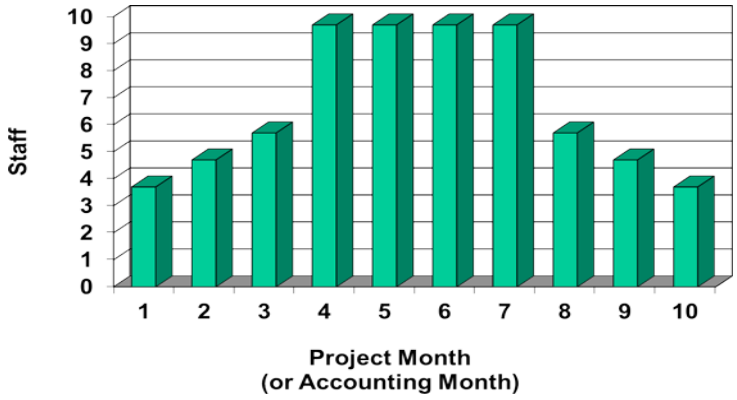
The **level loading staffing approach** uses 6.7 team members from project start to finish. At least one staff member is part-time working 70% of their time on this project.



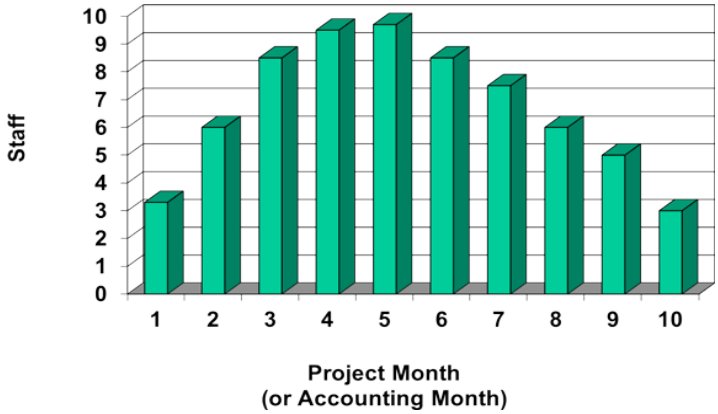
An alternative to level loading staff is to transition staff onto the program (ramping up to start of code) and transition them off as you begin integration. Transitioning staff on and off reflects the following profile:



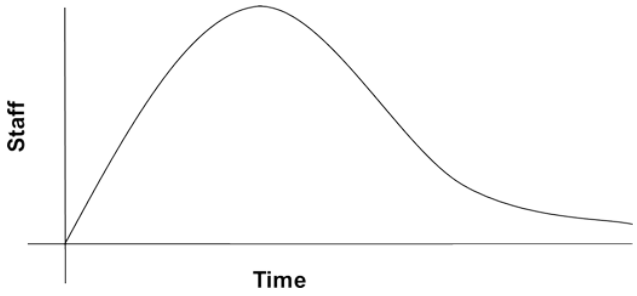
However, this profile does not fulfil the effort estimation of 67 SM. We are short 3 SMs in months 1 and 10, 2 SMs in months 2 and 9, 1 SM in months 3 and 8; that is we are short a total of 12 SMs. Hence we must alter the profile to expend 12 more SMs during months 4 through 7.



But this creates an increase from month 3 to month 4 of four staff personnel (from 5.7 to 9.7 staff) nearly doubling the size of the project. The process is iterated until you smooth the profile and comply with resource availability (or hiring capacity) within your organization. You attempt additional smoothing keeping the total staff months at 67.



The current literature suggests ramping up and down with a profile that approximates a Rayleigh distribution, which is similar to a bell curve skewed to the left.



Rayleigh Curve

It is important to determine your staffing strategy for each project; however, take into account that there are pros and cons with each approach.

Level Loaded Staffing Strategy	Ramp Up & Down Staffing Strategy
All team members are on project from start, allowing junior members to gain experience with analysis, design, integration and testing.	Senior staff perform the front and back end work for which they are most qualified including analysis, design, integration and testing.
Junior team members are more prepared for start of code, unit test, and integration.	Senior staff are not slowed down by effort required to train and mentor junior staff.
Typically it is difficult to staff all positions instantaneously, especially for larger programs.	There is the opportunity to search for resources over a longer period of time.
There may be some activities more suited for junior staff during early phases of program, for example, prototyping.	There is the opportunity to phase personnel to other efforts as the program begins to wind down.
There may be some activities more suited for junior staff during final phases, for example, software debugging.	There is more front end planning effort required with modern methods and languages, for example with using Object Orientation and C++.

Skill Mix Profile

The skill mix profile must also be considered when forming your team. Typically the planning and requirements phases up front and the integration and testing phases at the back end require more experienced personnel, while the middle development phase uses less experienced staff. The staffing ratios will fluctuate depending on the life cycle phase but in general you should typically assume there is about 10% senior staff, 40% mid-level staff, and 50% junior staff on a program. The chart defines the typical staff profile mix. In most organizations, a Master's Degree is equal to two years' experience.

Job Code	Job Description	% of Staff

Job Code	Job Description	% of Staff
5	• Technical leader or advisor leading a large team • BS plus 12 or more years' experience • MS plus 10 or more years' experience	10
4	• Technical leader leading a small team • BS plus 9 or more years' experience • MS plus 7 or more years' experience	20
3	• Individual contributor • BS plus 5 or more years' experience • MS plus 3 or more years' experience	20
2	• Individual contributor • BS plus 3 or more years' experience • MS plus 1 or more years' experience	25
1	• Entry level programmer • BS plus minimal experience • MS and no experience required	25

Total Labor Cost Comparison

It is important to understand total labor cost and its variables.

