UTD

Estimating Times and Costs

Dr. Mark C. Paulk

SE 4381, Software Project Planning and Management

Management Topics

1. Modern project management

PMBOK

2. Organization strategy and project selection

3. Organization: structure and culture

4. Defining the project

5. Estimating times and costs

6. Developing a project plan

7. Managing risk

8. Scheduling resources and cost

9. Reducing project duration

10. Leadership

11. Teams

12. Outsourcing

13. Monitoring progress

14. Project closure

15. International projects

16. Oversight

17. Agile PM

Critical chain project management

Estimating

The process of forecasting or approximating the time and cost of completing project deliverables

May need to estimate parameters other than time and cost

- **size – lines of code, function points, etc.**
- **critical resources**

Software project cost is largely driven by effort.

Schedule and effort are NOT the same.

Tools and other environmental factors also affect productivity.

Why Estimating Is Important

Support good decisions

Schedule work

Determine how long the project should take and its cost

Determine whether the project is worth doing

Develop cash flow needs

Determine how well the project is progressing

Develop time-phased budgets and establish the project baseline

Factors Influencing the Quality of Estimates

Planning horizon

- Boehm's cone of uncertainty

Project duration

- time to implement a new technology tends to expand in a nonlinear fashion

People

- matching skills to the task (productivity, learning time)
- team building (worked together before)
- turnover
- team size (more communication channels)
- typically people only have 5-6 productive hours available per working day

Project structure and organization

- dedicated project team… structured open team

Padding estimates

- probability of meeting your target: 50%? 90%
- every level in the management structure may add or remove padding in an arbitrary way

