This image cannot currently be displayed.

# Chapter 5

# Software Project Planning

# Declaration

- These slides are made for UIT, BU students only. I am not holding any copy write of it as I had collected these study materials from different books and websites etc. I have not mentioned those to avoid complexity.

# Topics as per Syllabus

**Techniques for Software Size and Cost Estimation: Software Project Planning:** Line of Codes method, Function Point Analysis for size estimation, Static Single variable and Static Multi Variable models for Cost Estimation. COCOMO and COCOMO-II.

# Software Project Planning

- After the finalization of SRS, we would like to estimate size, cost and development time of the project. Also, in many cases, customer may like to know the cost and development time even prior to finalization of the SRS.
- In order to conduct a successful software project, we must understand:
  - Scope of work to be done
  - Software Project Planning
  - The risk to be incurred
  - The resources required
  - The task to be accomplished
  - The cost to be expended
  - The schedule to be followed

# Software Project Planning

- Software planning begins before technical work starts, continues as the software evolves from concept to reality, and culminates only when the software is retired.

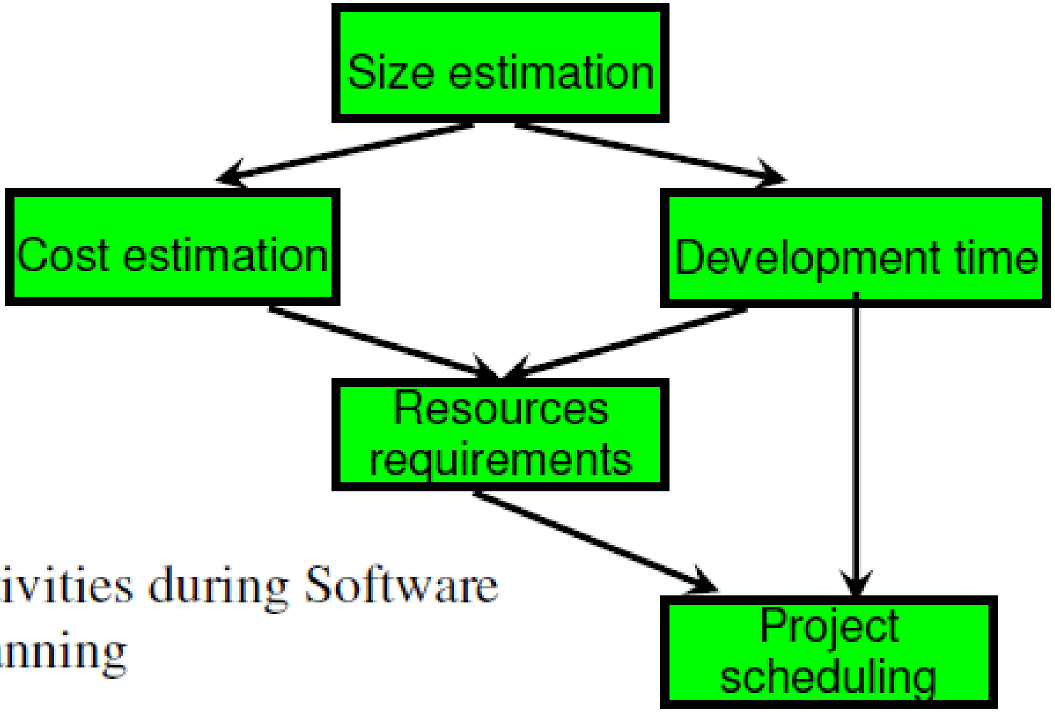


Fig. 1: Activities during Software Project Planning

# Project size estimation techniques

- Estimation of the size of software is an essential part of Software Project Management. It helps the project manager to further predict the effort and time which will be needed to build the project. Various measures are used in project size estimation. Some of these are:
  - Lines of Code
  - Number of entities in ER diagram
  - Total number of processes in detailed data flow diagram
  - Function points

# Lines of Code

- As the name suggest, LOC count the total number of lines of source code in a project. The units of LOC are:
  - KLOC- Thousand lines of code
  - NLOC- Non comment lines of code
  - KDSI- Thousands of delivered source instruction
- LOC is the simplest among all metrics available to estimate project size. This metric is very popular because it is the simplest to use. Using this metric, the project size is estimated by counting the number of source instructions in the developed program. Obviously, while counting the number of source instructions, lines used for commenting the code and the header lines should be ignored.

# Lines of Code

- Determining the LOC count at the end of a project is a very simple job. However, accurate estimation of the LOC count at the beginning of a project is very difficult. In order to estimate the LOC count at the beginning of a project, project managers usually divide the problem into modules, and each module into sub-modules and so on, until the sizes of the different leaf-level modules can be approximately predicted. To be able to do this, past experience in developing similar products is helpful. By using the estimation of the lowest level modules, project managers arrive at the total size estimation.

# Lines of Code

- **Advantages:**
  - Universally accepted and is used in many models like COCOMO.
  - Estimation is closer to developer's perspective.
  - Simple to use.
- **Disadvantages:**
  - Different programming languages contains different number of lines.
  - No proper industry standard exist for this technique.
  - It is difficult to estimate the size using this technique in early stages of project.

# Function Point Analysis - History

- The concept of Function Points was introduced by Alan Albrecht of IBM in 1979.
- In 1984, Albrecht refined the method.
- The first Function Point Guidelines were published in 1984.
- The International Function Point Users Group (IFPUG) is a US-based worldwide organization of Function Point Analysis metric software users. The **International Function Point Users Group (IFPUG)** is a non-profit, member-governed organization founded in 1986. IFPUG owns Function Point Analysis (FPA) as defined in ISO standard 20296:2009 which specifies the definitions, rules and steps for applying the IFPUG's functional size measurement (FSM) method. IFPUG maintains the Function Point Counting Practices Manual (CPM).

# Function Point Analysis - History

- CPM 2.0 was released in 1987, and since then there have been several iterations. CPM Release 4.3 was in 2010.
- The CPM Release 4.3.1 with incorporated ISO editorial revisions was in 2010. The ISO Standard (IFPUG FSM) - Functional Size Measurement that is a part of CPM 4.3.1 is a technique for measuring software in terms of the functionality it delivers. The CPM is an internationally approved standard under ISO/IEC 14143-1 Information Technology – Software Measurement.

# Function Point Analysis

- The steps in function point analysis are:
  - Count the number of functions of each proposed type.
  - Compute the Unadjusted Function Points (UFP).
  - Find Total Degree of Influence (TDI).
  - Compute Value Adjustment Factor (VAF).
  - Find the Function Point Count (FPC).

# Function Point Analysis

- **Count the number of functions of each proposed type:** Find the number of functions belonging to the following types:
  - External Inputs: Functions related to data entering the system.
  - External outputs: Functions related to data exiting the system.
  - External Inquiries: They leads to data retrieval from system but don’t change the system.
  - Internal Files: Logical files maintained within the system. Log files are not included here.
  - External interface Files: These are logical files held by other systems which are used by the system being analyzed.

# Function Point Analysis

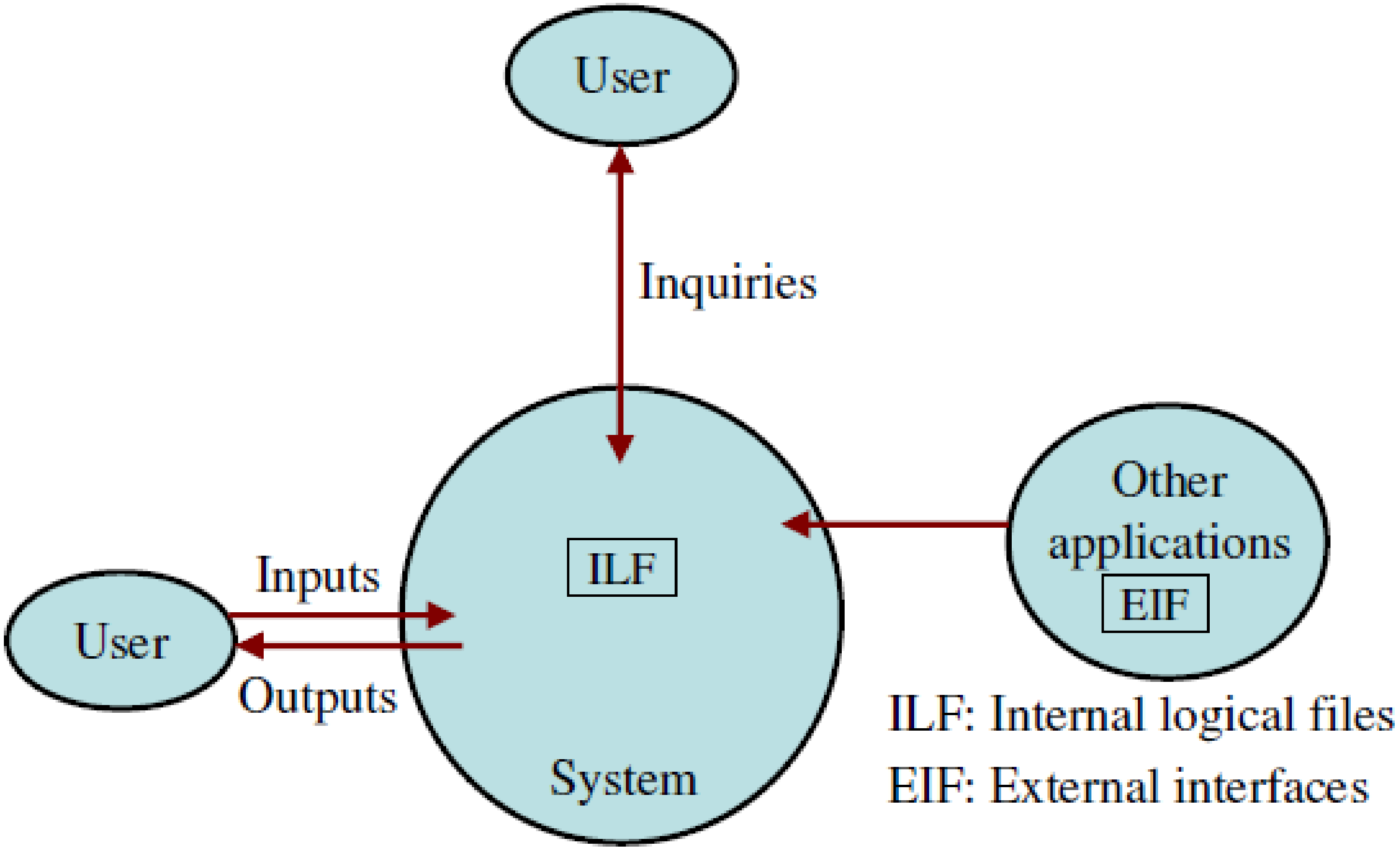
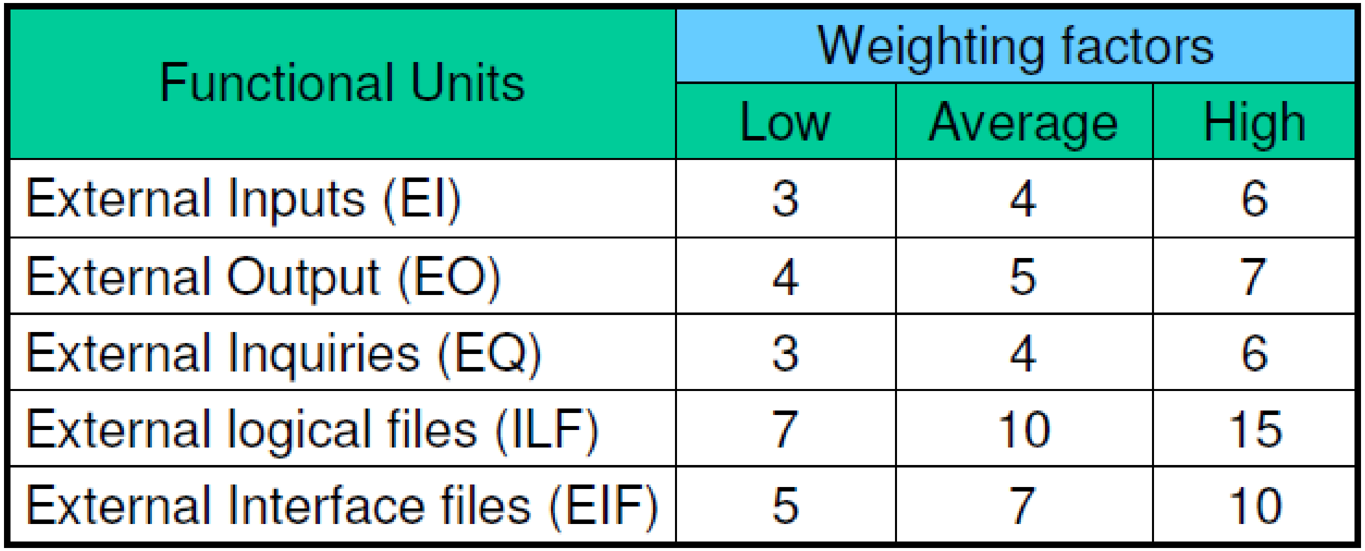


Fig. 3: FPAs functional units System

# Function Point Analysis

- **Compute the Unadjusted Function Points (UFP):** Categorize each of the five function types as simple, average or complex based on their complexity. Multiply count of each function type with its weighting factor and find the weighted sum. The weighting factors for each type based on their complexity are as follows:

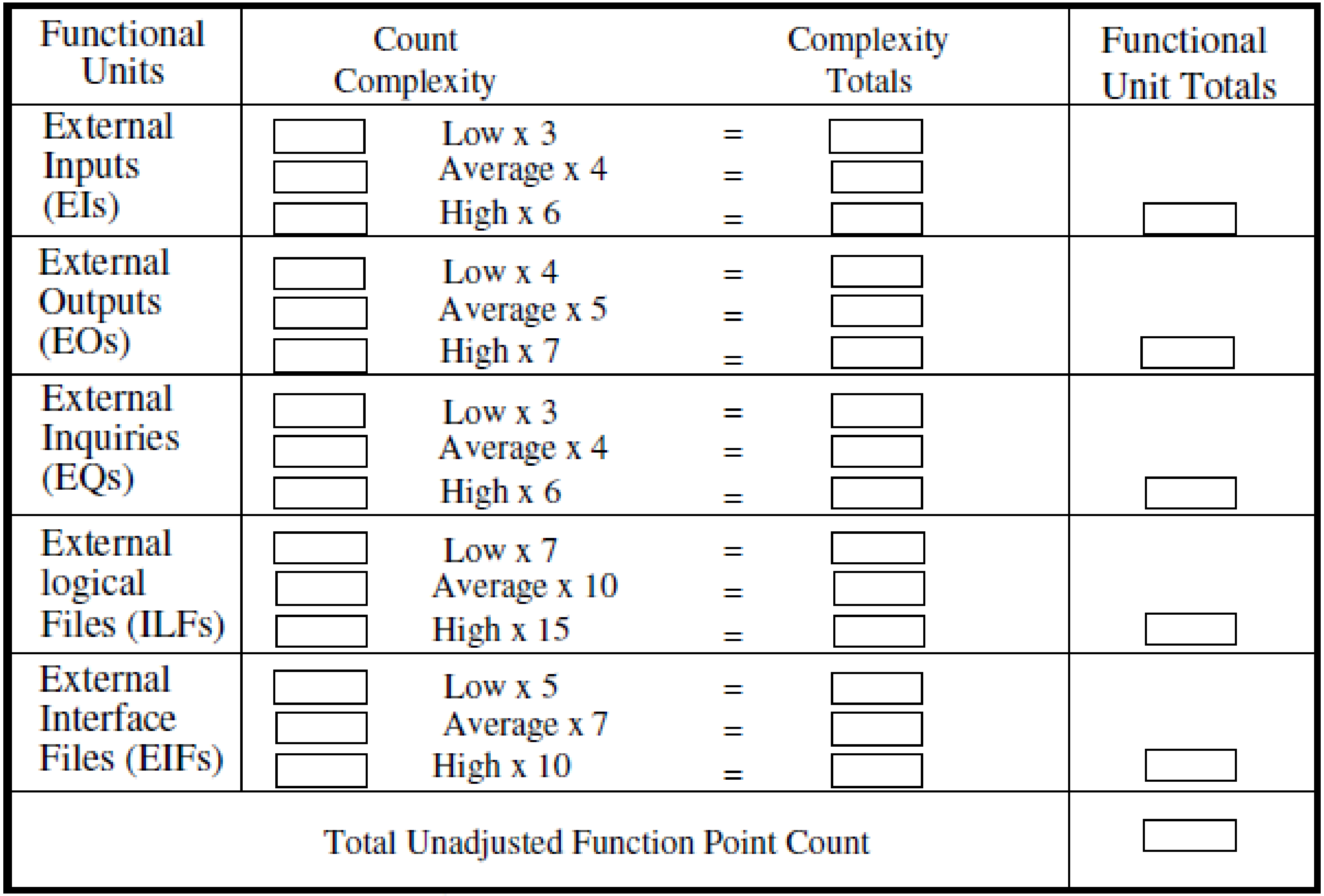


| Functional Units | Weighting factors | | |
|---|---|---|---|
| | Low | Average | High |
| External Inputs (EI) | 3 | 4 | 6 |
| External Output (EO) | 4 | 5 | 7 |
| External Inquiries (EQ) | 3 | 4 | 6 |
| External logical files (ILF) | 7 | 10 | 15 |
| External Interface files (EIF) | 5 | 7 | 10 |

Table 1 : Functional units with weighting factors

# Function Point Analysis

Table 2: UFP calculation table



| Functional Units | Count Complexity | Complexity Totals | Functional Unit Totals |
|---|---|---|---|
| External Inputs (EIs) | □ Low x 3<br>□ Average x 4<br>□ High x 6 | = □<br>= □<br>= □ | □ |
| External Outputs (EOs) | □ Low x 4<br>□ Average x 5<br>□ High x 7 | = □<br>= □<br>= □ | □ |
| External Inquiries (EQs) | □ Low x 3<br>□ Average x 4<br>□ High x 6 | = □<br>= □<br>= □ | □ |
| External logical Files (ILFs) | □ Low x 7<br>□ Average x 10<br>□ High x 15 | = □<br>= □<br>= □ | □ |
| External Interface Files (EIFs) | □ Low x 5<br>□ Average x 7<br>□ High x 10 | = □<br>= □<br>= □ | □ |
| Total Unadjusted Function Point Count | | | □ |

Software Engineering (3rd ed.), By K.K Aggarwal & Yogesh Singh, Copyright © New Age International Publishers, 2007

# Function Point Analysis

- The weighting factors are identified for all functional units and multiplied with the functional units accordingly. The procedure for the calculation of Unadjusted Function Point (UFP) is given in table shown above.

# Function Point Analysis

- The procedure for the calculation of UFP in mathematical form is given below:

$$UFP = \sum_{i=1}^{5} \sum_{J=1}^{3} Z_{ij} w_{ij}$$

- Where i indicate the row and j indicates the column of Table 1
- $W_{ij}$ : It is the entry of the $i^{th}$ row and $j^{th}$ column of the table 1
- $Z_{ij}$ : It is the count of the number of functional units of Type *i* that have been classified as having the complexity corresponding to column *j*.

# Function Point Analysis

- **Find Total Degree of Influence (TDI):** Use the ’14 general characteristics’ of a system to find the degree of influence of each of them. The sum of all 14 degrees of influences will give the TDI. The range of TDI is 0 to 70.
- **Compute Value Adjustment Factor (VAF):** Use the following formula to calculate VAF = (TDI * 0.01) + 0.65
- **Find the Function Point Count (FPC):** Use the following formula to calculate FPC = UFP * VAF

# Function Point Analysis

1. Does the system require reliable backup and recovery ?

2. Is data communication required ?

3. Are there distributed processing functions ?

4. Is performance critical ?

5. Will the system run in an existing heavily utilized operational environment ?

6. Does the system require on line data entry ?

7. Does the on line data entry require the input transaction to be built over multiple screens or operations ?

# Function Point Analysis

8. Are the master files updated on line ?

9. Is the inputs, outputs, files, or inquiries complex ?

10. Is the internal processing complex ?

11. Is the code designed to be reusable ?

12. Are conversion and installation included in the design ?

13. Is the system designed for multiple installations in different organizations ?

14. Is the application designed to facilitate change and ease of use by the user ?

# Function Point Analysis

- Functions points may compute the following important metrics:
  - Productivity = FP / persons-months
  - Quality = Defects / FP
  - Cost = Rupees / FP
  - Documentation = Pages of documentation / FP
- These metrics are controversial and are not universally acceptable.

# Function Point Analysis

Consider a project with the following functional units:

Number of user inputs = 50

Number of user outputs = 40

Number of user enquiries = 35

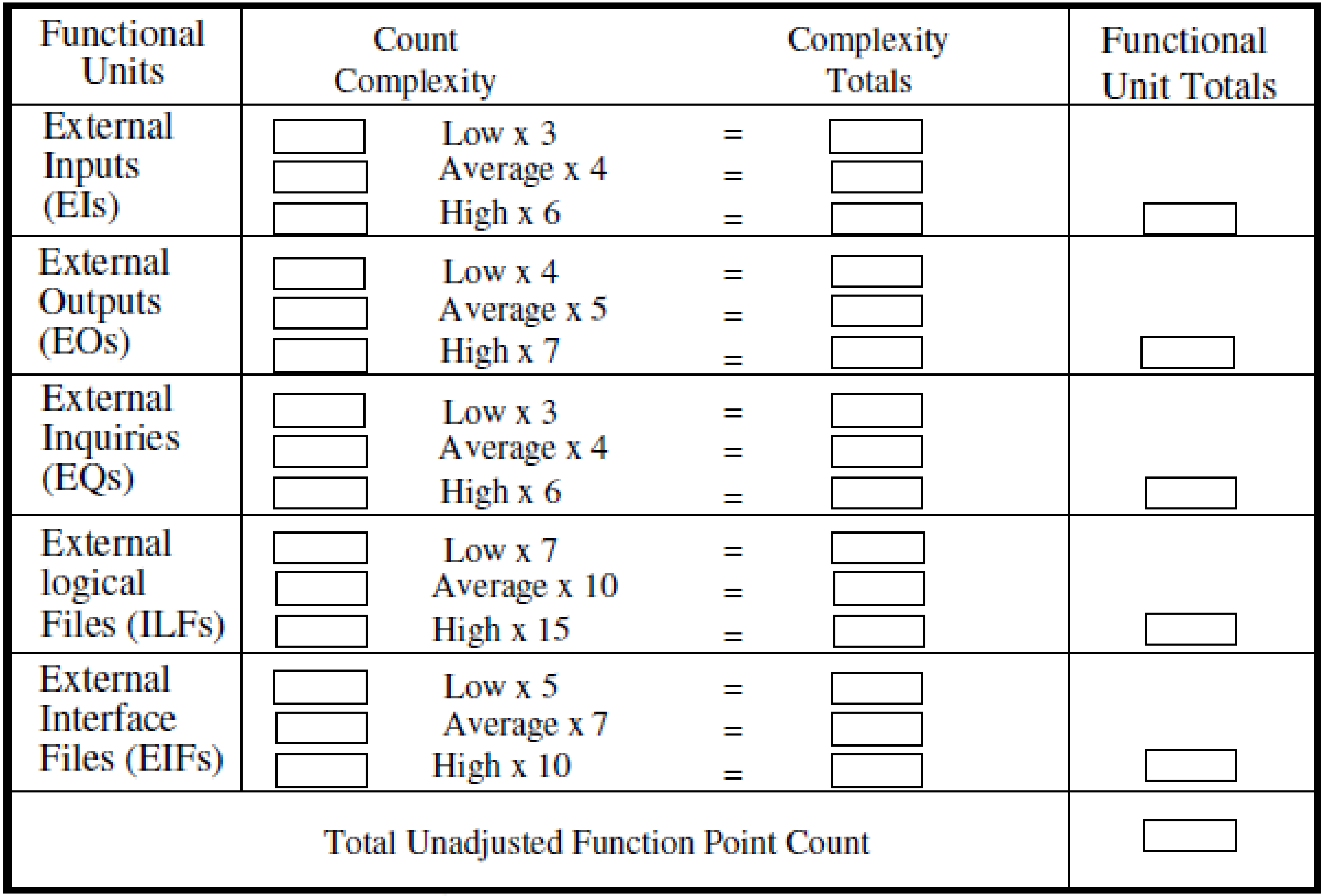
Number of user files = 06

Number of external interfaces = 04

Assume all complexity adjustment factors and weighting factors are average. Compute the function points for the project.

# Function Point Analysis

Table 2: UFP calculation table



| Functional Units | Count Complexity | | | Complexity Totals | Functional Unit Totals |
|---|---|---|---|---|---|
| External Inputs (EIs) | □ | Low x 3 | = | □ | |
| | □ | Average x 4 | = | □ | |
| | □ | High x 6 | = | □ | □ |
| External Outputs (EOs) | □ | Low x 4 | = | □ | |
| | □ | Average x 5 | = | □ | |
| | □ | High x 7 | = | □ | □ |
| External Inquiries (EQs) | □ | Low x 3 | = | □ | |
| | □ | Average x 4 | = | □ | |
| | □ | High x 6 | = | □ | □ |
| External logical Files (ILFs) | □ | Low x 7 | = | □ | |
| | □ | Average x 10 | = | □ | |
| | □ | High x 15 | = | □ | □ |
| External Interface Files (EIFs) | □ | Low x 5 | = | □ | |
| | □ | Average x 7 | = | □ | |
| | □ | High x 10 | = | □ | □ |
| Total Unadjusted Function Point Count | | | | | □ |

Software Engineering (3rd ed.), By K.K Aggarwal & Yogesh Singh, Copyright © New Age International Publishers, 2007

# Function Point Analysis

## Solution

We know

$$UFP = \sum_{i=1}^{5} \sum_{J=1}^{3} Z_{ij} w_{ij}$$

UFP = 50 x 4 + 40 x 5 + 35 x 4 + 6 x 10 + 4 x 7

= 200 + 200 + 140 + 60 + 28 = 628

CAF = $(0.65 + 0.01\ \Sigma F_i)$

= (0.65 + 0.01 (14 x 3)) = 0.65 + 0.42 = 1.07

FP = UFP x CAF

= 628 x 1.07 = 672

# Function Point Analysis

An application has the following:

10 low external inputs, 12 high external outputs, 20 low internal logical files, 15 high external interface files, 12 average external inquiries, and a value of complexity adjustment factor of 1.10.

What are the unadjusted and adjusted function point counts ?

# Function Point Analysis

Solution

Unadjusted function point counts may be calculated using as:

$$UFP = \sum_{i=1}^{5} \sum_{J=1}^{3} Z_{ij} w_{ij}$$

= 10 x 3 + 12 x 7 + 20 x 7 + 15 + 10 + 12 x 4

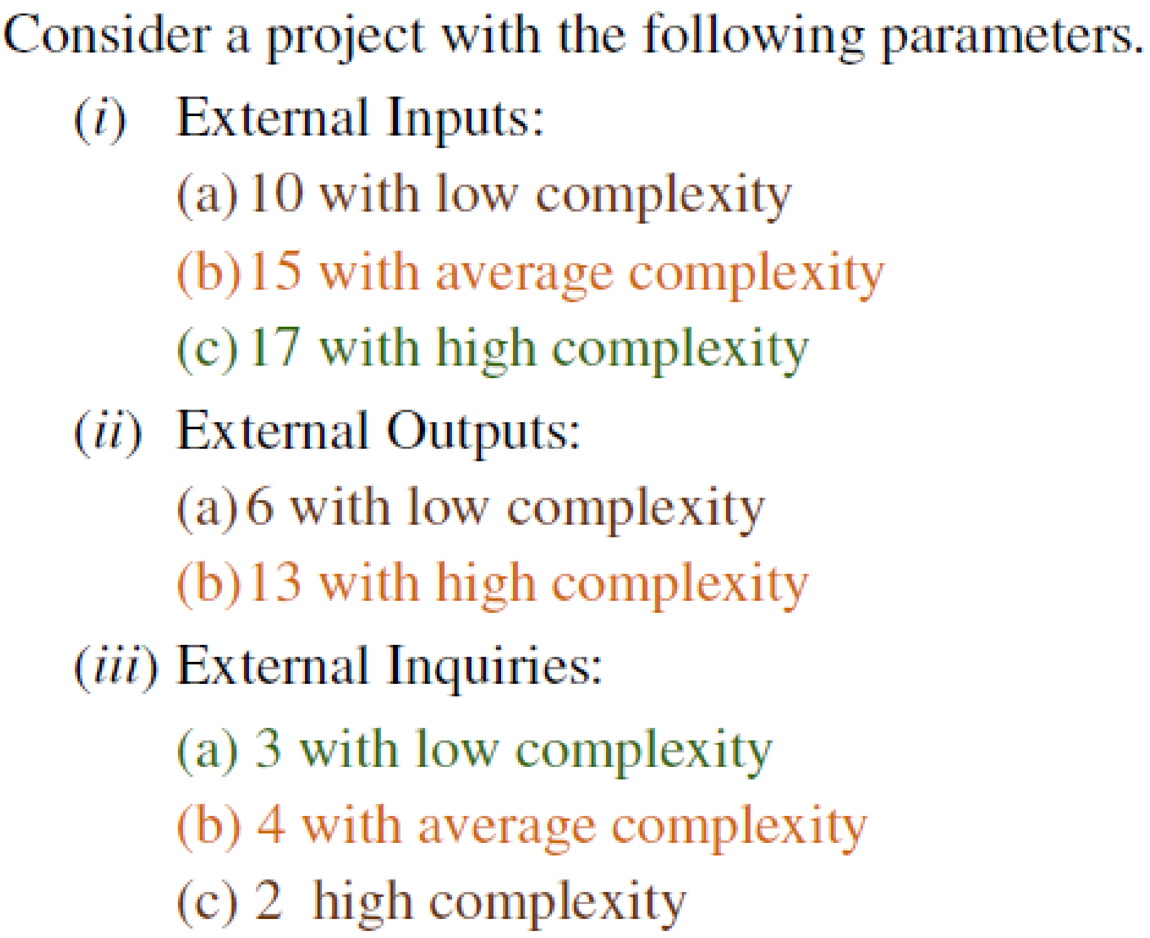
= 30 + 84 +140 + 150 + 48

= 452

FP = UFP x CAF

= 452 x 1.10 = 497.2.

# Function Point Analysis



Consider a project with the following parameters.