Total Labor Cost = $/hour * hours

$/Hour is based on employee salary which is loaded with:

- Fringe (or benefits): represents approximately 40%
- Overhead (or building & utilities): represents approximately 60%
- General and Administrative (G&A) (or corporate staff): represents approximately 10%

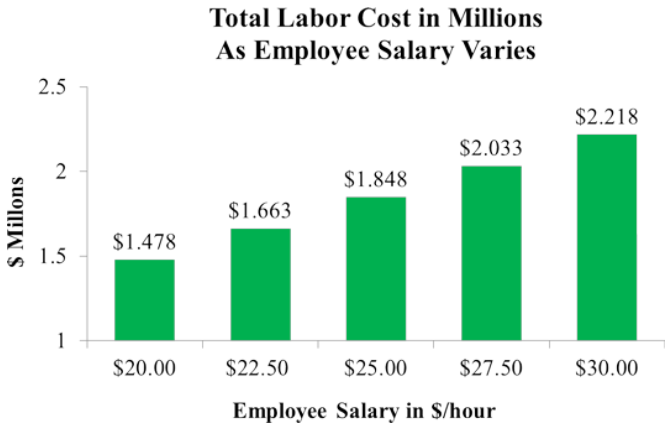
Salary burdens compound so one unit of employee salary becomes: 1 * 1.4 * 1.6 * 1.1 = 2.5 cost to the company

Hours are based on the:

- Productivity rate (LOC/day or LOC/hour), and
- LOC

As employee salary, fringe, overhead, G&A, productivity rate, and LOC vary, the total labor cost varies. Let's assume employee salary is $25/hour, fringe is 40%, overhead is 60%, G&A is 10%, your project's size is 30,000 LOC, and productivity rate is 8 LOC/day. The following graphs illustrate the effects as each variable changes.

The effect of **Employee Salary** on Total Labor Cost as the cost of employees' salary varies plus/minus 20% from $20/hour = $41.6K/year to $30/hour = $62.4K/year is shown in this graph.

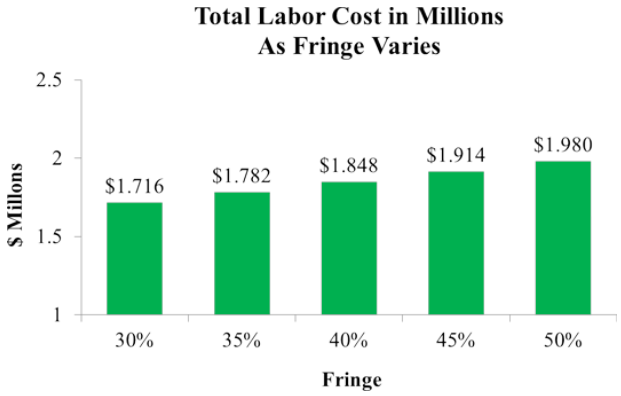


As you vary employees' salary, these results vary around $1.85M as seen in the following equation:

Total Labor Cost = $/hour * hours
= (Employee Salary * Fringe * Overhead * G&A) * (LOC / Productivity)
= ($25/hour * 1.4 * 1.6 * 1.1) * (30K LOC / (8 LOC/day / 8 hour/day))
= ($61.6/hour) * (30K hours)
Total Labor Cost = $1,848,000 = $1.848M

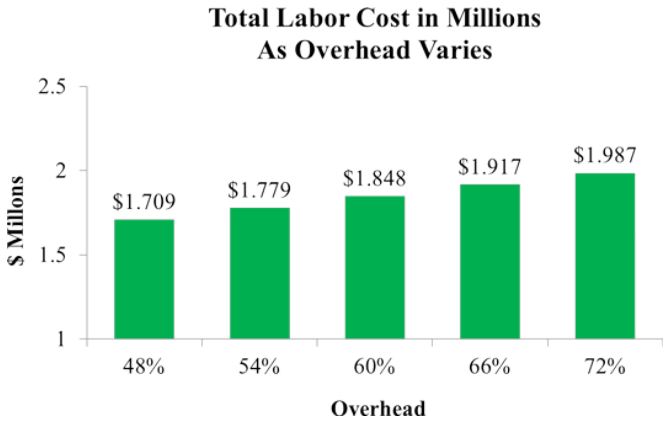
Thus as the employees' salary increases from $20/hour while holding the variables the same, total labor cost increases.

The effect of **Fringe** on Total Labor Cost as the cost of fringe varies plus/minus 25% from 30% to 50% is shown in this graph.



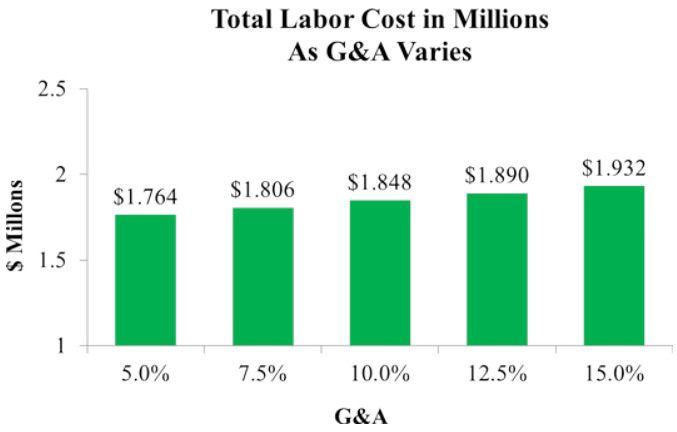
As the fringe increases and the other variables remain the same, the total labor cost increases. It is for this reason that many companies work hard to reduce the cost of fringe benefits.