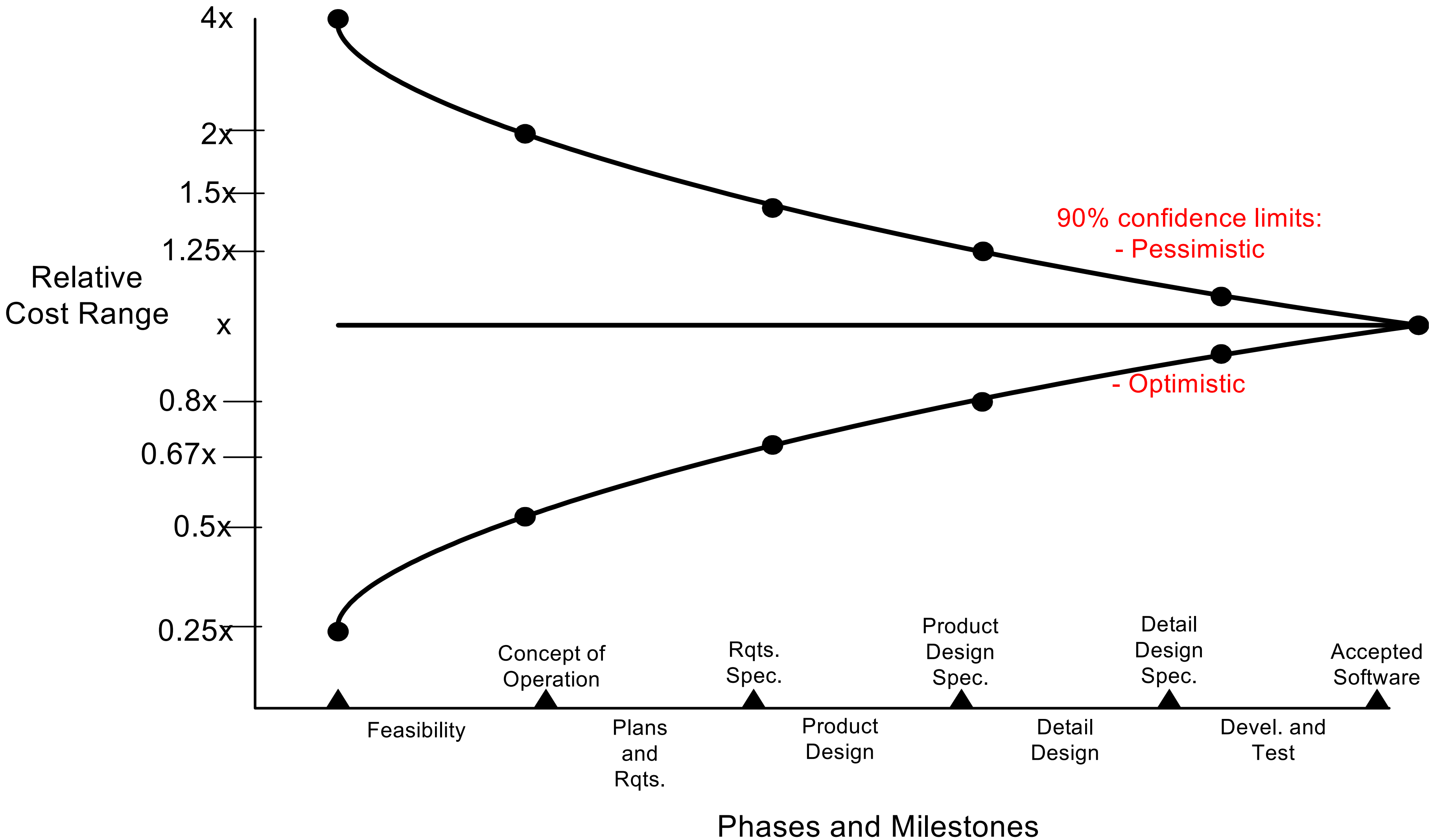
Organization culture

- is padding encouraged? estimation accuracy?
- is historical data available to support estimating?
- estimating skills?

Other factors

- equipment downtime
- holidays, vacation
- relative task / project / functional priorities

Boehm's Cone of Uncertainty



“When a person is forced to plan under severe schedule and budget constraints, planning consists mostly of a prioritized list and a lot of optimistic promises.”

-Hihn and Habib-Agahi

Expert Judgment by Analogy

Experts compare the target product to completed products

- **guesses can lead to hopelessly incorrect cost estimates**
- **experts may recollect completed products inaccurately**
- **human experts have biases**
- **the results of estimation by a broad group of experts may be accurate**

The Delphi technique is sometimes needed to achieve consensus.

- **see Boehm's 1981 discussion of Delphi (eLearning)**

Life Cycle Trumps Estimating?

The traditional waterfall life cycle model assumes complete knowledge about the product during planning.

- **this is rarely true**

Life cycle models that deal with ignorance and uncertainty are broadly characterized as incremental or iterative.

- **some software engineering methods, e.g., agile, are designed to be IID**

Bill Curtis: "The process of developing large software systems must be treated, at least in part, as a learning and communication process."

Top-Down Estimating

May be based on experience

May NOT reflect knowledge of the processes needed to do the work

- **time and cost of specific tasks are not considered**
- **encourages errors of omission and imposed times and costs**

May be guesstimates rather than estimates

Can become a self-fulfilling prophecy

May NOT represent low-cost, efficient methods

  - **impossible and impractical regions**

Top-Down vs Bottom-Up

Two major types of estimating

Top-down usually done by (senior) management

- **analogy**
- **consensus**
- **mathematical (parametric) models**

Bottom-up usually done by the people who are doing the work

- **based on WBS**
- **may be affected by dependencies between tasks**

Top-Down Estimates

Intended use

- **feasibility / conceptual phase**
- **rough time / cost estimate**
- **fund requirements**
- **resource capacity planning**

Preparation cost: 0.1 to 0.3% of total project cost

Accuracy: -20% to +60%

Top-Down Approaches

Consensus methods (can also be bottom-up)

- **Delphi method**

Ratio methods

- **e.g., number of square feet to cost a house**
- **software features and complexity**

Apportion methods

Function point methods for software and system projects (really middle up)

Learning curves

Bottom-Up Estimates

Provides the customer with an opportunity to compare the low-cost, efficient approach with any imposed constraints

