SOFTWARE ENGINEERING ECONOMICS

DEFINITIONS

- The dictionary defines "**economics**" as "a social science concerned chiefly with description and analysis of the production, distribution, and consumption of goods and services."
- ***Economics** is the study of how people make decisions* in resource-limited situations.

MACROECONOMICS

- ***Macroeconomics** is the study of how people make decisions* in resource-limited situations on a national or global scale.
- It deals with the **effects of decisions** that national leaders make on such issues as tax rates, interest rates, foreign and trade policy.

MICROECONOMICS

- ***Microeconomics** is the study of how people make decisions* in resource-limited situations on a more personal scale.
- It deals with the decisions that individuals and organizations make on such issues as how much insurance to buy, which word processor to buy, or what prices to charge for their products or services.

ECONOMICS AND SOFTWARE ENGINEERING MANAGEMENT

- If we look at the discipline of software engineering, we see that the **microeconomics** branch of economics deals more with the types of decisions we need to make as software engineers or managers.
- Clearly, we deal with **limited resources**.
- There is never enough **time or money** to cover all the good features we would like to put into our software products.

- And even in these days of cheap hardware and virtual memory, our more significant software products must always operate within a world of limited computer power and main memory.
- If you have been in the software engineering field for any length of time, I am sure you can think of a number of decision situations in which you had to determine some key software product feature as a function of some limiting critical resource.

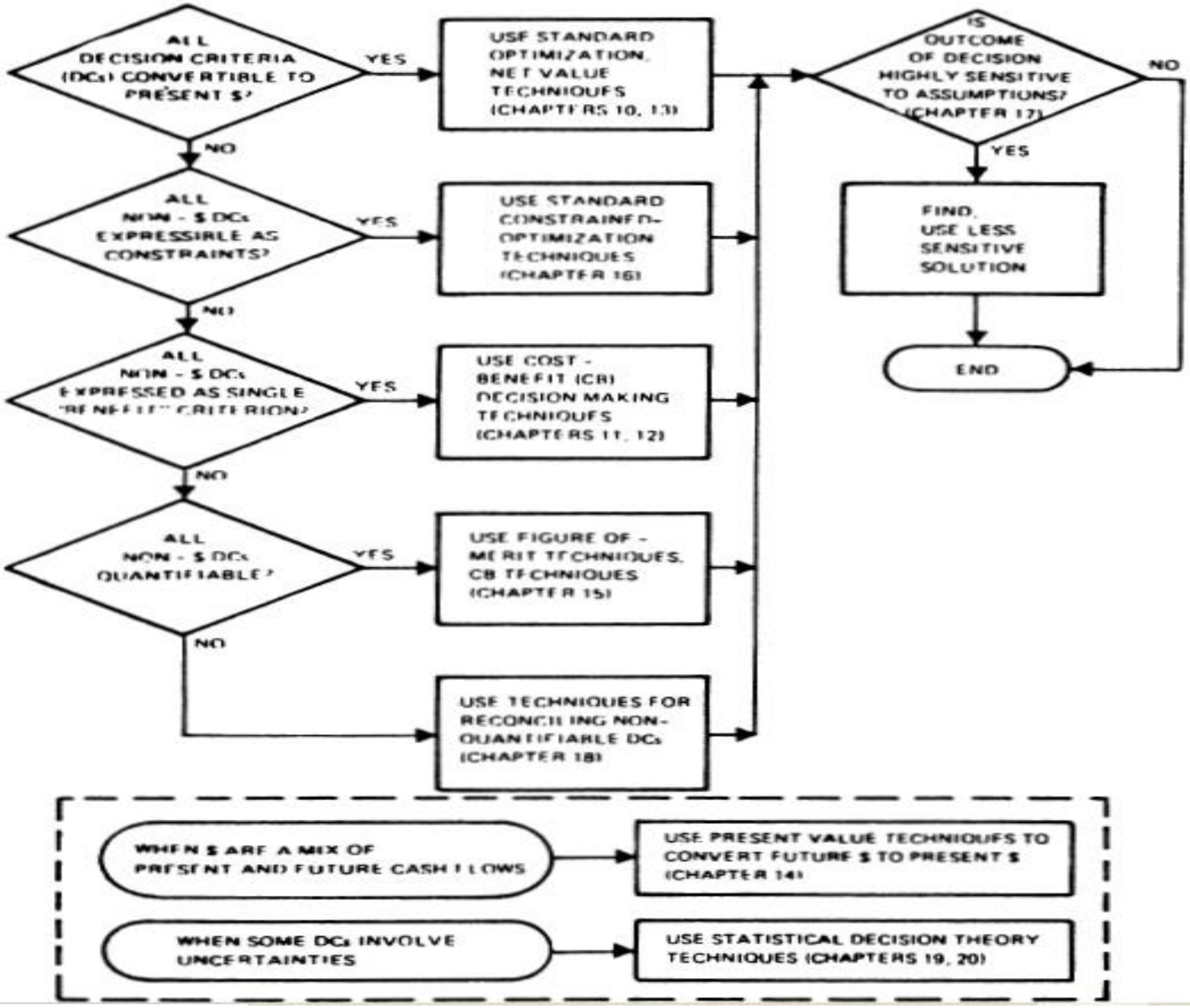
- Throughout the software life cycle, there are many decision situations involving limited resources in which software engineering economics techniques provide useful assistance.
- To provide a feel for the nature of these economic decision issues, an example is given below for each of the major phases in the software life cycle.
- ***Feasibility Phase:*** *How much should we invest in information* system analyses (user questionnaires and interviews, current-system analysis, workload characterizations, simulations, scenarios, prototypes) in order that we converge on an appropriate definition and concept of operation for the system we plan to implement?

- ***Plans and Requirements Phase:*** *How rigorously should* we specify requirements? How much should we invest in requirements validation activities (automated completeness, consistency, and traceability checks, analytic models, simulations, prototypes) before proceeding to design and develop a software system?
- ***Product Design Phase:*** *Should we organize the software* to make it possible to use a complex piece of existing software which generally but not completely meets our requirements?

- ***Programming Phase:*** *Given a choice between three data* storage and retrieval schemes which are primarily execution time-efficient, storage-efficient, and easy-to modify, respectively; which of these should we choose to implement?
- ***Integration and Test Phase:*** *How much testing and formal* verification should we perform on a product before releasing it to users?
- ***Maintenance Phase:*** *Given an extensive list of suggested* product improvements, which ones should we implement first?
- ***Phase out:*** *Given an aging, hard-to-modify software* product, should we replace it with a new product, restructure it, or leave it alone?

SOFTWARE ENGINEERING ECONOMICS ANALYSIS TECHNIQUES

- The microeconomics field provides a number of techniques for dealing with software life-cycle decision issues such as the ones given in the previous section. Fig. 1 presents an overall master key to these techniques and when to use them.
- As indicated in Fig. 1, **standard optimization techniques** can be used when we can find a single quantity such as dollars (or pounds, yen, cruzeiros, etc.) to serve as a "universal solvent" into which all of our decision variables can be converted.
- Or, if the nondollar objectives can be expressed as **constraints** (system availability must be at least 98 percent; throughput must be at least 150 transactions per second), then **standard constrained optimization techniques** can be used. And if cash flows occur at different times, then **present-value techniques** can be used to normalize them to a common point in time.



ALL DECISION CRITERIA (DCs) CONVERTIBLE TO PRESENT $?
YES
USE STANDARD OPTIMIZATION, NET VALUE TECHNIQUES (CHAPTERS 10, 13)
NO
ALL NON - $ DCs EXPRESSIBLE AS CONSTRAINTS?
YES
USE STANDARD CONSTRAINED-OPTIMIZATION TECHNIQUES (CHAPTER 16)
NO
ALL NON - $ DCs EXPRESSED AS SINGLE "BENEFIT" CRITERION?
YES
USE COST - BENEFIT (CB) DECISION MAKING TECHNIQUES (CHAPTERS 11, 12)
NO
ALL NON - $ DCs QUANTIFIABLE?
YES
USE FIGURE OF - MERIT TECHNIQUES, CB TECHNIQUES (CHAPTER 15)
NO
USE TECHNIQUES FOR RECONCILING NON-QUANTIFIABLE DCs (CHAPTER 18)
IS OUTCOME OF DECISION HIGHLY SENSITIVE TO ASSUMPTIONS? (CHAPTER 17)
NO
YES
FIND, USE LESS SENSITIVE SOLUTION
END
WHEN $ ARE A MIX OF PRESENT AND FUTURE CASH FLOWS
USE PRESENT VALUE TECHNIQUES TO CONVERT FUTURE $ TO PRESENT $ (CHAPTER 14)
WHEN SOME DCs INVOLVE UNCERTAINTIES
USE STATISTICAL DECISION THEORY TECHNIQUES (CHAPTERS 19, 20)

- More frequently, some of the resulting benefits from the software system are not expressible in dollars.
- In such situations, one alternative solution will not necessarily dominate another solution.
- An example situation is shown in Fig. 2, which compares the cost and benefits (here, in terms of throughput in transactions per second) of two alternative approaches to developing an operating system for a transaction processing system.