(*i*) External Inputs:

(a) 10 with low complexity

(b) 15 with average complexity

(c) 17 with high complexity

(*ii*) External Outputs:

(a) 6 with low complexity

(b) 13 with high complexity

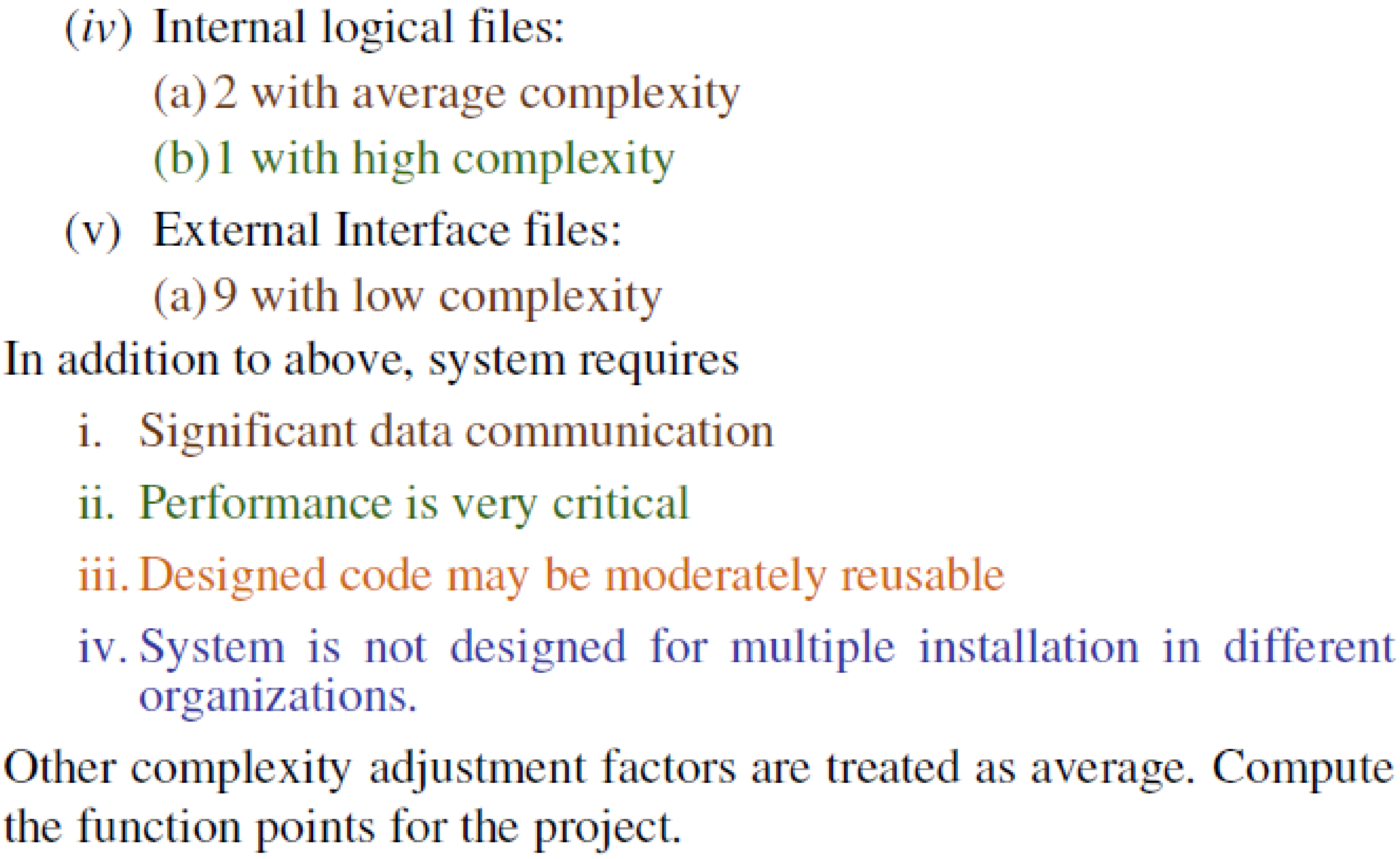
(*iii*) External Inquiries:

(a) 3 with low complexity

(b) 4 with average complexity

(c) 2 high complexity

# Function Point Analysis



(*iv*) Internal logical files:

(a) 2 with average complexity

(b) 1 with high complexity

(v) External Interface files:

(a) 9 with low complexity

In addition to above, system requires

i. Significant data communication

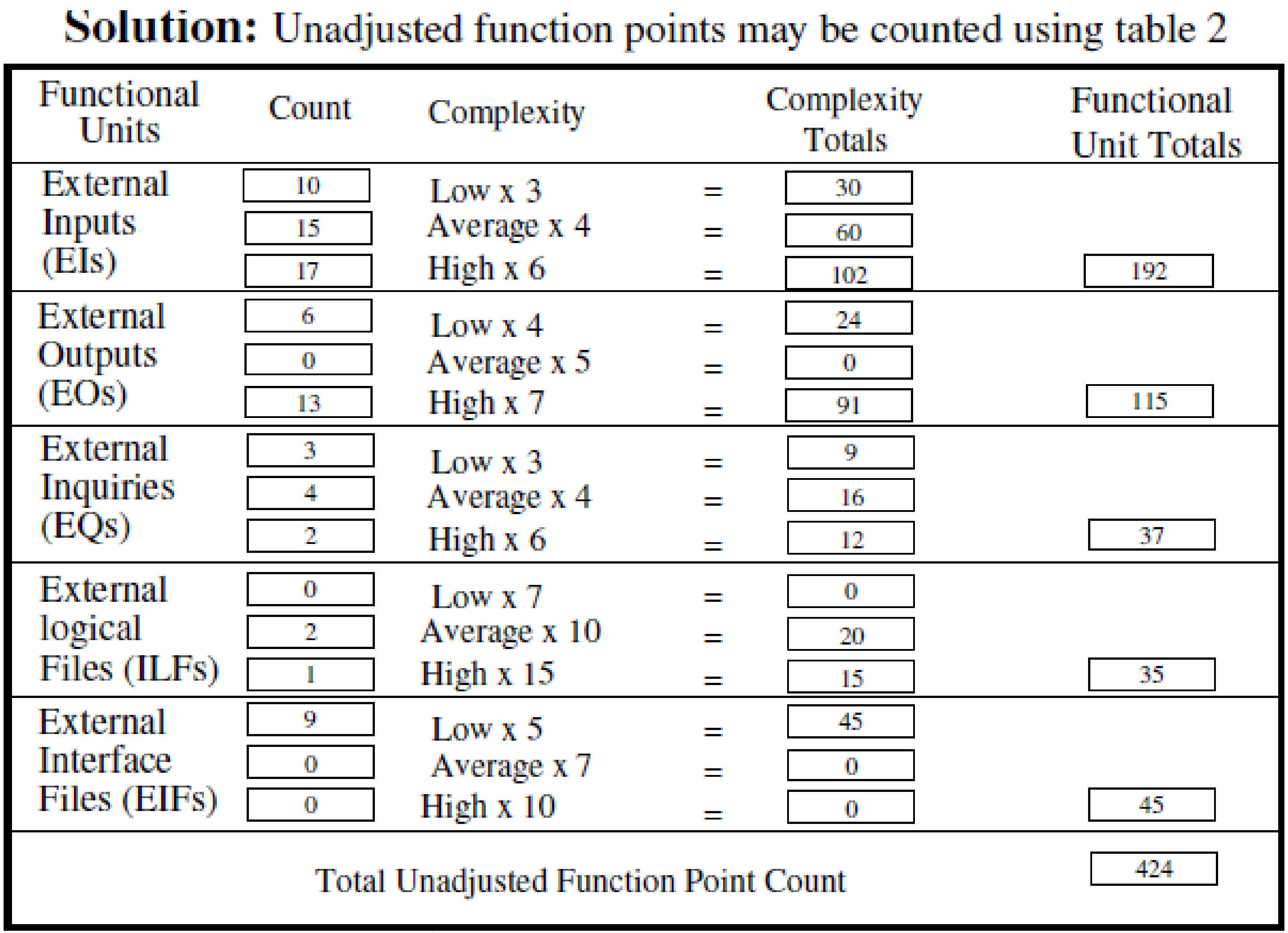
ii. Performance is very critical

iii. Designed code may be moderately reusable

iv. System is not designed for multiple installation in different organizations.

Other complexity adjustment factors are treated as average. Compute the function points for the project.

# Function Point Analysis



**Solution:** Unadjusted function points may be counted using table 2

| Functional Units | Count | Complexity | | Complexity Totals | Functional Unit Totals |
|---|---|---|---|---|---|
| External Inputs (EIs) | 10 | Low x 3 | = | 30 | |
| | 15 | Average x 4 | = | 60 | |
| | 17 | High x 6 | = | 102 | 192 |
| External Outputs (EOs) | 6 | Low x 4 | = | 24 | |
| | 0 | Average x 5 | = | 0 | |
| | 13 | High x 7 | = | 91 | 115 |
| External Inquiries (EQs) | 3 | Low x 3 | = | 9 | |
| | 4 | Average x 4 | = | 16 | |
| | 2 | High x 6 | = | 12 | 37 |
| External logical Files (ILFs) | 0 | Low x 7 | = | 0 | |
| | 2 | Average x 10 | = | 20 | |
| | 1 | High x 15 | = | 15 | 35 |
| External Interface Files (EIFs) | 9 | Low x 5 | = | 45 | |
| | 0 | Average x 7 | = | 0 | |
| | 0 | High x 10 | = | 0 | 45 |
| Total Unadjusted Function Point Count | | | | | 424 |

Software Engineering (3rd ed.), By K.K Aggarwal & Yogesh Singh, Copyright © New Age International Publishers, 2007

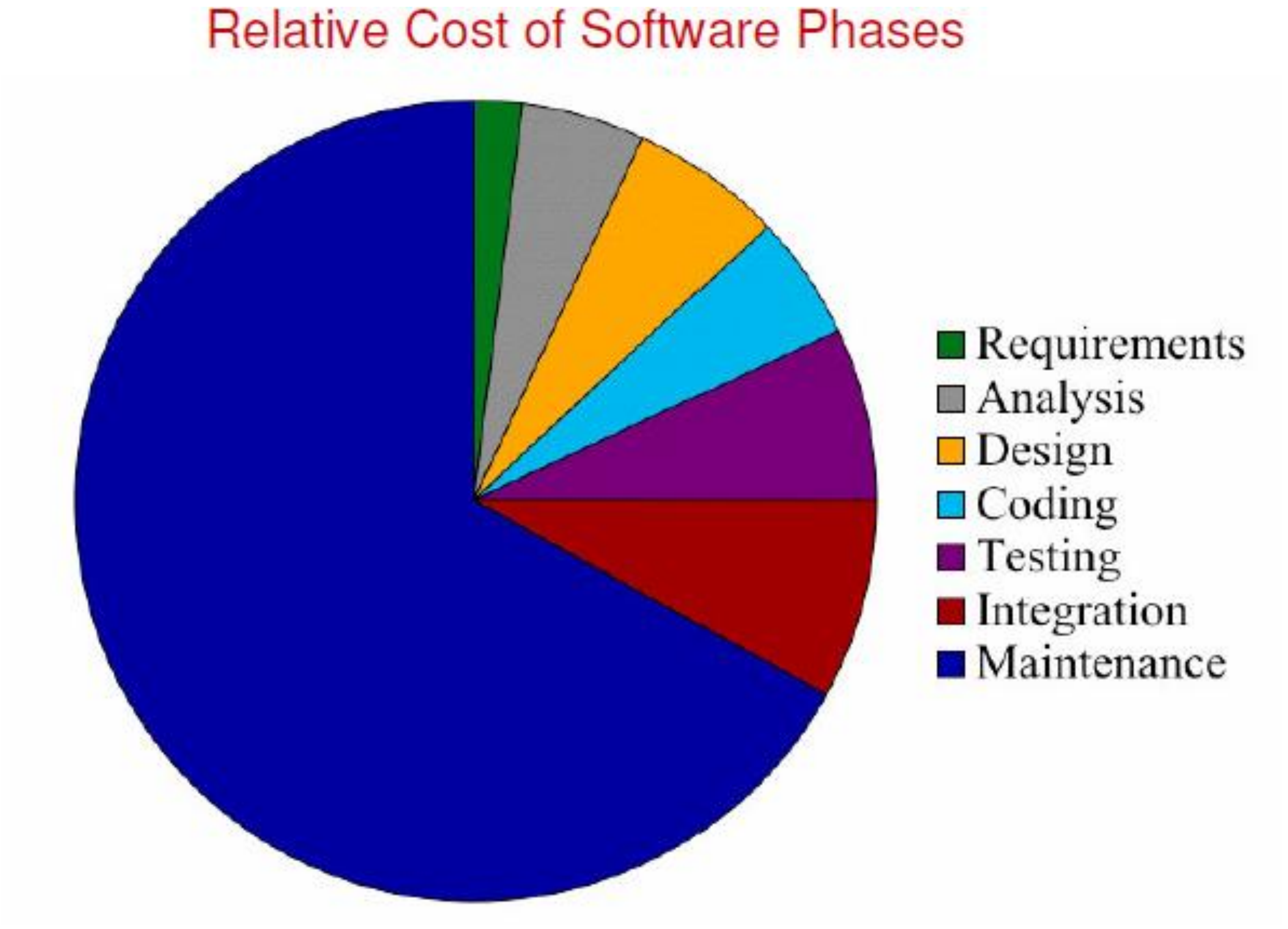
# Function Point Analysis

$$\sum_{i=1}^{14} F_i = 3+4+3+5+3+3+3+3+3+3+2+3+0+3=41$$

$$\begin{aligned}
\mathrm{CAF} &= (0.65 + 0.01 \text{ x } \Sigma \mathrm{F_i}) \\
&= (0.65 + 0.01 \text{ x } 41) \\
&= 1.06 \\
\mathrm{FP} &= \mathrm{UFP} \text{ x } \mathrm{CAF} \\
&= 424 \text{ x } 1.06 \\
&= 449.44
\end{aligned}$$

Hence $\boxed{\mathrm{FP} = 449}$

# Software Project Planning



# Heuristic Techniques

- Heuristic techniques assume that the relationships among the different project parameters can be modeled using suitable mathematical expressions. Once the basic (independent) parameters are known, the other (dependent) parameters can be easily determined by substituting the value of the basic parameters in the mathematical expression. Different heuristic estimation models can be divided into the following two classes: single variable model and the multi variable model.

# Single variable model

Methods using this model use an equation to estimate the desired values such as cost, time, effort, etc. They all depend on the same variable used as predictor (say, size). An example of the most common equations is :

$$C = a\,L^{b}$$

SEL (Software Engineering Laboratory) model

C is the cost, L is the size and a, b are constants

$$E = 1.4\,L^{0.93}$$

$$DOC = 30.4\,L^{0.90}$$

$$D = 4.6\,L^{0.26}$$

Effort (E in Person-months), documentation (DOC, in number of pages) and duration (D, in months) are calculated from the number of lines of code (L, in thousands of lines) used as a predictor.

# Single variable model

WF (Walston-Felix) model

These models are often based on equation (i), they actually depend on several variables representing various aspects of the software development environment, for example method used, user participation, customer oriented changes, memory constraints, etc.

$$E = 5.2\ L^{0.91}$$

$$D = 4.1\ L^{0.36}$$

# Multi variable model

- The productivity index uses 29 variables which are found to be highly correlated to productivity as follows:

$$\mathrm{I} = \sum_{i=1}^{29} W_i X_i$$

# Heuristic Techniques

- Compare the WF (Walston-Felix) model with the SEL (Software Engineering Laboratory) model on a software development expected to involve 8 person-years of effort. Software Project Planning

(a) Calculate the number of lines of source code that can be produced.

(b) Calculate the duration of the development.

(c) Calculate the productivity in LOC/PY

(d) Calculate the average manning

# Heuristic Techniques

## Solution

The amount of manpower involved = 8 PY = 96 person-months

**(a)** Number of lines of source code can be obtained by reversing equation to give:

$$L = (E/a)^{1/b}$$

Then

$$L(SEL) = (96/1.4)^{1/0.93} = 94264 \text{ LOC}$$

$$L\ (W - F) = 96/5.2)^{1/0.91} = 24632 \text{ LOC.}$$

# Heuristic Techniques

**(b)** Duration in months can be calculated by means of equation

$$D(SEL) = 4.6\ (L)^{0.26}$$

$$= 4.6\ (94.264)^{0.26} = 15 \text{ months}$$

$$D(W\text{-}F) = 4.1\ L^{0.36}$$

$$= 4.1(24.632)^{0.36} = 13 \text{ months}$$

**(c)** Productivity is the lines of code produced per person/month (year)

$$P(SEL) = \frac{94264}{8} = 11783\ LOC / Person - Years$$

$$P(W - F) = \frac{24632}{8} = 3079\ LOC / Person - Years$$

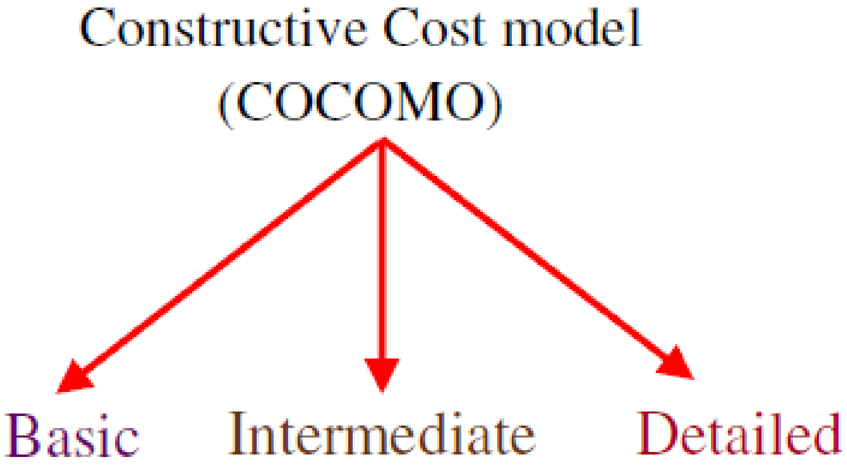
# Heuristic Techniques

**(d)** Average manning is the average number of persons required per month in the project.

$$M(SEL) = \frac{96P - M}{15M} = 6.4 Persons$$

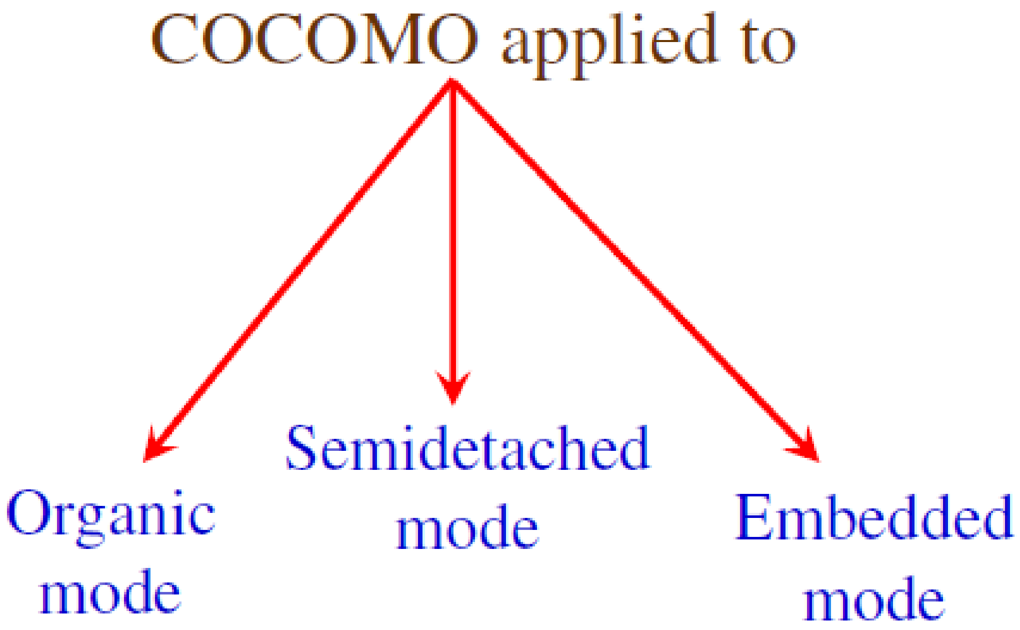
$$M(W - F) = \frac{96P - M}{13M} = 7.4 Persons$$

# Constructive Cost Model (COCOMO)



Model proposed by
B. W. Boehm's
through his book
Software Engineering Economics in 1981

# Constructive Cost Model (COCOMO)



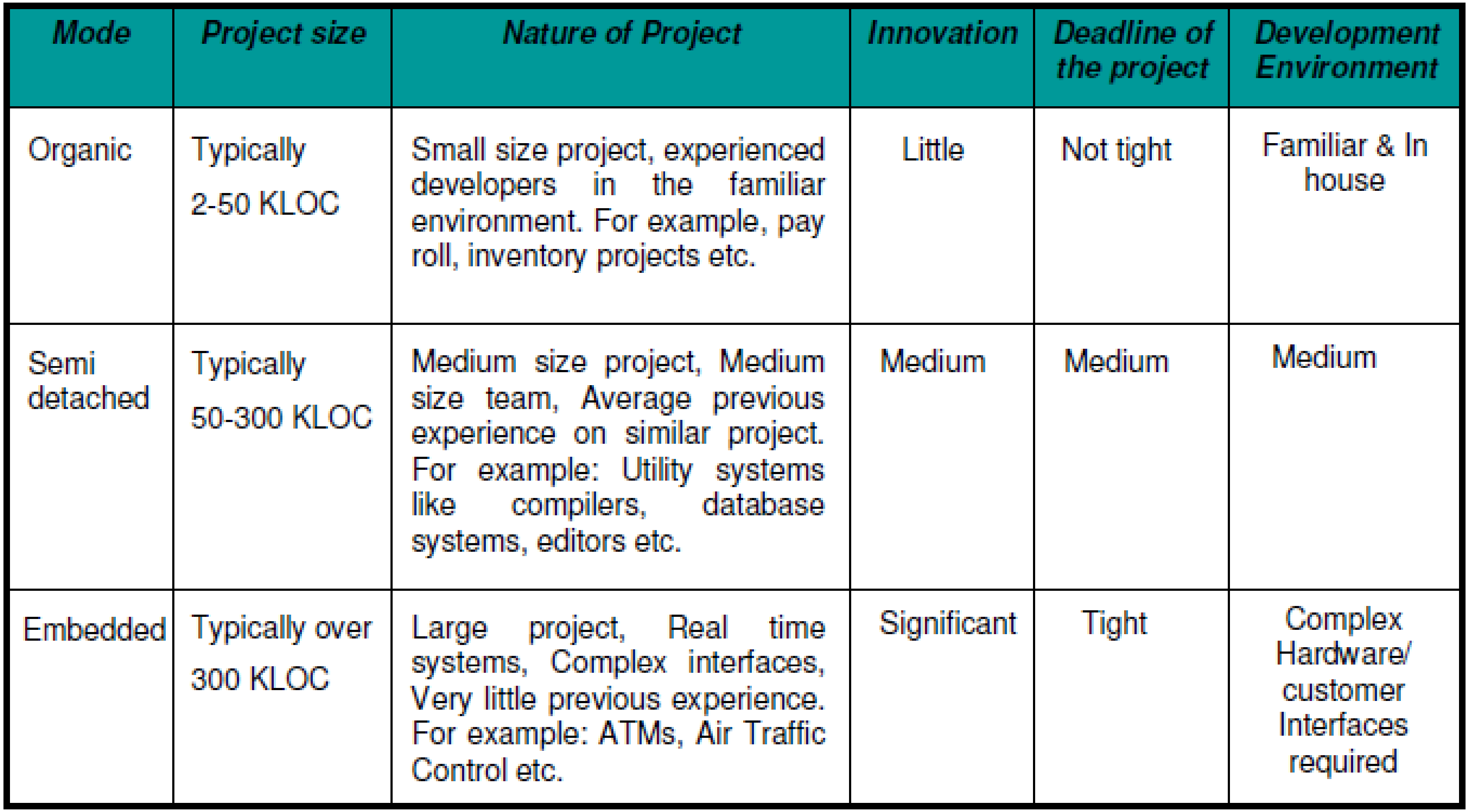
COCOMO applied to

Organic mode

Semidetached mode

Embedded mode

# Constructive Cost Model (COCOMO)



| *Mode* | *Project size* | *Nature of Project* | *Innovation* | *Deadline of the project* | *Development Environment* |
|---|---|---|---|---|---|
| Organic | Typically<br>2-50 KLOC | Small size project, experienced developers in the familiar environment. For example, pay roll, inventory projects etc. | Little | Not tight | Familiar & In house |
| Semi detached | Typically<br>50-300 KLOC | Medium size project, Medium size team, Average previous experience on similar project. For example: Utility systems like compilers, database systems, editors etc. | Medium | Medium | Medium |
| Embedded | Typically over<br>300 KLOC | Large project, Real time systems, Complex interfaces, Very little previous experience. For example: ATMs, Air Traffic Control etc. | Significant | Tight | Complex Hardware/ customer Interfaces required |

**Table 4:** The comparison of three COCOMO modes

# Constructive Cost Model (COCOMO)

## Basic Model

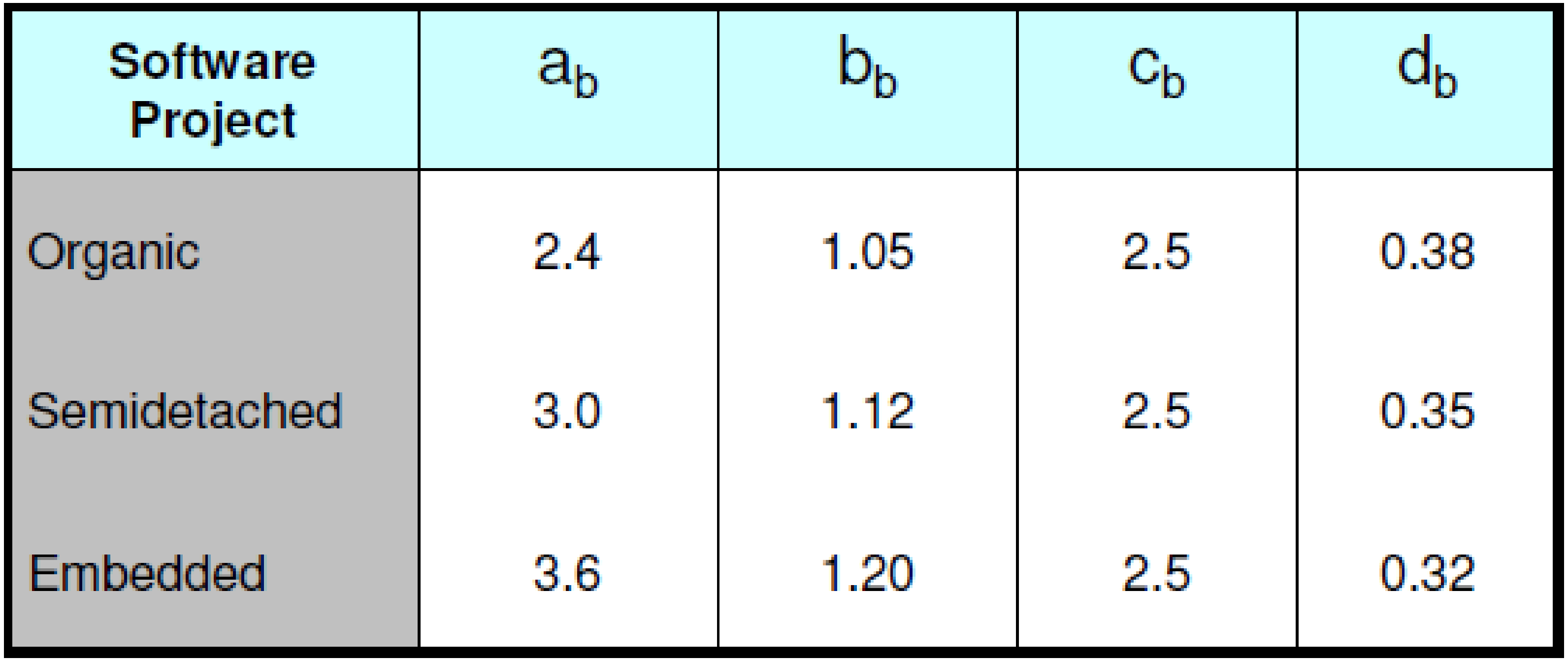
Basic COCOMO model takes the form

$$E = a_b (KLOC)^{b_b}$$

$$D = c_b (E)^{d_b}$$

where E is effort applied in Person-Months, and D is the development time in months. The coefficients $a_b$, $b_b$, $c_b$ and $d_b$ are given in table 4 (a).

# Constructive Cost Model (COCOMO)



| Software Project | $a_b$ | $b_b$ | $c_b$ | $d_b$ |
|---|---|---|---|---|
| Organic | 2.4 | 1.05 | 2.5 | 0.38 |
| Semidetached | 3.0 | 1.12 | 2.5 | 0.35 |
| Embedded | 3.6 | 1.20 | 2.5 | 0.32 |

**Table 4(a):** Basic COCOMO coefficients

# Constructive Cost Model (COCOMO)

When effort and development time are known, the average staff size to complete the project may be calculated as:

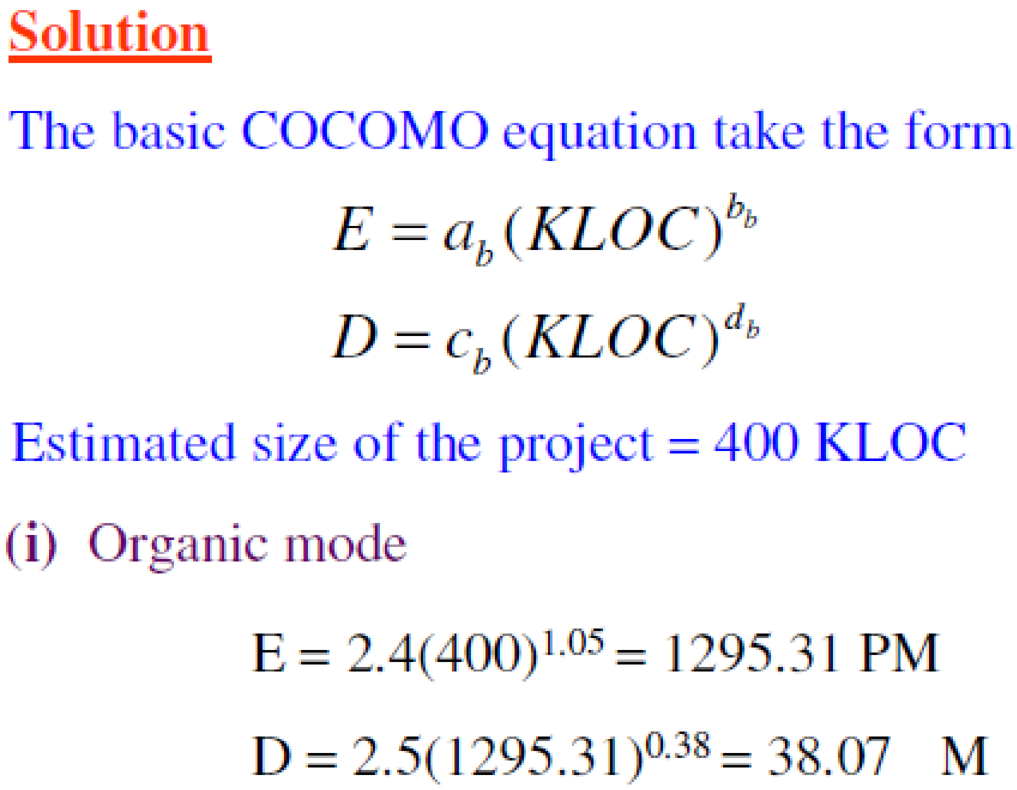
$$\text{Average staff size } (SS) = \frac{E}{D} \; Persons$$

When project size is known, the productivity level may be calculated as:

$$\text{Productivity } (P) = \frac{KLOC}{E} \; KLOC\,/\,PM$$

# Constructive Cost Model (COCOMO)

- Suppose that a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three modes i.e., organic, semidetached and embedded.