Intended use

- **budgeting**
- **scheduling**
- **resource requirements**
- **fund timing**

Preparation cost: 0.3 to 1.0% of total project cost

Accuracy: -10% to +30%

Bottom-Up Approaches

Template methods

- **derived from a “standard” past project**

Parametric procedures applied to specific tasks

Range estimating

- **low, average, high estimates**
- **PERT methodology**

Phase estimating

- **significant amount of uncertainty**
- **begin with a top-down estimate**
- **detailed estimate for the immediate phase**
- **macro estimates for remaining phases**

Seven Guidelines for Estimating Work Packages

Responsibility

- **estimates should be made by the people most familiar with the task – team leaders or workers**

Use several people to estimate

- **Delphi method to aggregate judgments**

Normal conditions

- **reflect "normal" efficient use of resources**
- **defer considering conflicts for resourcing or concurrency to scheduling**

Time units

- **use consistent time units appropriate for the kind of work**

Independence

- consider each task time estimate independently of other activities

Contingencies

- work package estimates should not include allowances for contingencies
- contingencies (management reserves, buffers) should be addressed at a higher level than estimating tasks

Adding risk assessment to the estimate helps avoid surprises to stakeholders

- software project management IS risk management

Types of Costs

Direct costs

- **labor**
- **materials**
- **equipment**
- **other**

Direct project overhead costs

General and administrative (G&A) overhead costs

- **organization costs that are not directly linked to a specific project**

Conditions for Preferring Top-Down or Bottom-Up Estimating

Condition	Top-Down Estimates	Bottom-Up Estimates
Strategic decision making	**X**	
Cost and time important		**X**
High uncertainty	**X (agile!)**	
Internal, small project	**X**	
Fixed-price contract		**X**
Customer wants details		**X**
Unstable scope	**X (agile!)**	

Refining Estimates

Why, after doing detailed estimating, might you want to adjust your estimates by a significant amount?

Interaction costs are hidden in estimates

- **assumed independence is wrong!**

Normal conditions do not apply

- **especially wrt resources**

Things go wrong…

Changes in project scope and plans

- **ignorance and uncertainty**

Zero-Based Budgeting

Make a list of all the features you want in the final product

Prioritize them from most to least important

Write down the costs and (schedule) significance of each feature

When you run out of resources, draw a line under the last feature you can accomplish within your budget and schedule

- **items above the line can be done**
- **items below the line should be scrubbed**

Negotiate with the stakeholders based on reality…

Safety Mechanisms for Estimating

Be pessimistic in your estimates.