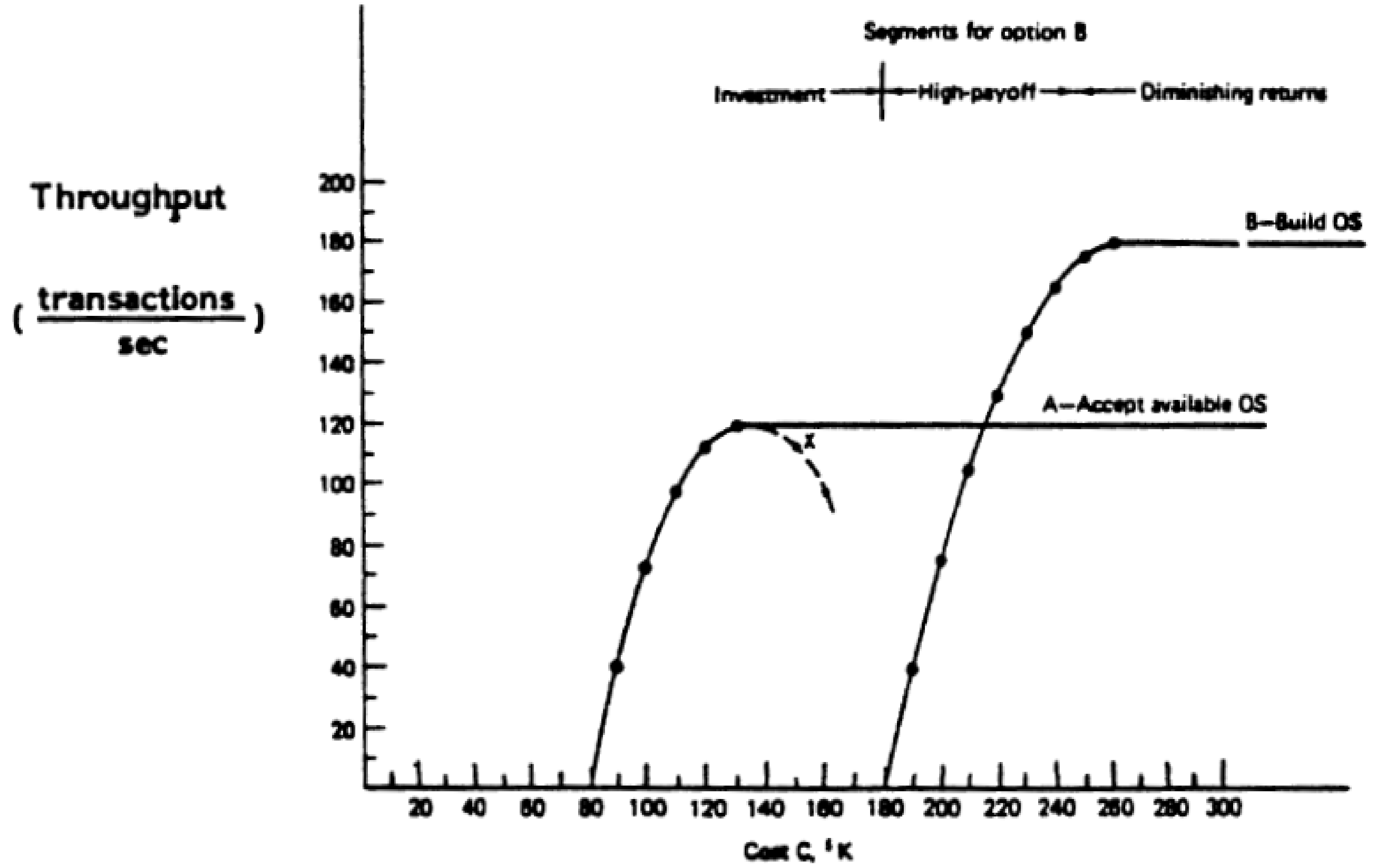


Fig. 2. Cost-effectiveness comparison, transaction processing system options.

- ***Option A:*** *Accept an available operating system. This* will require only $80K in software costs, but will achieve a peak performance of 120 transactions per second, using five $10K minicomputer processors, because of a high multiprocessor overhead factor.
- ***Option B:*** *Build a new operating system. This system* would be more efficient and would support a higher peak throughput, but would require $180K in software costs.

- The cost-versus-performance curve for these two options are shown in Fig. 2. Here, neither option dominates the other, and various cost-benefit decision-making techniques (maximum profit margin, cost benefit ratio, return on investments, etc.) must be used to choose between Options **A and B.**
- In general, software engineering decision problems **are** even more complex than Fig. 2, as Options A and B will have several important criteria on which they differ (e.g., robustness, ease of tuning, ease of change, functional capability).
- If these criteria are **quantifiable**, then some type of **figure of merit** can be defined to support a comparative analysis of the prefer- ability of one option over another.
- If some of, the criteria are unquantifiable (user goodwill, programmer morale, etc.), then some techniques for comparing unquantifiable criteria need to be used.

ANALYZING RISK, UNCERTAINTY, AND THE VALUE OF IN FORMATION

- In software engineering, our decision issues are generally even more complex than those discussed above. This is because the outcome of many of our options cannot be determined in advance.
- In such circumstances, we are faced with a problem of decision making under uncertainty, with a considerable risk of an undesired outcome.
- The main economic analysis techniques available to support us in resolving such problems are the following.
- 1) Techniques for decision making under complete uncertainty, such as the maximax rule, the maximin rule, and the Laplace rule [38]. These techniques are generally inadequate for practical software engineering decisions.

- 2) Expected-value techniques, in which we estimate the probabilities of occurrence of each outcome (successful or unsuccessful development of the new operating system) and complete the expected payoff of each option:

$$\begin{aligned} \text{EV} &= \text{Prob(success)} * \text{Payoff(successful OS)} \\ &\quad + \text{Prob(failure)} * \text{Payoff(unsuccessful OS)}. \end{aligned}$$

- 3) Techniques in which we reduce uncertainty by *buying information. For example, prototyping is a way of buying information* to reduce our uncertainty about the likely success or failure of a multiprocessor operating system; by developing a rapid prototype of its high-risk elements, we can get a clearer picture of our likelihood of successfully developing the full operating system.

- In general, prototyping and other options for buying information are most valuable aids for software engineering decisions.
- However, they always raise the following question: "how much information-buying is enough?"
- In principle,. this question can be answered via statistical decision theory techniques involving the use of Bayes' Law, which allows us to calculate the expected payoff from a software project as a function of our level of investment in a prototype or other information-buying option. (Some examples, of the use of Bayes' Law to estimate the appropriate level of investment in a prototype are given in [[1, ch. 20] .)
- In practice, the use of Bayes' Law involves the estimation of a number of conditional probabilities which are not easy to estimate accurately.

CONDITIONS

- However, the **Bayes' Law approach** can be translated into a number of *value-of-information guidelines, or* conditions under which it makes good sense to decide on investing in more information before committing ourselves to a particular course of action.
- ***Condition 1:*** *There exist attractive alternatives whose payoff vanes greatly, depending on some critical states of nature. I f not, we can commit ourselves to one of the attractive alternatives* with no risk of significant loss.
- ***Condition 2:*** *The critical states of nature have an appreciable probability of occurring. If not, we can again commit ourselves* without major risk. For situations with extremely high variations in payoff, the appreciable probability level is lower than in situations with smaller variations in payoff.

- ***Condition 3:*** *The investigations have a high probability of accurately identifying the occurrence of the critical states of nature. If not, the investigations will not do much to reduce* our risk of loss due to making the wrong decision.
- ***Condition 4****: The required cost and schedule of the investigations do not overly curtail their net value. It does us little* good to obtain results which cost more than they can save us, or which arrive too late to help us make a decision.
- ***Condition 5****: There exist significant side benefits derived from performing the investigations. Again, we may be able to* justify an investigation solely on the basis of its value in training, team-building, customer relations, or design validation.

SOME PITFALLS AVOIDED BY USING THE VALUE-OF-INFORMATION APPROACH

- The pitfalls below are expressed in terms of some frequently expressed but faulty pieces of software engineering advice.
- ***Pitfall 1:*** *Always use a simulation to investigate the feasibility of complex real-time software.*
- ***Pitfall 2:*** *Always build the software twice.*
- ***Pitfall 3:*** *Build the software purely top-down.*
- ***Pitfall 4:*** *Every piece of code should be proved correct.*
- ***Pitfall 5:*** *Nominal-case testing is sufficient.*

SOFTWARE ENGINEERING ECONOMICS

Estimation Factors

FUNDAMENTAL ESTIMATION QUESTIONS

- How much effort is required to complete an activity?
- How much calendar time is needed to complete an activity?
- What is the total cost of an activity?
- Project estimation and scheduling are interleaved management activities.

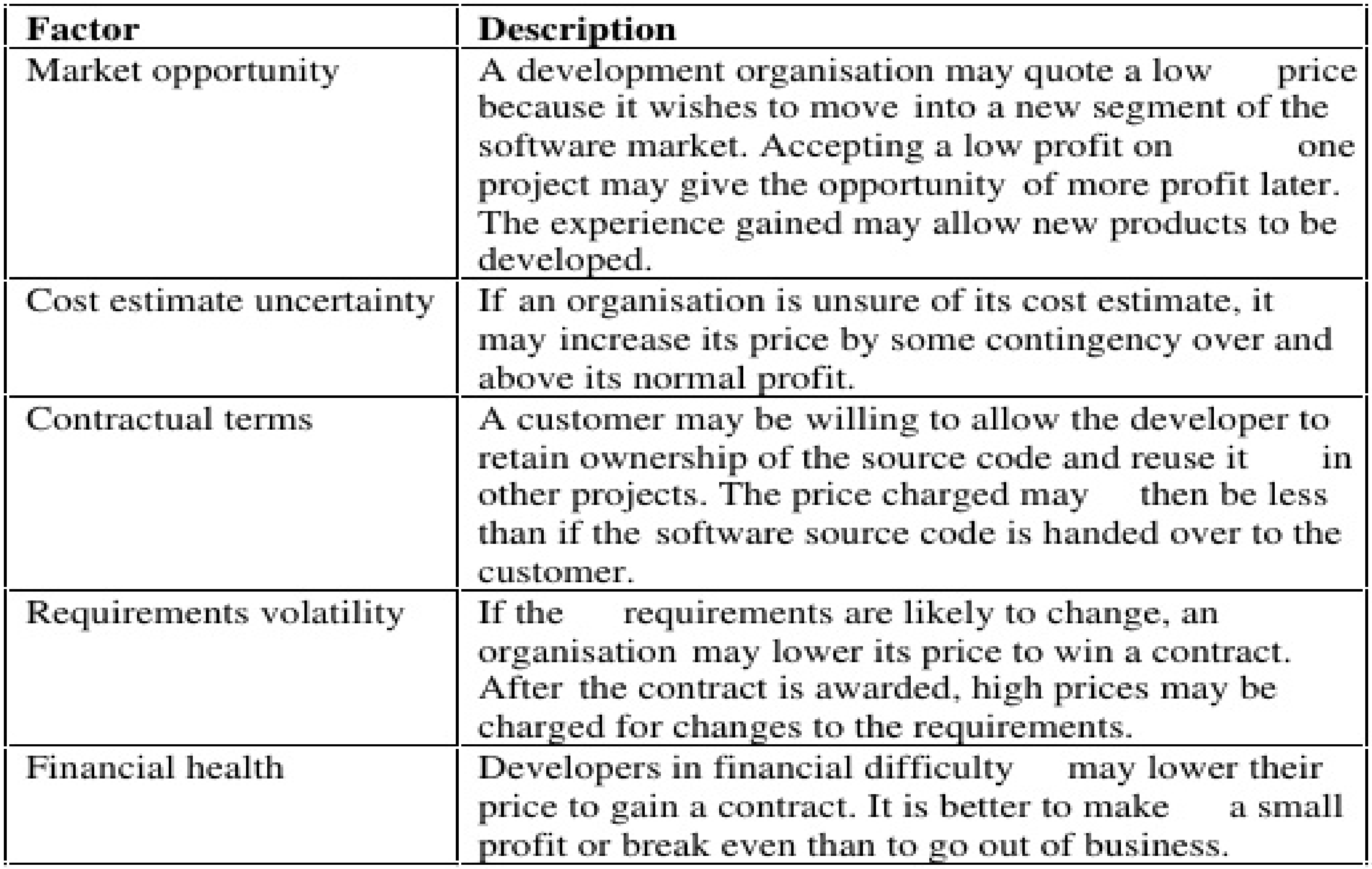
SOFTWARE COST COMPONENTS

- Hardware and software costs.
- Travel and training costs.
- Effort costs (the dominant factor in most projects)
  - The salaries of engineers involved in the project;
  - Social and insurance costs.
- Effort costs must take overheads into account
  - Costs of building, heating, lighting.
  - Costs of networking and communications.
  - Costs of shared facilities (e.g library, staff restaurant, etc.).