The effect of **Overhead** on Total Labor Cost as the cost of overhead varies plus/minus 20% from 48% to 72% is shown in this graph.



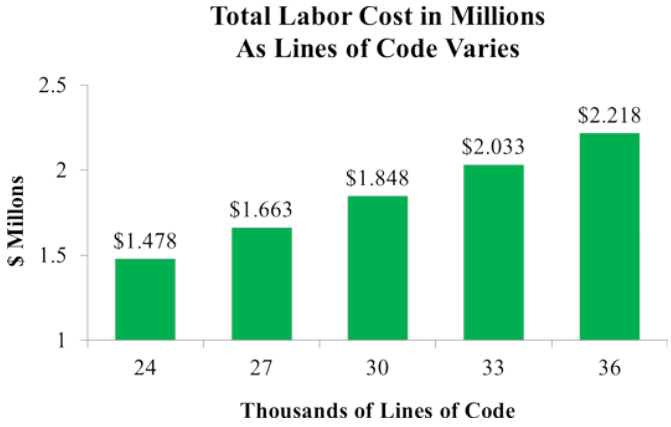
Similarly as the overhead costs increase and the other variables remain the same, the total labor cost increases. Again many companies work hard to reduce overhead costs.

The effect of **G&A** on Total Labor Cost as the cost of G&A varies plus/minus 50% from 5% to 15% is shown in this graph.



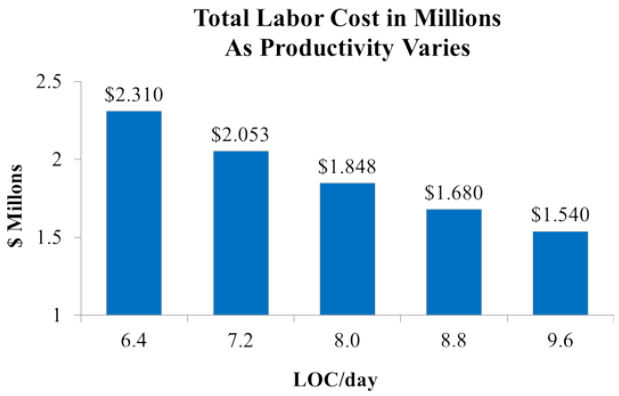
As the G&A costs increase and the other variables remain the same, the total labor cost increases. Again many companies work hard to reduce G&A costs.

The effect of LOC on Total Labor Cost as the number of LOC varies plus/minus 20% from 24,000 LOC to 36,000 LOC is shown in this graph.



As size of the program increases in terms of LOC and the other variables remain the same, the total labor cost increases.

Lastly the effect of **Productivity Rate** on Total Labor Cost as the productivity rate varies plus/minus 20% from 6.4 LOC/day to 9.6 LOC/day is shown in this graph.



Not surprisingly as employees become more productive, the total labor cost decreases. For this reason many companies implement process improvement plans.

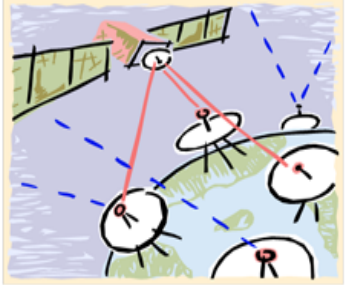
It is employee salary, the number of LOC, and the productivity rate that have the largest impacts on total labor cost. For these reasons, software program/project managers work hard to ensure that they properly staff their program or project, build a quality product, increase productivity, and keep costs low, while ensuring that they deliver the product and its component on time. You should now have a better appreciation of the implications when a single variable changes, but in reality, it is not single variables, but multiple variables that are changing simultaneously making the total cost of labor very complicated.

Software Estimation Methods

Software Estimation Methods

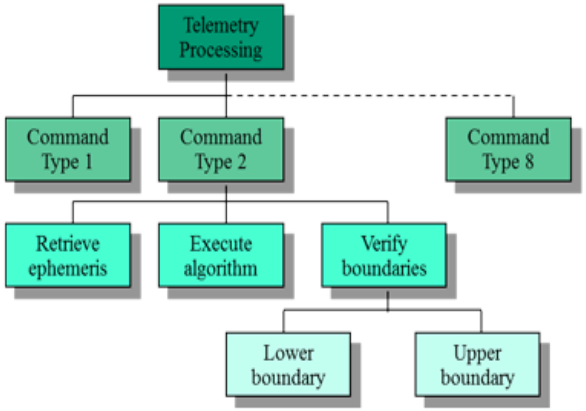
- Lines of Code (LOC)
- Function Points / Feature Points
- User Displays / Panels
- Classes / Modules

Lines of Code



The Satellite Command & Control example:

- Satellite command and control consists of:
  - Telemetry processing
  - Hardware control
  - User Interface
  - Off-line processing
- Decompose each functional area (e.g., telemetry processing)