**Solution**

The basic COCOMO equation take the form:

$$E = a_b (KLOC)^{b_b}$$

$$D = c_b (KLOC)^{d_b}$$

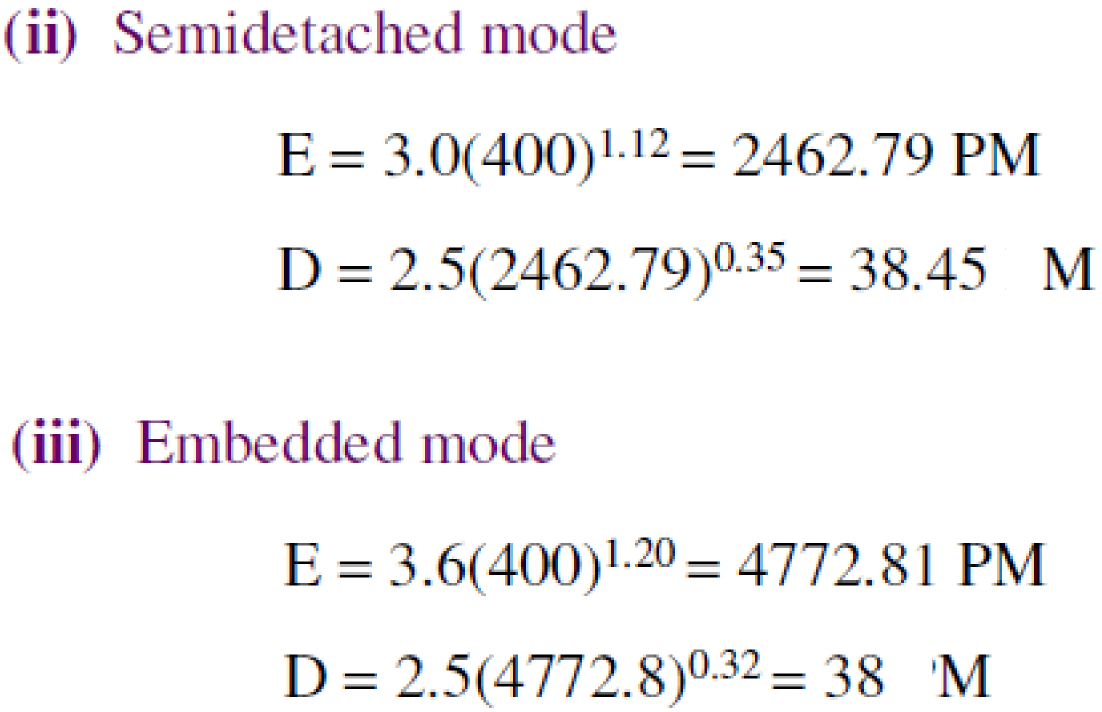
Estimated size of the project = 400 KLOC

(i) Organic mode

$$E = 2.4(400)^{1.05} = 1295.31 \text{ PM}$$

$$D = 2.5(1295.31)^{0.38} = 38.07 \quad \text{M}$$

# Constructive Cost Model (COCOMO)



**(ii)** Semidetached mode

$$E = 3.0(400)^{1.12} = 2462.79 \text{ PM}$$

$$D = 2.5(2462.79)^{0.35} = 38.45 \quad \text{M}$$

**(iii)** Embedded mode

$$E = 3.6(400)^{1.20} = 4772.81 \text{ PM}$$

$$D = 2.5(4772.8)^{0.32} = 38 \quad \text{M}$$

# Constructive Cost Model (COCOMO)

- A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the effort, development time, average staff size and productivity of the project.

# Constructive Cost Model (COCOMO)

**Solution**

The semi-detached mode is the most appropriate mode; keeping in view the size, schedule and experience of the development team.

Hence $E = 3.0(200)^{1.12} = 1133.12$ PM

$D = 2.5(1133.12)^{0.35} = 29.3$ 'M

$$\text{Average staff size } (SS) = \frac{E}{D}\, Persons$$

$$= \frac{1133.12}{29.3} = 38.67\, Persons$$

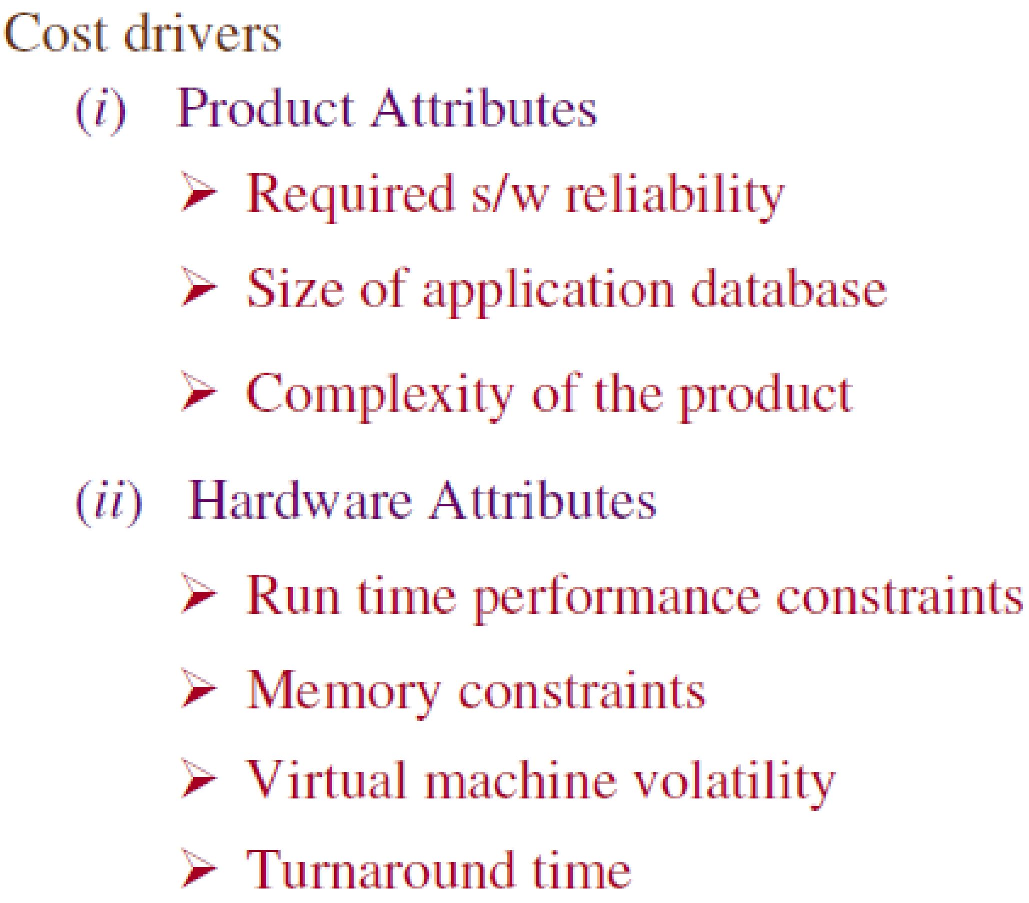
# Constructive Cost Model (COCOMO)

$$\text{Productivity} = \frac{KLOC}{E} = \frac{200}{1133.12} = 0.1765\, KLOC / PM$$

$$P = 176\, LOC / PM$$

# Constructive Cost Model (COCOMO)

## Intermediate Model



Cost drivers

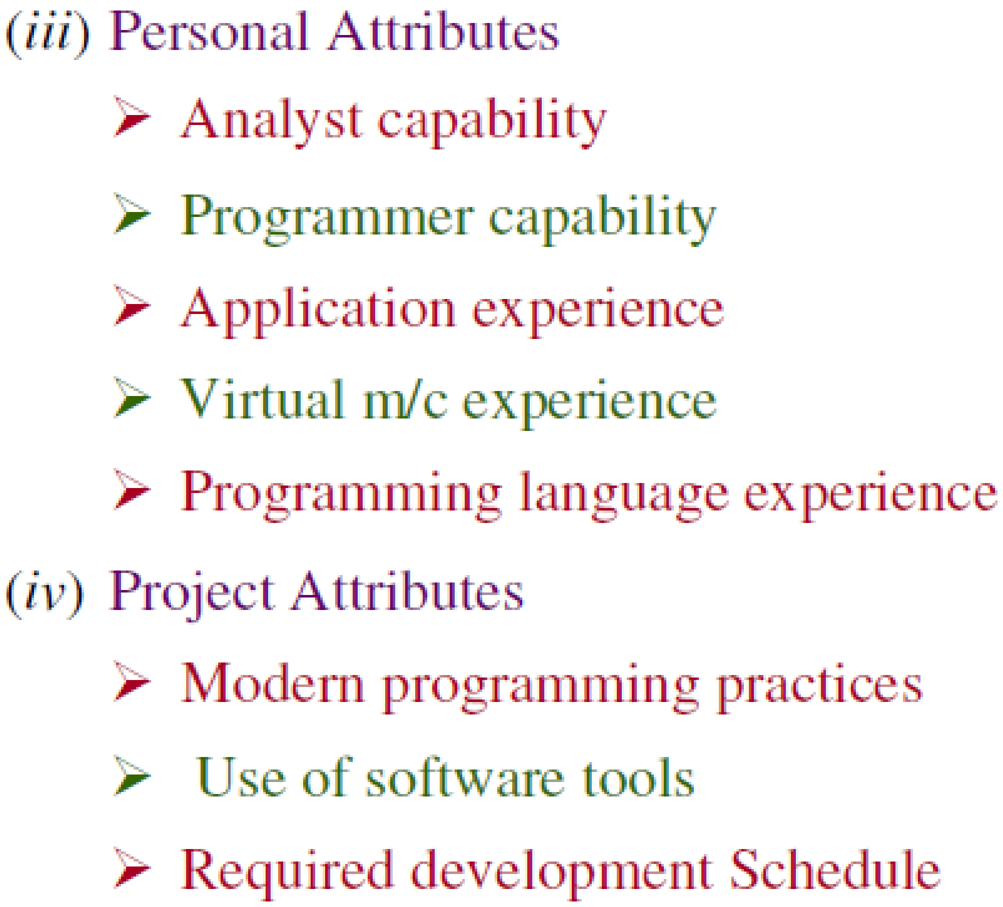
(*i*) Product Attributes

- ➢ Required s/w reliability
- ➢ Size of application database
- ➢ Complexity of the product

(*ii*) Hardware Attributes

- ➢ Run time performance constraints
- ➢ Memory constraints
- ➢ Virtual machine volatility
- ➢ Turnaround time

# Constructive Cost Model (COCOMO)



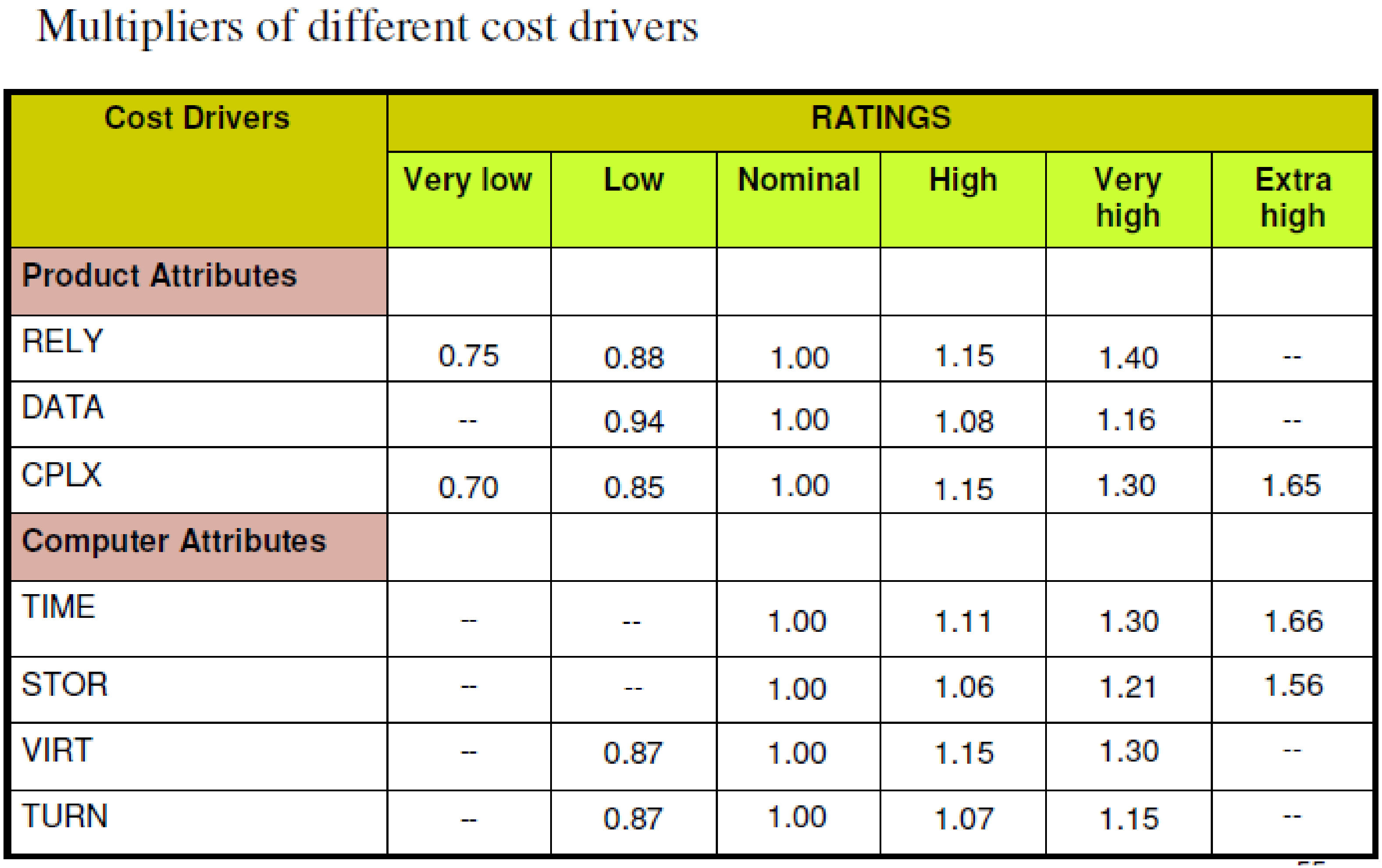
(*iii*) Personal Attributes

- Analyst capability
- Programmer capability
- Application experience
- Virtual m/c experience
- Programming language experience

(*iv*) Project Attributes

- Modern programming practices
- Use of software tools
- Required development Schedule

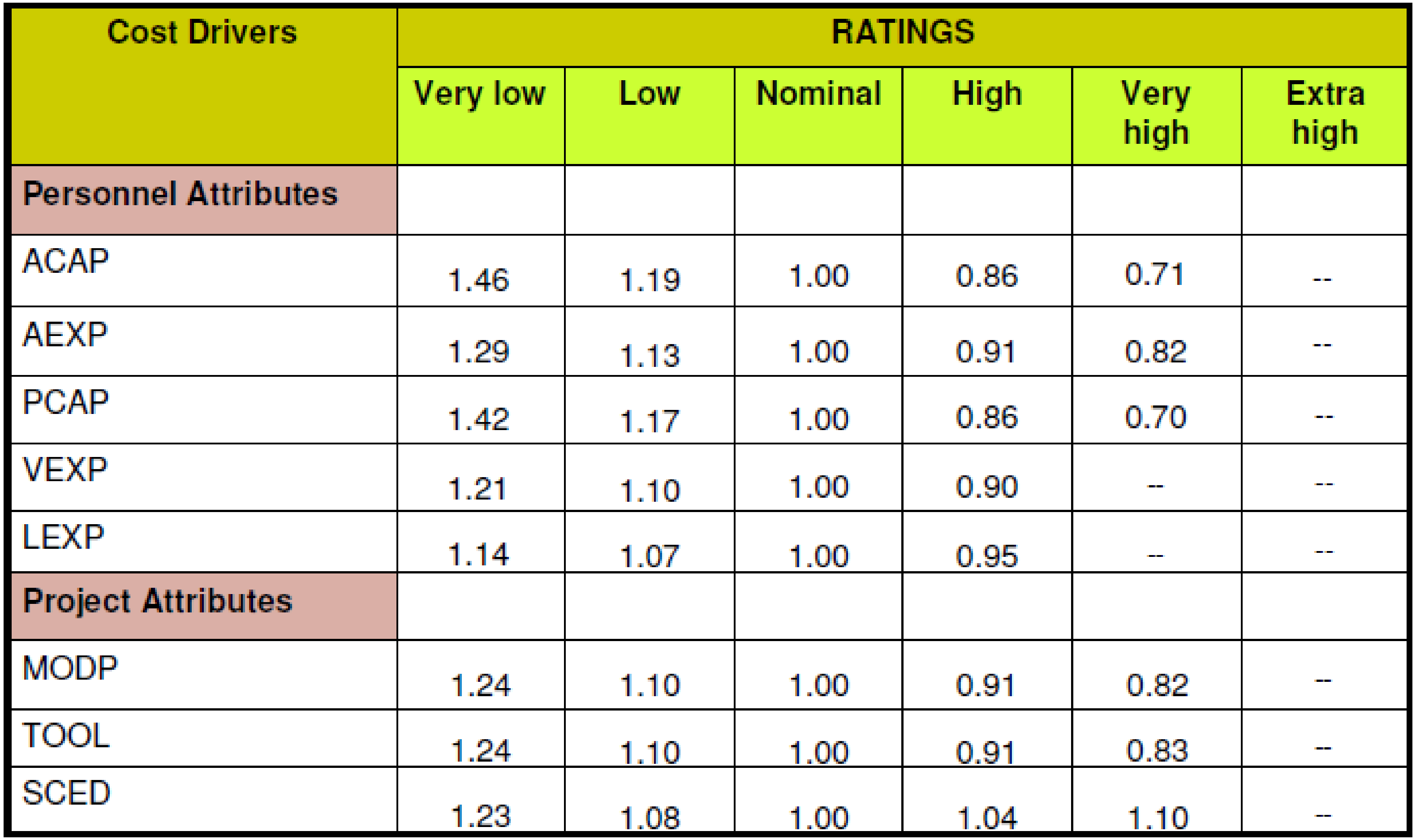
# Constructive Cost Model (COCOMO)



Multipliers of different cost drivers

| Cost Drivers | RATINGS | | | | | |
|---|---|---|---|---|---|---|
| | Very low | Low | Nominal | High | Very high | Extra high |
| **Product Attributes** | | | | | | |
| RELY | 0.75 | 0.88 | 1.00 | 1.15 | 1.40 | -- |
| DATA | -- | 0.94 | 1.00 | 1.08 | 1.16 | -- |
| CPLX | 0.70 | 0.85 | 1.00 | 1.15 | 1.30 | 1.65 |
| **Computer Attributes** | | | | | | |
| TIME | -- | -- | 1.00 | 1.11 | 1.30 | 1.66 |
| STOR | -- | -- | 1.00 | 1.06 | 1.21 | 1.56 |
| VIRT | -- | 0.87 | 1.00 | 1.15 | 1.30 | -- |
| TURN | -- | 0.87 | 1.00 | 1.07 | 1.15 | -- |

# Constructive Cost Model (COCOMO)



| Cost Drivers | RATINGS | | | | | |
|---|---|---|---|---|---|---|
| | Very low | Low | Nominal | High | Very high | Extra high |
| **Personnel Attributes** | | | | | | |
| ACAP | 1.46 | 1.19 | 1.00 | 0.86 | 0.71 | -- |
| AEXP | 1.29 | 1.13 | 1.00 | 0.91 | 0.82 | -- |
| PCAP | 1.42 | 1.17 | 1.00 | 0.86 | 0.70 | -- |
| VEXP | 1.21 | 1.10 | 1.00 | 0.90 | -- | -- |
| LEXP | 1.14 | 1.07 | 1.00 | 0.95 | -- | -- |
| **Project Attributes** | | | | | | |
| MODP | 1.24 | 1.10 | 1.00 | 0.91 | 0.82 | -- |
| TOOL | 1.24 | 1.10 | 1.00 | 0.91 | 0.83 | -- |
| SCED | 1.23 | 1.08 | 1.00 | 1.04 | 1.10 | -- |

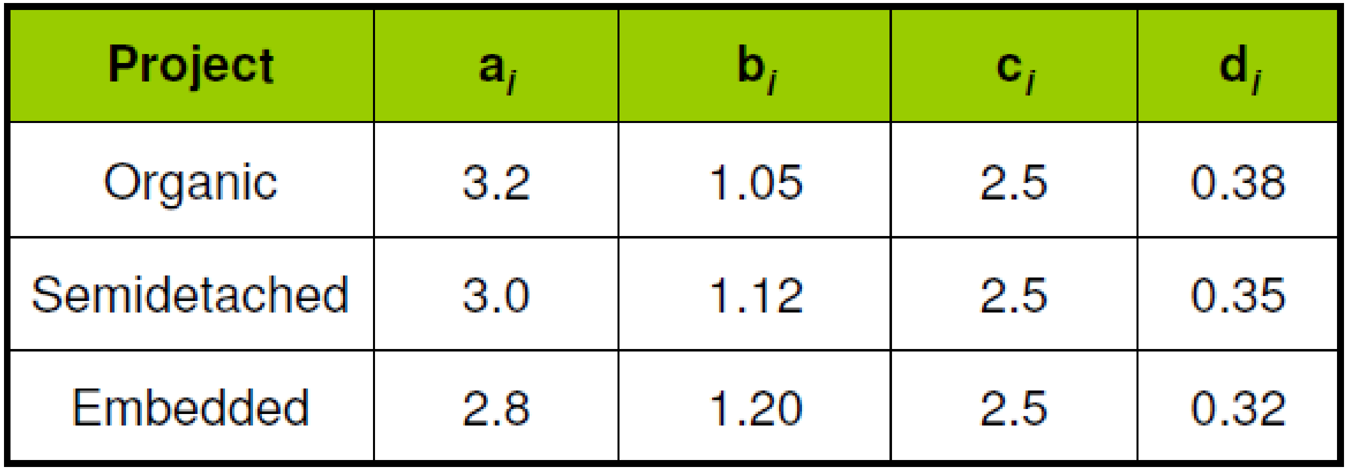
**Table 5:** Multiplier values for effort calculations

# Constructive Cost Model (COCOMO)

Intermediate COCOMO equations

$$E = a_i (KLOC)^{b_i} * EAF$$

$$D = c_i (E)^{d_i}$$

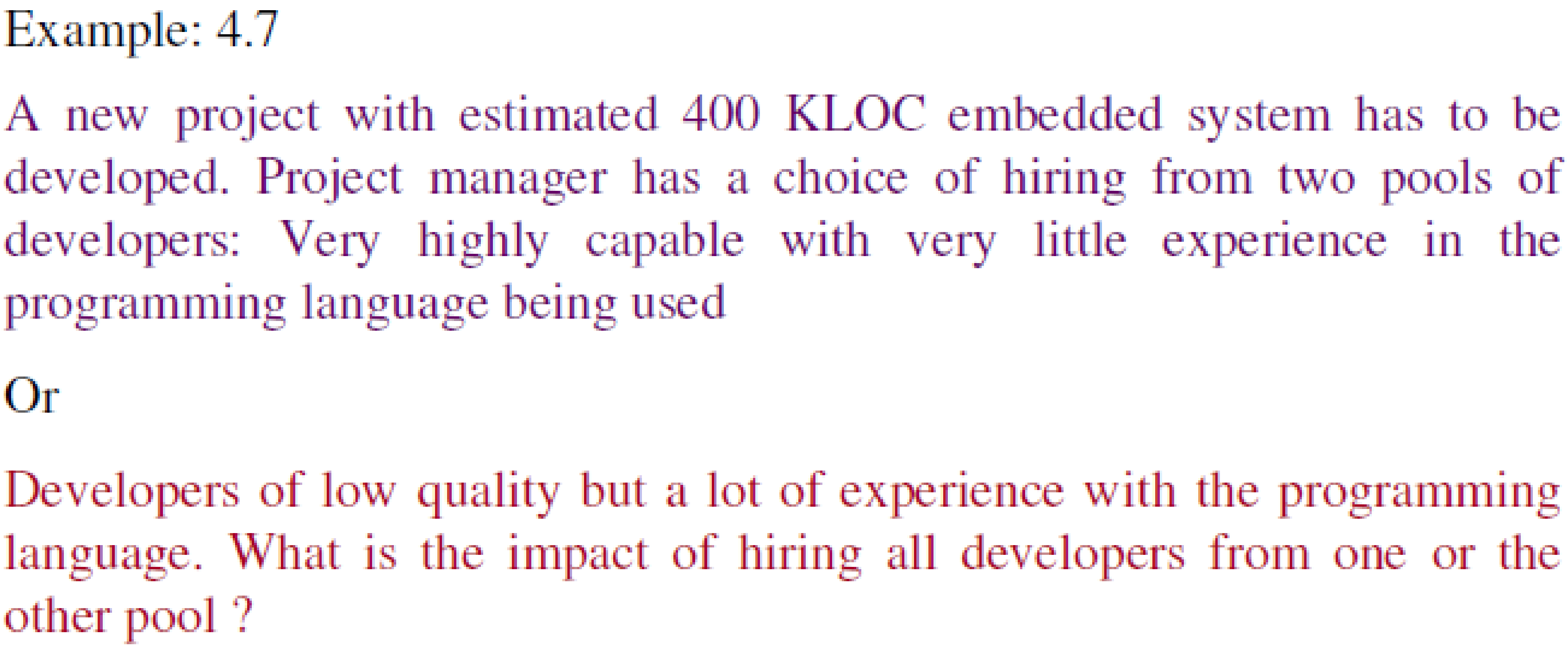


| Project | a*i* | b*i* | c*i* | d*i* |
|---|---|---|---|---|
| Organic | 3.2 | 1.05 | 2.5 | 0.38 |
| Semidetached | 3.0 | 1.12 | 2.5 | 0.35 |
| Embedded | 2.8 | 1.20 | 2.5 | 0.32 |

**Table 6:** Coefficients for intermediate COCOMO

EAF (Effort Adjustment Factor)

# Constructive Cost Model (COCOMO)



Example: 4.7

A new project with estimated 400 KLOC embedded system has to be developed. Project manager has a choice of hiring from two pools of developers: Very highly capable with very little experience in the programming language being used

Or

Developers of low quality but a lot of experience with the programming language. What is the impact of hiring all developers from one or the other pool ?

# Constructive Cost Model (COCOMO)

**Solution**

This is the case of embedded mode and model is intermediate COCOMO.

Hence $E = a_i (KLOC)^{d_i}$

$= 2.8\ (400)^{1.20} = 3712$ PM

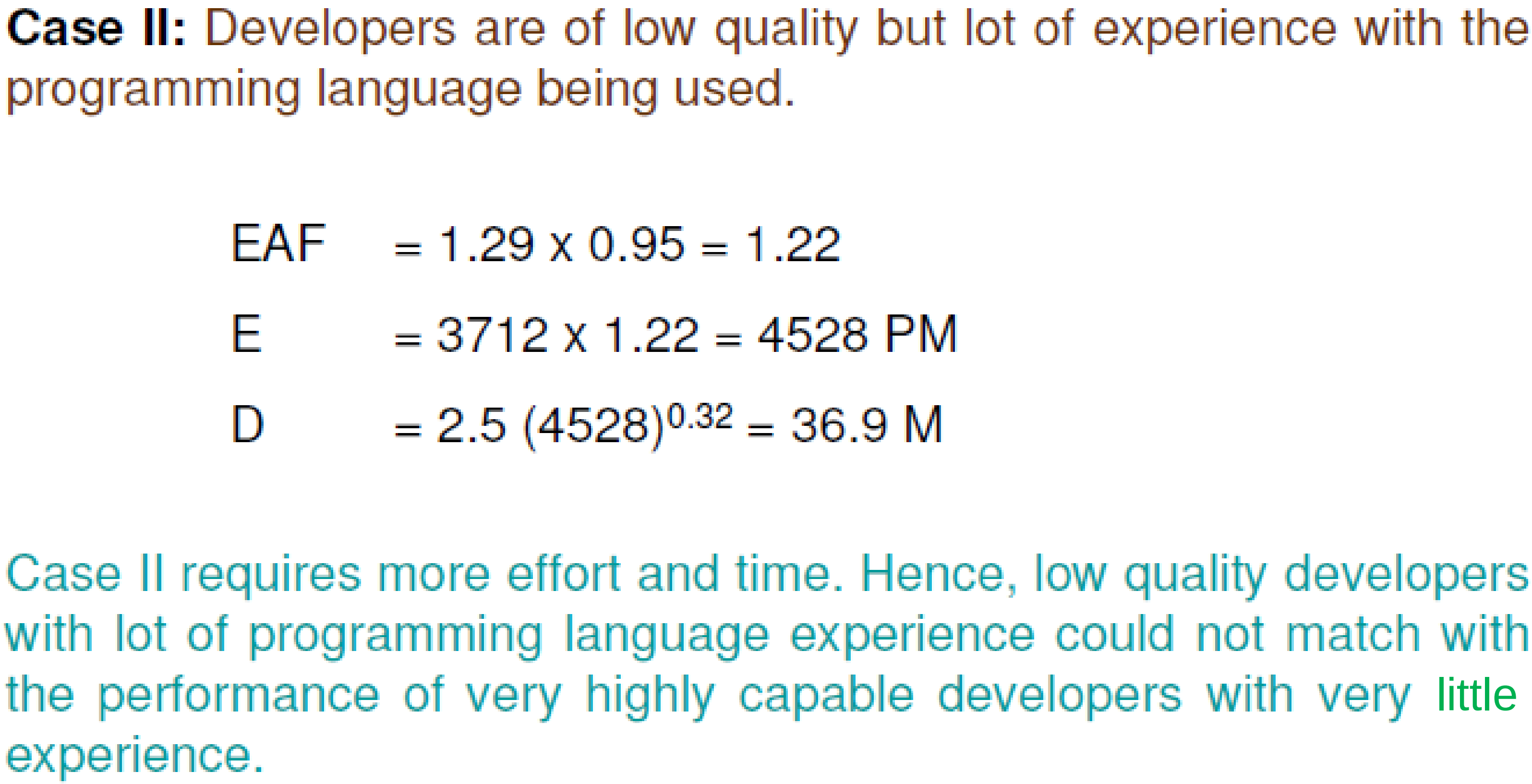
**Case I:** Developers are very highly capable with very little experience in the programming being used.

EAF $= 0.82 \times 1.14 = 0.9348$

E $= 3712 \times .9348 = 3470$ PM

D $= 2.5\ (3470)^{0.32} = 33.9$ M

# Constructive Cost Model (COCOMO)



**Case II:** Developers are of low quality but lot of experience with the programming language being used.

$$EAF = 1.29 \times 0.95 = 1.22$$

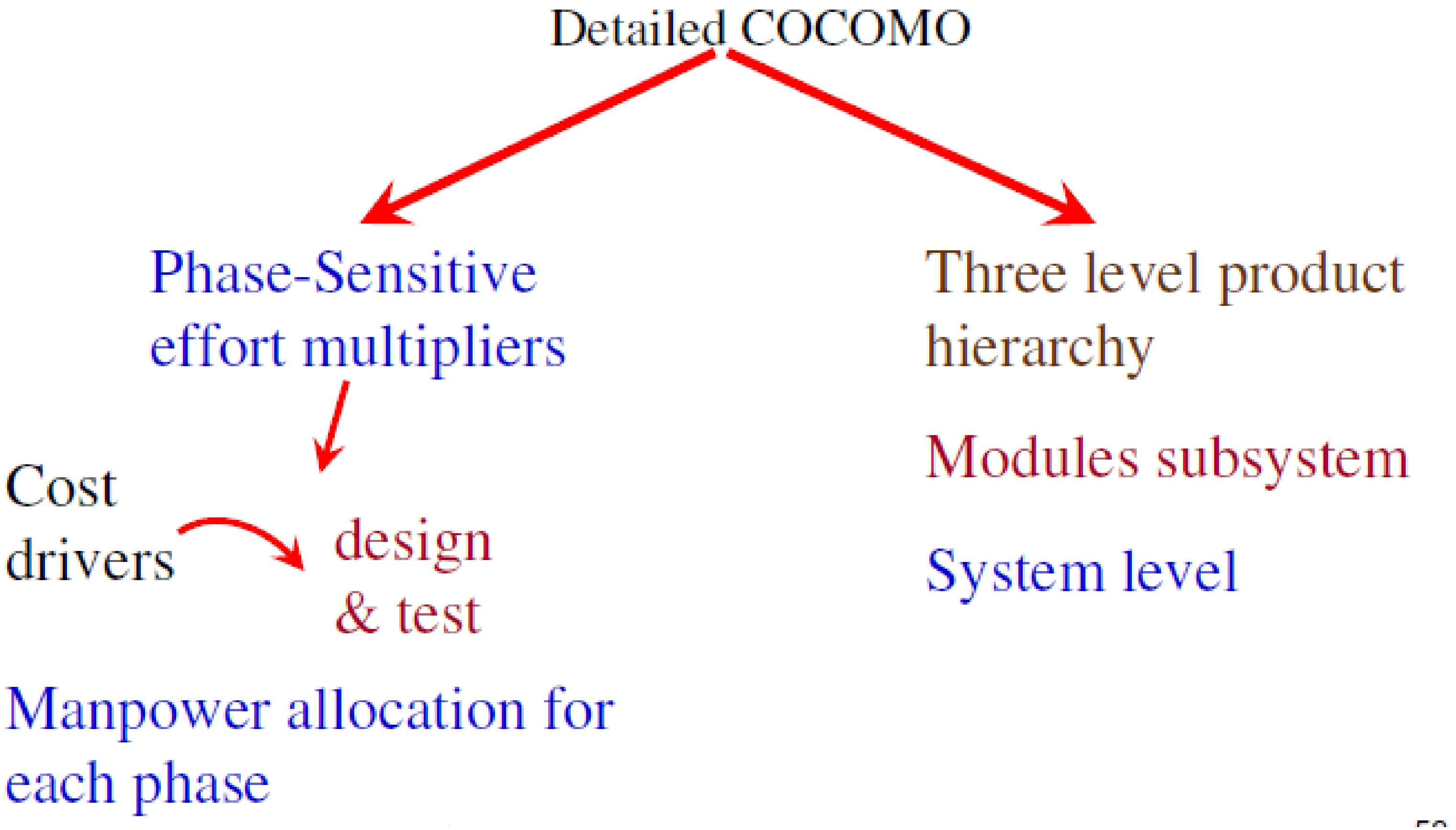
$$E = 3712 \times 1.22 = 4528 \text{ PM}$$

$$D = 2.5\ (4528)^{0.32} = 36.9 \text{ M}$$

Case II requires more effort and time. Hence, low quality developers with lot of programming language experience could not match with the performance of very highly capable developers with very little experience.