COSTING AND PRICING

- Estimates are made to discover the cost, to the developer, of producing a software system.
- There is not a simple relationship between the development cost and the price charged to the customer.
- Broader organisational, economic, political and business considerations influence the price charged.

SOFTWARE PRICING FACTORS



Factor	Description
Market opportunity	A development organisation may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the opportunity of more profit later. The experience gained may allow new products to be developed.
Cost estimate uncertainty	If an organisation is unsure of its cost estimate, it may increase its price by some contingency over and above its normal profit.
Contractual terms	A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may then be less than if the software source code is handed over to the customer.
Requirements volatility	If the requirements are likely to change, an organisation may lower its price to win a contract. After the contract is awarded, high prices may be charged for changes to the requirements.
Financial health	Developers in financial difficulty may lower their price to gain a contract. It is better to make a small profit or break even than to go out of business.

SOFTWARE PRODUCTIVITY

- A measure of the rate at which individual engineers involved in software development produce software and associated documentation.
- Not quality-oriented although quality assurance is a factor in productivity assessment.
- Essentially, we want to measure useful functionality produced per time unit.

PRODUCTIVITY MEASURES

- Size related measures based on some output from the software process.
  - This may be lines of delivered source code, object code instructions, etc.
- Function-related measures based on an estimate of the functionality of the delivered software.
  - Function-points are the best known of this type of measure.

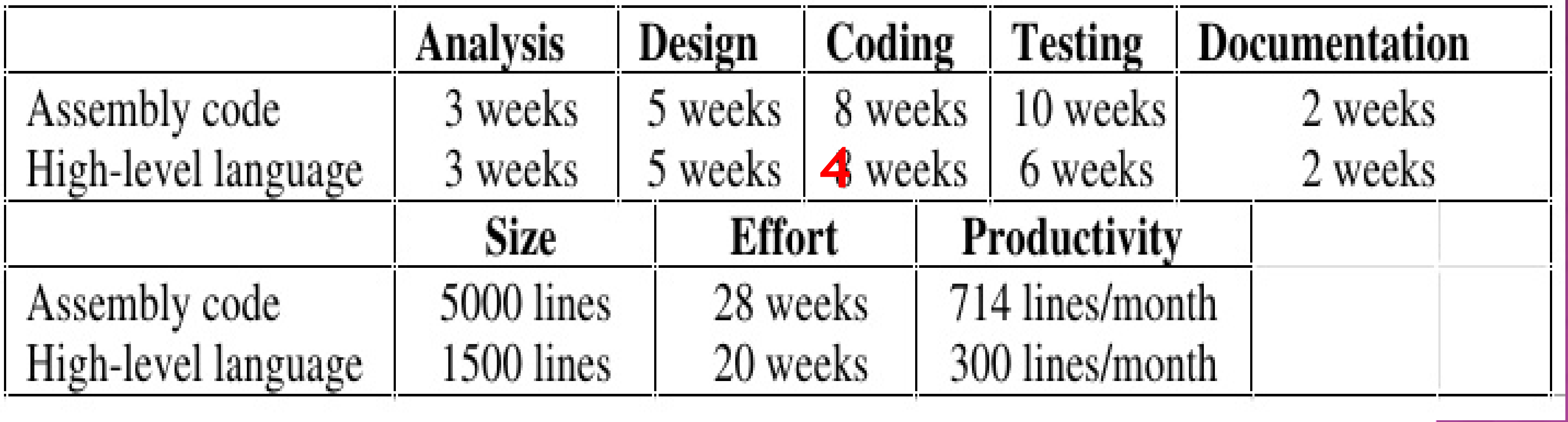
LINES OF CODE

- What's a line of code? –
  - The measure was first proposed when programs were typed on cards with one line per card;
- This model assumes that there is a linear relationship between system size and volume of documentation.

PRODUCTIVITY COMPARISONS

- The lower level the language, the more productive the programmer
  - The same functionality takes more code to implement in a lower-level language than in a high-level language.
- The more verbose the programmer, the higher the productivity
  - Measures of productivity based on lines of code suggest that programmers who write verbose code are more productive than programmers who write compact code.

SYSTEM DEVELOPMENT TIMES



	Analysis	Design	Coding	Testing	Documentation
Assembly code	3 weeks	5 weeks	8 weeks	10 weeks	2 weeks
High-level language	3 weeks	5 weeks	4 weeks	6 weeks	2 weeks

	Size	Effort	Productivity
Assembly code	5000 lines	28 weeks	714 lines/month
High-level language	1500 lines	20 weeks	300 lines/month

FUNCTION POINTS

- Based on a combination of program characteristics
  - external inputs and outputs;
  - user interactions;
  - external interfaces;
  - files used by the system.
- A weight is associated with each of these and the function point count is computed by multiplying each raw count by the weight and summing all values.

FUNCTION POINTS

- The function point count is modified by complexity of the project
- FPs can be used to estimate LOC depending on the average number of LOC per FP for a given language
  - LOC = AVC * number of function points;
  - AVC is a language-dependent factor varying from 200-300 for assemble language to 2-40 for a 4GL;
- FPs are very subjective. They depend on the estimator
  - Automatic function-point counting is impossible.

OBJECT POINTS

- Object points (alternatively named application points) are an alternative function-related measure to function points when 4Gls or similar languages are used for development.
- Object points are NOT the same as object classes.
- The number of object points in a program is a weighted estimate of
  - The number of separate screens that are displayed;
  - The number of reports that are produced by the system;
  - The number of program modules that must be developed to supplement the database code;

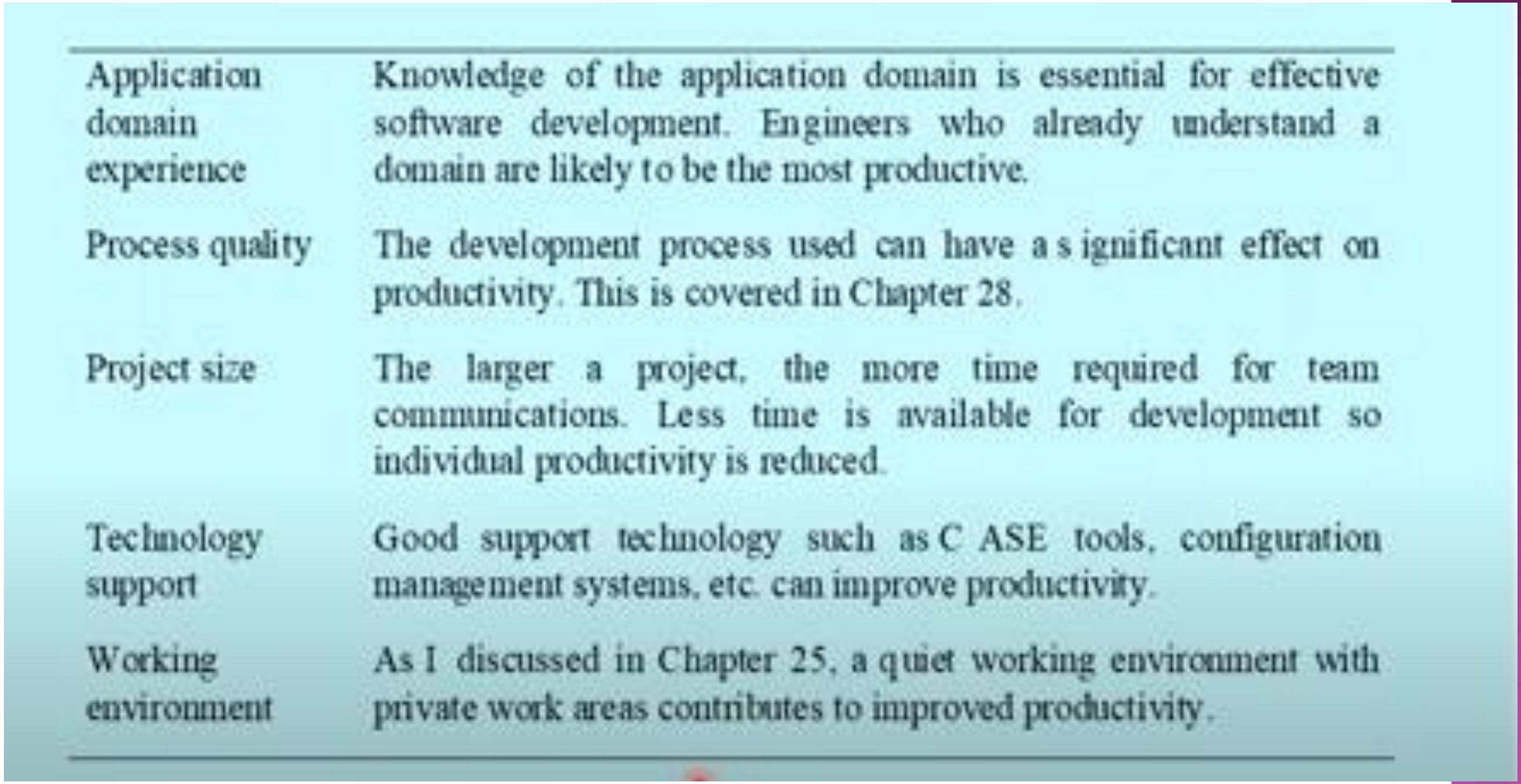
OBJECT POINT ESTIMATION

- Object points are easier to estimate from a specification than function points as they are simply concerned with screens, reports and programming language modules.
- They can therefore be estimated at a fairly early point in the development process.
- At this stage, it is very difficult to estimate the number of lines of code in a system.

PRODUCTIVITY ESTIMATES

- Real-time embedded systems, 40-160 LOC/P-month.
- Systems programs, 150-400 LOC/P-month.
- Commercial applications, 200-900 LOC/P-month.
- In object points, productivity has been measured between 4 and 50 object points/month depending on tool support and developer capability.

FACTORS AFFECTING PRODUCTIVITY



Application domain experience	Knowledge of the application domain is essential for effective software development. Engineers who already understand a domain are likely to be the most productive.
Process quality	The development process used can have a s ignificant effect on productivity. This is covered in Chapter 28.
Project size	The larger a project, the more time required for team communications. Less time is available for development so individual productivity is reduced.
Technology support	Good support technology such as C ASE tools, configuration management systems, etc. can improve productivity.
Working environment	As I discussed in Chapter 25, a quiet working environment with private work areas contributes to improved productivity.