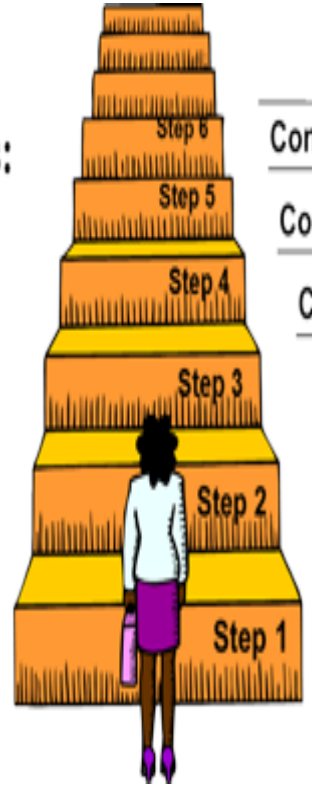
Could assume that telemetry processing requires 18,000 LOCs or could further decompose into command types and decompose each command type into associated processing

- Lower boundary = 200 LOCs
- Upper boundary = 300 LOC
- Verify boundaries = 500 LOCs
- Execute algorithm = 600 LOCs
- Retrieve ephemeris = 450 LOCs
- Command type 2 = 1550 LOCs
- Command type 1 = 950 LOCs
- etc.

Function Points

Function Points Process

Six step process:



Convert to LOCs (optional)

Compute adjusted function points

Calculate complexity adjustment factors

Evaluate environmental factors

Estimate weight complexity

Count function points

1. Count Function Points

Group	Definition	Counting Criteria	If Missed
Outputs (Transactional)	Items of business info processed by the system for the end user (e.g. a report generated by software)	Count each unique user data control output procedurally generated that leaves the application boundary. Considered unique if it has a different format or processing logic	4–7 points
Inputs (Transactional)	Items of business data sent by the user to the system for processing and add/ delete/modify (e.g. add new customer)	Count each unique user data or control input that enters the application boundary and updates a logical internal file, data set, table, or independent data item	3–6 points
Inquiries (Transactional)	Inquiries into a DB or master file that look for specific data, use simple keys, require immediate response, and do not update	Count each unique input/output combination in which the on-line user-defined input causes & generates an immediate on-line output. Those functions failing the key-response-update test should be listed under inputs and outputs (queries)	3–7 points
Logical Files (Data)	Data stored for an application, as logically viewed by the user	Count each major logical group of user data or control info permanently maintained within the application boundary and available to users via inputs, outputs, inquiries, and interfaces (I/Fs)	24 points Is a file missing? This is a big penalty.
External Interface Files (Data)	Data stored elsewhere by another application but used by the one under evaluation	Count each major logical file within the application boundary that is sent to, shared with, or sends to another application. Count each flow of data or control info in each direction with each external entity	16 points What if you miss an I/F?

2. Estimate Weight Complexity

	Simple	Average	Complex	Function Points
# Outputs	___ x 4 =	___ x 5 =	___ x 7 =	
# Inputs	___ x 3 =	___ x 4 =	___ x 6 =	
# Inquiries[1] **Outputs** **Inputs**	 ___ x 4 = ___ x 3 =	 ___ x 5 = ___ x 4 =	 ___ x 7 = ___ x 6 =	
# Files	___ x 7 =	___ x 10 =	___ x 15 =	
# Interfaces	___ x 5 =	___ x 7 =	___ x 10 =	
			Total:	

Note [1]: select the greater of the output and input portion

3. Evaluate Environmental Factors (F_i)

1. Data communications
2. Distributed data or processing
3. Performance objectives
4. Heavily used configurations
5. Transaction rate
6. On-line data entry
7. End user efficiency
8. On-line update
9. Complex processing
10. Reusability
11. Conversion installation ease
12. Operational ease
13. Multiple site usage
14. Facilitate change

Use for all Environmental Factors Except "Reusability"

Adjustment Factor Value	System Influence	% Affects or Required by the Application
0	None	0%
1	Minor (insignificant)	1–20%
2	Moderate	21–40%
3	Average	41–60%
4	Significant	61–80%
5	Strong Throughout	81–100%

Use for "Reusability"

Adjustment Factor Value	System Influence	% Affects or Required by the Application
0	None	0%
1	Minor (insignificant)	1–20%
2	Moderate	21–30%
3	Average	31–40%
4	Significant	41–50%
5	Strong Throughout	> 50%

Adjusted Function Points

4. Calculate Complexity Adjustment Factor (CAF)

- Total Degree of Influence (N) = ΣF_i (Sum of the 14 environmental factor values)
- Complexity Adjustment Factor (CAF) = 0.65 + (.01 x N)

5. Compute Adjusted Function Points (FP)

- $FP_{adjusted} = FP_{unadjusted}$ x CAF

6. Convert to Lines of Code (LOC) if needed

Lines of Code to Function Point Translation Table

Language	Level	Avg LOC/ AFP
Basic Assembler	1	320
Macro Assembler	1.5	213
C	2.5	128
ALGOL	3	105
COBOL	3	105
FORTRAN	3	105
JOVIAL	3	105
Mixed Languages (default)	3	105
Other Languages (default)	3	105
Pascal	3.5	91
RPG	4	80
MODULA-2	4.5	80
PL/1	4.5	80
Ada	4.5	71
BASIC	5	64

Language	Level	Avg LOC/ AFP
FORTH	5	64
LISP	5	64
PROLOG	5	64
LOGO	5.5	58
English-based languages	6	53
Data-based languages	8	40
Decision support languages	9	35
APL	10	32
Statistical languages	10	32
OBJECTIVE-C	12	27
SMALLTALK	15	21
Menu-driven generators	20	16
Database query languages	25	13
Spreadsheet languages	50	6
Graphic Icon languages	75	4

Note: The LOC/AFP values provided in the table may vary based on the factors of the development environment and the specific version of the programming language used. The estimator should use the numbers with caution based on experience. *Function Point Analysis*, Brian J. Dreger

Cost Models

Cost models are used to estimate manpower and schedule based on the software project profile. Three major categories are used for estimation techniques in the computer models:

- Regression-based models are derived and analyzed from historical data to express mathematical relationships among project variables. Examples are Barry Boehm's COCOMO and Ray Kile's REVIC.
- Empirical-based models are derived from observing the past performance of projects' from similar domains. Capers Jones' Checkpoint is an example.
- Theory-based models are based on underlying theoretical considerations for software development processes. These models are the most expensive as they require extensive research. Examples are Larry Putnam's SLIM and Lockheed Martin's PRICE-S.