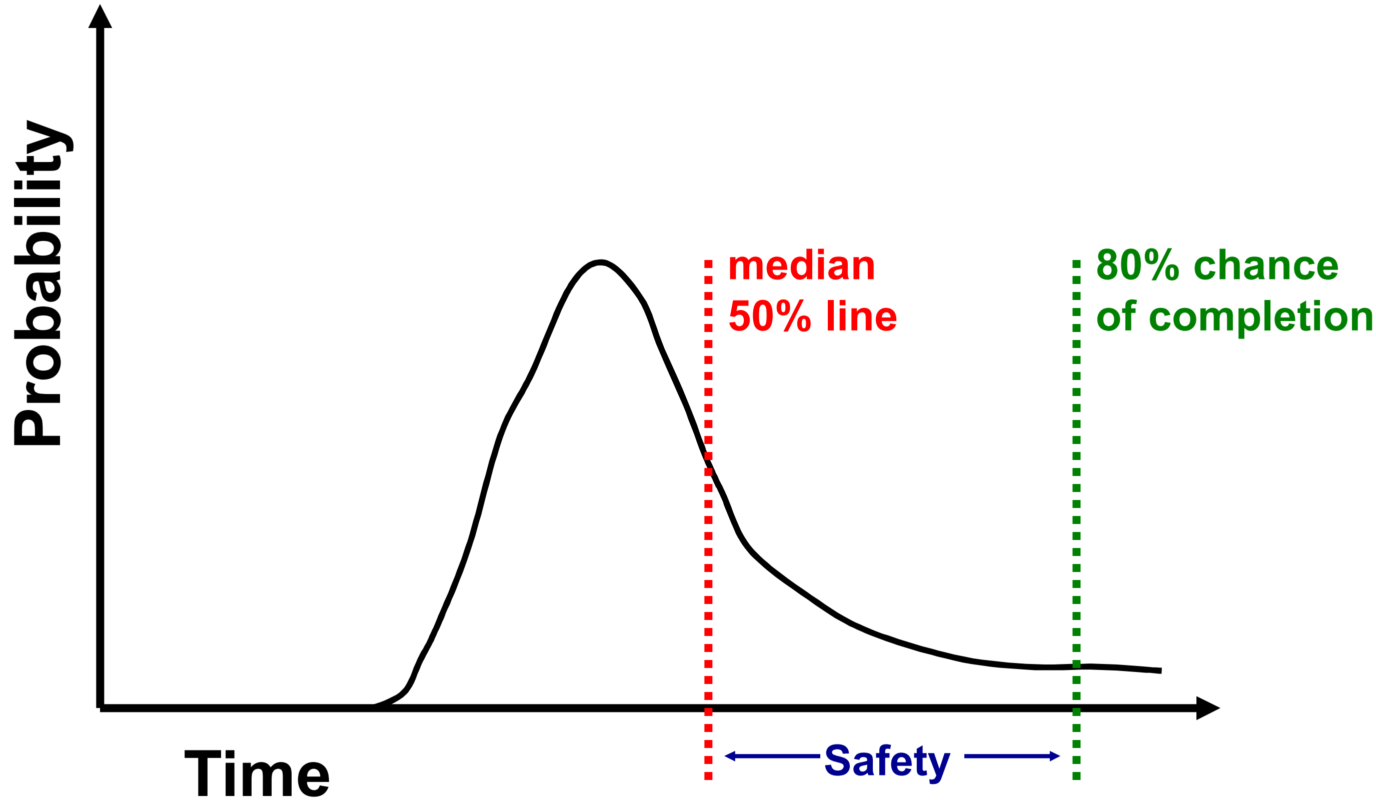
Each level in the management hierarchy should add their own safety factor.

Protect your estimates from a global cut (to win the contract…).

Wasting the safety

- ***the student syndrome – start at the last minute***
- ***multi-tasking***
- ***dependencies between steps cause delays to accumulate and advances to be wasted***

CCPM: Safety



The higher the uncertainty, the longer the tail of the distribution.

- ***Critical Chain, page 44***

Contributors to Poor Estimation

Lack of estimating experience

Lack of historical data on which to base estimates

Lack of a systematic estimation process, sound techniques, or models suited to the project's needs

Failure to include essential project activities and products within the scope of the estimates

Unrealistic expectations or assumptions

Failure to recognize and address the uncertainty inherent in project estimates

Estimating Rules of Thumb

(Jones, 1996)

Raising the number of function points to the 1.15 power predicts approximate page counts for paper documents associated with software projects.

Creeping user requirements will grow at an average rate of 1% per month over the entire development schedule.

Raising the number of function points to the 1.25 power predicts the approximate defect potential for new software projects. (1.27 for enhancement projects)

Each software review, inspection, or test step will find and remove 30% of the bugs that are present.

Raising the number of function points to the 0.4 power predicts the approximate development schedule in calendar months.

Dividing the number of function points by 150 predicts the approximate number of personnel required for the application.

Parametric Cost Estimation Models

A metric is used as an input to a model to compute cost and duration

- **a parametric model is unbiased, and therefore in principle superior to expert opinion**
- **estimates are only as good as the underlying assumptions…**

Examples

- **SLIM Model**
- **Price S Model**
- **<u>CO</u>nstructive <u>CO</u>st <u>MO</u>del (COCOMO)**

First-Order Models

The first-order model is the most rudimentary… simply a productivity constant.

$E_d = C_k S_e$

- **where E_d is development effort, C_k is productivity factor, and S_e is estimated SLOC**

The weakness of this model is its insensitivity to the magnitude of the effective product size. Productivity is, or at least should be, decreased for larger projects.

Second-Order Models

The second-order model incorporates an entropy factor to account for the impact of a large number of communications paths in large teams.