QUALITY AND PRODUCTIVITY

- All metrics based on volume/unit time are flawed because they do not take quality into account.
- Productivity may generally be increased at the cost of quality. It is not clear how productivity/quality metrics are related.
- If requirements are constantly changing then an approach based on counting lines of code is not meaningful as the program itself is not static;

ESTIMATION TECHNIQUES

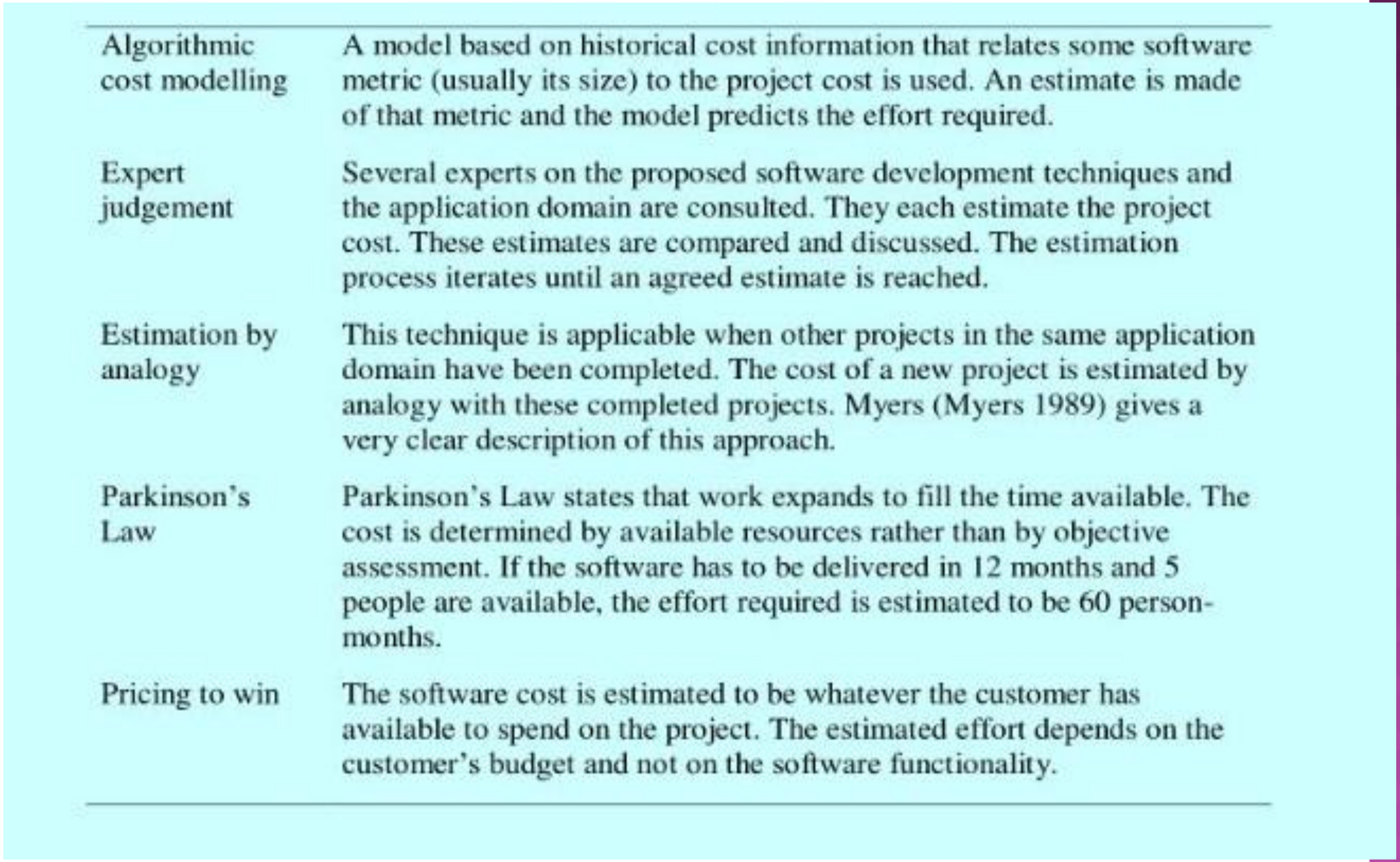
ESTIMATION TECHNIQUES

- There is no simple way to make an accurate estimate of the effort required to develop a software system
    - Initial estimates are based on inadequate information in a user requirements definition;
    - The software may run on unfamiliar computers or use new technology;
    - The people in the project may be unknown.
- Project cost estimates may be self-fulfilling
    - The estimate defines the budget and the product is adjusted to meet the budget.

ESTIMATION TECHNIQUES

- Algorithmic cost modelling.
- Expert judgement.
- Estimation by analogy.
- Parkinson's Law.
- Pricing to win.

ESTIMATION TECHNIQUES



Algorithmic cost modelling	A model based on historical cost information that relates some software metric (usually its size) to the project cost is used. An estimate is made of that metric and the model predicts the effort required.
Expert judgement	Several experts on the proposed software development techniques and the application domain are consulted. They each estimate the project cost. These estimates are compared and discussed. The estimation process iterates until an agreed estimate is reached.
Estimation by analogy	This technique is applicable when other projects in the same application domain have been completed. The cost of a new project is estimated by analogy with these completed projects. Myers (Myers 1989) gives a very clear description of this approach.
Parkinson's Law	Parkinson's Law states that work expands to fill the time available. The cost is determined by available resources rather than by objective assessment. If the software has to be delivered in 12 months and 5 people are available, the effort required is estimated to be 60 person-months.
Pricing to win	The software cost is estimated to be whatever the customer has available to spend on the project. The estimated effort depends on the customer's budget and not on the software functionality.

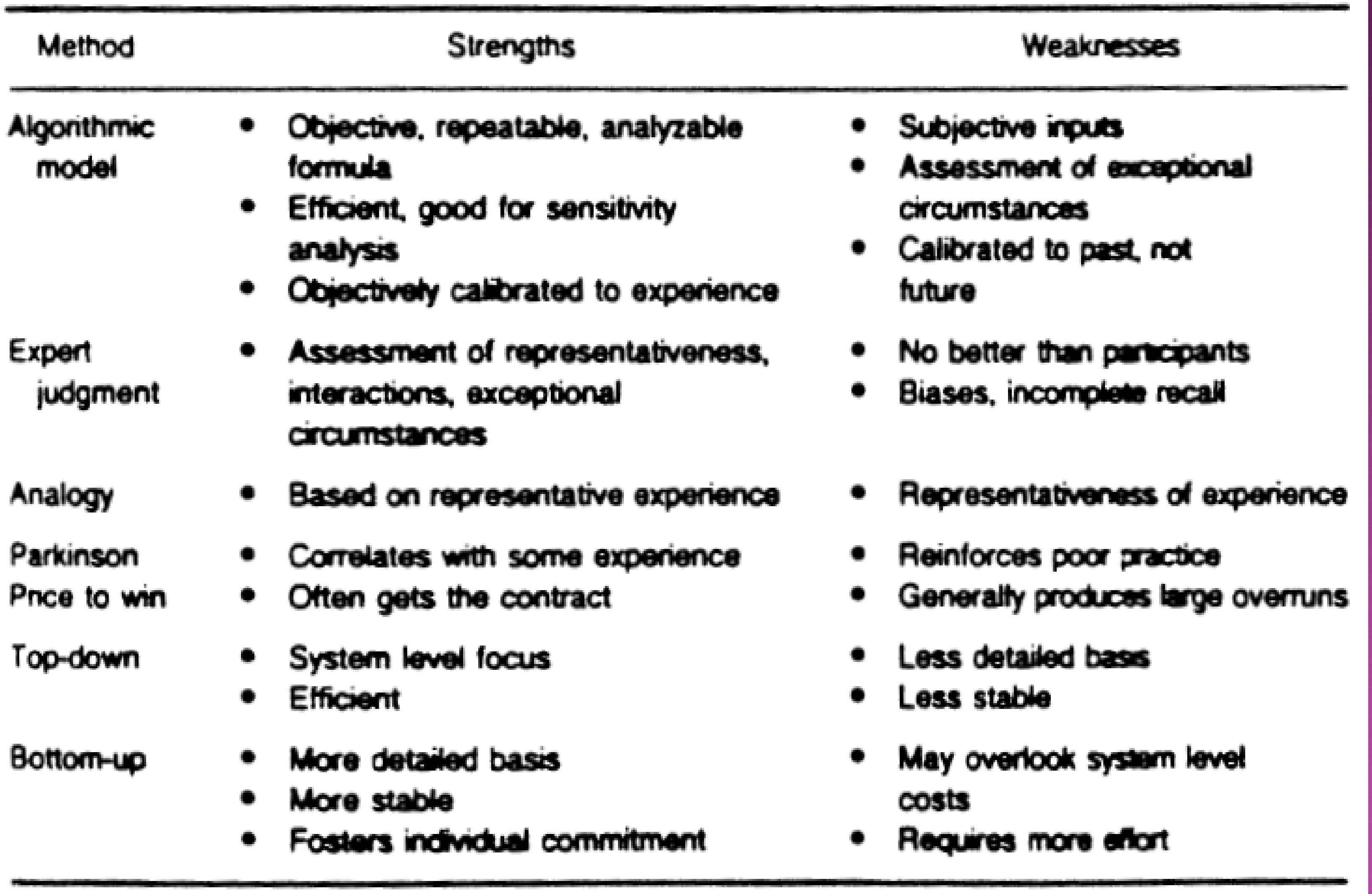
ESTIMATION METHODS

- Each method has strengths and weaknesses.
- Estimation should be based on several methods.
- If these do not return approximately the same result, then you have insufficient information available to make an estimate.
- Some action should be taken to find out more in order to make more accurate estimates.
- Pricing to win is sometimes the only applicable method.

ESTIMATION METHODS

- Each method has strengths and weaknesses.
- Estimation should be based on several methods.
- If these do not return approximately the same result, then you have insufficient information available to make an estimate.
- Some action should be taken to find out more in order to make more accurate estimates.
- Pricing to win is sometimes the only applicable method.

STRENGTHS AND WEAKNESSES OF SOFTWARE COST-ESTIMATION METHODS



Method	Strengths	Weaknesses
Algorithmic model	• Objective, repeatable, analyzable formula • Efficient, good for sensitivity analysis • Objectively calibrated to experience	• Subjective inputs • Assessment of exceptional circumstances • Calibrated to past, not future
Expert judgment	• Assessment of representativeness, interactions, exceptional circumstances	• No better than participants • Biases, incomplete recall
Analogy	• Based on representative experience	• Representativeness of experience
Parkinson	• Correlates with some experience	• Reinforces poor practice
Price to win	• Often gets the contract	• Generally produces large overruns
Top-down	• System level focus • Efficient	• Less detailed basis • Less stable
Bottom-up	• More detailed basis • More stable • Fosters individual commitment	• May overlook system level costs • Requires more effort

TOP-DOWN AND BOTTOM-UP ESTIMATION

- Any of these approaches may be used top-down or bottom-up.
- Top-down
  - Start at the system level and assess the overall system functionality and how this is delivered through sub-systems.
- Bottom-up
  - Start at the component level and estimate the effort required for each component. Add these efforts to reach a final estimate.

TOP-DOWN ESTIMATION

- Usable without knowledge of the system architecture and the components that might be part of the system.
- Takes into account costs such as integration, configuration management and documentation.
- Can underestimate the cost of solving difficult low-level technical problems.

BOTTOM-UP ESTIMATION

- Usable when the architecture of the system is known and components identified.
- This can be an accurate method if the system has been designed in detail.
- It may underestimate the costs of system level activities such as integration and documentation.

PRICING TO WIN

- The project costs whatever the customer has to spend on it.
- Advantages:
  - You get the contract.
- Disadvantages:
  - The probability that the customer gets the system he or she wants is small. Costs do not accurately reflect the work required.