# Constructive Cost Model (COCOMO)

## Detailed COCOMO Model



Detailed COCOMO

Phase-Sensitive effort multipliers

Cost drivers

design & test

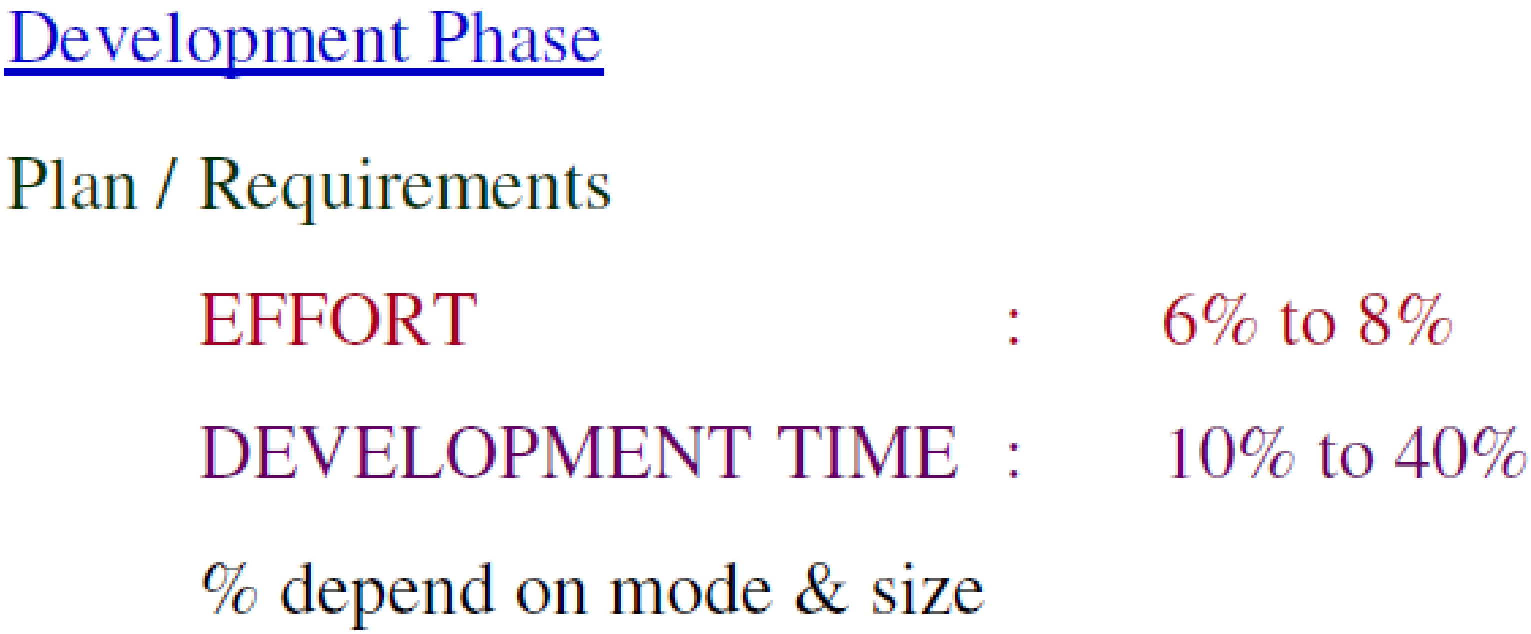
Manpower allocation for each phase

Three level product hierarchy

Modules subsystem

System level

# Constructive Cost Model (COCOMO)



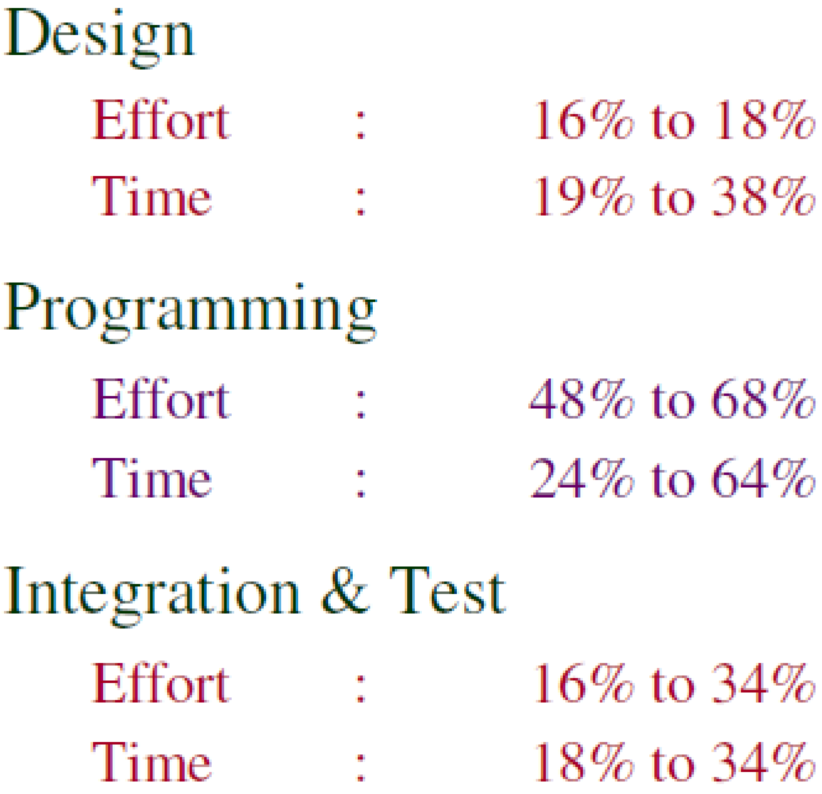
## Development Phase

Plan / Requirements

| | | |
|---|---|---|
| EFFORT | : | 6% to 8% |
| DEVELOPMENT TIME | : | 10% to 40% |

% depend on mode & size

# Constructive Cost Model (COCOMO)



Design

| | | |
|---|---|---|
| Effort | : | 16% to 18% |
| Time | : | 19% to 38% |

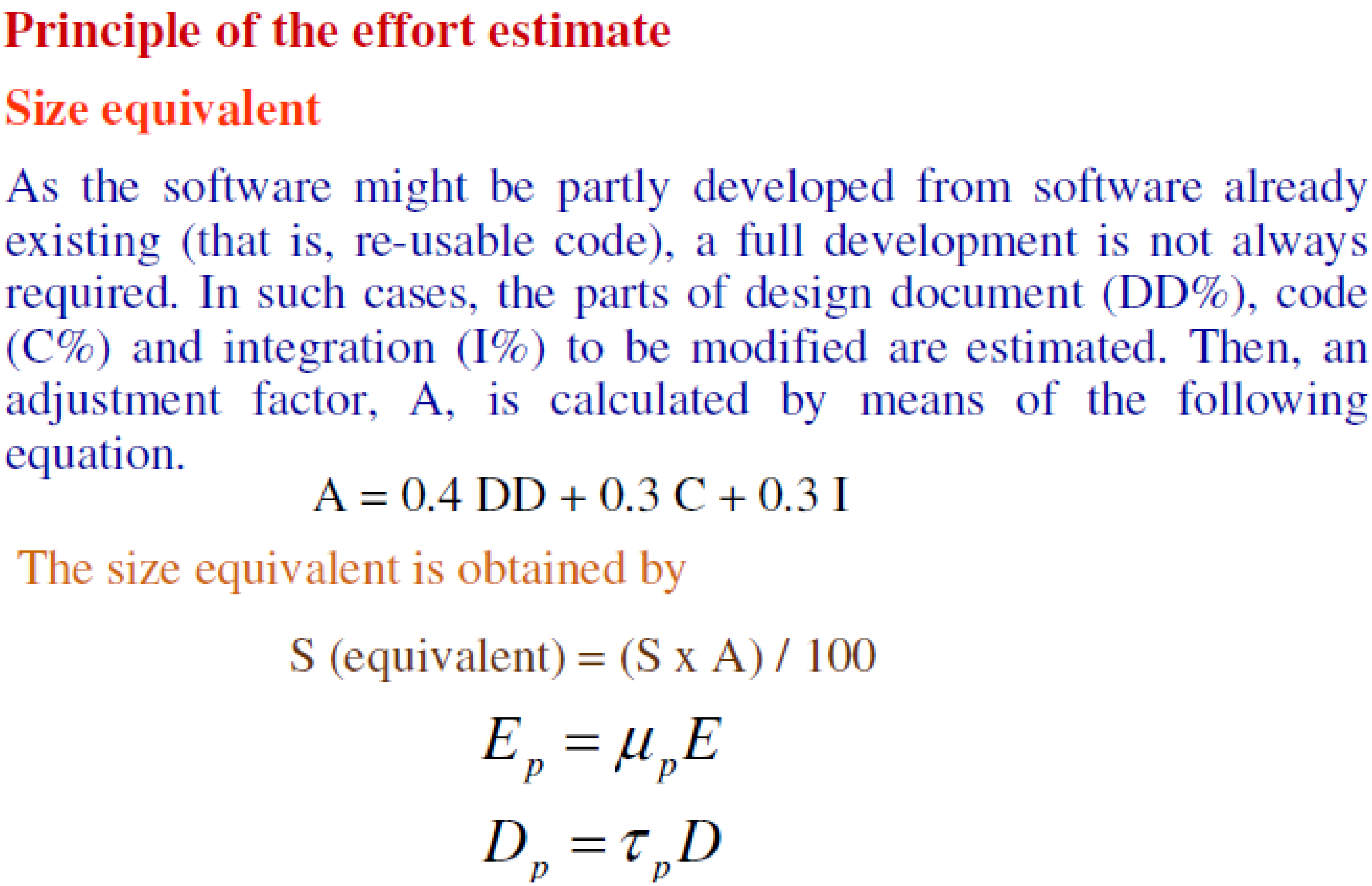
Programming

| | | |
|---|---|---|
| Effort | : | 48% to 68% |
| Time | : | 24% to 64% |

Integration & Test

| | | |
|---|---|---|
| Effort | : | 16% to 34% |
| Time | : | 18% to 34% |

# Constructive Cost Model (COCOMO)



## Principle of the effort estimate

### Size equivalent

As the software might be partly developed from software already existing (that is, re-usable code), a full development is not always required. In such cases, the parts of design document (DD%), code (C%) and integration (I%) to be modified are estimated. Then, an adjustment factor, A, is calculated by means of the following equation.

$$A = 0.4\ DD + 0.3\ C + 0.3\ I$$

The size equivalent is obtained by

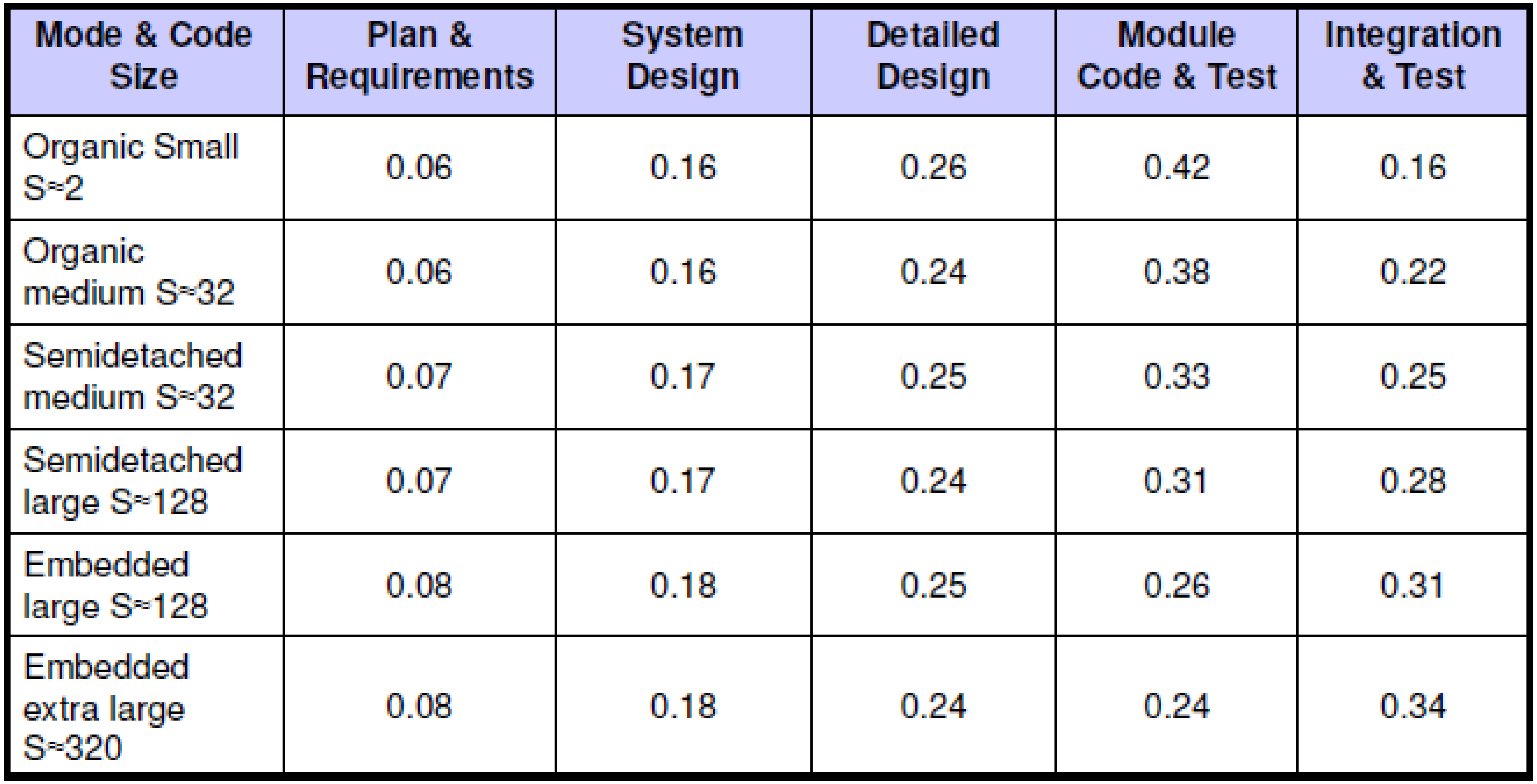
$$S\ (\text{equivalent}) = (S \times A) / 100$$

$$E_p = \mu_p E$$

$$D_p = \tau_p D$$

# Constructive Cost Model (COCOMO)

Lifecycle Phase Values of $\mu_p$

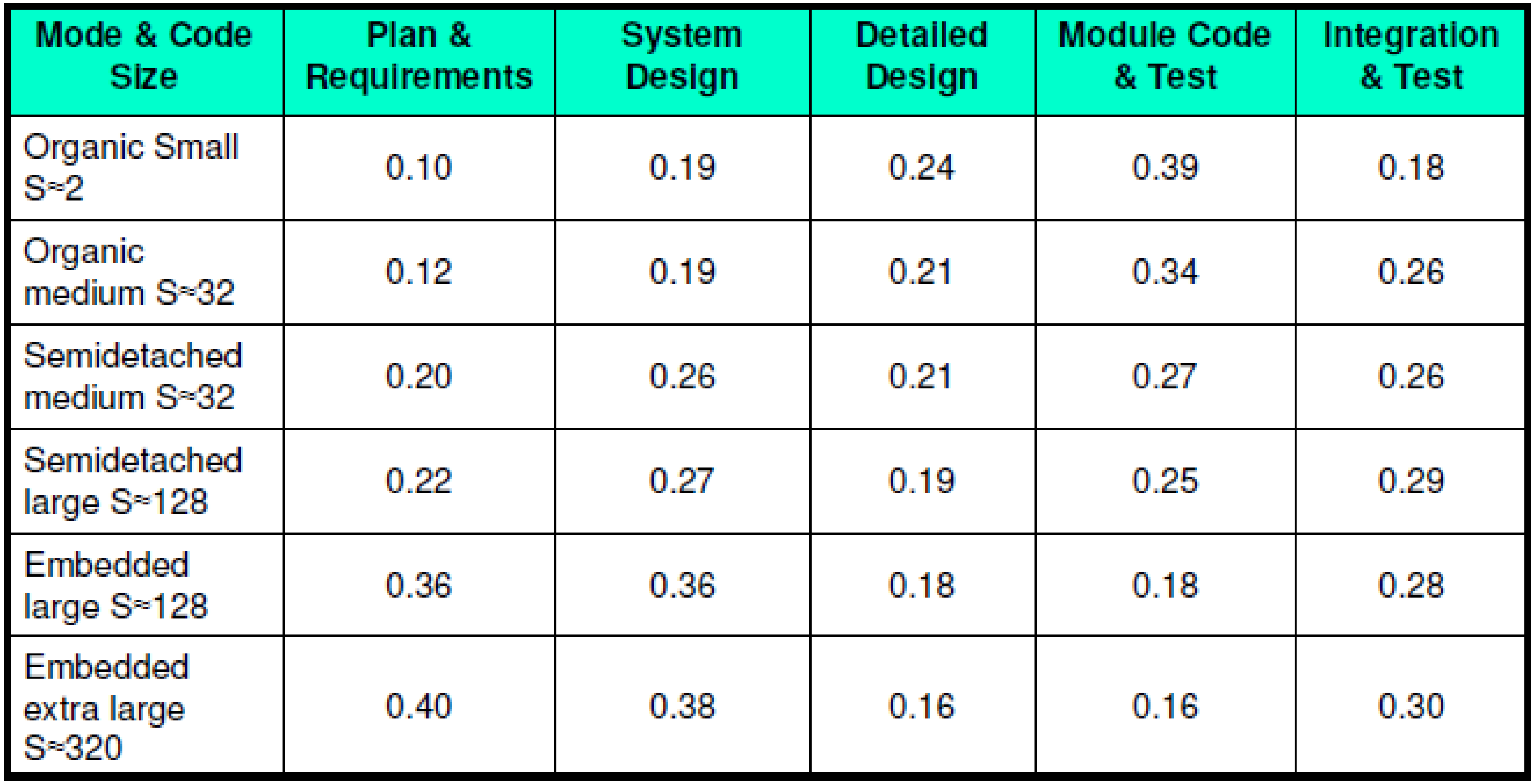


| Mode & Code Size | Plan & Requirements | System Design | Detailed Design | Module Code & Test | Integration & Test |
|---|---|---|---|---|---|
| Organic Small S≈2 | 0.06 | 0.16 | 0.26 | 0.42 | 0.16 |
| Organic medium S≈32 | 0.06 | 0.16 | 0.24 | 0.38 | 0.22 |
| Semidetached medium S≈32 | 0.07 | 0.17 | 0.25 | 0.33 | 0.25 |
| Semidetached large S≈128 | 0.07 | 0.17 | 0.24 | 0.31 | 0.28 |
| Embedded large S≈128 | 0.08 | 0.18 | 0.25 | 0.26 | 0.31 |
| Embedded extra large S≈320 | 0.08 | 0.18 | 0.24 | 0.24 | 0.34 |

**Table 7 :** Effort and schedule fractions occurring in each phase of the lifecycle

# Constructive Cost Model (COCOMO)

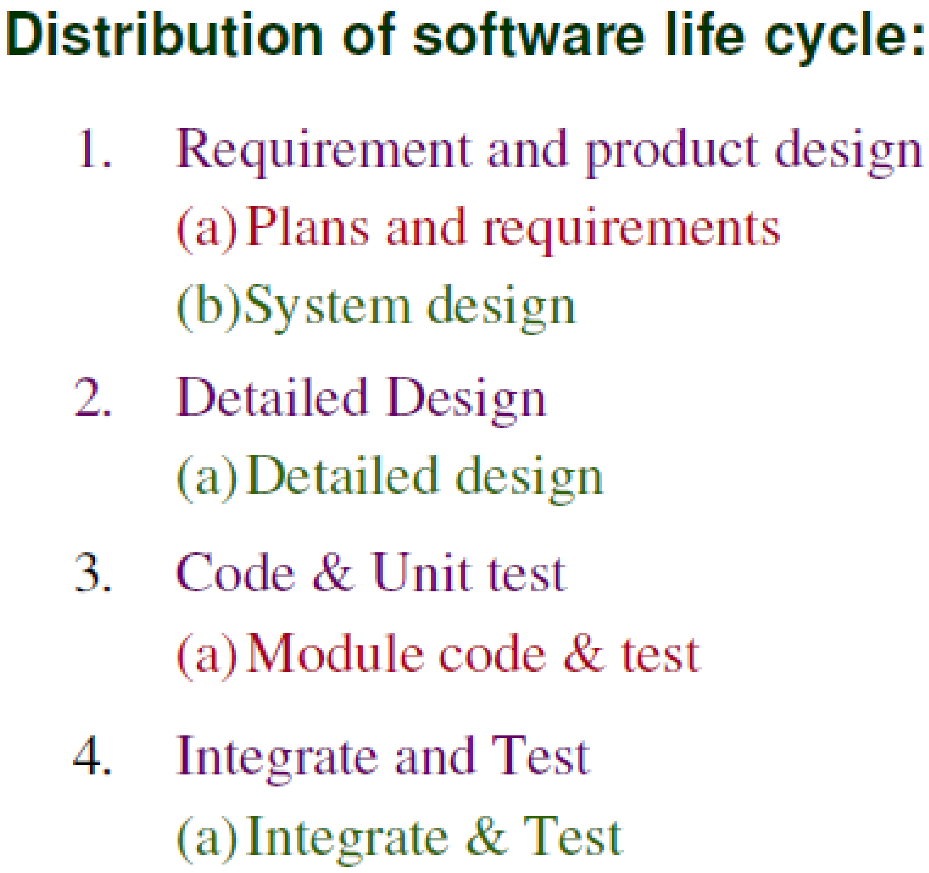
Lifecycle Phase Values of $\tau_p$



| Mode & Code Size | Plan & Requirements | System Design | Detailed Design | Module Code & Test | Integration & Test |
|---|---|---|---|---|---|
| Organic Small S≈2 | 0.10 | 0.19 | 0.24 | 0.39 | 0.18 |
| Organic medium S≈32 | 0.12 | 0.19 | 0.21 | 0.34 | 0.26 |
| Semidetached medium S≈32 | 0.20 | 0.26 | 0.21 | 0.27 | 0.26 |
| Semidetached large S≈128 | 0.22 | 0.27 | 0.19 | 0.25 | 0.29 |
| Embedded large S≈128 | 0.36 | 0.36 | 0.18 | 0.18 | 0.28 |
| Embedded extra large S≈320 | 0.40 | 0.38 | 0.16 | 0.16 | 0.30 |

**Table 7 :** Effort and schedule fractions occurring in each phase of the lifecycle

# Constructive Cost Model (COCOMO)



**Distribution of software life cycle:**

1. Requirement and product design

   (a) Plans and requirements

   (b) System design

2. Detailed Design

   (a) Detailed design

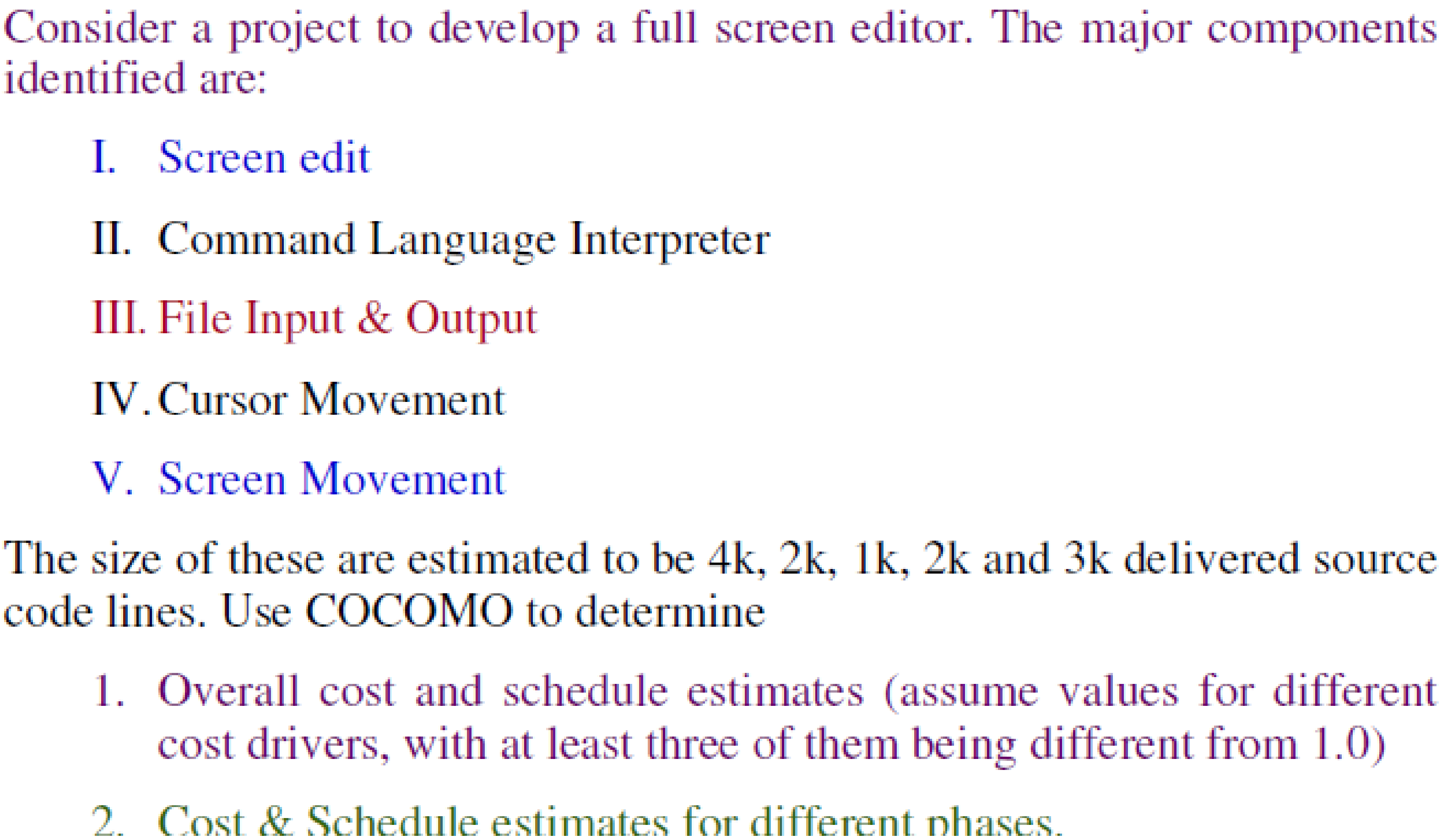
3. Code & Unit test

   (a) Module code & test

4. Integrate and Test

   (a) Integrate & Test

# Constructive Cost Model (COCOMO)



Consider a project to develop a full screen editor. The major components identified are:

I. Screen edit

II. Command Language Interpreter

III. File Input & Output

IV. Cursor Movement

V. Screen Movement

The size of these are estimated to be 4k, 2k, 1k, 2k and 3k delivered source code lines. Use COCOMO to determine

1. Overall cost and schedule estimates (assume values for different cost drivers, with at least three of them being different from 1.0)
2. Cost & Schedule estimates for different phases.