- **entropy < 1 shows a productivity increase with size, > 1 represents a productivity decrease with size**

$$E_d = C_k \, S_e^{\beta}$$

- **where β is the entropy factor**

Most used entropy values of approximately 1.2.

The major weakness of this model is its inability to adjust the productivity factor to account for variations between projects in development environments.

Third-Order Models

The third-order model incorporates a set of environment factors to adjust the productivity factor to fit a larger range of problems.

$$E_d = C_k \left(\Pi_{i=1}^{n} f_i\right) S_e^{\beta}$$

- **where f_i is the i^{th} environment factor**

For example, COCOMO II has 17 environment factors (and 5 scaling adjustment factors for β)

Input to Cost Model

LOC (or FP) is accurately known only when the product finished.

Estimation based on LOC is therefore doubly dangerous.

- **to start the estimation process, LOC (FP) in the finished product must be estimated**
- **the LOC (FP) estimate is then used to estimate the cost of the product — an uncertain input to an uncertain cost estimator**

Software Size Measures

Source lines of code (SLOC), lines of code (LOC), delivered source instructions (DSI), …

Function points,

- **number of inputs, outputs, inquiries, data files, and interfaces**
- **typical US productivity: 5 function points per person month**

feature points, …

Backfiring

C. Jones, <u>Applied Software Measurement, Third Edition</u>, 2008. (Table 2-14)

QSM, "Function Point Languages Table," <URL: http://www.qsm.com/resources/function-point-languages-table>

Language	Jones (avg)	QSM (avg)
Basic assembly	320	119
C	128	97
Fortran	107	
Java		53
Pascal	91	
Ada83	71	
C++	53	50
SQL	12	21

Putnam's SLIM Cost Model

Equations that relate effort and development time:

$$(E / B)^{1/3} * t_d^{4/3} = Size / PP$$

$$K / t_d^3 = MB$$

where

E = effort	**Size = source lines of code**
B = a constant	**PP = process productivity**
t_d = development time	**K = E / 0.39**
	MB = manpower buildup

L.H. Putnam, "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," IEEE Transactions on Software Engineering, April, 1978.

Using the SLIM Model

We estimate size.

- **B is a function of project size**
- **size is measured in (effective) source lines of code**

We know process productivity and manpower buildup from calibrating previous projects.

Effort and time are measured in years.

Impossible & Impractical Regions

L.H. Putnam and W. Myers, Industrial Strength Software: Effective Management Using Measurement, 1997.

- **if we knew enough to draw the critical path (or PERT) diagram, the length of the critical path would represent the minimum schedule**
  - **the impossible region is that less than the minimum…**
- **it is usually impractical to plan a development time much greater than 130% of the minimum**
- ***lengthening the development time (just two or three months) leads to greatly reduced cost***

Death march projects attempt to enter the impossible region…

- ***E. Yourdon, Death March: The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects, 1997.***

Tradeoffs in Effort and Schedule

(Putnam 1997, pp. 104-105)

System Char	Development Time (months)			
	Min	110%	120%	130%
10 KSLOC				
Dev time	7.0	7.7	8.4	9.2
Effort	12.3	8.4	5.9	4.3
100 KSLOC				
Dev time	18.9	20.8	22.7	24.5
Effort	587.5	401.3	283.3	205.7
1 MSLOC				
Dev time	50.6	55.7	60.8	65.8
Effort	11,353.6	7,754.7	5,475.3	3,975.2

COCOMO Model

The underlying COCOMO effort model is

$$\textbf{effort} = a \times (\textbf{size})^{b}$$

Intermediate COCOMO (1981)

- **three values for (*a*, *b*)**

COCOMO II (1995)

- ***a* varies depending on the values of 17 multiplicative cost factors**
- ***b* varies depending on the values of 5 scale factor parameters**

Validating Intermediate COCOMO

Intermediate COCOMO (1981) was validated with respect to a broad sample.

Actual values are within 20% of predicted values about 68% of the time.