PRICING TO WIN

- This approach may seem unethical and un-businesslike.
- However, when detailed information is lacking it may be the only appropriate strategy.
- The project cost is agreed on the basis of an outline proposal and the development is constrained by that cost.
- A detailed specification may be negotiated or an evolutionary approach used for system development.

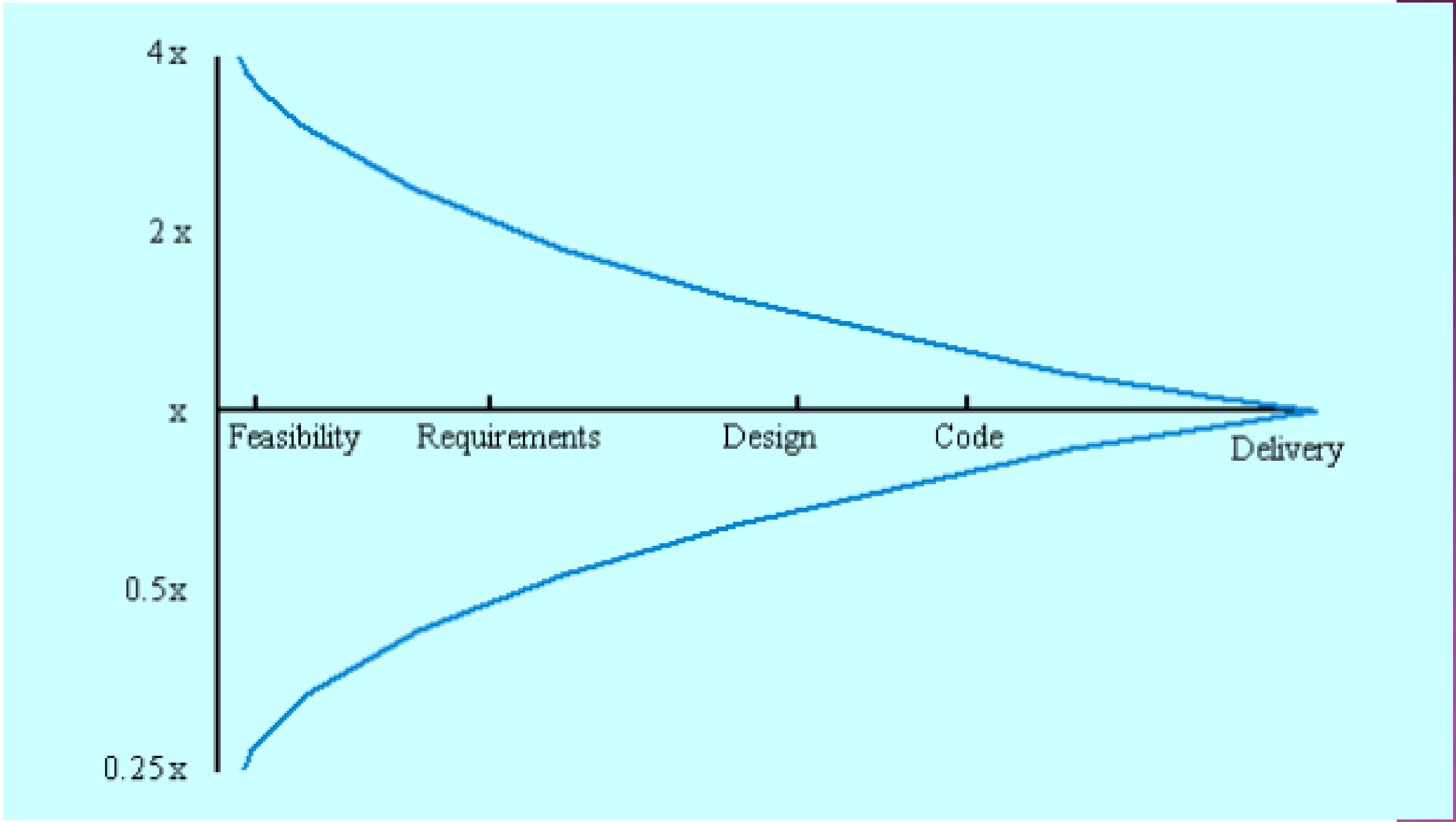
ALGORITHMIC COST MODELLING

- Cost is estimated as a mathematical function of product, project and process attributes whose values are estimated by project managers:
  - Effort = A x Size^B x M
  - A is an organisation-dependent constant, B reflects the disproportionate effort for large projects and M is a multiplier reflecting product, process and people attributes.
- The most commonly used product attribute for cost estimation is code size.
- Most models are similar but they use different values for A, B and M.

ESTIMATION ACCURACY

- The size of a software system can only be known accurately when it is finished.
- Several factors influence the final size
  - Use of COTS and components;
  - Programming language;
  - Distribution of system.
- As the development process progresses then the size estimate becomes more accurate.

ESTIMATE UNCERTAINTY



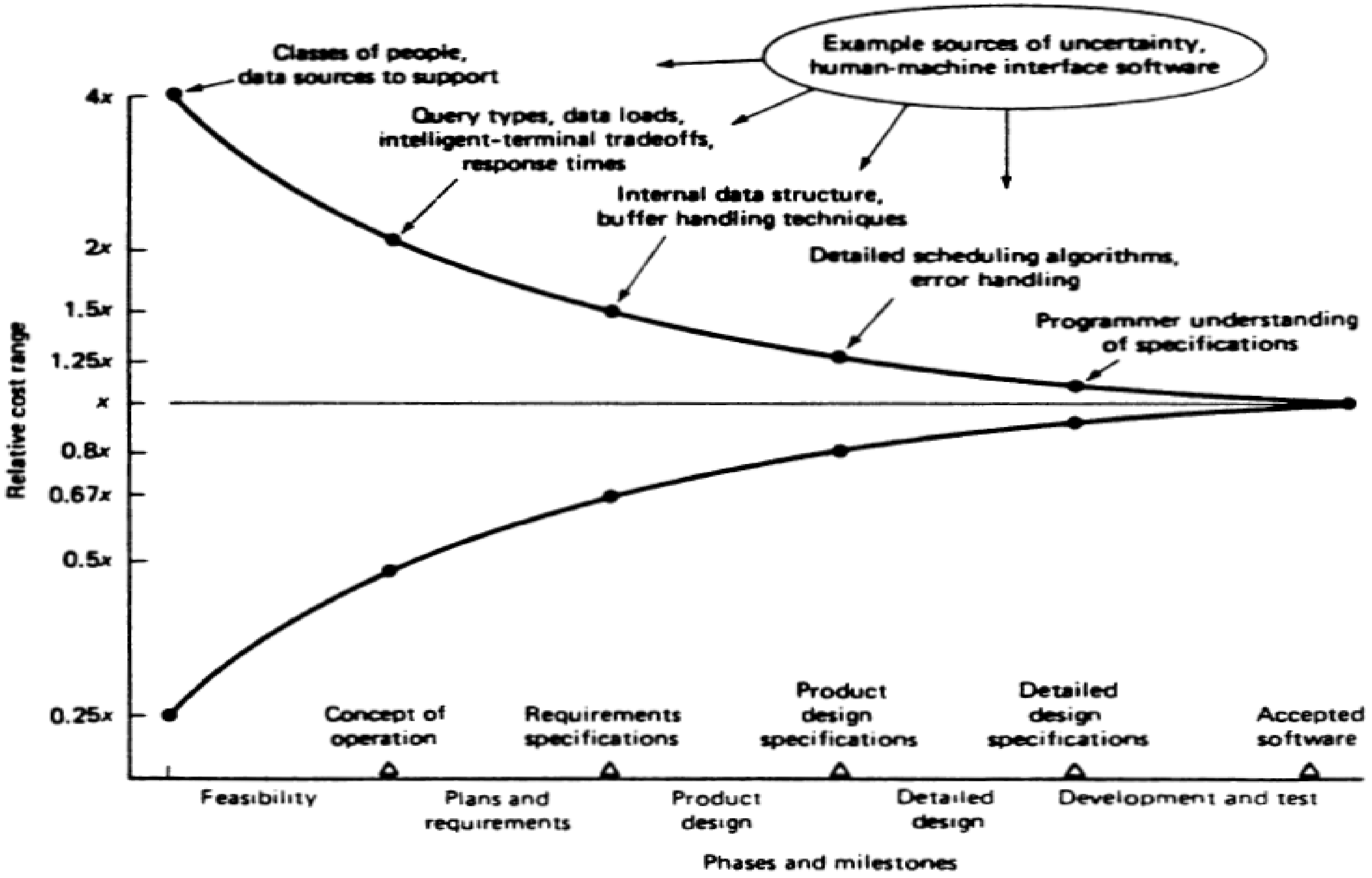


Fig. 3. Software cost estimation accuracy versus phase.

DETAILED/ COMPLETE COCOMO MODEL

DISADVANTAGE OF BASIC AND INTERMEDIATE COCOMO MODEL

- Consider a software product as a single homogeneous entity.
- Large software systems are made up of several smaller sub-systems or modules with different characteristics.
- Some sub-system may be considered as organic type, some may be considered as embedded.
- For some the reliability requirements may be high and so on.

COMPLETE COCOMO MODEL

- Complete COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step (analysis, design etc.) of the software engineering process.
- The complete model uses different effort multipliers for each cost driver attribute.
- These phase sensitive effort multipliers are each to determine the amount of effort required to complete each phase.

COMPLETE COCOMO

- In complete cocomo, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.
- The complete COCOMO model considers different subsystems of the software.
- Cost of each sub-system is estimated separately.
- Costs of the sub-systems are added to obtain total cost.
- Reduce the margin of error in the final estimation.