Typical inputs to the models include software size attributes, product attributes, computer attributes, personnel attributes, and project attributes. Typical outputs from the models are labor cost in total staff months (effort), development time in accounting months (schedule), and the cost and schedule by activity. This chart shows a comparison of four cost models and the parameters required for that tool.

Model Parameters	SLIM	Price-S	COCOMO	Jensen
Size Attributes				
Source Instructions	✔		✔	✔
Number of routines		✔		
Number of data items			✔	
Documentation				✔
Number of personnel	✔			✔
Product Attributes				
Type	✔	✔		
Complexity	✔	✔	✔	✔
Language	✔			
Reuse	✔	✔	✔	✔
Required reliability	✔	✔	✔	✔
Computer Attributes				
Time constraint	✔	✔	✔	✔
Storage constraint	✔	✔	✔	✔
Hardware configuration		✔		
Development		✔	✔	✔
Personnel Attributes				

Model Parameters	SLIM	Price-S	COCOMO	Jensen
Personnel capability	✔	✔	✔	✔
Personnel continuity				
Hardware experience	✔	✔	✔	✔
Applications experience	✔	✔	✔	✔
Language experience	✔	✔	✔	✔
Project Attributes				
Tools and techniques	✔	✔	✔	✔
Requirements definition				✔
Requirements volatility		✔	✔	✔
Schedule	✔	✔	✔	✔
Security				
Computer access	✔		✔	✔
Travel/rehosting multisite		✔		✔
Support software maturity			✔	
Total	16	18	17	20

You will certainly note that each tool requires different parameters. You should also notice that no tool uses personnel continuity or security. Perhaps as future tools are created, these variables will be considered.

COnstructive COst MOdel

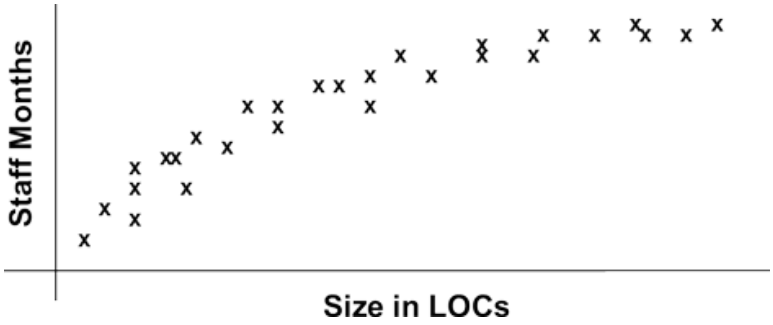
The **Co**nstructive **Co**st **Mo**del or COCOMO as it is generally referred to, is the most widely used software cost and schedule estimation tool still today. Barry Boehm, when he was working at TRW, developed COCOMO in the mid-1970s. It is non-proprietary; various versions can be found on the internet. As with all models, it is critical to know what activities are covered by COCOMO; it addresses:

- Software Design
- Code and Unit Test
- Unit and Computer Software Component (CSC) Integration into the larger Computer Software Configuration Items (CSCI)
- CSCI Test (or System Test) and Sell-off

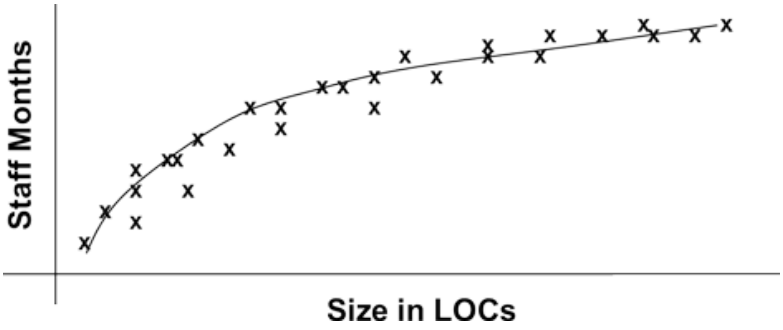
Note that software requirements analysis is not covered as COCOMO assumes that this activity is a system engineering function.

In 1987, Ray Kile updated COCOMO and developed the **Rev**ised **I**ntermediate **C**OCOMO (REVIC) to support Ada, software reuse, and incremental development model. Corporate experience has shown that the REVIC equations result in lower staffing levels and longer schedules. The original COCOMO and the even later revised COCOMO II are closer to company's experience. Some organizations have checked results with a Function Point model or other methods. For this reason, we will use COCOMO in its original form in our examples.

Barry Boehm collected data from 63 completed projects. He counted the LOC and obtained labor hours from the TRW time accounting system. He assumed 160 work hours in a month.



Then he performed Regression Modeling analysis to fit a curve to the data collected for 63 projects producing equations with coefficients and exponent to predict future staff months (or effort) based on estimated size.