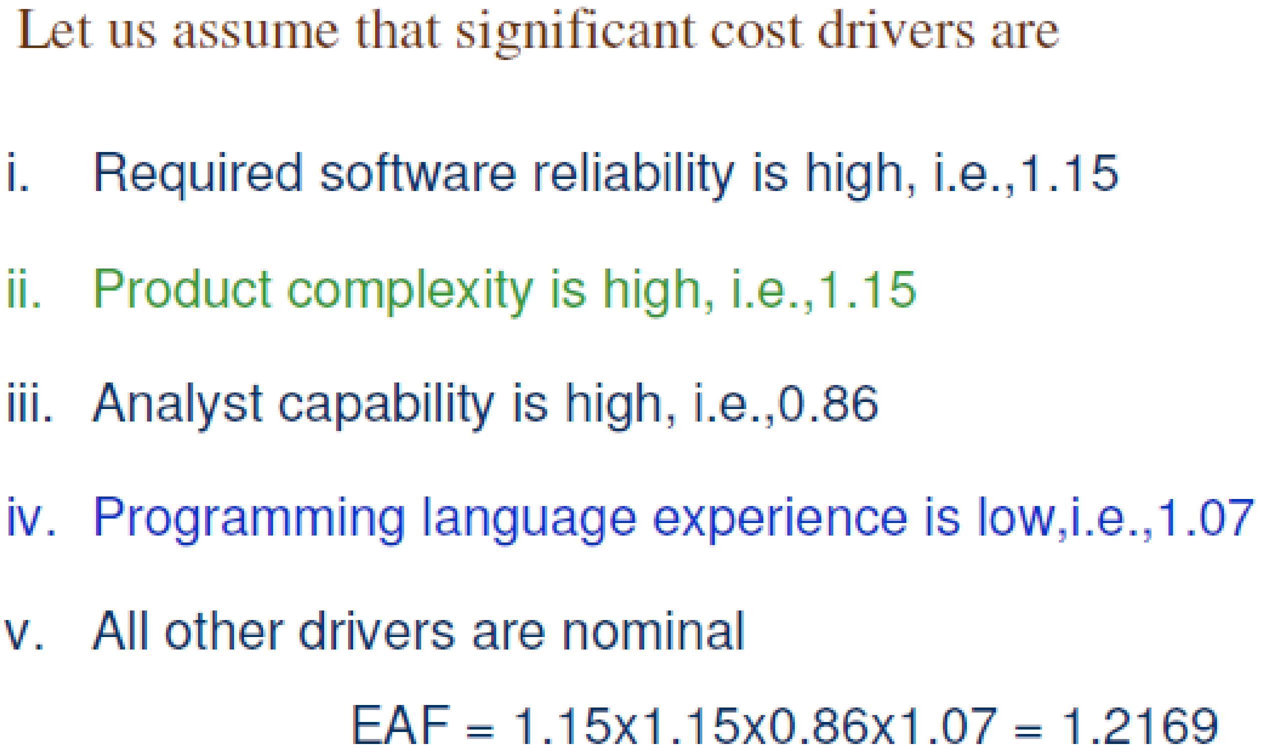
# Constructive Cost Model (COCOMO)

## Solution

Size of five modules are:

| | |
|---|---|
| Screen edit | = 4 KLOC |
| Command language interpreter | = 2 KLOC |
| File input and output | = 1 KLOC |
| Cursor movement | = 2 KLOC |
| Screen movement | = 3 KLOC |
| **Total** | **= 12 KLOC** |

# Constructive Cost Model (COCOMO)



Let us assume that significant cost drivers are

i. Required software reliability is high, i.e.,1.15

ii. Product complexity is high, i.e.,1.15

iii. Analyst capability is high, i.e.,0.86

iv. Programming language experience is low,i.e.,1.07

v. All other drivers are nominal

$$EAF = 1.15x1.15x0.86x1.07 = 1.2169$$

# Constructive Cost Model (COCOMO)

(a) The initial effort estimate for the project is obtained from the following equation

$$E = a_i\,(KLOC)^{bi} \times EAF$$

$$= 3.2(12)^{1.05} \times 1.2169 = 52.91\ PM$$

Development time $D = C_i(E)^{di}$

$$= 2.5(52.91)^{0.38} = 11.29\ M$$

(b) Using the following equations and referring Table 7, phase wise cost and schedule estimates can be calculated.

$$E_p = \mu_p E$$

$$D_p = \tau_p D$$

# Constructive Cost Model (COCOMO)

Since size is only 12 KLOC, it is an organic small model. Phase wise effort distribution is given below:

| | |
|---|---|
| System Design | = 0.16 x 52.91 = 8.465 PM |
| Detailed Design | = 0.26 x 52.91 = 13.756 PM |
| Module Code & Test | = 0.42 x 52.91 = 22.222 PM |
| Integration & Test | = 0.16 x 52.91 = 8.465 Pm |

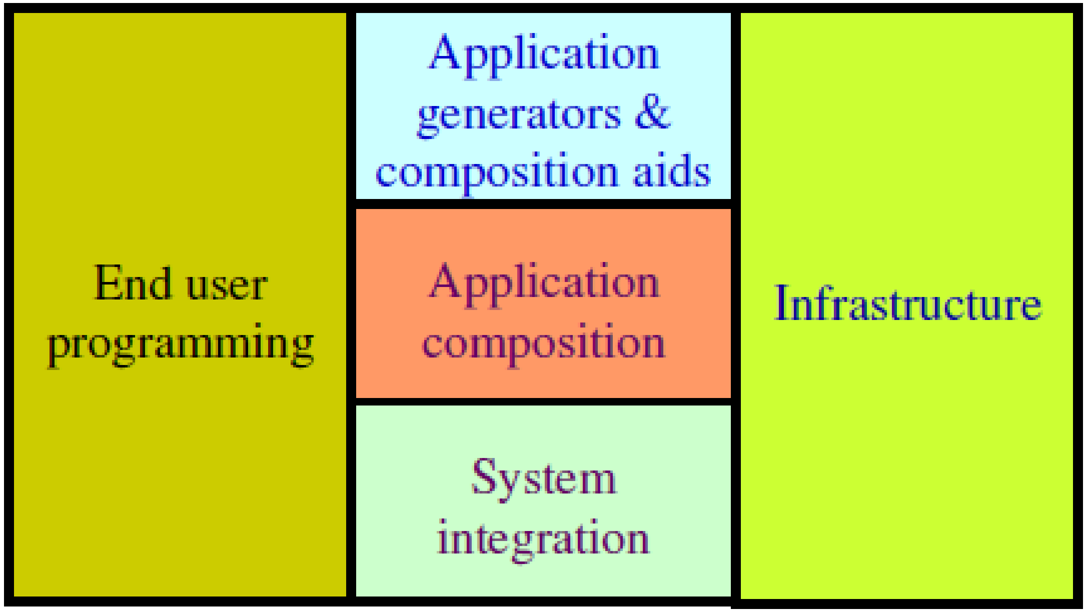
Now Phase wise development time duration is

| | |
|---|---|
| System Design | = 0.19 x 11.29 = 2.145 M |
| Detailed Design | = 0.24 x 11.29 = 2.709 M |
| Module Code & Test | = 0.39 x 11.29 = 4.403 M |
| Integration & Test | = 0.18 x 11.29 = 2.032 M |

# COCOMO-II

## COCOMO-II

The following categories of applications / projects are identified by COCOMO-II and are shown in fig. 4 shown below:



End user programming

Application generators & composition aids

Application composition

System integration

Infrastructure

**Fig. 4 :** Categories of applications / projects

# COCOMO-II

**1. End User Programming:**
Application generators are used in this sub-model. End user write the code by using these application generators.
**Example –** Spreadsheets, report generator, etc.

# COCOMO-II

**2. Intermediate Sector:**

**(a). Application Generators and Composition Aids –**
This category will create largely prepackaged capabilities for user programming. Their product will have many reusable components. Typical firms operating in this sector are Microsoft, Lotus, Oracle, IBM, Borland, Novell.

**(b). Application Composition Sector –**
This category is too diversified and to be handled by prepackaged solutions. It includes GUI, Databases, domain specific components such as financial, medical or industrial process control packages.

**(c). System Integration –**
This category deals with large scale and highly embedded systems.

# COCOMO-II

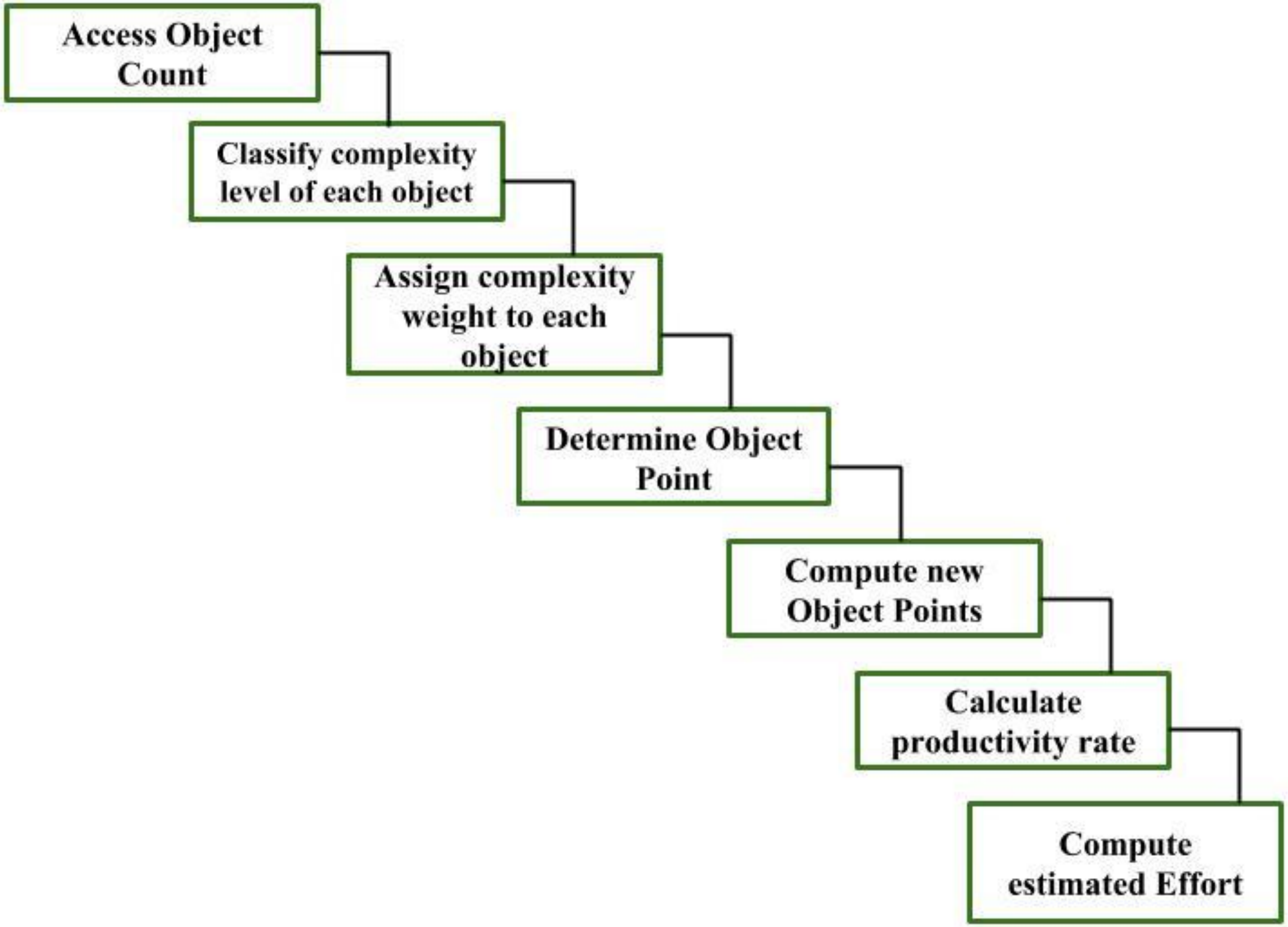
**3. Infrastructure Sector:**
This category provides infrastructure for the software development like Operating System, Database Management System, User Interface Management System, Networking System, etc.

# COCOMO-II

- **Stages of COCOMO II:**
- **Stage-I:**
  It supports estimation of prototyping. For this it uses *Application Composition Estimation Model*. This model is used for the prototyping stage of application generator and system integration.
- **Stage-II:**
  It supports estimation in the early design stage of the project, when we less know about it. For this it uses *Early Design Estimation Model*. This model is used in early design stage of application generators, infrastructure, system integration.
- **Stage-III:**
  It supports estimation in the post architecture stage of a project. For this it uses *Post Architecture Estimation Model*. This model is used after the completion of the detailed architecture of application generator, infrastructure, system integration.

# COCOMO-II

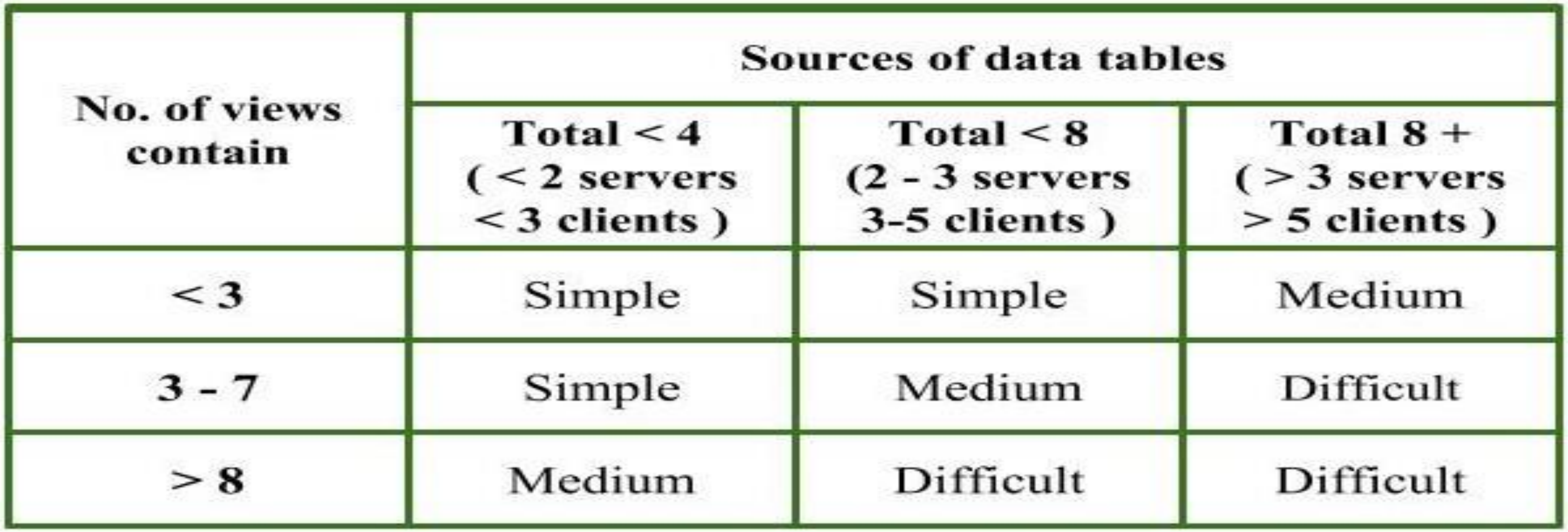
***Application Composition Estimation Model***


Access Object Count
Classify complexity level of each object
Assign complexity weight to each object
Determine Object Point
Compute new Object Points
Calculate productivity rate
Compute estimated Effort

# COCOMO-II

i. Assess object counts: Estimate the number of screens, reports and 3 GL components that will comprise this application.

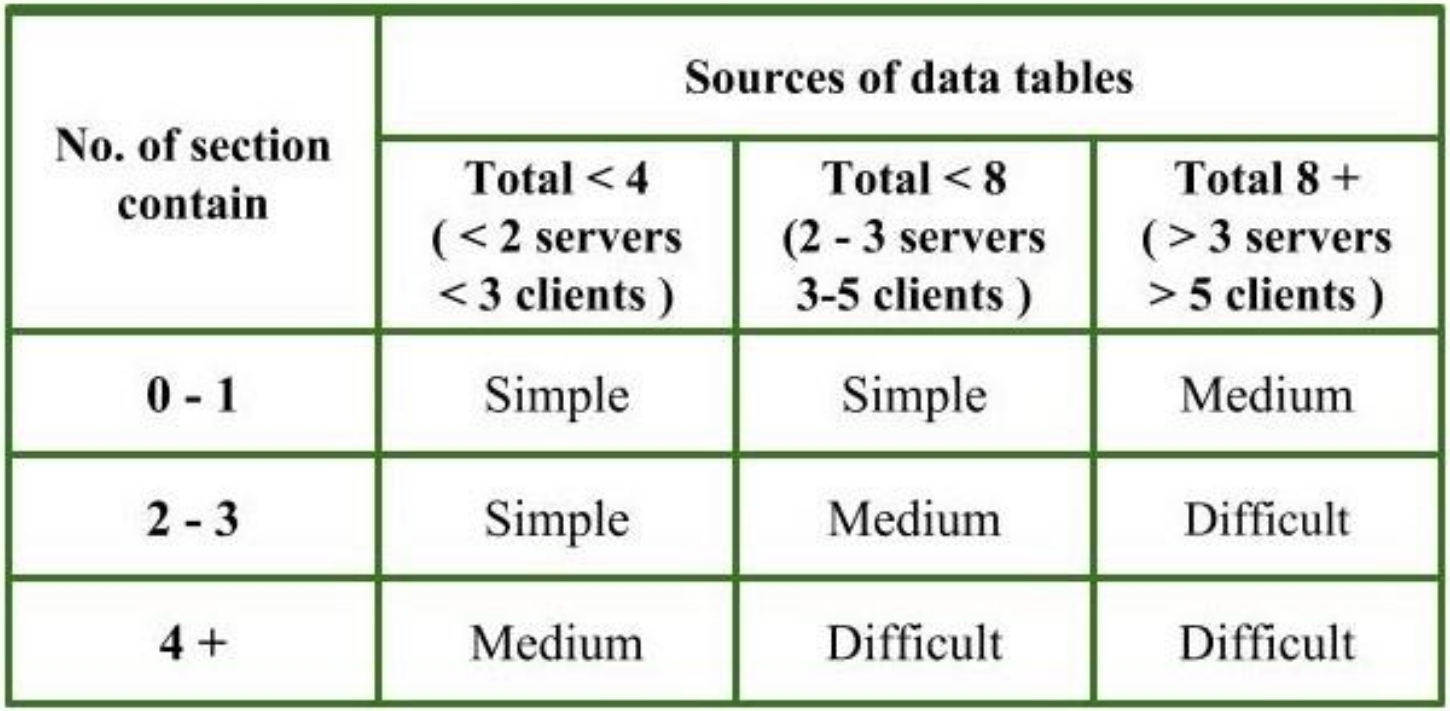
ii. Classification of complexity levels: We have to classify each object instance into simple, medium and difficult complexity levels depending on values of its characteristics.



| No. of views contain | Sources of data tables | | |
|---|---|---|---|
| | Total < 4<br>( < 2 servers<br>< 3 clients ) | Total < 8<br>(2 - 3 servers<br>3-5 clients ) | Total 8 +<br>( > 3 servers<br>> 5 clients ) |
| < 3 | Simple | Simple | Medium |
| 3 - 7 | Simple | Medium | Difficult |
| > 8 | Medium | Difficult | Difficult |

For Screens

# COCOMO-II

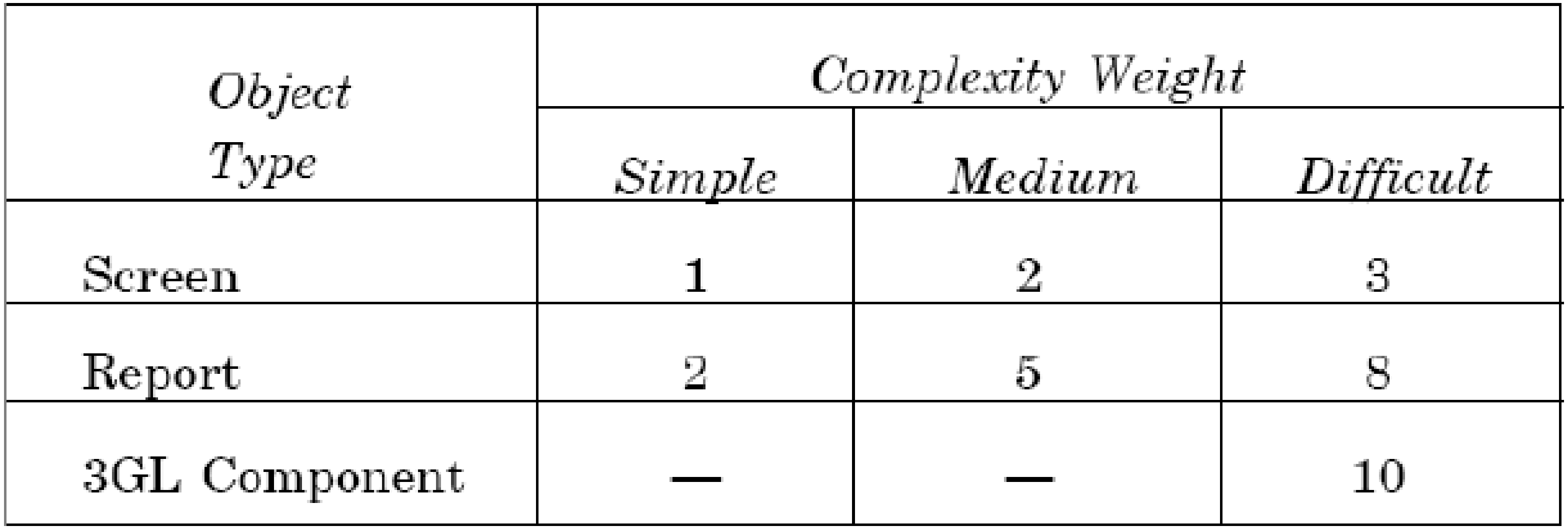


| No. of section contain | Sources of data tables | | |
|---|---|---|---|
| | Total < 4<br>( < 2 servers<br>< 3 clients ) | Total < 8<br>(2 - 3 servers<br>3-5 clients ) | Total 8 +<br>( > 3 servers<br>> 5 clients ) |
| 0 - 1 | Simple | Simple | Medium |
| 2 - 3 | Simple | Medium | Difficult |
| 4 + | Medium | Difficult | Difficult |

**For Reports**

# COCOMO-II

iii. Assign complexity weight to each object : The weights are used for three object types i.e., screen, report and 3GL components using the Table 10.



| *Object Type* | *Complexity Weight* | | |
|---|---|---|---|
| | *Simple* | *Medium* | *Difficult* |
| Screen | 1 | 2 | 3 |
| Report | 2 | 5 | 8 |
| 3GL Component | — | — | 10 |

**Table 10:** Complexity weights for each level

# COCOMO-II

iv. Determine object points: Add all the weighted object instances to get one number and this known as object-point count.

v. Compute new object points: We have to estimate the percentage of reuse to be achieved in a project. Depending on the percentage reuse, the new object points (NOP) are computed.

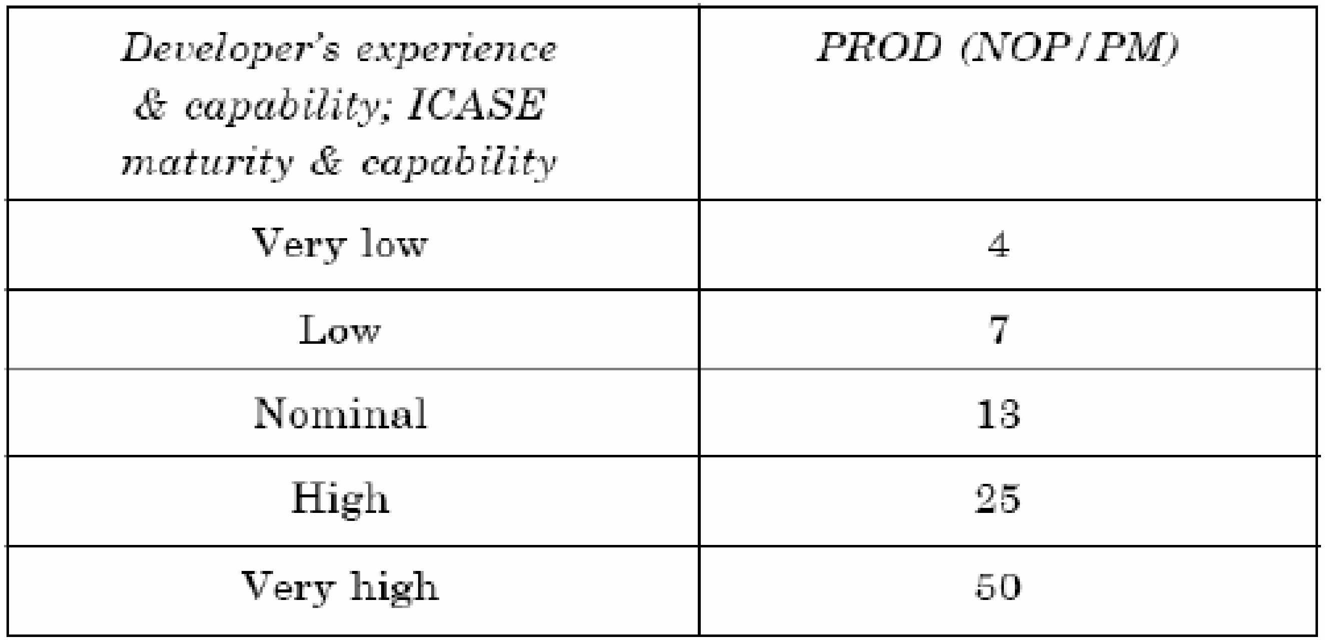
$$\textbf{NOP} = \frac{(\text{object points}) * (100\text{-}\%\text{reuse})}{100}$$

NOP are the object points that will need to be developed and differ from the object point count because there may be reuse.

# COCOMO-II

vi. Calculation of productivity rate: The productivity rate can be calculated as:

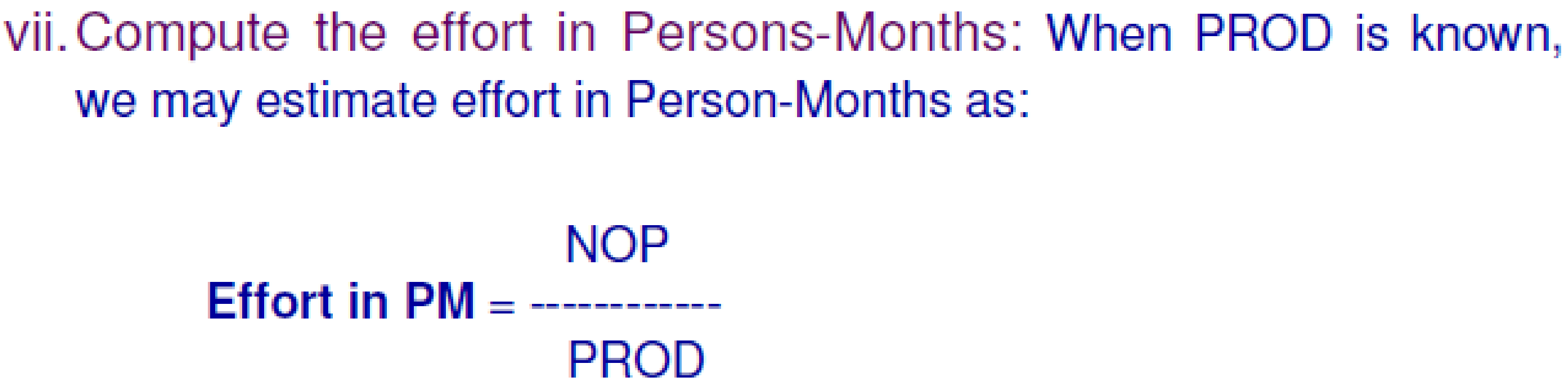
Productivity rate (PROD) = NOP/Person month



| *Developer's experience & capability; ICASE maturity & capability* | *PROD (NOP/PM)* |
|---|---|
| Very low | 4 |
| Low | 7 |
| Nominal | 13 |
| High | 25 |
| Very high | 50 |

**Table 11:** Productivity values

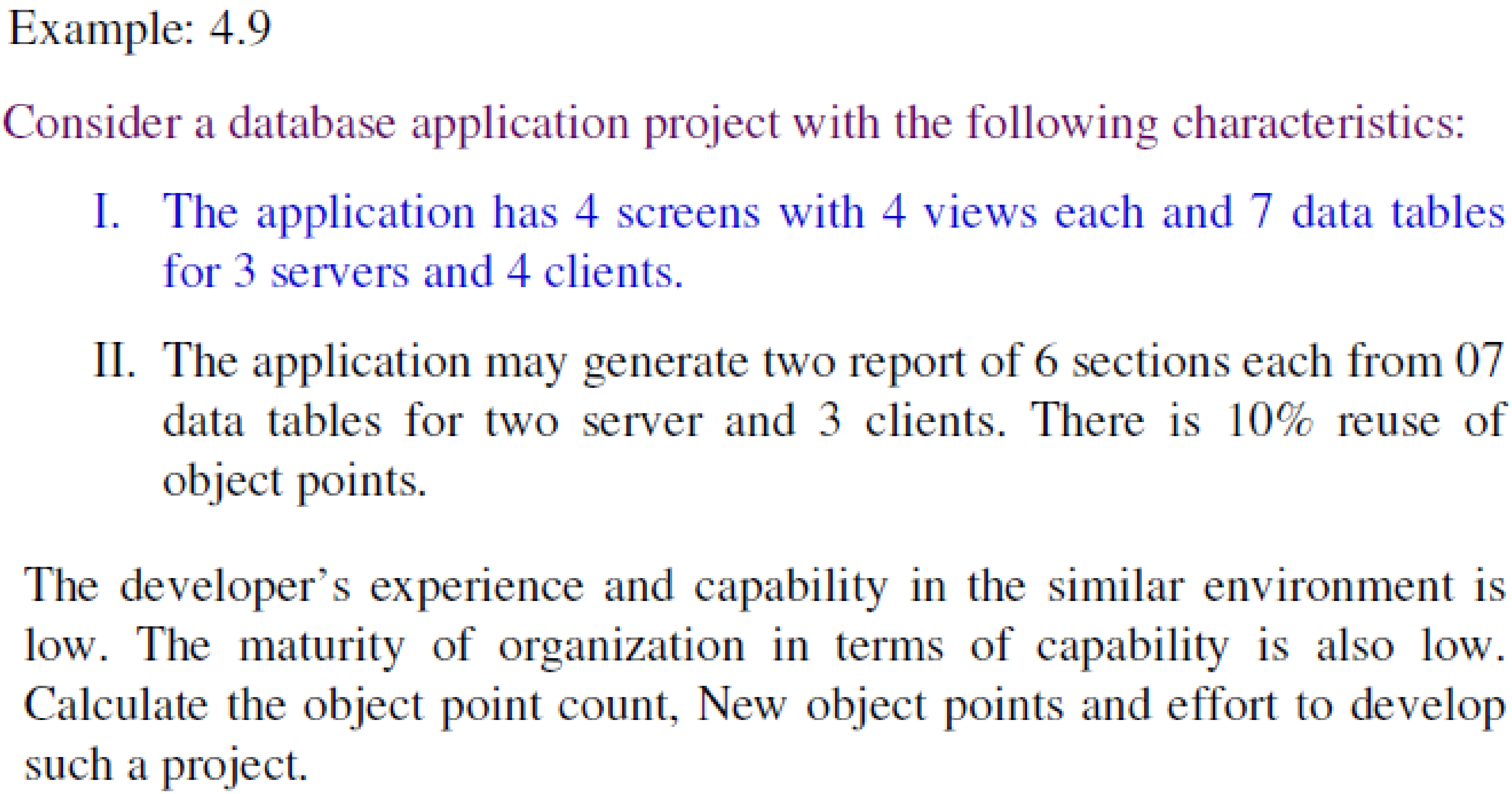
# COCOMO-II



vii. Compute the effort in Persons-Months: When PROD is known, we may estimate effort in Person-Months as:

$$\textbf{Effort in PM} = \frac{NOP}{PROD}$$

# COCOMO-II



Example: 4.9

Consider a database application project with the following characteristics:

I. The application has 4 screens with 4 views each and 7 data tables for 3 servers and 4 clients.

II. The application may generate two report of 6 sections each from 07 data tables for two server and 3 clients. There is 10% reuse of object points.

The developer's experience and capability in the similar environment is low. The maturity of organization in terms of capability is also low. Calculate the object point count, New object points and effort to develop such a project.

# COCOMO-II

## Solution

This project comes under the category of application composition estimation model.

Number of screens = 4 with 4 views each

Number of reports = 2 with 6 sections each

From Table 9 we know that each screen will be of medium complexity and each report will be difficult complexity.