EXAMPLE OF COMPLETE COCOMO MODEL

- A Management Information System (MIS) for an organization having offices at several places across the country.
  - Database part (Semi-detached)
  - Graphical user interface (GUI) part (Organic)
  - Communication part (Embedded)
- The costs for these three components can be estimated separately, and summed up to give the overall cost of the system

- The Six phases of complete COCOMO are:
  - Planning and requirements
  - System design
  - Detailed design
  - Module code and test
  - Integration and test
  - Cost Constructive model
- The effort is calculated as a function of program size and a set of cost drivers are given according to each phase of the software lifecycle

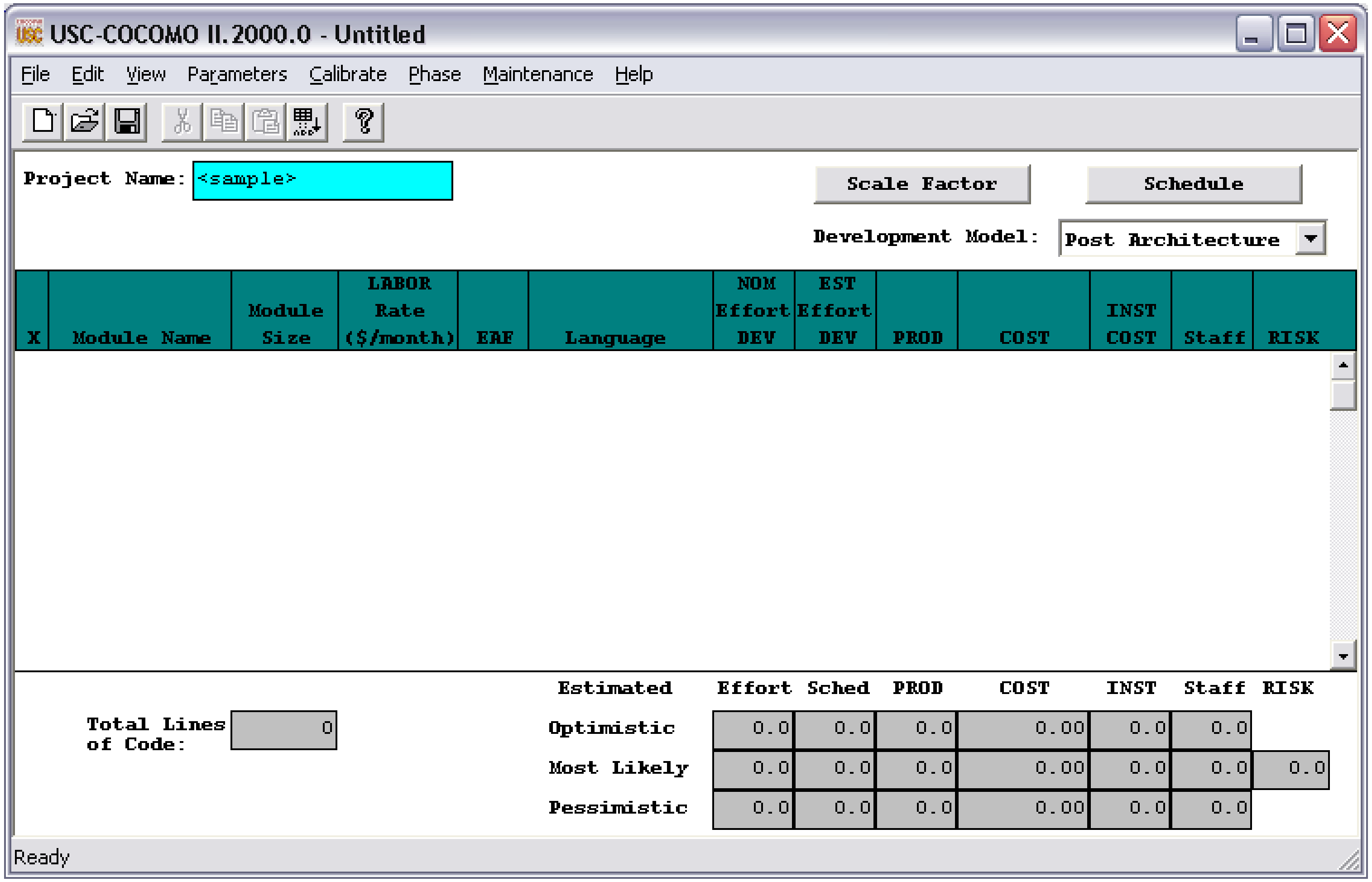
ADVANTAGES AND DISADVANTAGES OF COCOMO MODEL

- **Advantages**
  - Easy to estimate the total cost of the project.
  - Easy to implement with various factors.
  - Works on historical data and hence is more predictable and accurate.
- **Disadvantages**
  - It ignores requirements, customer skills, and hardware issues.
  - Extremely vulnerable to misclassification of the development mode
  - It mostly depends on time factors.
  - It is hard to accurately estimate KDSI early on in the project, when most effort estimates are required

COCOMO TOOLS

- USC Tool
- Calico
- Spreadsheet
  - http://www.softstarsystems.com/calico.htm

USC TOOL



USC-COCOMO II.2000.0 - Untitled

File Edit View Parameters Calibrate Phase Maintenance Help

Project Name: <sample>

Scale Factor | Schedule

Development Model: Post Architecture

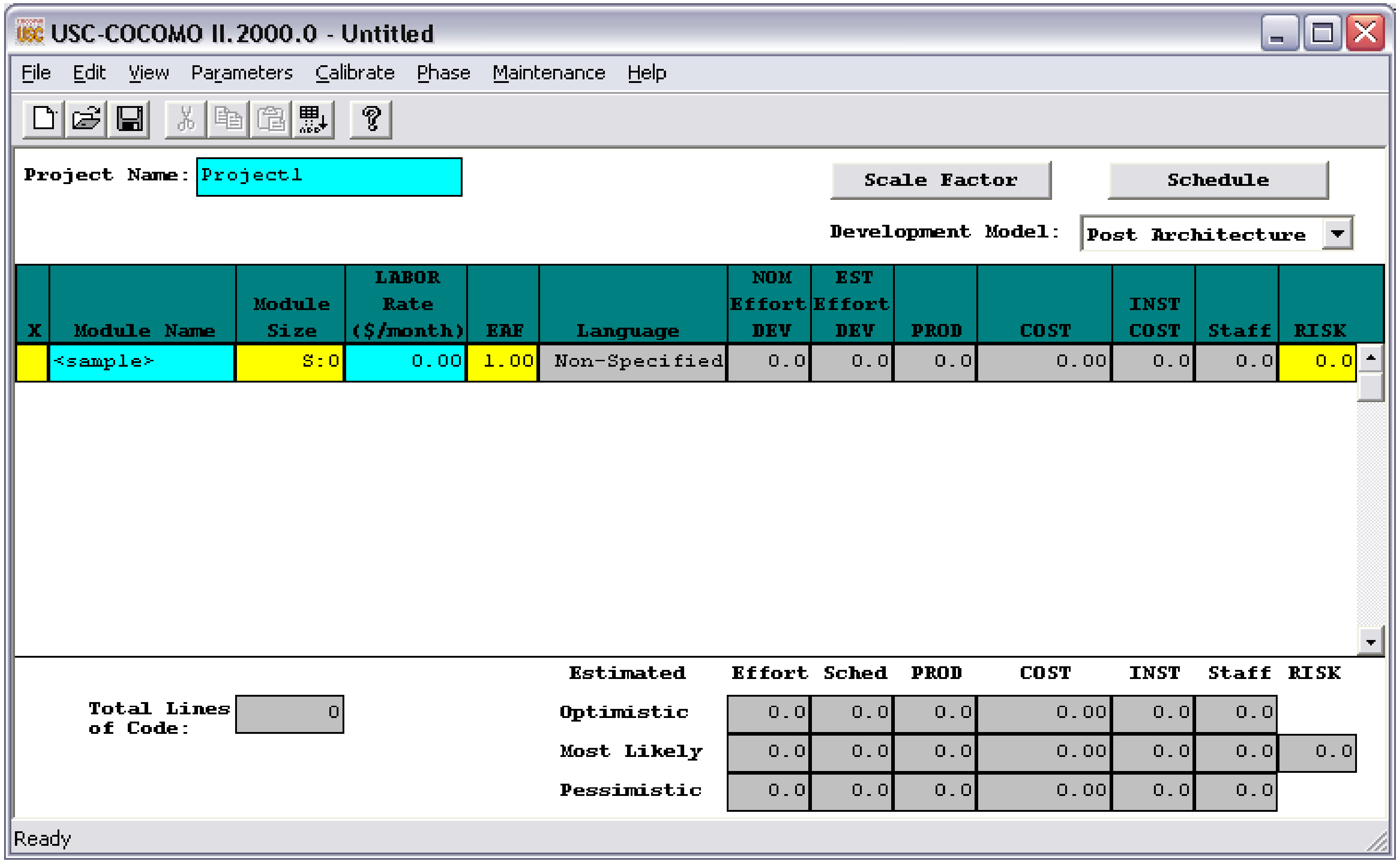
X	Module Name	Module Size	LABOR Rate ($/month)	EAF	Language	NOM Effort DEV	EST Effort DEV	PROD	COST	INST COST	Staff	RISK

Total Lines of Code: 0

Estimated	Effort	Sched	PROD	COST	INST	Staff	RISK
Optimistic	0.0	0.0	0.0	0.00	0.0	0.0	
Most Likely	0.0	0.0	0.0	0.00	0.0	0.0	0.0
Pessimistic	0.0	0.0	0.0	0.00	0.0	0.0	

Ready

USC TOOL


USC-COCOMO II.2000.0 - Untitled
File Edit View Parameters Calibrate Phase Maintenance Help
Scale Factor
Schedule

Project Name: Project1

Development Model: Post Architecture

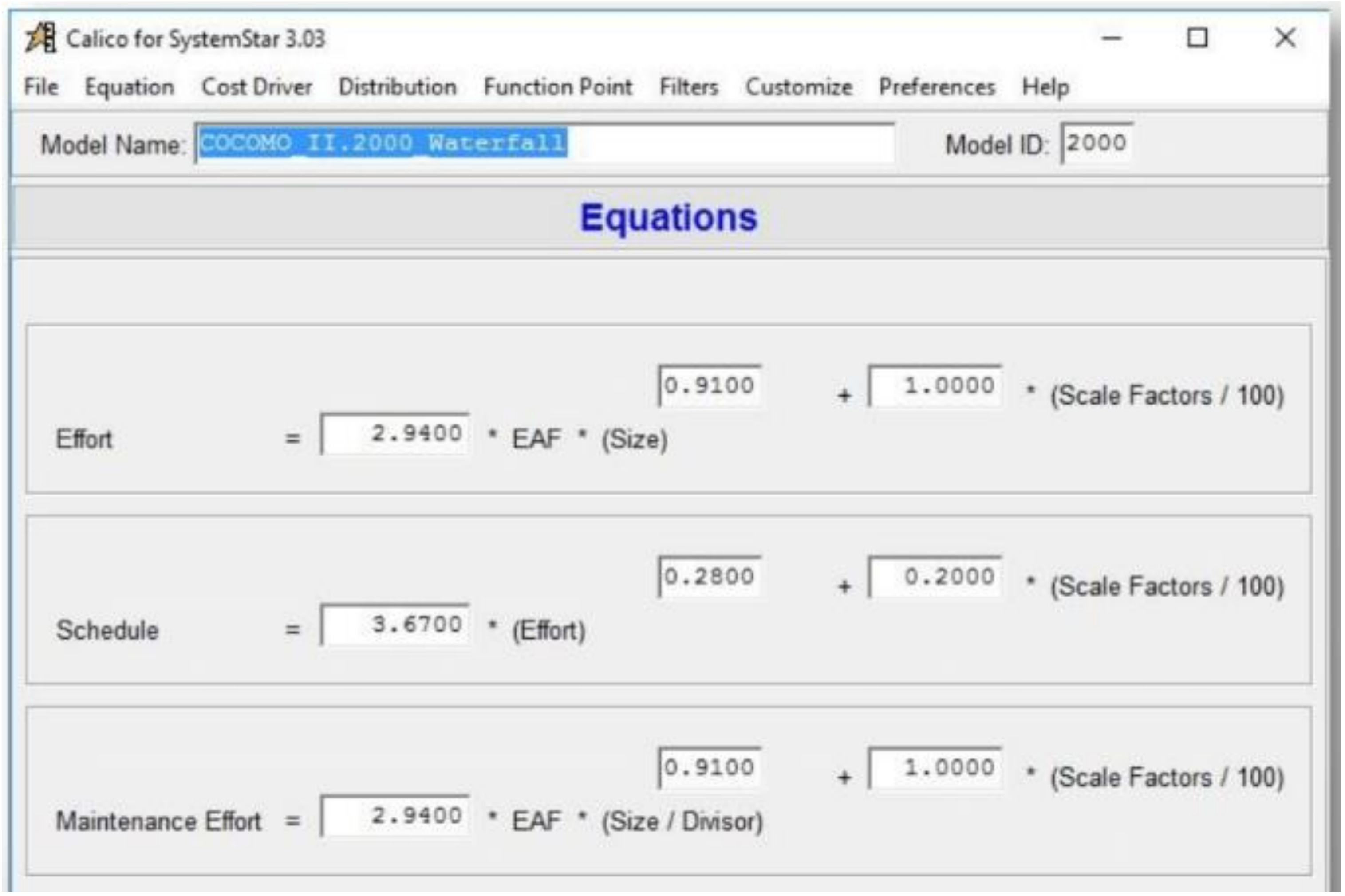
X	Module Name	Module Size	LABOR Rate ($/month)	EAF	Language	NOM Effort DEV	EST Effort DEV	PROD	COST	INST COST	Staff	RISK
	<sample>	S:0	0.00	1.00	Non-Specified	0.0	0.0	0.0	0.00	0.0	0.0	0.0