To use the COCOMO tool, it requires the user to select a development mode for the project and one of the available models.

There are three development modes: Organic, Semi-detached, and Embedded:

1. Organic uses a small team in a familiar environment and with a familiar application. Generally the team is working to a set of less-than-rigid requirements, e.g., data reduction, scientific models, business models, simple inventory control, or simple production control.
2. Semi-detached is the intermediate mode. It has a mixture of organic and embedded mode characteristics; for example, most transaction processing systems, ambitious inventory system, ambitious inventory control, and ambitious production control would be semi-detached.
3. Embedded has tight constraints on software. There are complex interactions with hardware and the operating environment, the cost of validation and change is high.

Embedded mode includes unique applications, e.g., avionics, large and complex transaction processing, aircraft on-board collision avoidance system, and complex command and control systems.

The user also chooses one of three available models: Basic, Intermediate, and Detailed:

4. The Basic model uses a single equation for selected mode.
5. The Intermediate introduces multipliers (or project characteristics) which are incorporated into the cost driver equations.
6. The Detailed adds the phases of the project and subsystems to use the multipliers in the cost driver equations for each phase of project and/or for each subsystem.

The equations are:

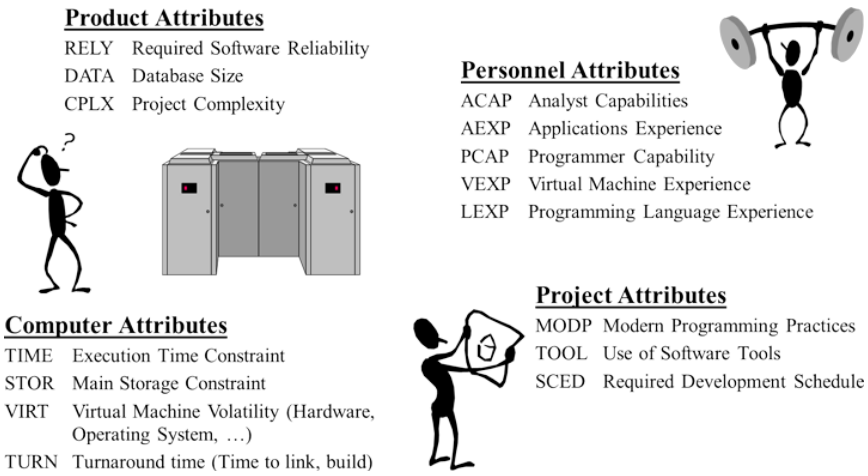
Model	Mode	Effort	Schedule
Basic	Organic	Semi-detached	Embedded
	$SM = 2.4 (KDSI)^{1.05}$	$SM = 3.0 (KDSI)^{1.12}$	$SM = 3.6 (KDSI)^{1.20}$
	$T_{DEV} = 2.5 (SM)^{0.38}$	$T_{DEV} = 2.5 (SM)^{0.35}$	$T_{DEV} = 2.5 (SM)^{0.32}$
Intermediate & Detailed	Organic	Semi-detached	Embedded
	$SM = 3.2 (PEM) (KDSI)^{1.05}$	$SM = 3.0 (PEM) (KDSI)^{1.12}$	$SM = 2.8 (PEM) (KDSI)^{1.20}$
	$T_{DEV} = 2.5 (SM)^{0.38}$	$T_{DEV} = 2.5 (SM)^{0.35}$	$T_{DEV} = 2.5 (SM)^{0.32}$

SM: Staff Months
PEM: Product of Effort Multipliers (covered next)
KDSI: Thousand Delivered Source Instructions
T_{DEV}: Time for Development

Note that the same equations are used for both the Intermediate and Detailed models. For the Detailed model, you would use these equations for each phase or each major subsystem. The user then runs the model, assesses the results, refines the parameters, and runs again as necessary.

Effort Multipliers

The effort multipliers are divided among four attribute categories: product, computer, personnel, and project.



For the target machine, the user determines for each cost driver (or cost attribute) the effort multiplier rating that will be used and justifies the reason for choosing that rating. This table shows the various effort multiplier ratings:

Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
RELY	Slight inconvenience	Easily recover losses	Recoverable losses	High financial loss	Risk to human life	
DATA		D/P < 10	10 < D/P < 100	100 < D/P < 1000	D/P > 1000	
CPLX	Refer to next table					
TIME			< 50% use of avail exec time	70%	85%	95%
STOR			< 50% use of avail exec time	70%	85%	95%
VIRT		Major chg 1 per 12 mos, min; 1 per mo	Maj: 1/6 mos Min: 1/2 wks	Maj: 1/2 mos Min: 1/week	Maj: 1/2 wks Min: 1/2 days	
TURN		Interactive	< 4 hrs	4–12 hrs	> 12 hrs	
ACAP	15th percentile	25th percentile	55th percentile	75th percentile	90th percentile	
AEXP	< 4 mos experience	1 year	3 years	6 years	12 years	
PCAP	15th percentile	25th percentile	55th percentile	75th percentile	90th percentile	
VEXP	< 1 month	4 months	1 year	3 years		

Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
VEXP	< 1 month	4 months	1 year	3 years		
MODP	No use	Beginning use	Some use	General use	Routine use	
TOOL	Basic use of microprocessor tools	Basic use of small tools	Strong use of small tools and basic use of large tools	Strong use of large tools and use of test tools	Advanced use of large tools	
SCED	75% of nominal	85% of nominal	100% of nominal	130% of nominal	160% of nominal	

Note: for the Data row: D: Database size in bytes, P: LOC delivered

You will see the ratings go from Very Low to Extra High. Not all cost drivers used every rating. This reason is based on Barry Boehm's regression modeling analysis. These terms and their ratings are:

Required Software Reliability (RELY) is defined in terms of the effect it has on the user from a Slight Inconvenience (Very Low rating) to Risk to Human Life (Very High rating). As you can see there is no Extra High rating.