- **Intermediate COCOMO was the most accurate estimation method of its time.**

Major problem

- **If the estimate of the number of lines of codes of the target product is incorrect, then everything is incorrect.**

COCOMO II

1995 extension to 1981 COCOMO that incorporates

- **object orientation**
- **modern life-cycle models**
- **rapid prototyping**
- **fourth-generation languages**
- **COTS software**

COCOMO II is more complex than the first version…

Three COCOMO II Models

Application composition model for the early phases

- **based on feature points (similar to function points)**

Early design model

- **based on function points**

Post-architecture model

- **based on function points or KDSI**

COCOMO II.2000 Effort Estimation Post-Architecture Model

Person months = PM

$PM = A * (Size \wedge E) * \Pi (EM_i)$

A = 2.94 (can be calibrated)

Size is in KSLOC

- **do not worry about the difference between KDSI and KSLOC for our purposes**

$E = B + 0.01 * \Sigma (SF_i)$

B = 0.91 (can be calibrated)

COCOMO II Effort Multipliers (EM)

RELY – required software reliability

DATA – database size

CPLX – product complexity

RUSE – develop for reuse

DOCU – documentation match to lifecyle needs

TIME – time constraint

STOR – storage constraint

PVOL – platform volatility

ACAP – analyst capability

PCAP – programmer capability

APEX – applications experience

PLEX – platform experience

LTEX – language and tool experience

PCON – personnel continuity

TOOL – use of software tools

SITE – multi-site development

SCED – required development schedule

Scale is categorized from very low to extra high.

See the table on eLearning for the values.

- ***nominal value is 1 (effort multipliers)***

COCOMO II Scale Factors (SF)

PREC – precedentedness

FLEX – development flexibility

RESL – architecture and risk resolution

TEAM – team cohesion

PMAT – process maturity

Note that nominal values for the Scale Factors are not 1.

- ***extra high values are 0 (sum of scale factors)***

Transaction Processing System Example

COCOMO II Post-Architecture Model

Software Cost Estimation with COCOMO II, 2000

Size: 44,700 SLOC estimated

Cost drivers (multiplicative factors)

- **ACAP – High**
- **PCON – High**
- **APEX – High**
- **PLEX – High**
- **TOOL – High**
- **SITE – Low**

Scale factors

- **PREC – High**
- **FLEX – High**
- **RESL – High**
- **TEAM – Very High**
- **PMAT – Nominal**

TPS Effort Estimation

$$E = B + 0.01 * \Sigma (SF_i)$$
$$= 0.91 + 0.01 (2.48 + 2.03 + 2.83 + 1.10 + 4.68)$$
$$= 1.0412$$

$$Person\_months = A * (Size \wedge E) * \Pi (EM_i)$$
$$= 2.94 * (44.7 \wedge 1.0412) * (0.85 * 0.90 * 0.88 * 0.91 * 0.90 * 1.09)$$
$$\approx 92 \text{ months}$$