Using Table 10 of complexity weights, we may calculate object point count.

$$= 4 \text{ x } 2 + 2 \text{ x } 8 = 24$$

$$\textbf{NOP} = \frac{24 * (100 - 10)}{100} = 21.6$$

# COCOMO-II

Table 11 gives the low value of productivity (PROD) i.e. 7.

$$\text{Efforts in PM} = \frac{NOP}{PROD}$$

$$\text{Efforts} = \frac{21.6}{7} = 3.086 \text{ PM}$$

# COCOMO-II

## The Early Design Model

The COCOMO-II models use the base equation of the form

$$PM_{nominal} = A * (size)^{B}$$

**where**

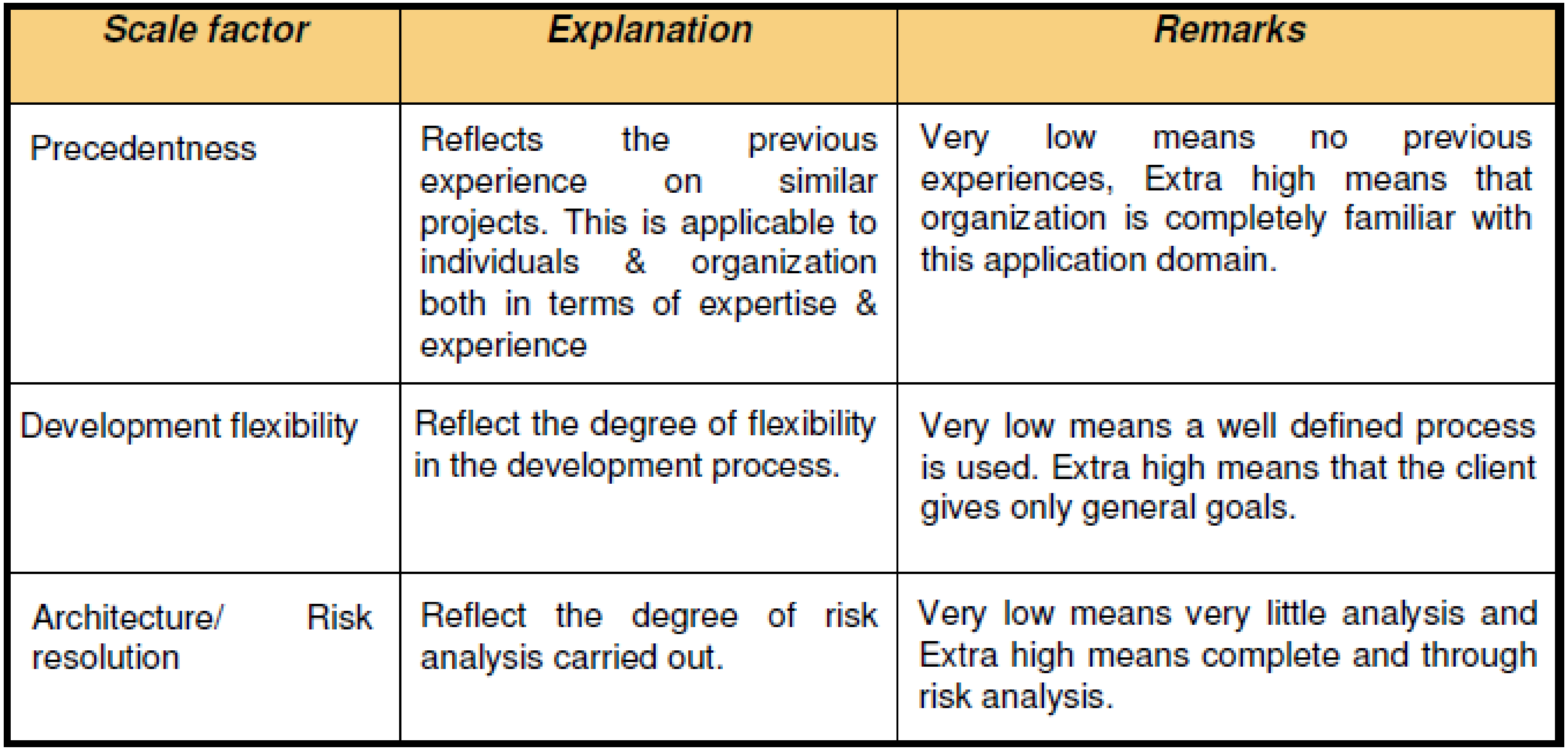
**$PM_{nominal}$ =** Effort of the project in person months

**A =** Constant representing the nominal productivity, provisionally set to 2.5

**B =** Scale factor

**Size =** Software size

# COCOMO-II

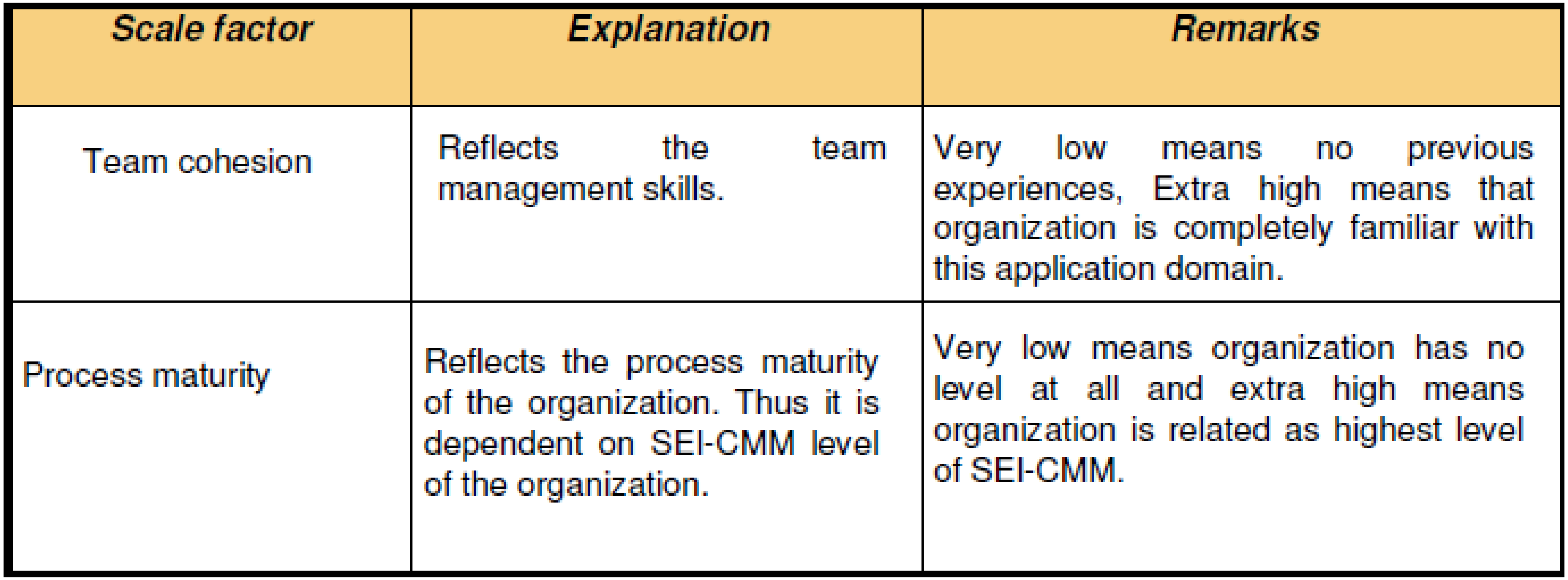


| *Scale factor* | *Explanation* | *Remarks* |
|---|---|---|
| Precedentness | Reflects the previous experience on similar projects. This is applicable to individuals & organization both in terms of expertise & experience | Very low means no previous experiences, Extra high means that organization is completely familiar with this application domain. |
| Development flexibility | Reflect the degree of flexibility in the development process. | Very low means a well defined process is used. Extra high means that the client gives only general goals. |
| Architecture/ Risk resolution | Reflect the degree of risk analysis carried out. | Very low means very little analysis and Extra high means complete and through risk analysis. |

**Cont…**

**Table 12:** Scaling factors required for the calculation of the value of B

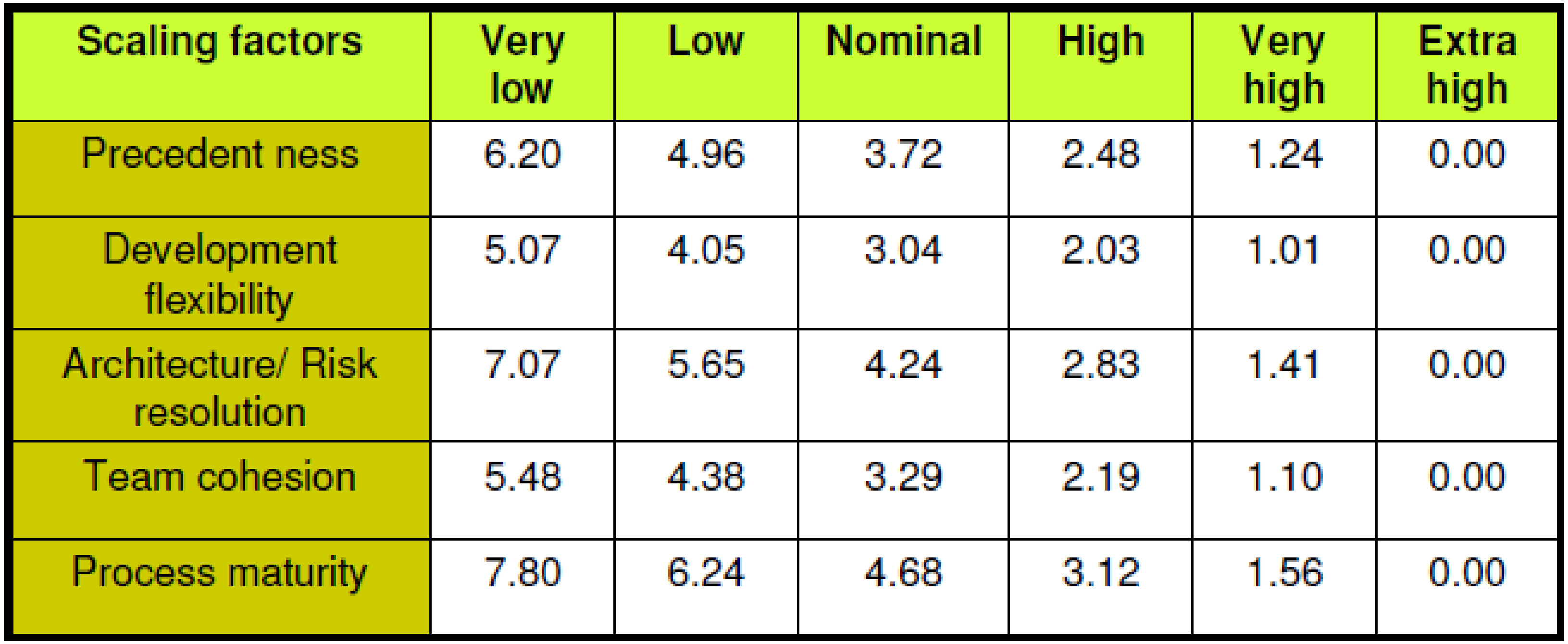
# COCOMO-II



| Scale factor | Explanation | Remarks |
|---|---|---|
| Team cohesion | Reflects the team management skills. | Very low means no previous experiences, Extra high means that organization is completely familiar with this application domain. |
| Process maturity | Reflects the process maturity of the organization. Thus it is dependent on SEI-CMM level of the organization. | Very low means organization has no level at all and extra high means organization is related as highest level of SEI-CMM. |

**Table 12:** Scaling factors required for the calculation of the value of B

# COCOMO-II



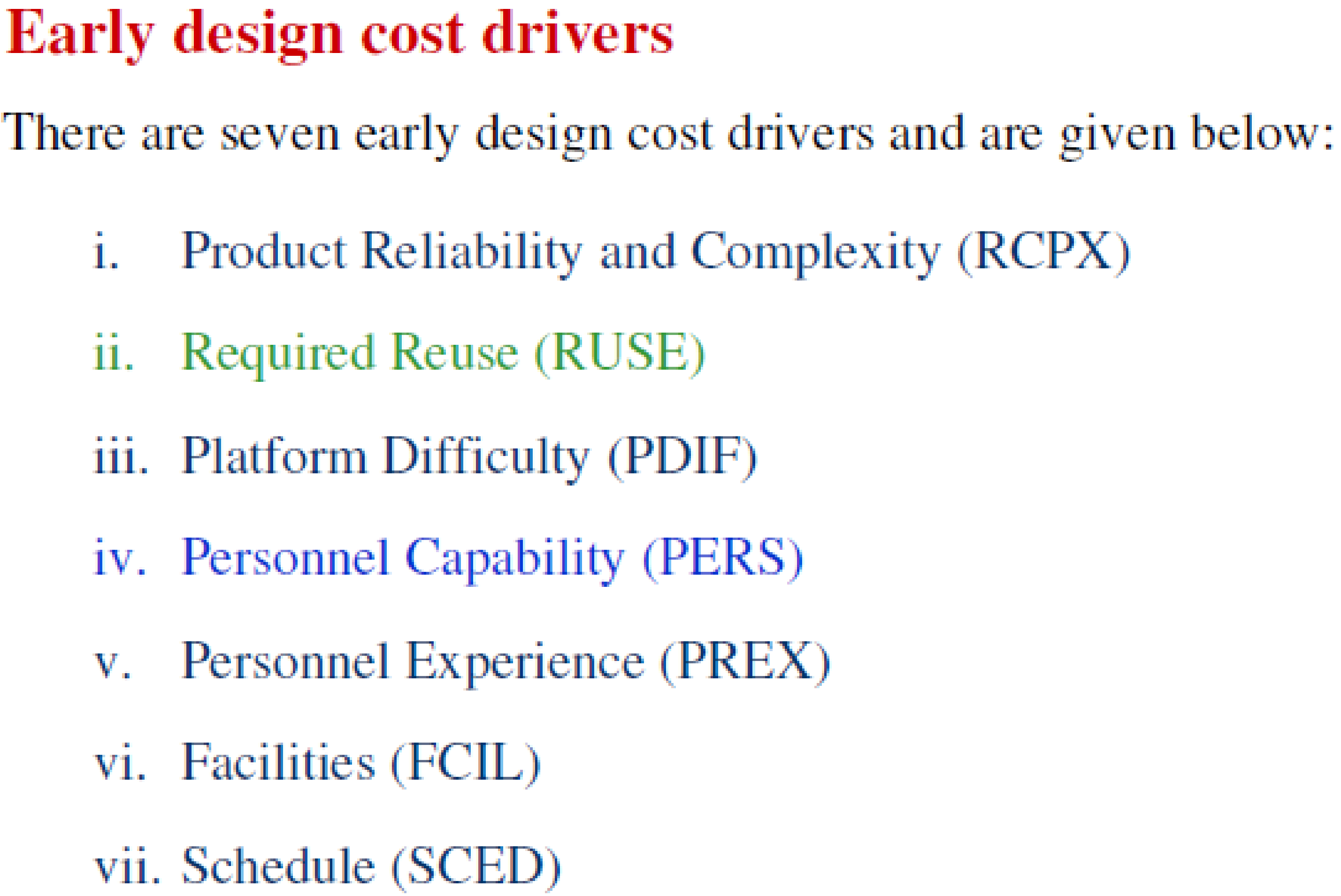
| Scaling factors | Very low | Low | Nominal | High | Very high | Extra high |
|---|---|---|---|---|---|---|
| Precedent ness | 6.20 | 4.96 | 3.72 | 2.48 | 1.24 | 0.00 |
| Development flexibility | 5.07 | 4.05 | 3.04 | 2.03 | 1.01 | 0.00 |
| Architecture/ Risk resolution | 7.07 | 5.65 | 4.24 | 2.83 | 1.41 | 0.00 |
| Team cohesion | 5.48 | 4.38 | 3.29 | 2.19 | 1.10 | 0.00 |
| Process maturity | 7.80 | 6.24 | 4.68 | 3.12 | 1.56 | 0.00 |

**Table 13:** Data for the Computation of B

The value of B can be calculated as:

$$B=0.91 + 0.01 * (\text{Sum of rating on scaling factors for the project})$$

# COCOMO-II



## Early design cost drivers

There are seven early design cost drivers and are given below:

i. Product Reliability and Complexity (RCPX)

ii. Required Reuse (RUSE)

iii. Platform Difficulty (PDIF)

iv. Personnel Capability (PERS)

v. Personnel Experience (PREX)

vi. Facilities (FCIL)

vii. Schedule (SCED)

# COCOMO-II

## Post architecture cost drivers

There are 17 cost drivers in the Post Architecture model. These are rated on a scale of 1 to 6 as given below :

| *Very Low* | *Low* | *Nominal* | *High* | *Very High* | *Extra High* |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |

The list of seventeen cost drivers is given below :

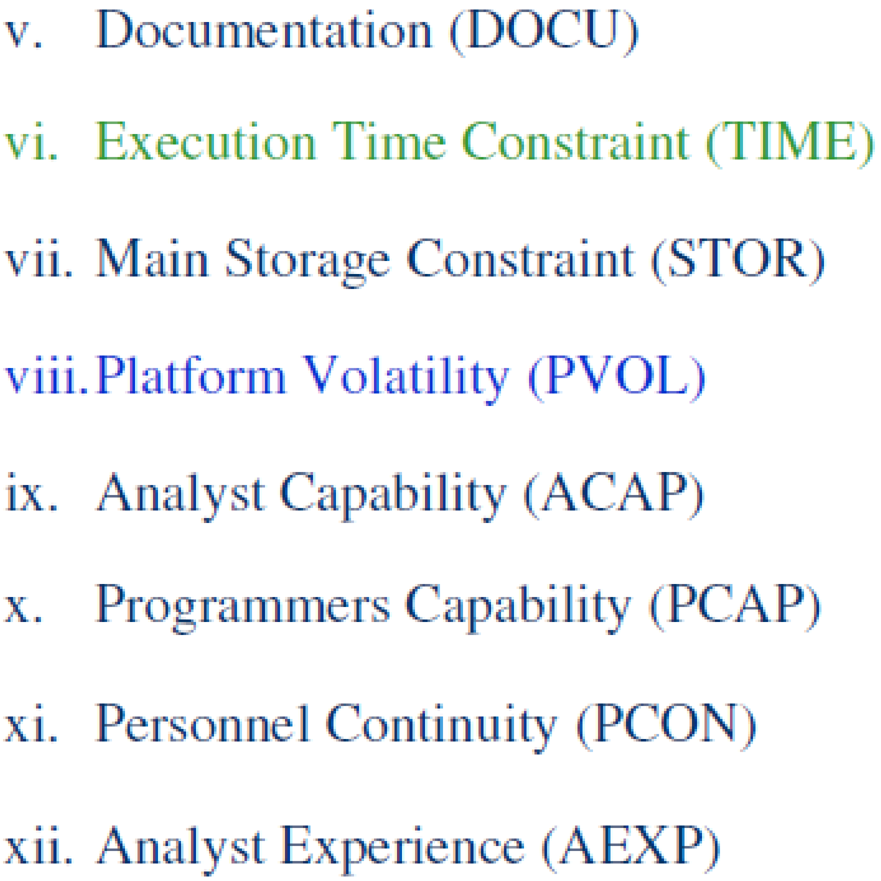
i. Reliability Required (RELY)

ii. Database Size (DATA)

iii. Product Complexity (CPLX)

iv. Required Reusability (RUSE)

# COCOMO-II



v. Documentation (DOCU)

vi. Execution Time Constraint (TIME)

vii. Main Storage Constraint (STOR)

viii. Platform Volatility (PVOL)

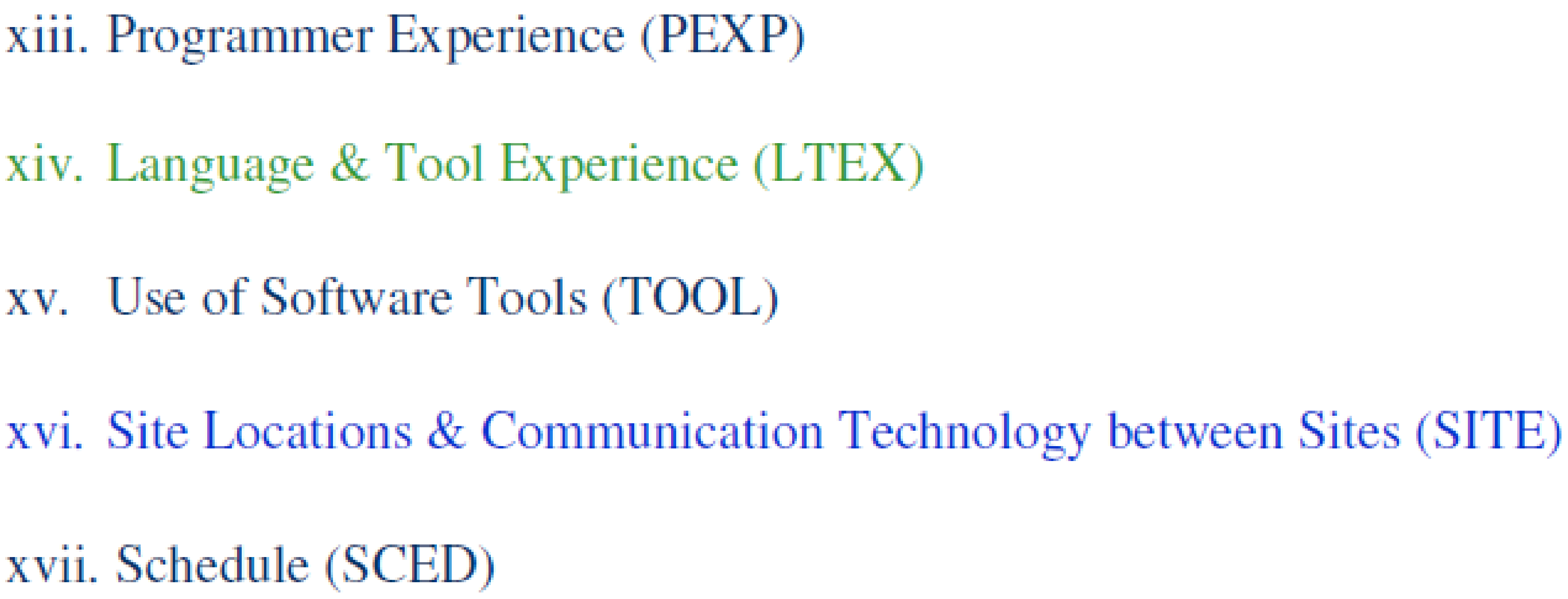
ix. Analyst Capability (ACAP)

x. Programmers Capability (PCAP)

xi. Personnel Continuity (PCON)

xii. Analyst Experience (AEXP)

# COCOMO-II



xiii. Programmer Experience (PEXP)

xiv. Language & Tool Experience (LTEX)

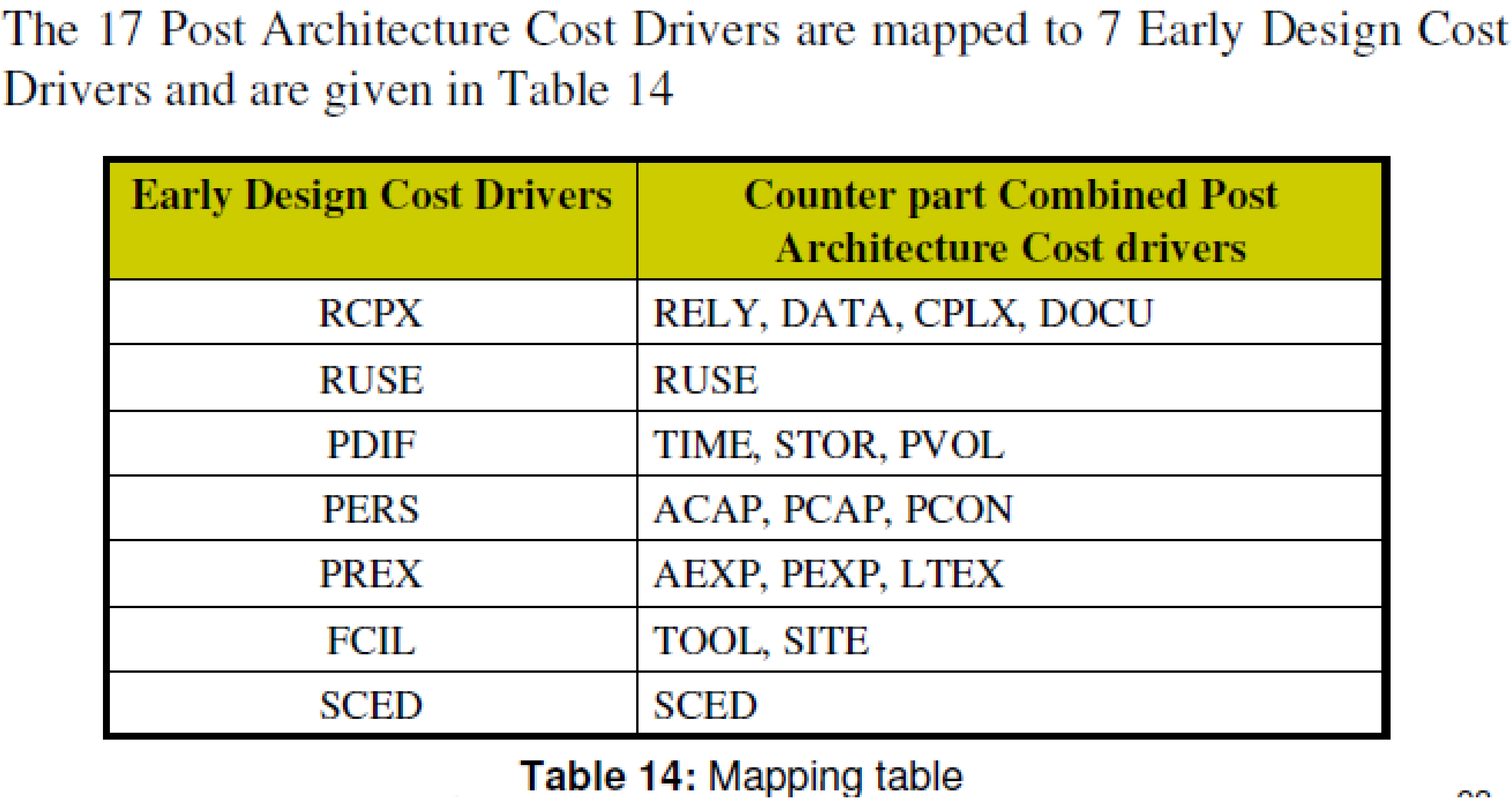
xv. Use of Software Tools (TOOL)

xvi. Site Locations & Communication Technology between Sites (SITE)

xvii. Schedule (SCED)

# COCOMO-II

## Mapping of early design cost drivers and post architecture cost drivers



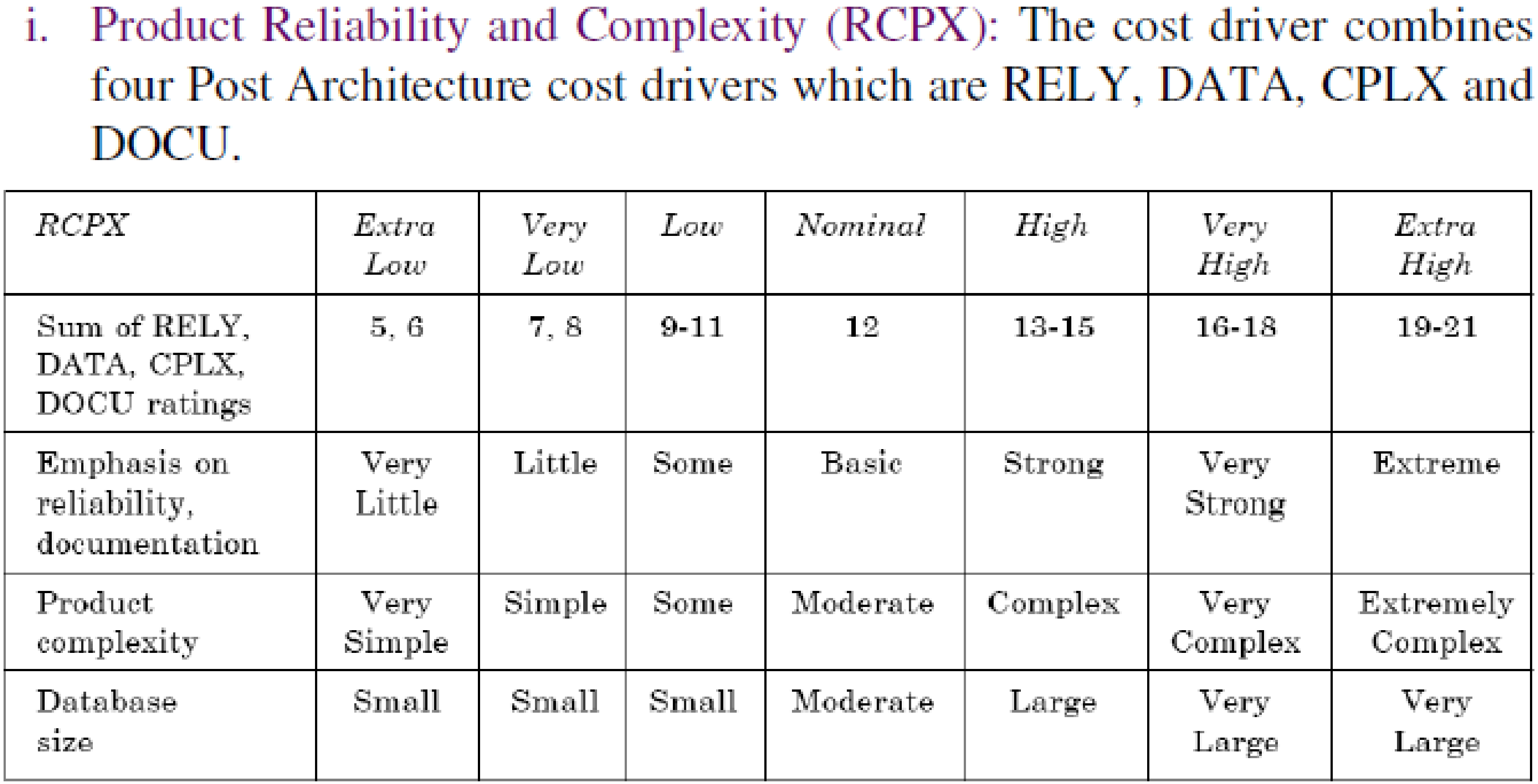
The 17 Post Architecture Cost Drivers are mapped to 7 Early Design Cost Drivers and are given in Table 14

| Early Design Cost Drivers | Counter part Combined Post Architecture Cost drivers |
|---|---|
| RCPX | RELY, DATA, CPLX, DOCU |
| RUSE | RUSE |
| PDIF | TIME, STOR, PVOL |
| PERS | ACAP, PCAP, PCON |
| PREX | AEXP, PEXP, LTEX |
| FCIL | TOOL, SITE |
| SCED | SCED |

**Table 14:** Mapping table

# COCOMO-II

## Product of cost drivers for early design model

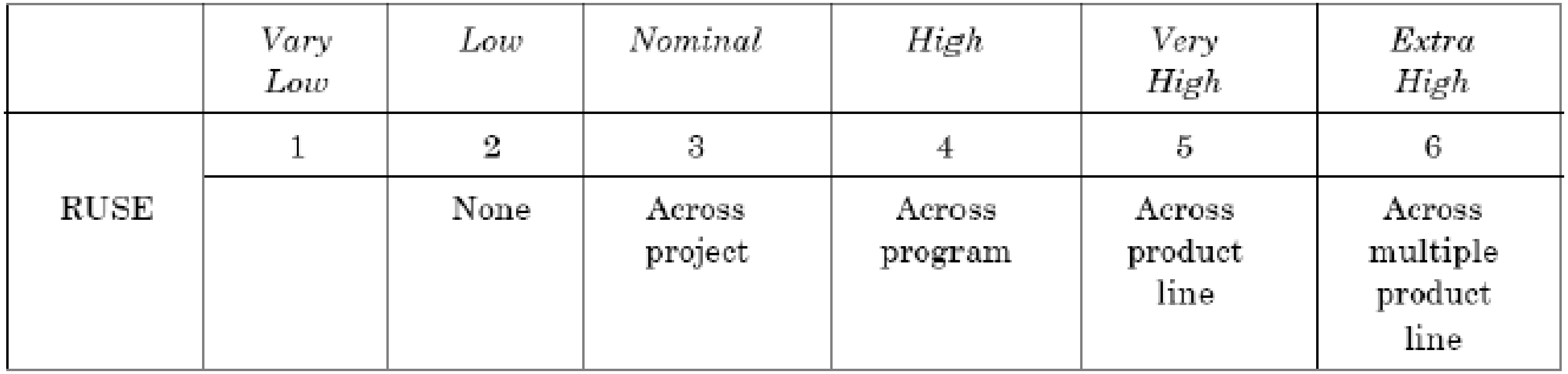


i. Product Reliability and Complexity (RCPX): The cost driver combines four Post Architecture cost drivers which are RELY, DATA, CPLX and DOCU.

| *RCPX* | *Extra Low* | *Very Low* | *Low* | *Nominal* | *High* | *Very High* | *Extra High* |
|---|---|---|---|---|---|---|---|
| Sum of RELY, DATA, CPLX, DOCU ratings | 5, 6 | 7, 8 | 9-11 | 12 | 13-15 | 16-18 | 19-21 |
| Emphasis on reliability, documentation | Very Little | Little | Some | Basic | Strong | Very Strong | Extreme |
| Product complexity | Very Simple | Simple | Some | Moderate | Complex | Very Complex | Extremely Complex |
| Database size | Small | Small | Small | Moderate | Large | Very Large | Very Large |