Database size (DATA) is the measure of data in the system, that is, the ratio D/P where D is the database size in bytes and P is the estimated delivered program instructions or delivered LOC. The rating is from Low or a ratio of less than 10 to Very High with a rating greater than 1000.

Product Complexity (CPLX) is comprised of four types of operations: Control Operations, Computational Operations, Device-Dependent Operations, and Data Management Operations. It is determined for each subsystem when using the Detailed Model. This cost driver spans the entire range from Very Low to Extra High. Here is a selection of the operations:

CPLX - Product Complexity (or each subsystem for Detailed Model)

Rating	Control Operations	Computational Operations	Device-dependent Operations	Data Management Operations
Very Low	Straight-line code with few non-nested SP operators: DOs, CASEs; Simple predicates	Evaluation of simple expressions A = B + C*D	Simple read/write statements with simple formats	Simple arrays in main memory
Low	Straightforward nesting of SP operators, mostly simple predicates	Evaluation of some complex expressions	Device timing-dependent coding	
Nominal	Mostly simple nesting, some intermodule control, decision tables	Structured numerical analysis		
High	Highly nested SP operators, compound predicates, queue & stack control	Unstructured evaluation of		
Very High	Reentrant & recursive coding, fixed priority interrupt handling			
Extra High	Multiple resource scheduling, dynamically changing priorities, microcode level control			

The Available Execution Time Constraint (TIME) is what percentage of the available execution time your system will use. It can range from less than 50% use of the available execution time (Nominal rating) to 95% (Extra High).

Similarly the Available Storage Constraint (STOR) is what percentage of the available storage your system will use. It can range from less than 50% use of the available storage (Nominal rating) to 95% (Extra High).

Virtual System Volatility (VIRT) is the anticipated number or major and minor changes to the system over a period of time. It ranges from a single major change every 12 months and one minor change each month for a Low rating to a major change every two weeks and a minor change every two days for a Very High rating. For a given software product, the underlying virtual machine is the complexity of the hardware and software (including the operating system and database management system it calls upon) to accomplish the tasks.

Computer Turnaround Time (TURN) is the time it takes to get a response back to the user. Most applications today are interactive. But there are applications which take a long time such as payroll systems which typically run overnight. This cost driver ranges from Interactive at Low to greater than 12 hours at Very High. Since COCOMO can be calibrated to your company's operating norms, this is probably a cost driver that would change.

Analyst Capability (ACAP) is the average percentile of the analysts' capability working on this project ranging from 15% or Very Low to 90% or Very High.

Application Experience (AEXP) is the average level of experience working with in this application domain. It ranges from a team with less than four months experience (Very Low) to a team with 12 years' experience (Very High.)

Programmer Capability (PCAP) is the average percentile of the programmer or software developers' capability working on this project ranging from 15% or Very Low to 90% or Very High.

Virtual Machine Experience (VEXP) is the team's experience with virtual machines. This is another parameter that is less useful in today's environment. It ranges from less than one month (Very Low) to three years (High.)

Program Language Experience (LEXP) is the team's experience with the specific programming language. It ranges from less than one month (Very Low) to three years (High.)

Modern Programming Practices (MODP) are the level of use of current programming practices. It ranges from No Use (Very Low) to Routine Use (Very High.)

The Use of Tools or Tools Experience (TOOL) is the level and number of tools available to the team. It ranges from basic use of microprocessor tools (Very Low), basic use of small tools (Low), strong use of small tools and basic use of large tools (Nominal), strong use of large tools and use of test tools (High) to advanced use of large tools (Very High.)

Lastly the Required Development Schedule (SCED) is the schedule that the customer requires. Ideally the team should deliver the product in the nominal time frame or at the exact time due. If the product is delivered in 75% of the time, it is rated Very Low. If it is delivered late at 160% of nominal, it is rated (Very High.)

The Effort Multiplier ratings are overlaid on the chart to reveal these Effort Multiplier Values.

Attributes		Very Low	Low	Nominal	High	Very High	Extra High
Product Attributes							
RELY	Required Software Reliability	.75	.88	1.00	1.15	1.40	
DATA	Database Size		.94	1.00	1.08	1.16	
CPLX	Product Complexity	.70	.85	1.00	1.15	1.30	1.65
Computer Attributes							
TIME	Execution Time Constraint			1.00	1.11	1.30	1.66
STOR	Main Storage Constraint			1.00	1.06	1.21	1.56
VIRT	virtual Machine		.87	1.00	1.15	1.30	

Attributes		Very Low	Low	Nominal	High	Very High	Extra High
	Volatility						
TURN	Computer Turnaround Time		.87	1.00	1.07	1.15	
Personnel Attributes							
ACAP	Analyst Capability	1.46	1.19	1.00	.86	.71	
AEXP	Applications Experience	1.29	1.13	1.00	.91	.82	
PCAP	Programmer Capability	1.42	1.17	1.00	.86	.70	
VEXP	Virtual Machine Experience	1.21	1.10	1.00	.90		
LEXP	Programming Language Experience	1.14	1.07	1.00	.95		
Project Attributes							
MODP	Use of Modern Programming Practice	1.24	1.10	1.00	.91	.82	
TOOL	Use of Software Tools	1.24	1.10	1.00	.91	.83	
SCED	Required Development Schedule	1.23	1.08	1.00	1.04	1.10	

SM: Staff Months
PEM: Product of Effort Multipliers (covered next)
KDSI: Thousand Delivered Source Instructions
T_{DEV}: Time for Development

Note that the same equations are used for both the Intermediate and Detailed models. For the Detailed model, you would use these equations for each phase or each major subsystem. The user then runs the model, assesses the results, refines the parameters, and runs again as necessary.