Total Lines of Code: 0

Estimated	Effort	Sched	PROD	COST	INST	Staff	RISK
Optimistic	0.0	0.0	0.0	0.00	0.0	0.0	
Most Likely	0.0	0.0	0.0	0.00	0.0	0.0	0.0
Pessimistic	0.0	0.0	0.0	0.00	0.0	0.0	

Ready

CALICO TOOL


Calico for SystemStar 3.03
File Equation Cost Driver Distribution Function Point Filters Customize Preferences Help

Model Name: COCOMO II.2000 Waterfall

Model ID: 2000

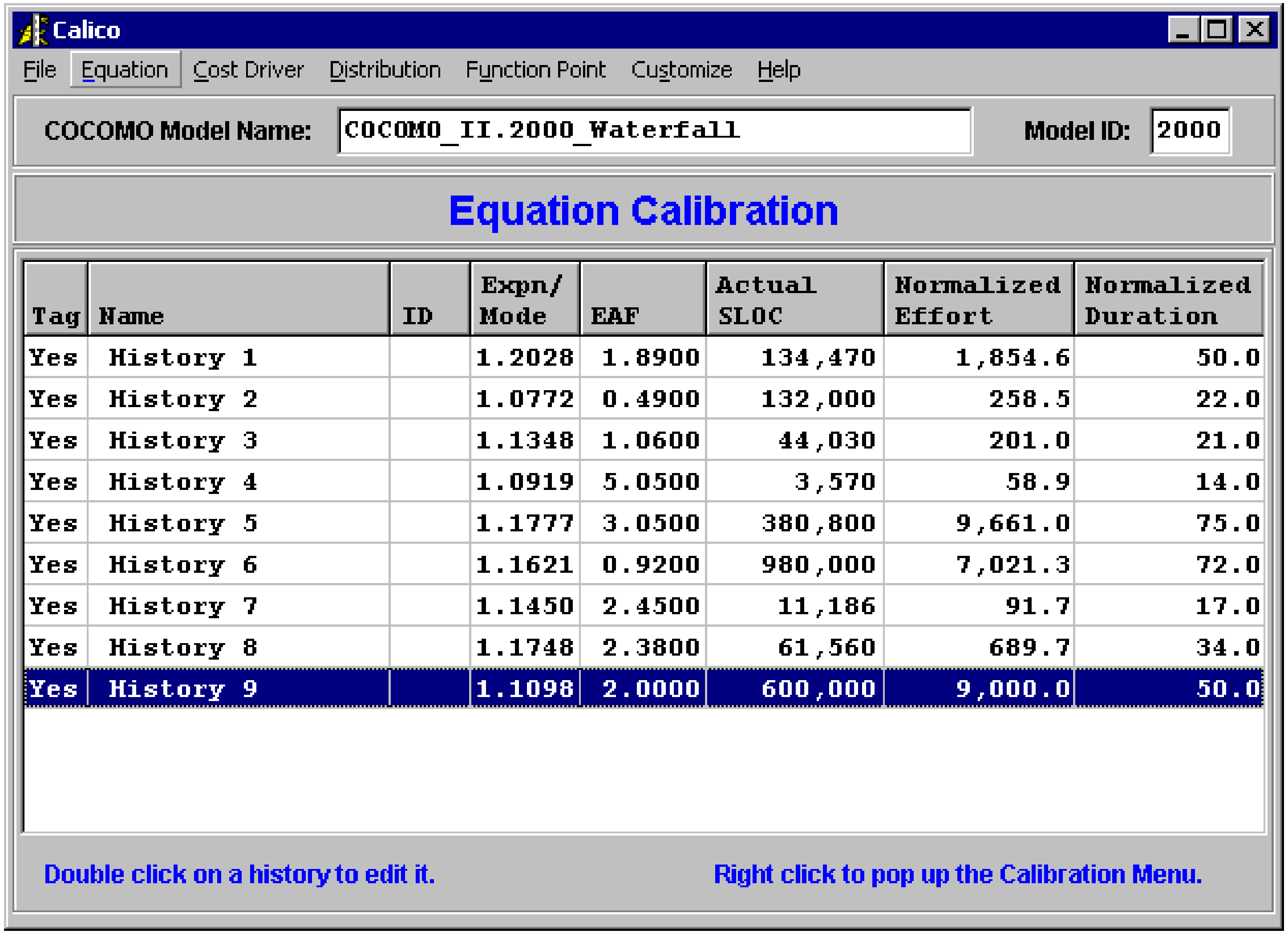
Equations

Effort = 2.9400 * EAF * (Size)^(0.9100 + 1.0000 * (Scale Factors / 100))

Schedule = 3.6700 * (Effort)^(0.2800 + 0.2000 * (Scale Factors / 100))

Maintenance Effort = 2.9400 * EAF * (Size / Divisor)^(0.9100 + 1.0000 * (Scale Factors / 100))

CALICO TOOL



COCOMO Model Name: COCOMO_II.2000_Waterfall

Model ID: 2000

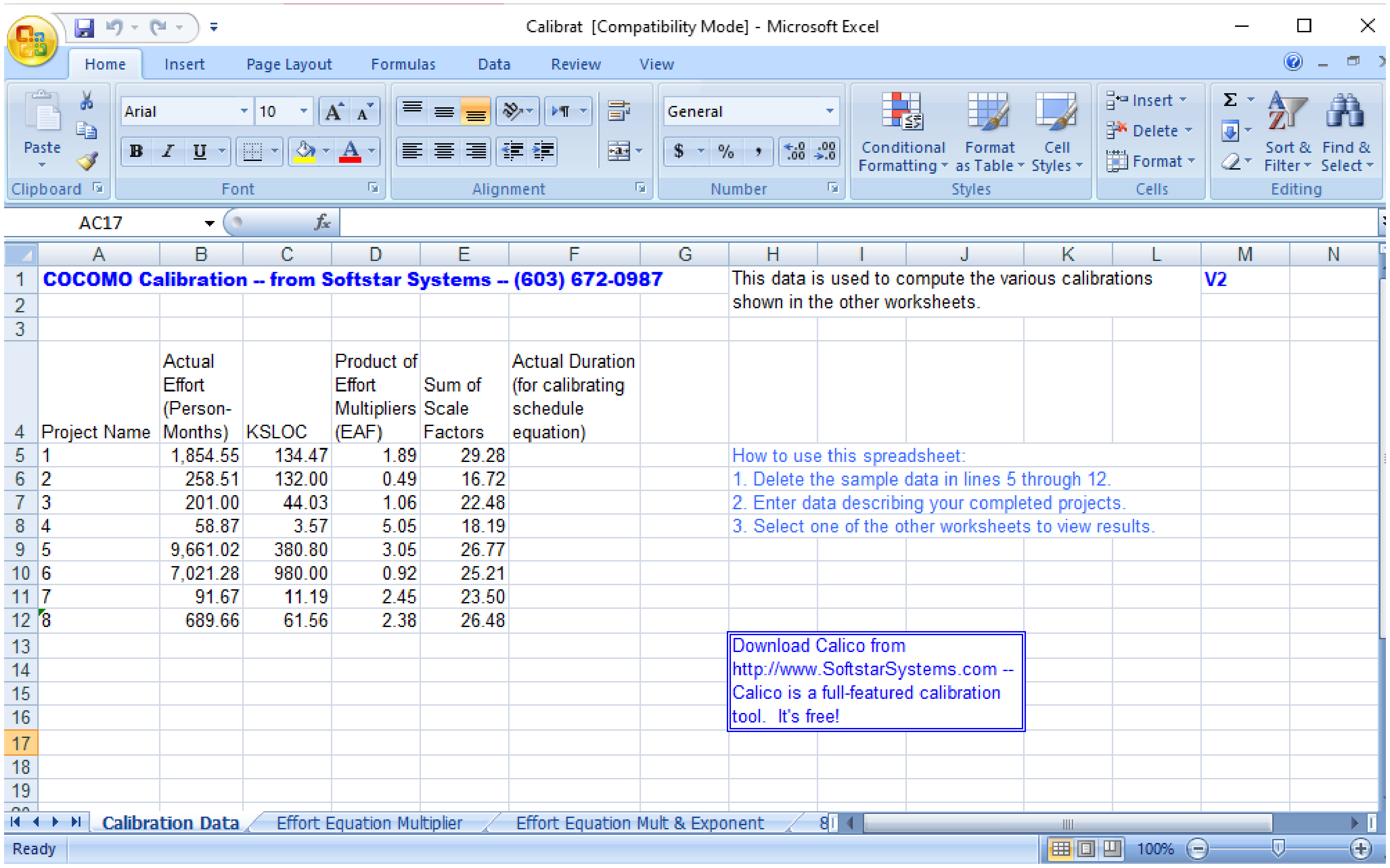
Equation Calibration

Tag	Name	ID	Expn/ Mode	EAF	Actual SLOC	Normalized Effort	Normalized Duration
Yes	History 1		1.2028	1.8900	134,470	1,854.6	50.0
Yes	History 2		1.0772	0.4900	132,000	258.5	22.0
Yes	History 3		1.1348	1.0600	44,030	201.0	21.0
Yes	History 4		1.0919	5.0500	3,570	58.9	14.0
Yes	History 5		1.1777	3.0500	380,800	9,661.0	75.0
Yes	History 6		1.1621	0.9200	980,000	7,021.3	72.0
Yes	History 7		1.1450	2.4500	11,186	91.7	17.0
Yes	History 8		1.1748	2.3800	61,560	689.7	34.0
Yes	History 9		1.1098	2.0000	600,000	9,000.0	50.0

Double click on a history to edit it.

Right click to pop up the Calibration Menu.