# COCOMO-II

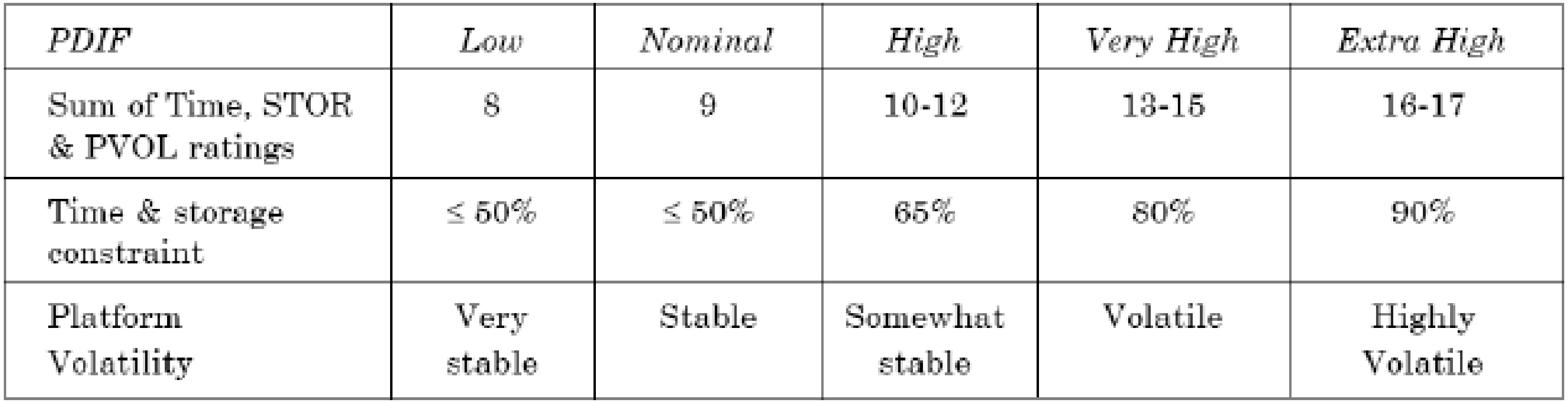
ii. Required Reuse (RUSE) : This early design model cost driver is same as its Post architecture Counterpart. The RUSE rating levels are (as per Table 16):



| | *Vary Low* | *Low* | *Nominal* | *High* | *Very High* | *Extra High* |
|---|---|---|---|---|---|---|
| RUSE | 1 | 2 | 3 | 4 | 5 | 6 |
| | | None | Across project | Across program | Across product line | Across multiple product line |

# COCOMO-II

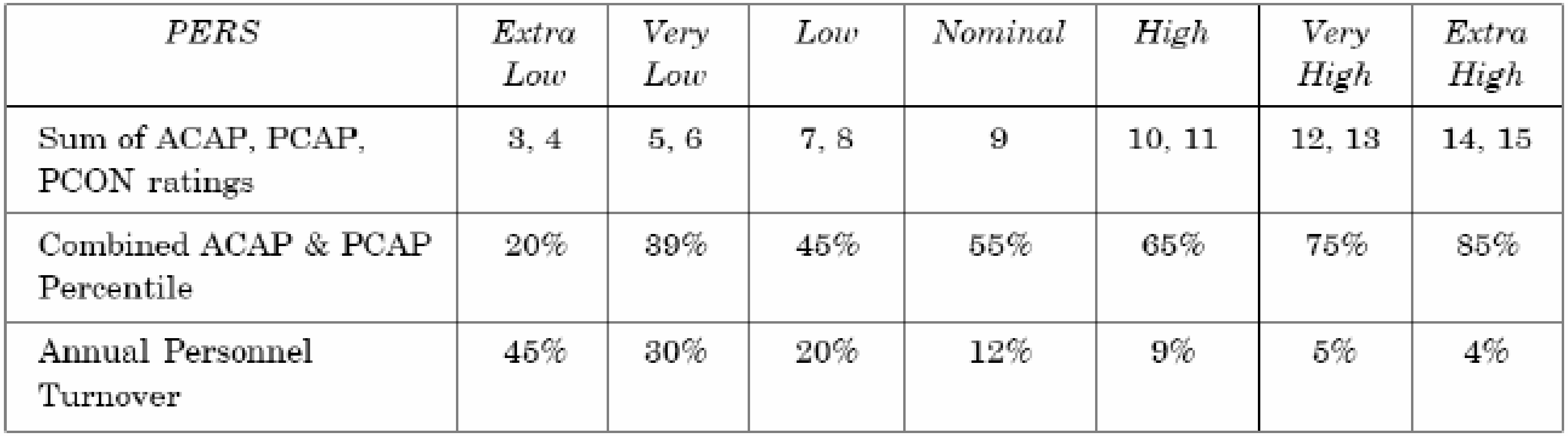
iii. Platform Difficulty (PDIF) : This cost driver combines TIME, STOR and PVOL of Post Architecture Cost Drivers.



| *PDIF* | *Low* | *Nominal* | *High* | *Very High* | *Extra High* |
|---|---|---|---|---|---|
| Sum of Time, STOR & PVOL ratings | 8 | 9 | 10-12 | 13-15 | 16-17 |
| Time & storage constraint | ≤ 50% | ≤ 50% | 65% | 80% | 90% |
| Platform Volatility | Very stable | Stable | Somewhat stable | Volatile | Highly Volatile |

# COCOMO-II

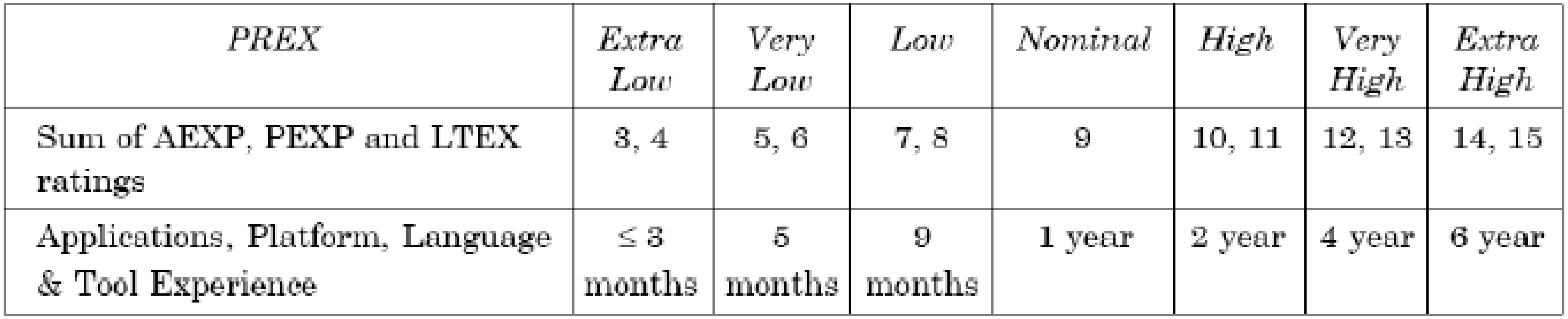
iv. Personnel Capability (PERS) : This cost driver combines three Post Architecture Cost Drivers. These drivers are ACAP, PCAP and PCON.



| *PERS* | *Extra Low* | *Very Low* | *Low* | *Nominal* | *High* | *Very High* | *Extra High* |
|---|---|---|---|---|---|---|---|
| Sum of ACAP, PCAP, PCON ratings | 3, 4 | 5, 6 | 7, 8 | 9 | 10, 11 | 12, 13 | 14, 15 |
| Combined ACAP & PCAP Percentile | 20% | 39% | 45% | 55% | 65% | 75% | 85% |
| Annual Personnel Turnover | 45% | 30% | 20% | 12% | 9% | 5% | 4% |

# COCOMO-II

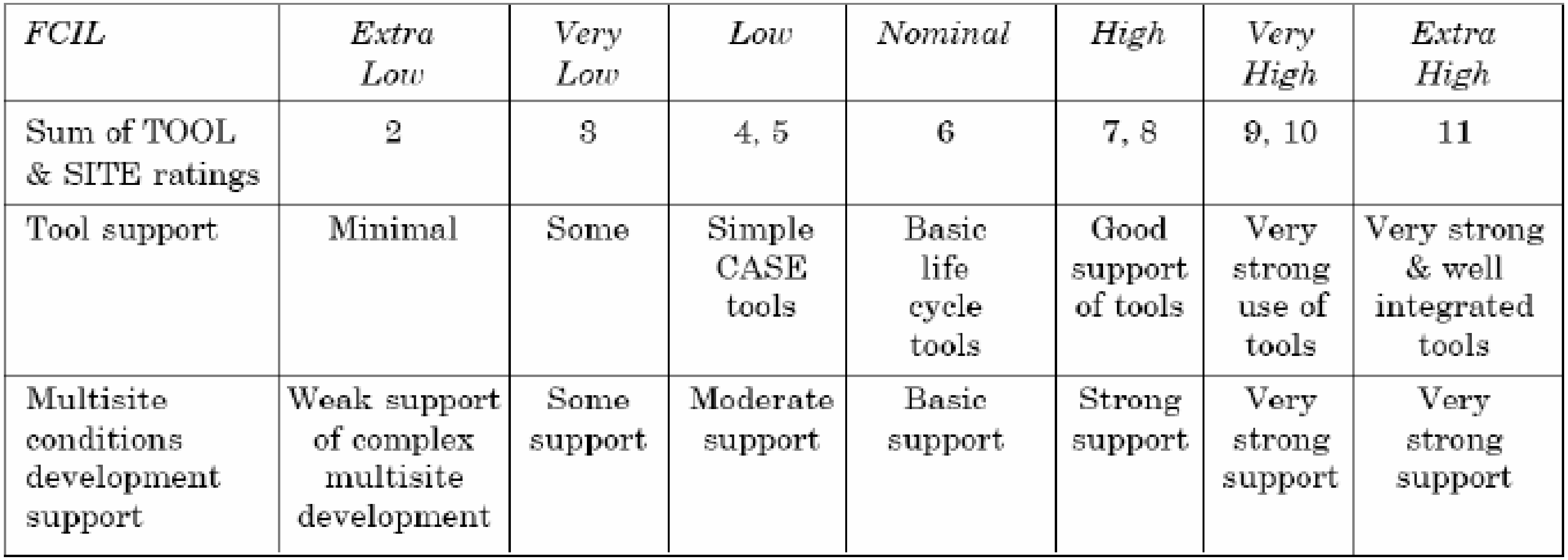
v. Personnel Experience (PREX) : This early design driver combines three Post Architecture Cost Drivers, which are AEXP, PEXP and LTEX.



| *PREX* | *Extra Low* | *Very Low* | *Low* | *Nominal* | *High* | *Very High* | *Extra High* |
|---|---|---|---|---|---|---|---|
| Sum of AEXP, PEXP and LTEX ratings | 3, 4 | 5, 6 | 7, 8 | 9 | 10, 11 | 12, 13 | 14, 15 |
| Applications, Platform, Language & Tool Experience | ≤ 3 months | 5 months | 9 months | 1 year | 2 year | 4 year | 6 year |

# COCOMO-II

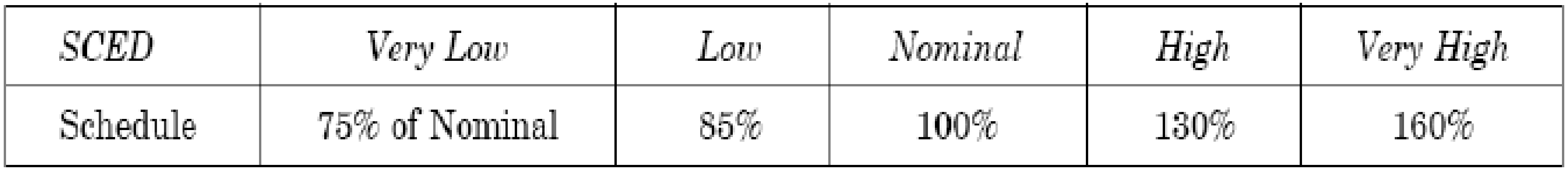
vi. Facilities (FCIL): This depends on two Post Architecture Cost Drivers, which are TOOL and SITE.



| *FCIL* | *Extra Low* | *Very Low* | *Low* | *Nominal* | *High* | *Very High* | *Extra High* |
|---|---|---|---|---|---|---|---|
| Sum of TOOL & SITE ratings | 2 | 3 | 4, 5 | 6 | 7, 8 | 9, 10 | 11 |
| Tool support | Minimal | Some | Simple CASE tools | Basic life cycle tools | Good support of tools | Very strong use of tools | Very strong & well integrated tools |
| Multisite conditions development support | Weak support of complex multisite development | Some support | Moderate support | Basic support | Strong support | Very strong support | Very strong support |

# COCOMO-II

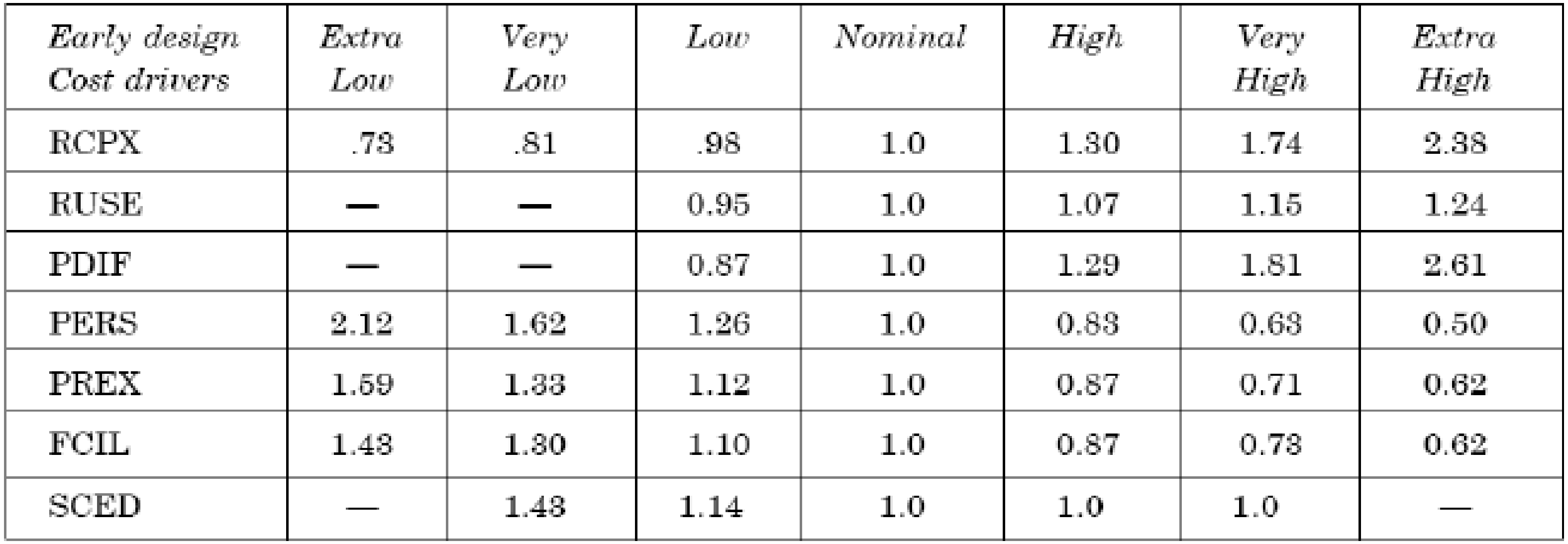
vii.Schedule (SCED) : This early design cost driver is the same as Post Architecture Counterpart and rating level are given below using table 16.



| *SCED* | *Very Low* | *Low* | *Nominal* | *High* | *Very High* |
|---|---|---|---|---|---|
| Schedule | 75% of Nominal | 85% | 100% | 130% | 160% |

# COCOMO-II

The seven early design cost drivers have been converted into numeric values with a Nominal value 1.0. These values are used for the calculation of a factor called "Effort multiplier" which is the product of all seven early design cost drivers. The numeric values are given in Table 15.



| *Early design Cost drivers* | *Extra Low* | *Very Low* | *Low* | *Nominal* | *High* | *Very High* | *Extra High* |
|---|---|---|---|---|---|---|---|
| RCPX | .73 | .81 | .98 | 1.0 | 1.30 | 1.74 | 2.38 |
| RUSE | — | — | 0.95 | 1.0 | 1.07 | 1.15 | 1.24 |
| PDIF | — | — | 0.87 | 1.0 | 1.29 | 1.81 | 2.61 |
| PERS | 2.12 | 1.62 | 1.26 | 1.0 | 0.83 | 0.63 | 0.50 |
| PREX | 1.59 | 1.33 | 1.12 | 1.0 | 0.87 | 0.71 | 0.62 |
| FCIL | 1.43 | 1.30 | 1.10 | 1.0 | 0.87 | 0.73 | 0.62 |
| SCED | — | 1.43 | 1.14 | 1.0 | 1.0 | 1.0 | — |

**Table 15:** Early design parameters

# COCOMO-II

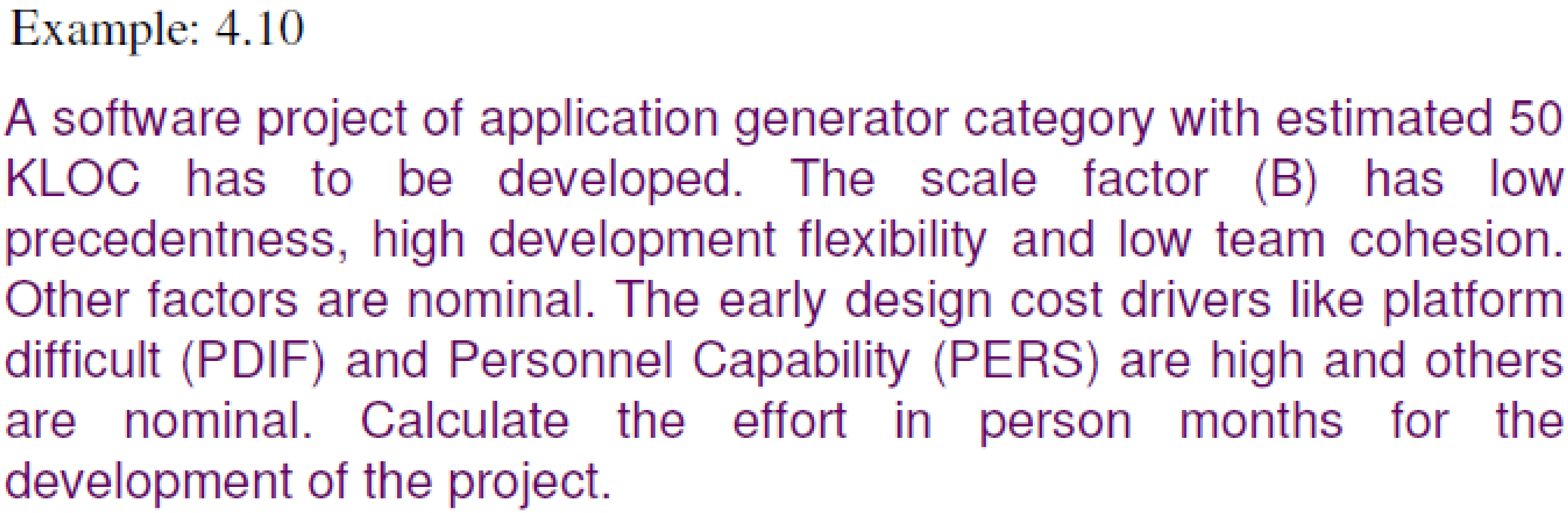
The early design model adjusts the nominal effort using 7 effort multipliers (EMs). Each effort multiplier (also called drivers) has 7 possible weights as given in Table 15. These factors are used for the calculation of adjusted effort as given below:

$$PM_{adjusted} = PM_{\text{nominal}} \times \left[ \prod_{i=7}^{7} EM_i \right]$$

$PM_{\text{adjusted}}$ effort may very even up to 400% from $PM_{\text{nominal}}$

Hence $PM_{\text{adjusted}}$ is the fine tuned value of effort in the early design phase

# COCOMO-II



Example: 4.10

A software project of application generator category with estimated 50 KLOC has to be developed. The scale factor (B) has low precedentness, high development flexibility and low team cohesion. Other factors are nominal. The early design cost drivers like platform difficult (PDIF) and Personnel Capability (PERS) are high and others are nominal. Calculate the effort in person months for the development of the project.

# COCOMO-II

**Solution**

Here $B = 0.91 + 0.01 * (\text{Sum of rating on scaling factors for the project})$

$= 0.91 + 0.01 * (4.96 + 2.03 + 4.24 + 4.38 + 4.68)$

$= 0.91 + 0.01(20.29) = 1.1129$

$PM_{nominal} = A^{*}(size)^{B}$

$= 2.5 * (50)^{1.1129} = 194.41$ Person months

The 7 cost drivers are

PDIF = high (1.29)

PERS = high (0.83)

RCPX = nominal (1.0)

RUSE = nominal (1.0)

PREX = nominal (1.0)

FCIL = nominal (1.0)

SCEO = nominal (1.0)

# COCOMO-II

$$PM_{adjusted} = PM_{\text{nominal}} \times \left[ \prod_{i=7}^{7} EM_i \right]$$

= 194.41 * [1.29 x 0.83)

= 194.41 x 1.07

= 208.155 Person months

# COCOMO-II

## Post Architecture Model

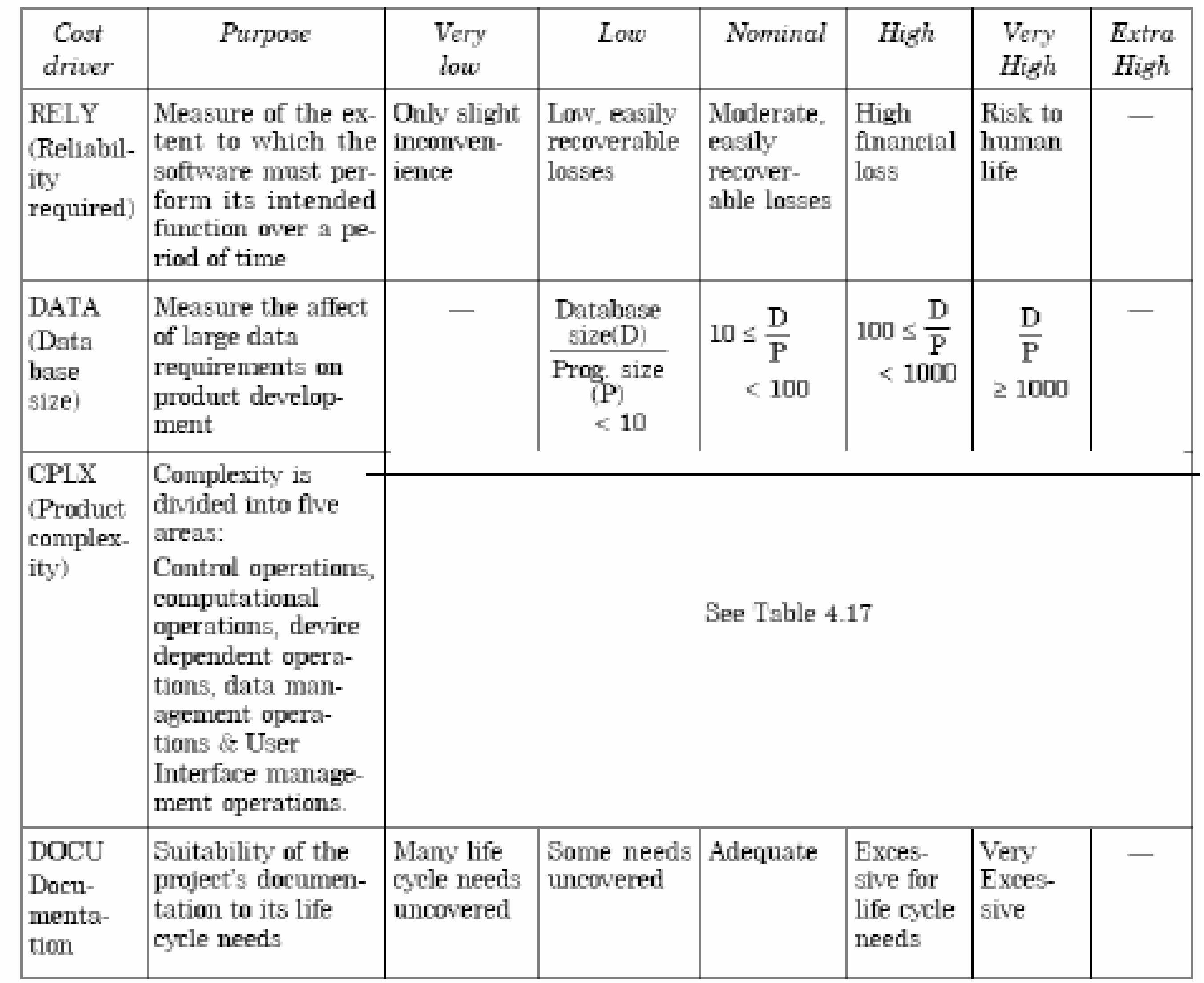
The post architecture model is the most detailed estimation model and is intended to be used when a software life cycle architecture has been completed. This model is used in the development and maintenance of software products in the application generators, system integration or infrastructure sectors.

$$PM_{adjusted} = PM_{\text{nominal}} \times \left[ \prod_{i=7}^{17} EM_i \right]$$

EM : Effort multiplier which is the product of 17 cost drivers.

The 17 cost drivers of the Post Architecture model are described in the table 16.

# COCOMO-II

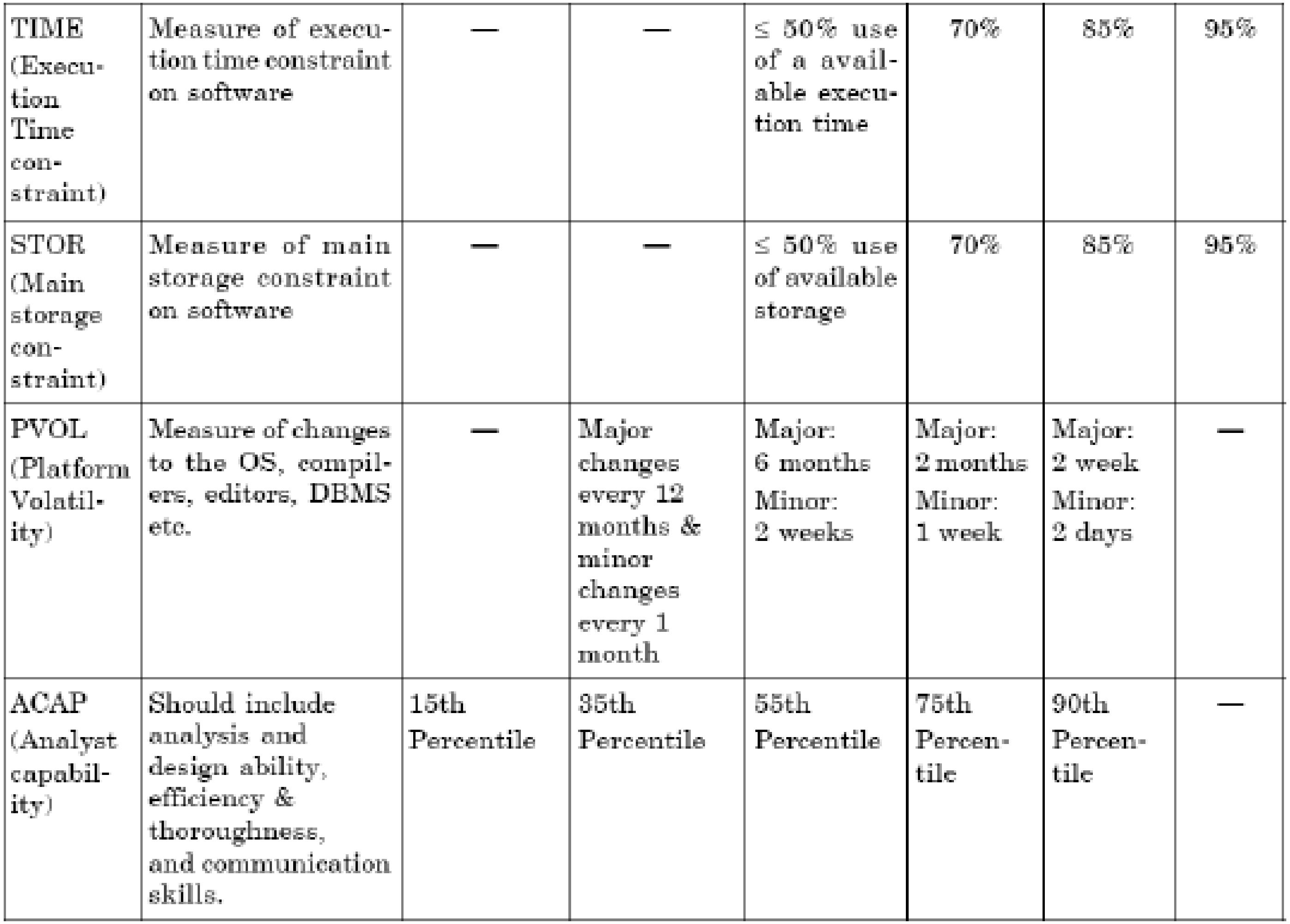


| *Cost driver* | *Purpose* | *Very low* | *Low* | *Nominal* | *High* | *Very High* | *Extra High* |
|---|---|---|---|---|---|---|---|
| RELY (Reliability required) | Measure of the extent to which the software must perform its intended function over a period of time | Only slight inconvenience | Low, easily recoverable losses | Moderate, easily recoverable losses | High financial loss | Risk to human life | — |
| DATA (Data base size) | Measure the affect of large data requirements on product development | — | $\frac{\text{Database size(D)}}{\text{Prog. size (P)}} < 10$ | $10 \le \frac{D}{P} < 100$ | $100 \le \frac{D}{P} < 1000$ | $\frac{D}{P} \ge 1000$ | — |
| CPLX (Product complexity) | Complexity is divided into five areas:<br>Control operations, computational operations, device dependent operations, data management operations & User Interface management operations. | See Table 4.17 | | | | | |
| DOCU Documentation | Suitability of the project's documentation to its life cycle needs | Many life cycle needs uncovered | Some needs uncovered | Adequate | Excessive for life cycle needs | Very Excessive | — |