Predicting

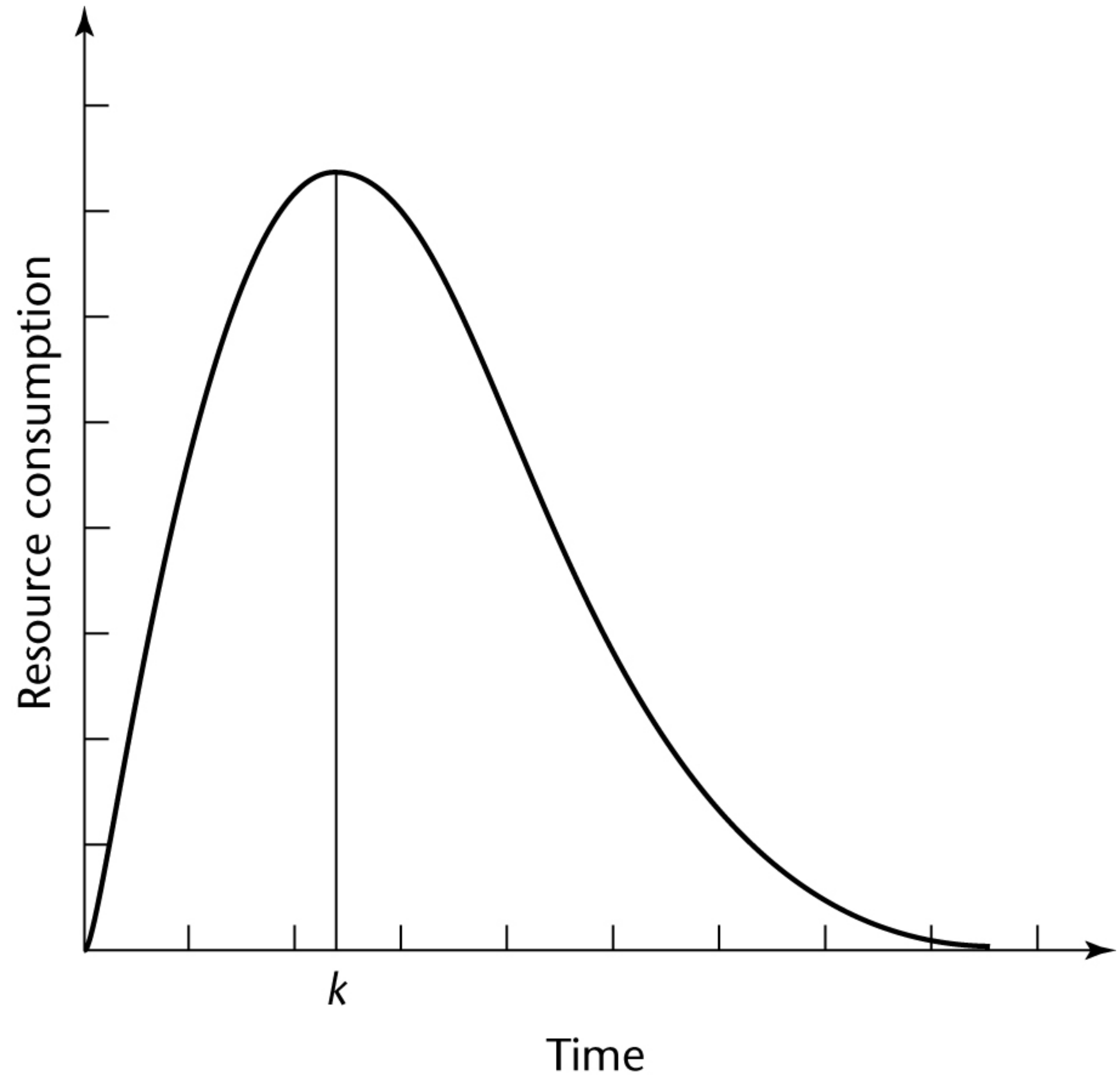
COCOMO involves replacing one difficult prediction problem (effort prediction) with another prediction problem which may be no easier (size prediction).

- **N. Fenton, "Software Measurement: A Necessary Scientific Basis," IEEE Transactions on Software Engineering, March 1994.**

Use of Resources Varies with Time

Rayleigh curves depict resource consumption.

The entire software development plan is a function of time.