Cost Trade-Off Analysis

COCOMO can be used to compare alternative approaches and the cost and schedule impacts. Using the previous example, we can determine if the use of average analysts and programmers each earning a salary of $5K/SM is more cost effective than using more experienced analysts and programmers with a salary of $6K/SM.

Trade Data	More Experienced Personnel	Average Personnel

Trade Data	More Experienced Personnel	Average Personnel
Cost/Staff Month	$6K/Staff Month	$5K/Staff Month
Analyst Capability	0.86	1.0
Programmer Capability	0.86	1.0
Effort Adjustment Factor	1.17	1.17 / (0.86 * 0.86) = 1.58
Estimated Effort	52 Staff Months	70 Staff Months
Estimated Cost	$6K/SM * 52 SM = $312K	$5K/SM * 70 SM = $350K

1.17/52 = 1.58/X
X = (1.58)*(52)/(1.17)
X = 70

COnstructive COst MOdel (COCOMO): A Microprocessor Example

Microprocessor Software Example

- Microprocessor-based communications software
- 10,000 LOC communication processing software
- Embedded mode, Intermediate model
- Local use - moderate effect of failures
- 20,000 byte database
- Uses 70% of available CPU capacity
- Uses 45K of 64K storage
- Senior analysts: 75th percentile
- Applications experience: 3 years
- Programmer capability: 75th percentile
- Virtual machine experience: 6 months
- Programming language experience: 12 months
- Most programming practices in use more than 1 year
- Tools used at basic mini level
- 9 month schedule

Source: Boehm, Software Engineering Economics, 1981.

Attributes	Very Low	Low	Nominal	High	Very High	Extra High
Product Attributes						
RELY	Effect: slight	Easily recover loses	Recoverable losses	High financial loss	Risk to human life	
DATA		D/P < 10	10 < D/P < 100	100 < D/P < 1000	D/P > 1000	
CPLX	Very low	Low complexity	Nominal	Highly complex	Very complex	
Computer Attributes						
TIME			< 50% use of time	70%	85%	95%
STOR			< 50% use of storage	70%	85%	95%
VIRT		Major chg 1 per 12 mos, min: 1 per mo	Maj: 1/6 mos Min: 1/2 wks	Maj: 1/2 mos Min: 1/week	Maj: 1/2 wks Min: 1/2 days	
TURN		Interactive	< 4 hrs	4 – 12 hrs	> 12 hrs	
Personnel Attributes						
ACAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
AEXP	< 4 mos exp	1 year	3 years	6 years	12 years	
PCAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
VEXP	< 1 month	4 months	1 year	3 years		
LEXP	< 1 month	4 months	1 year	3 years		
Project Attributes						
MODP	No use	Beginning use	Some use	General use	Routine use	
TOOL	Basic micro	Basic mini tools	Strong mini	Strong maxi	Adv maxi	
SCED	75% of nominal	85% of nominal	100% of nominal	130% of nominal	160% of nominal	

Attributes		Very Low	Low	Nominal	High	Very High	Extra High
Product Attributes							
RELY	Required software reliability	.75	.88	1.00	1.15	1.40	
DATA	Database size		.94	1.00	1.08	1.16	
CPLX	Product complexity	.70	.85	1.00	1.15	1.30	1.65
Computer Attributes							
TIME	Execution time constraint			1.00	1.11	1.30	1.66
STOR	Main storage constraint			1.00	1.06	1.21	1.56
VIRT	Virtual machine volatility		.87	1.00	1.15	1.30	
TURN	Computer turnaround time		.87	1.00	1.07	1.15	
Personnel Attributes							
ACAP	Analyst capability	1.46	1.19	1.00	.86	.71	
AEXP	Applications Experience	1.29	1.13	1.00	.91	.82	
PCAP	Programmer capability	1.42	1.17	1.00	.86	.70	
VEXP	Virtual machine experience	1.21	1.10	1.00	.90		
LEXP	Programming language experience	1.14	1.07	1.00	.95		
Project Attributes							
MODP	Use of modern programming practice	1.24	1.10	1.00	.91	.82	
TOOL	Use of software tools	1.24	1.10	1.00	.91	.83	
SCED	Required development schedule	1.23	1.08	1.00	1.04	1.10	

Effort and Schedule

Expected Effort in Staff Months:

$$SM = 2.8 * PEM * (KDSI)^{1.2}$$
$$= 2.8 * 1.17 * (10)^{1.2}$$
$$= 2.8 * 1.17 * 15.849$$
$$= 52 \text{ SM}$$

Expected Schedule in Months:

$$T_{DEV} = 2.5 * (SM)^{0.32}$$
$$= 2.5 * (52)^{0.32}$$
$$= 9 \text{ Months}$$

Average Staff (Level Loaded):

$$\text{Staff} = 52 \text{ SM} / 9\text{M}$$
$$= 5.8 \text{ Staff}$$