SPREADSHEET



COCOMO Calibration -- from Softstar Systems -- (603) 672-0987

This data is used to compute the various calibrations shown in the other worksheets.

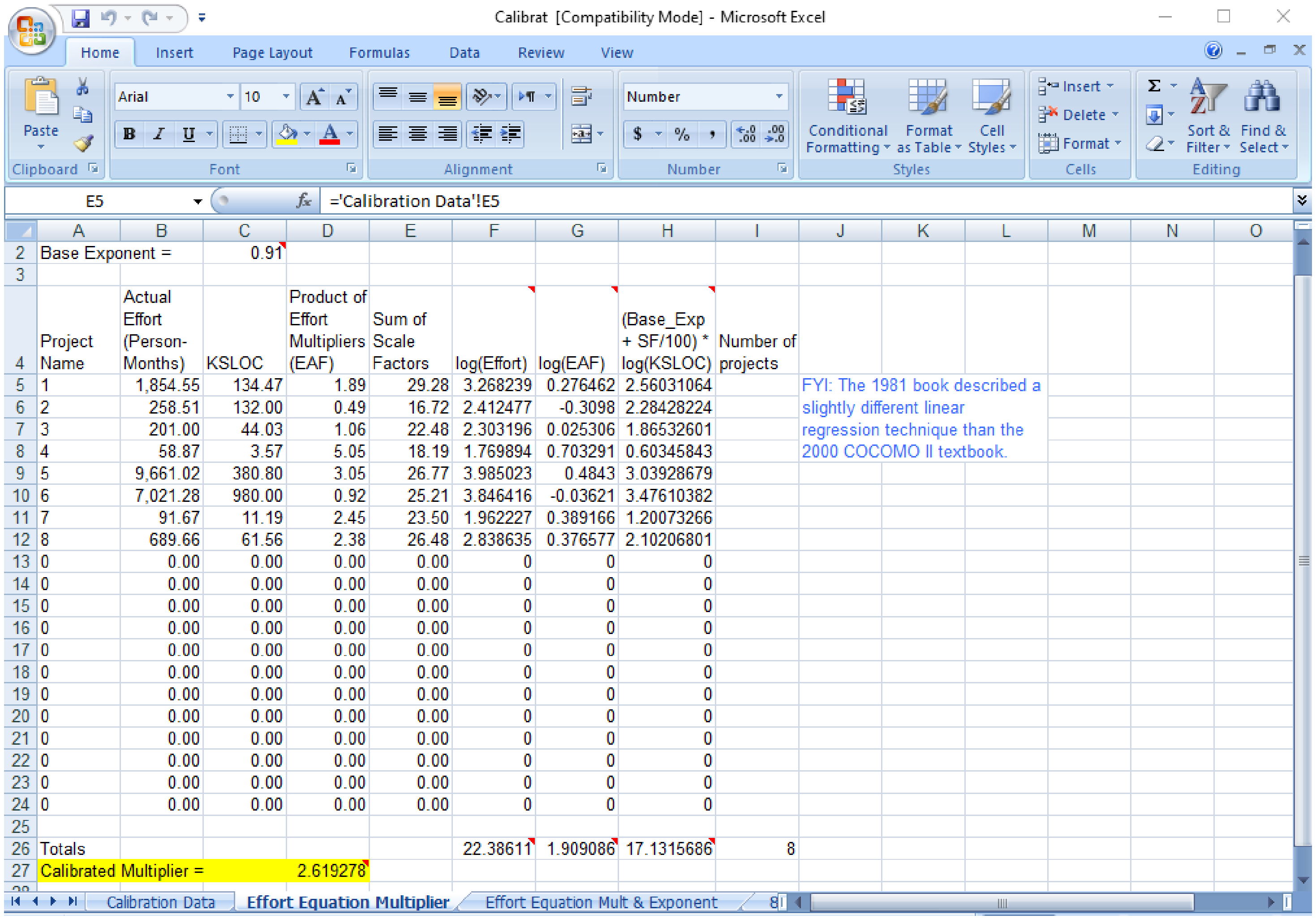
V2

Project Name	Actual Effort (Person-Months)	KSLOC	Product of Effort Multipliers (EAF)	Sum of Scale Factors	Actual Duration (for calibrating schedule equation)
1	1,854.55	134.47	1.89	29.28	
2	258.51	132.00	0.49	16.72	
3	201.00	44.03	1.06	22.48	
4	58.87	3.57	5.05	18.19	
5	9,661.02	380.80	3.05	26.77	
6	7,021.28	980.00	0.92	25.21	
7	91.67	11.19	2.45	23.50	
8	689.66	61.56	2.38	26.48	

How to use this spreadsheet:
1. Delete the sample data in lines 5 through 12.
2. Enter data describing your completed projects.
3. Select one of the other worksheets to view results.

Download Calico from http://www.SoftstarSystems.com -- Calico is a full-featured calibration tool. It's free!

SPREADSHEET

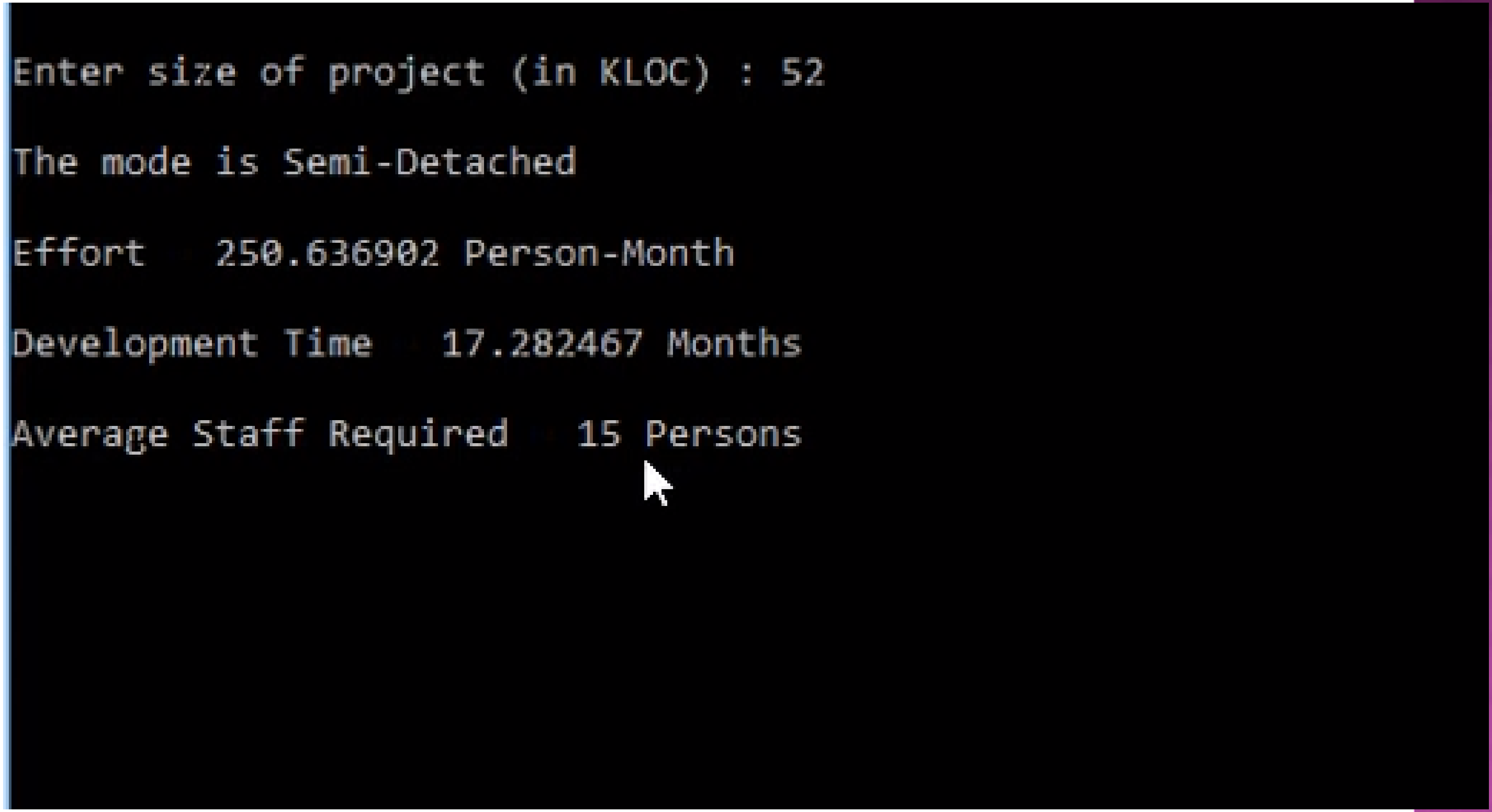


E5 | ='Calibration Data'!E5

	A	B	C	D	E	F	G	H	I
2	Base Exponent =		0.91						
3									
4	Project Name	Actual Effort (Person-Months)	KSLOC	Product of Effort Multipliers (EAF)	Sum of Scale Factors	log(Effort)	log(EAF)	(Base_Exp + SF/100) * log(KSLOC)	Number of projects
5	1	1,854.55	134.47	1.89	29.28	3.268239	0.276462	2.56031064	
6	2	258.51	132.00	0.49	16.72	2.412477	-0.3098	2.28428224	
7	3	201.00	44.03	1.06	22.48	2.303196	0.025306	1.86532601	
8	4	58.87	3.57	5.05	18.19	1.769894	0.703291	0.60345843	
9	5	9,661.02	380.80	3.05	26.77	3.985023	0.4843	3.03928679	
10	6	7,021.28	980.00	0.92	25.21	3.846416	-0.03621	3.47610382	
11	7	91.67	11.19	2.45	23.50	1.962227	0.389166	1.20073266	
12	8	689.66	61.56	2.38	26.48	2.838635	0.376577	2.10206801	
13	0	0.00	0.00	0.00	0.00	0	0	0	
14	0	0.00	0.00	0.00	0.00	0	0	0	
15	0	0.00	0.00	0.00	0.00	0	0	0	
16	0	0.00	0.00	0.00	0.00	0	0	0	
17	0	0.00	0.00	0.00	0.00	0	0	0	
18	0	0.00	0.00	0.00	0.00	0	0	0	
19	0	0.00	0.00	0.00	0.00	0	0	0	
20	0	0.00	0.00	0.00	0.00	0	0	0	
21	0	0.00	0.00	0.00	0.00	0	0	0	
22	0	0.00	0.00	0.00	0.00	0	0	0	
23	0	0.00	0.00	0.00	0.00	0	0	0	
24	0	0.00	0.00	0.00	0.00	0	0	0	
25									
26	Totals					22.38611	1.909086	17.1315686	8
27	Calibrated Multiplier =			2.619278					

FYI: The 1981 book described a slightly different linear regression technique than the 2000 COCOMO II textbook.

C++ PROGRAM FOR BASIC EQUATION



```
Enter size of project (in KLOC) : 52

The mode is Semi-Detached

Effort   250.636902 Person-Month

Development Time   17.282467 Months

Average Staff Required   15 Persons
```