Rayleigh Staffing Profile with sub-phases shown

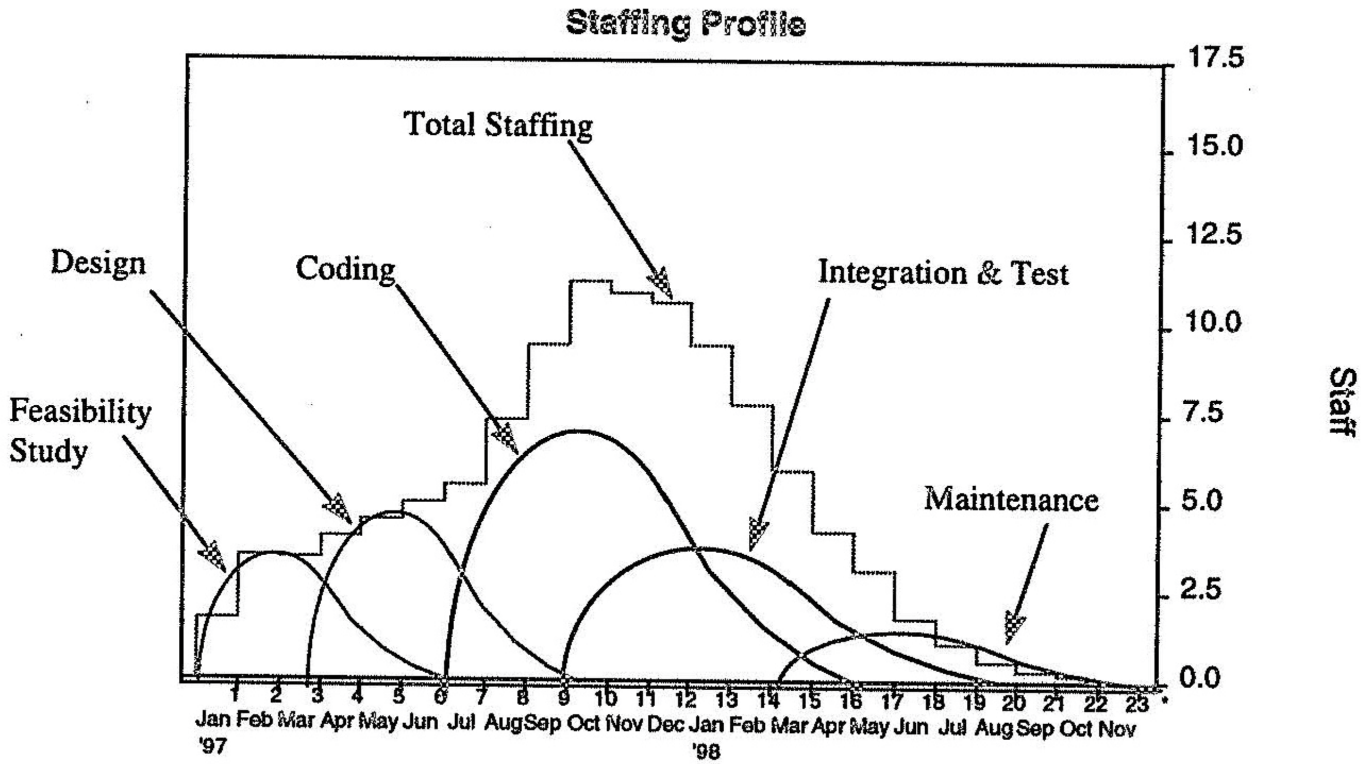


Figure 3-4. The set of small curves represents phases of the overall project. The sum of the effort under the small curves equals the total effort under the large curve. Although managers may not staff according to this curve, inevitably the way work gets done follows this pattern.

(Putnam 1997, p. 21)

COCOMO II.2000 Schedule Estimation Post-Architecture Model

TDEV = C * [PM_{NS} ^ (D + 0.2 * <E – B>)] * SCED% / 100

C = 3.67 (can be calibrated)

PM_{NS} is the estimated PM excluding the SCED effort multiplier

D = 0.28 (can be calibrated)

E – B = 0.01 * Σ (SF_i)

Schedule Compression

SCED% is the compression/expansion percentage in the SCED effort multiplier rating scale

- **very low 75%**
- **low 85%**
- **nominal 100%**
- **high 130%**
- **very high 160%**

TPS Schedule Estimation

E – B = 1.0412 – 0.91 = 0.1312

Person_months = PM_{NS} ≈ 92 person-months
- **SCED is nominal**

Time_to_develop = TDEV
= C * [PM_{NS} ^ (D + 0.2 * <E – B>)] * SCED% / 100
= 3.67 * [92 ^ (0.28 + 0.2 * <0.1312>)] * 1.0
≈ 15 calendar-months

Program Evaluation and Review Technique (PERT)

Developed in 1958 to schedule contractors on the Polaris submarine project

Uses three time estimates for each activity in project network

- **beta distribution is used to capture skewed-to-the-right distributions of activities**
  - **delays accumulate**

Weighted average activity time =
(optimistic estimate +
4 * most likely estimate +
pessimistic estimate) / 6

Standard deviation = (pessimistic – optimistic) / 6

Questions and Answers