Table 16: Post Architecture Cost Driver rating level summary

Cor

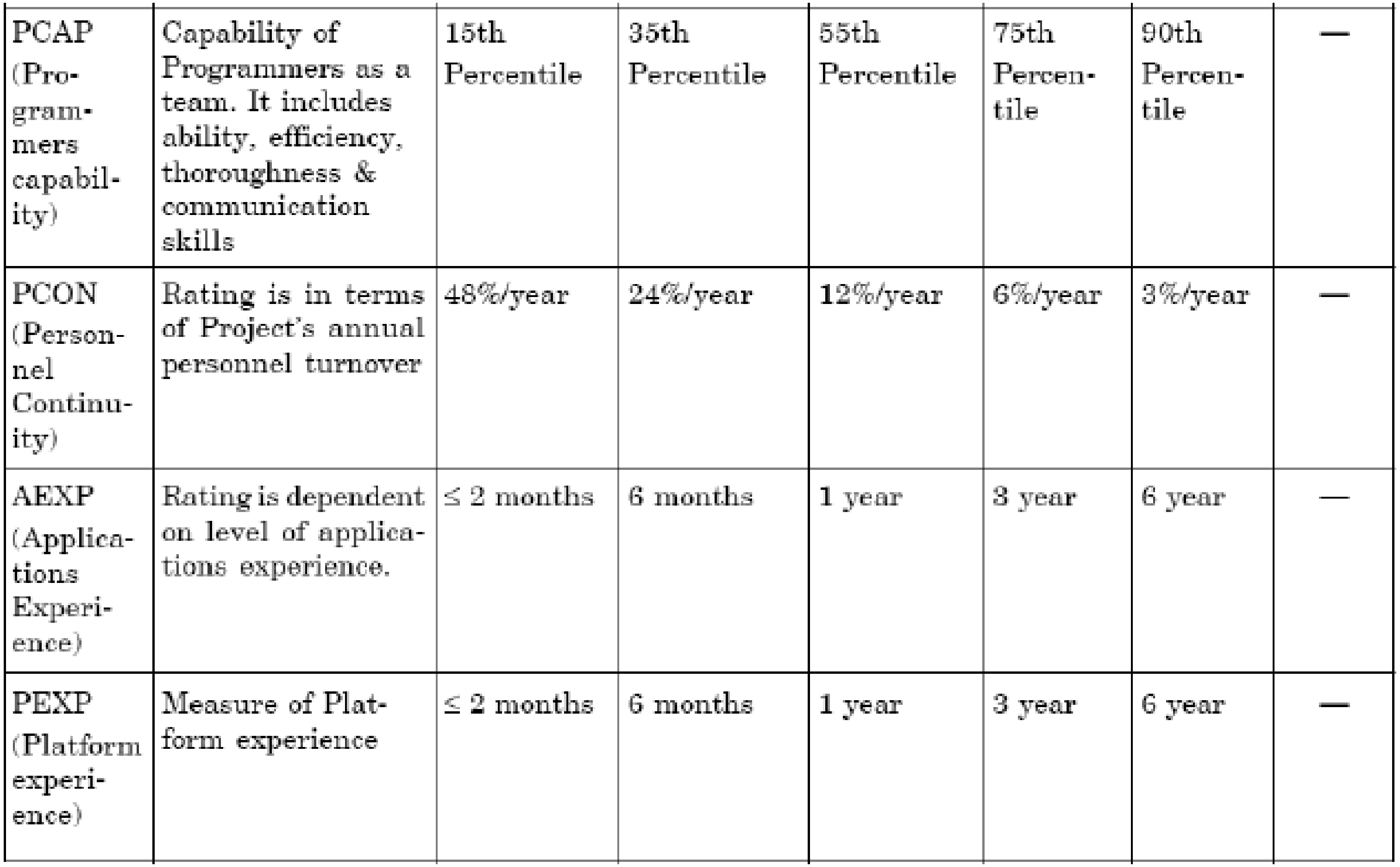
# COCOMO-II



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TIME (Execution Time constraint) | Measure of execution time constraint on software | — | — | ≤ 50% use of a available execution time | 70% | 85% | 95% |
| STOR (Main storage constraint) | Measure of main storage constraint on software | — | — | ≤ 50% use of available storage | 70% | 85% | 95% |
| PVOL (Platform Volatility) | Measure of changes to the OS, compilers, editors, DBMS etc. | — | Major changes every 12 months & minor changes every 1 month | Major:<br>6 months<br>Minor:<br>2 weeks | Major:<br>2 months<br>Minor:<br>1 week | Major:<br>2 week<br>Minor:<br>2 days | — |
| ACAP (Analyst capability) | Should include analysis and design ability, efficiency & thoroughness, and communication skills. | 15th Percentile | 35th Percentile | 55th Percentile | 75th Percentile | 90th Percentile | — |

**Table 16:** Post Architecture Cost Driver rating level summary

# COCOMO-II

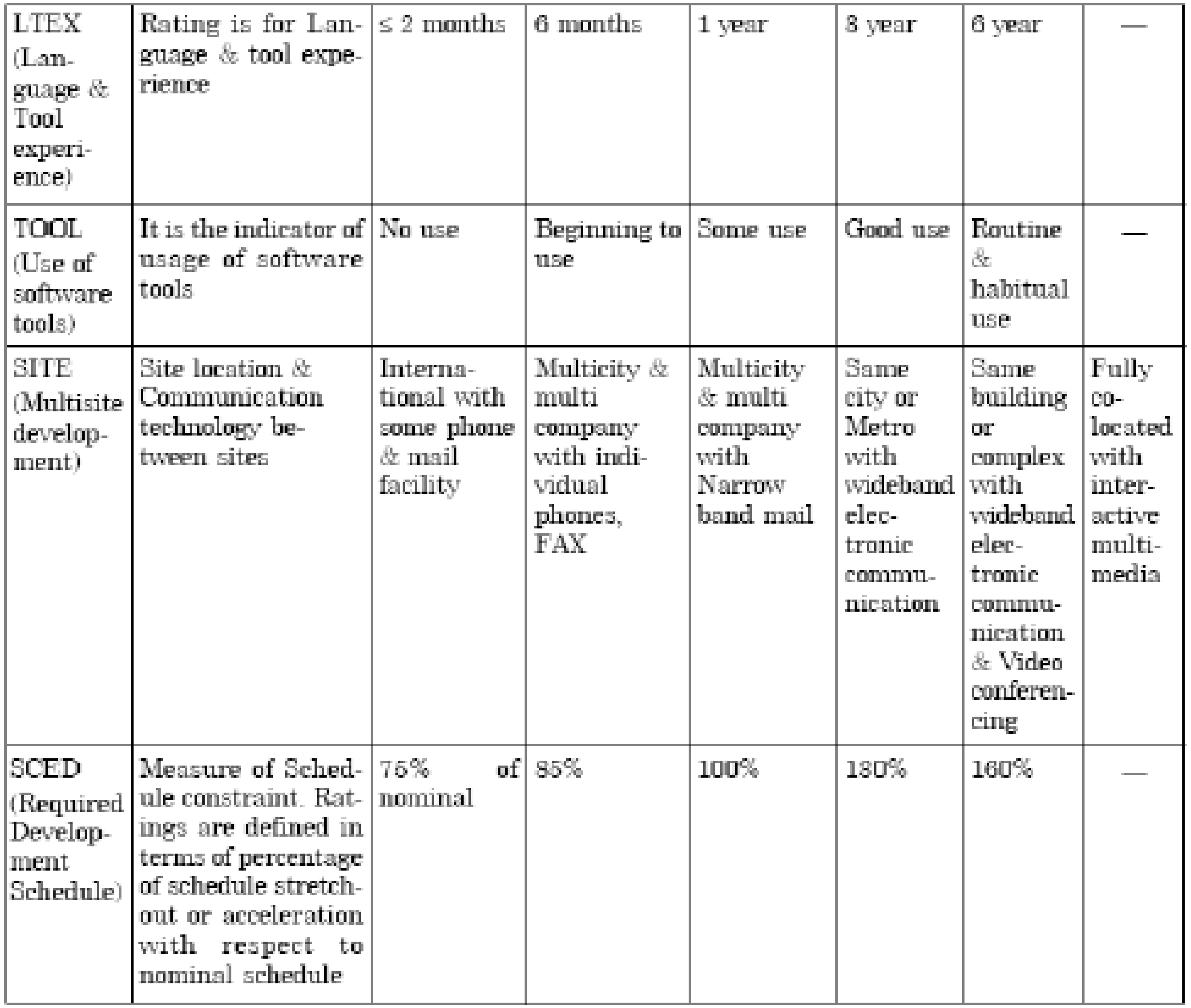


| | | | | | | | |
|---|---|---|---|---|---|---|---|
| PCAP (Programmers capability) | Capability of Programmers as a team. It includes ability, efficiency, thoroughness & communication skills | 15th Percentile | 35th Percentile | 55th Percentile | 75th Percentile | 90th Percentile | — |
| PCON (Personnel Continuity) | Rating is in terms of Project's annual personnel turnover | 48%/year | 24%/year | 12%/year | 6%/year | 3%/year | — |
| AEXP (Applications Experience) | Rating is dependent on level of applications experience. | ≤ 2 months | 6 months | 1 year | 3 year | 6 year | — |
| PEXP (Platform experience) | Measure of Platform experience | ≤ 2 months | 6 months | 1 year | 3 year | 6 year | — |

**Table 16:** Post Architecture Cost Driver rating level summary

Cont

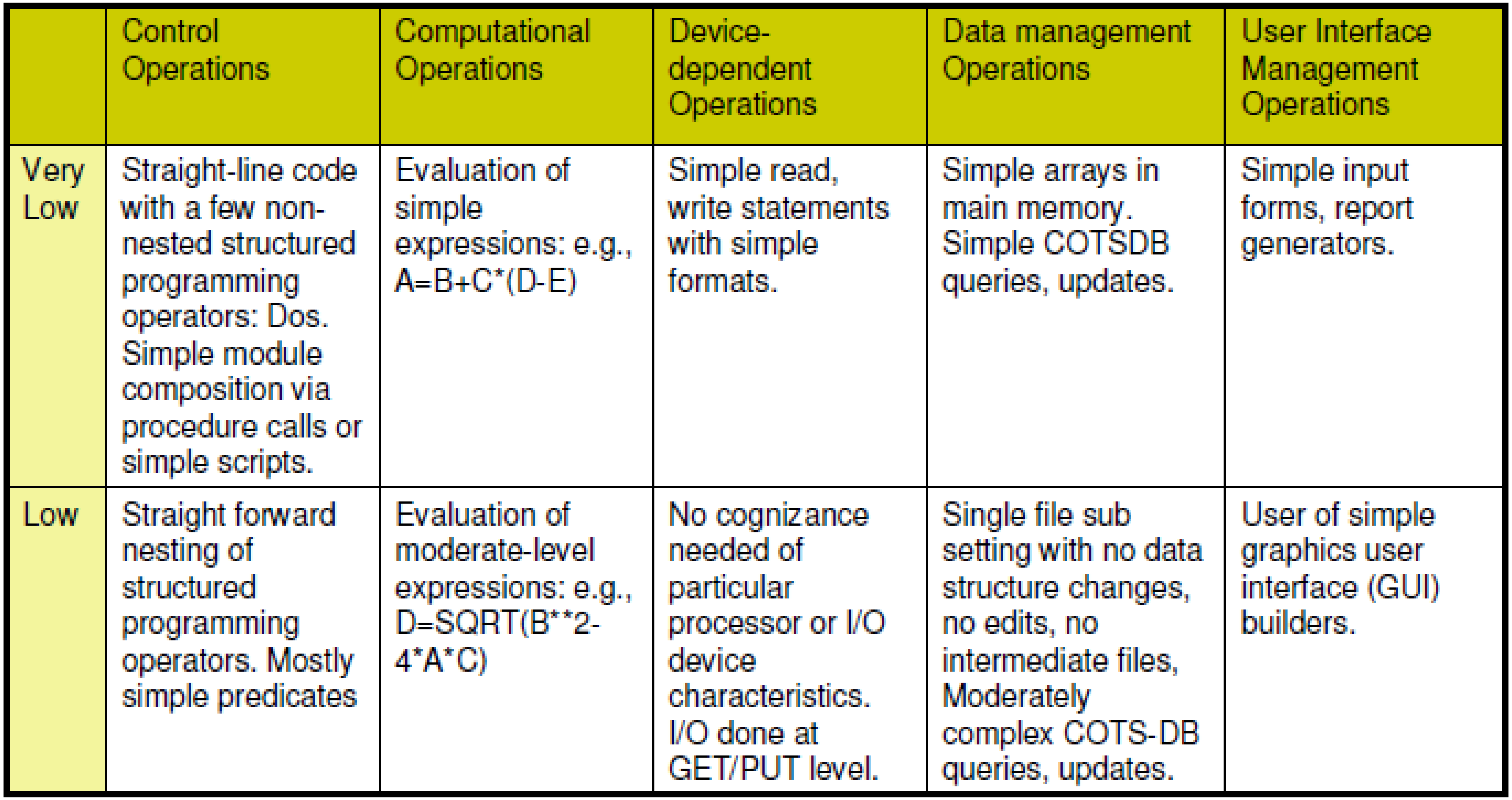
# COCOMO-II



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| LTEX (Language & Tool experience) | Rating is for Language & tool experience | ≤ 2 months | 6 months | 1 year | 3 year | 6 year | — |
| TOOL (Use of software tools) | It is the indicator of usage of software tools | No use | Beginning to use | Some use | Good use | Routine & habitual use | — |
| SITE (Multisite development) | Site location & Communication technology between sites | International with some phone & mail facility | Multicity & multi company with individual phones, FAX | Multicity & multi company with Narrow band mail | Same city or Metro with wideband electronic communication | Same building or complex with wideband electronic communication & Video conferencing | Fully co-located with interactive multimedia |
| SCED (Required Development Schedule) | Measure of Schedule constraint. Ratings are defined in terms of percentage of schedule stretch-out or acceleration with respect to nominal schedule | 75% of nominal | 85% | 100% | 130% | 160% | — |

Table 16: Post Architecture Cost Driver rating level summary

# COCOMO-II

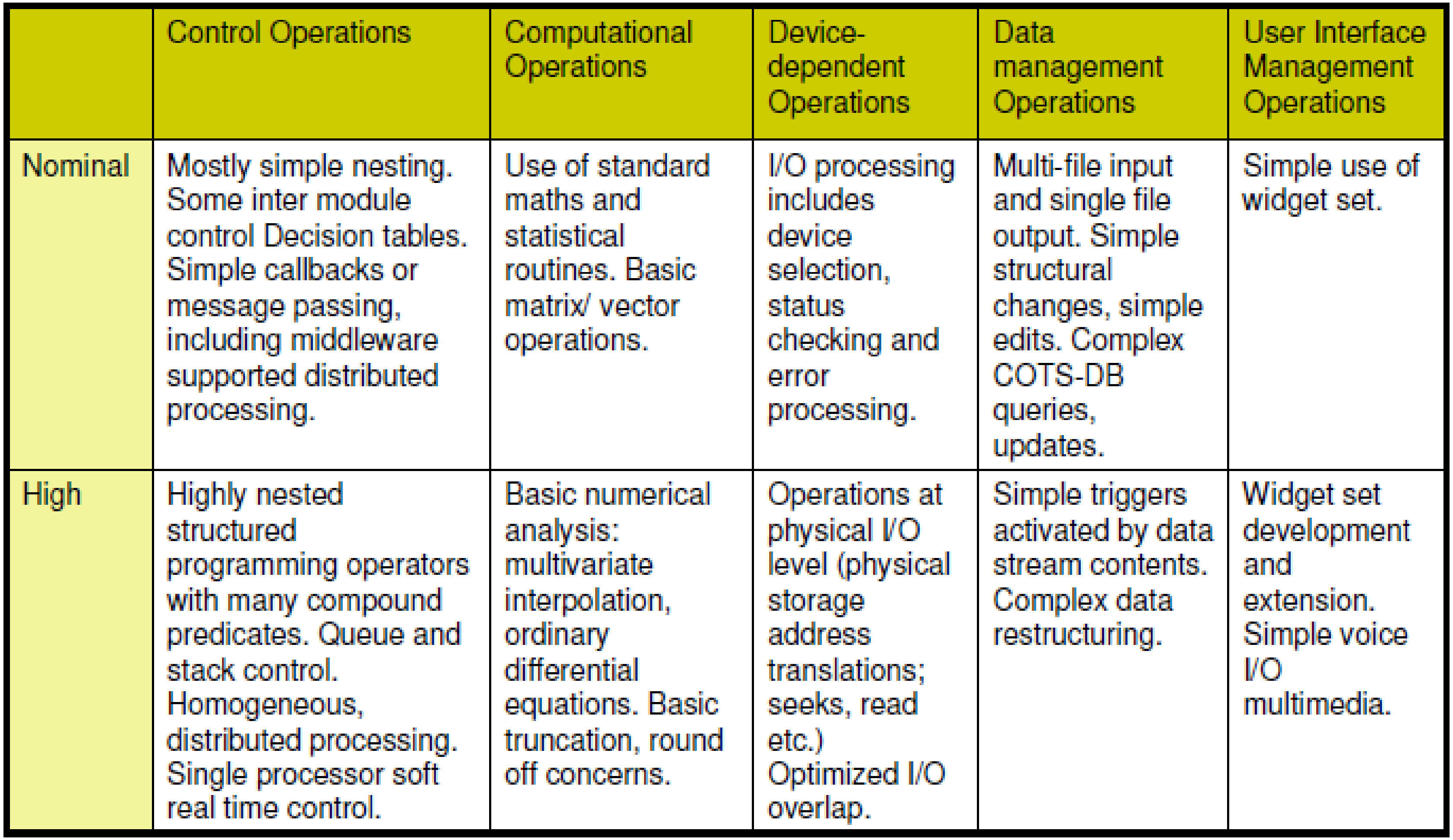


| | Control Operations | Computational Operations | Device-dependent Operations | Data management Operations | User Interface Management Operations |
|---|---|---|---|---|---|
| Very Low | Straight-line code with a few non-nested structured programming operators: Dos. Simple module composition via procedure calls or simple scripts. | Evaluation of simple expressions: e.g., A=B+C*(D-E) | Simple read, write statements with simple formats. | Simple arrays in main memory. Simple COTSDB queries, updates. | Simple input forms, report generators. |
| Low | Straight forward nesting of structured programming operators. Mostly simple predicates | Evaluation of moderate-level expressions: e.g., D=SQRT(B**2-4*A*C) | No cognizance needed of particular processor or I/O device characteristics. I/O done at GET/PUT level. | Single file sub setting with no data structure changes, no edits, no intermediate files, Moderately complex COTS-DB queries, updates. | User of simple graphics user interface (GUI) builders. |

**Table 17:** Module complexity ratings

**Cont…**

# COCOMO-II

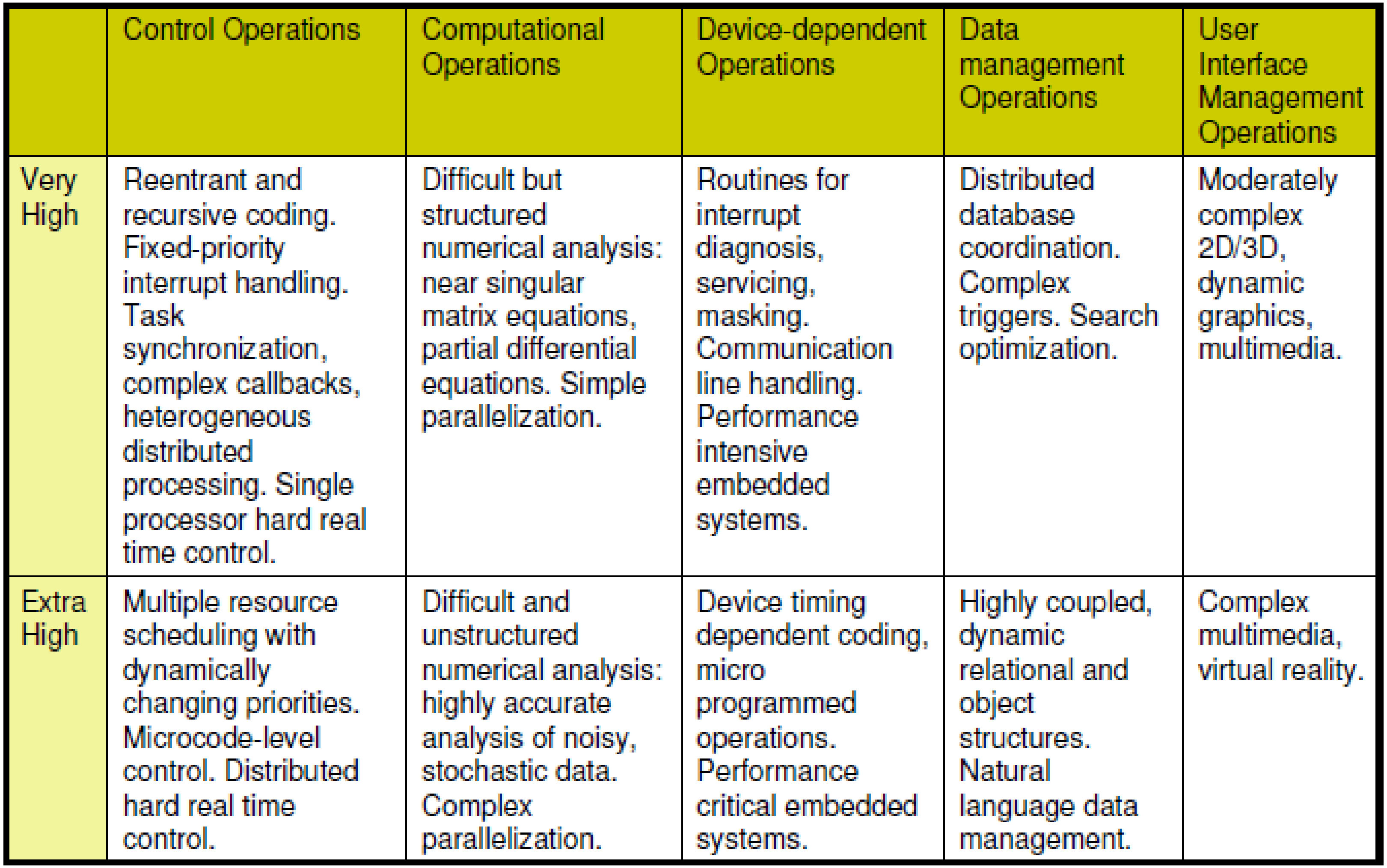


| | Control Operations | Computational Operations | Device-dependent Operations | Data management Operations | User Interface Management Operations |
|---|---|---|---|---|---|
| Nominal | Mostly simple nesting. Some inter module control Decision tables. Simple callbacks or message passing, including middleware supported distributed processing. | Use of standard maths and statistical routines. Basic matrix/ vector operations. | I/O processing includes device selection, status checking and error processing. | Multi-file input and single file output. Simple structural changes, simple edits. Complex COTS-DB queries, updates. | Simple use of widget set. |
| High | Highly nested structured programming operators with many compound predicates. Queue and stack control. Homogeneous, distributed processing. Single processor soft real time control. | Basic numerical analysis: multivariate interpolation, ordinary differential equations. Basic truncation, round off concerns. | Operations at physical I/O level (physical storage address translations; seeks, read etc.) Optimized I/O overlap. | Simple triggers activated by data stream contents. Complex data restructuring. | Widget set development and extension. Simple voice I/O multimedia. |

**Table 17:** Module complexity ratings

Cont…

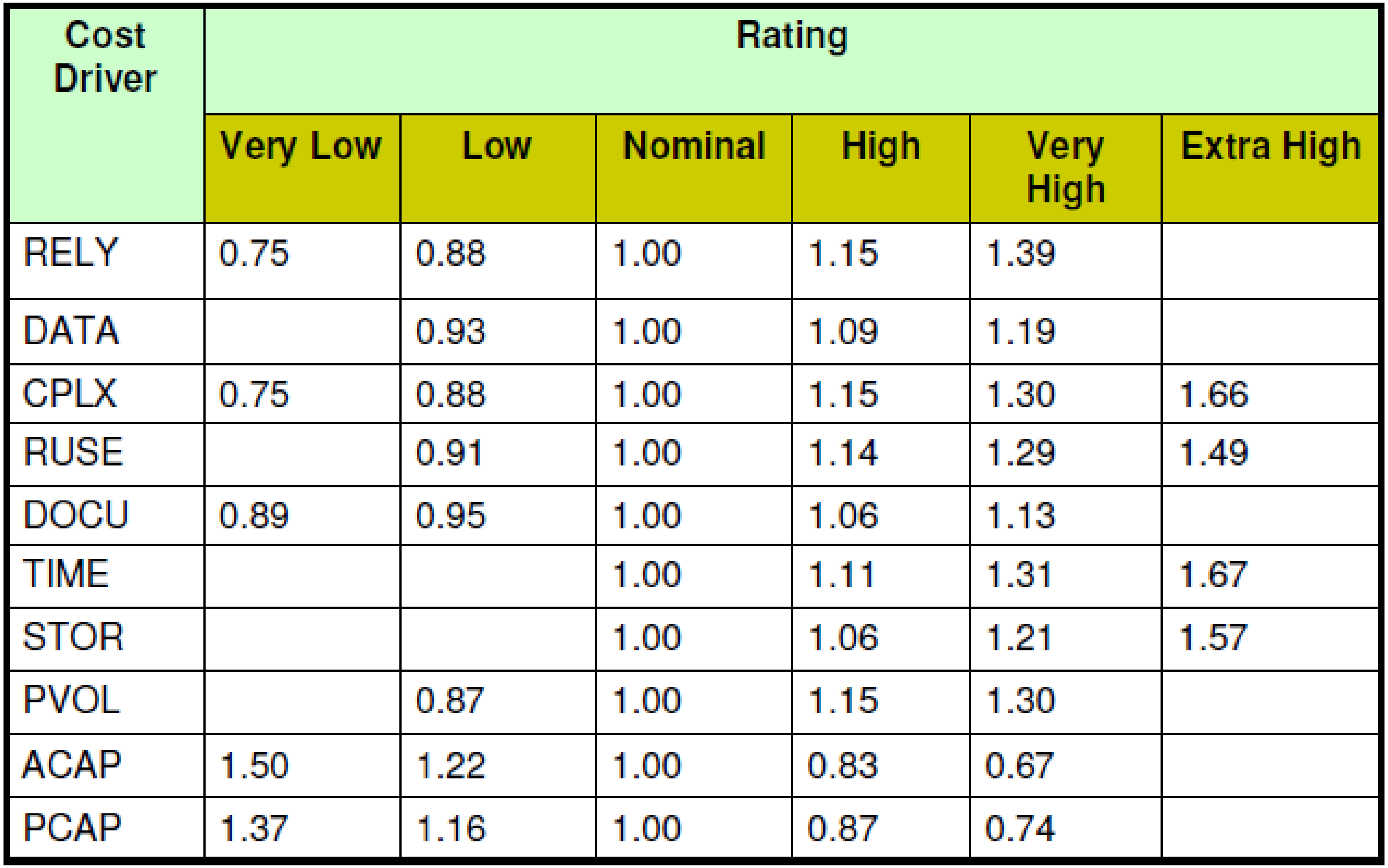
# COCOMO-II



| | Control Operations | Computational Operations | Device-dependent Operations | Data management Operations | User Interface Management Operations |
|---|---|---|---|---|---|
| Very High | Reentrant and recursive coding. Fixed-priority interrupt handling. Task synchronization, complex callbacks, heterogeneous distributed processing. Single processor hard real time control. | Difficult but structured numerical analysis: near singular matrix equations, partial differential equations. Simple parallelization. | Routines for interrupt diagnosis, servicing, masking. Communication line handling. Performance intensive embedded systems. | Distributed database coordination. Complex triggers. Search optimization. | Moderately complex 2D/3D, dynamic graphics, multimedia. |
| Extra High | Multiple resource scheduling with dynamically changing priorities. Microcode-level control. Distributed hard real time control. | Difficult and unstructured numerical analysis: highly accurate analysis of noisy, stochastic data. Complex parallelization. | Device timing dependent coding, micro programmed operations. Performance critical embedded systems. | Highly coupled, dynamic relational and object structures. Natural language data management. | Complex multimedia, virtual reality. |

**Table 17:** Module complexity ratings

# COCOMO-II

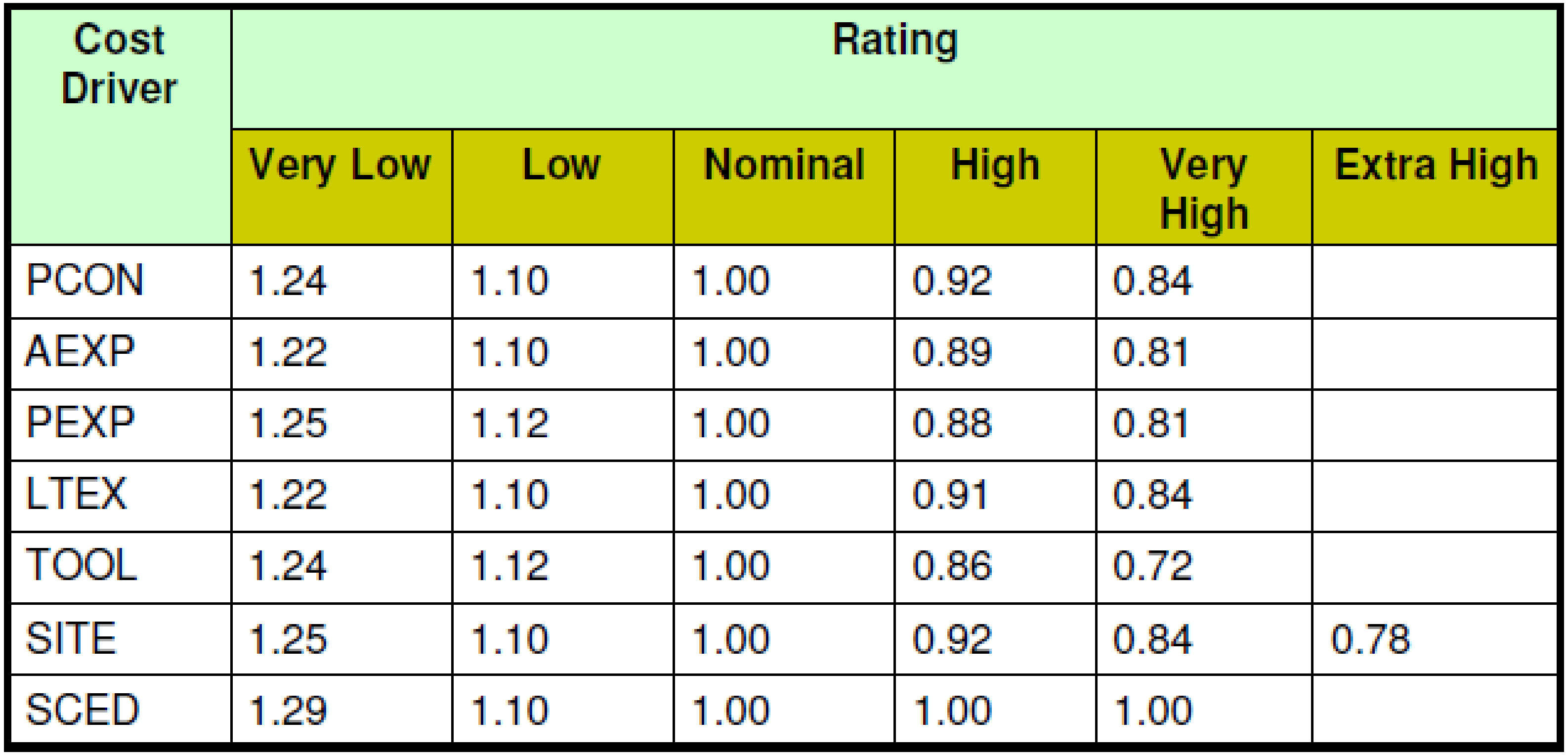


| Cost Driver | Rating | | | | | |
|---|---|---|---|---|---|---|
| | Very Low | Low | Nominal | High | Very High | Extra High |
| RELY | 0.75 | 0.88 | 1.00 | 1.15 | 1.39 | |
| DATA | | 0.93 | 1.00 | 1.09 | 1.19 | |
| CPLX | 0.75 | 0.88 | 1.00 | 1.15 | 1.30 | 1.66 |
| RUSE | | 0.91 | 1.00 | 1.14 | 1.29 | 1.49 |
| DOCU | 0.89 | 0.95 | 1.00 | 1.06 | 1.13 | |
| TIME | | | 1.00 | 1.11 | 1.31 | 1.67 |
| STOR | | | 1.00 | 1.06 | 1.21 | 1.57 |
| PVOL | | 0.87 | 1.00 | 1.15 | 1.30 | |
| ACAP | 1.50 | 1.22 | 1.00 | 0.83 | 0.67 | |
| PCAP | 1.37 | 1.16 | 1.00 | 0.87 | 0.74 | |

**Table 18:** 17 Cost Drivers

**Cont...**

# COCOMO-II



| Cost Driver | Rating | | | | | |
|---|---|---|---|---|---|---|
| | Very Low | Low | Nominal | High | Very High | Extra High |
| PCON | 1.24 | 1.10 | 1.00 | 0.92 | 0.84 | |
| AEXP | 1.22 | 1.10 | 1.00 | 0.89 | 0.81 | |
| PEXP | 1.25 | 1.12 | 1.00 | 0.88 | 0.81 | |
| LTEX | 1.22 | 1.10 | 1.00 | 0.91 | 0.84 | |
| TOOL | 1.24 | 1.12 | 1.00 | 0.86 | 0.72 | |
| SITE | 1.25 | 1.10 | 1.00 | 0.92 | 0.84 | 0.78 |
| SCED | 1.29 | 1.10 | 1.00 | 1.00 | 1.00 | |

**Table 18:** 17 Cost Drivers

# COCOMO-II

## Size measurement

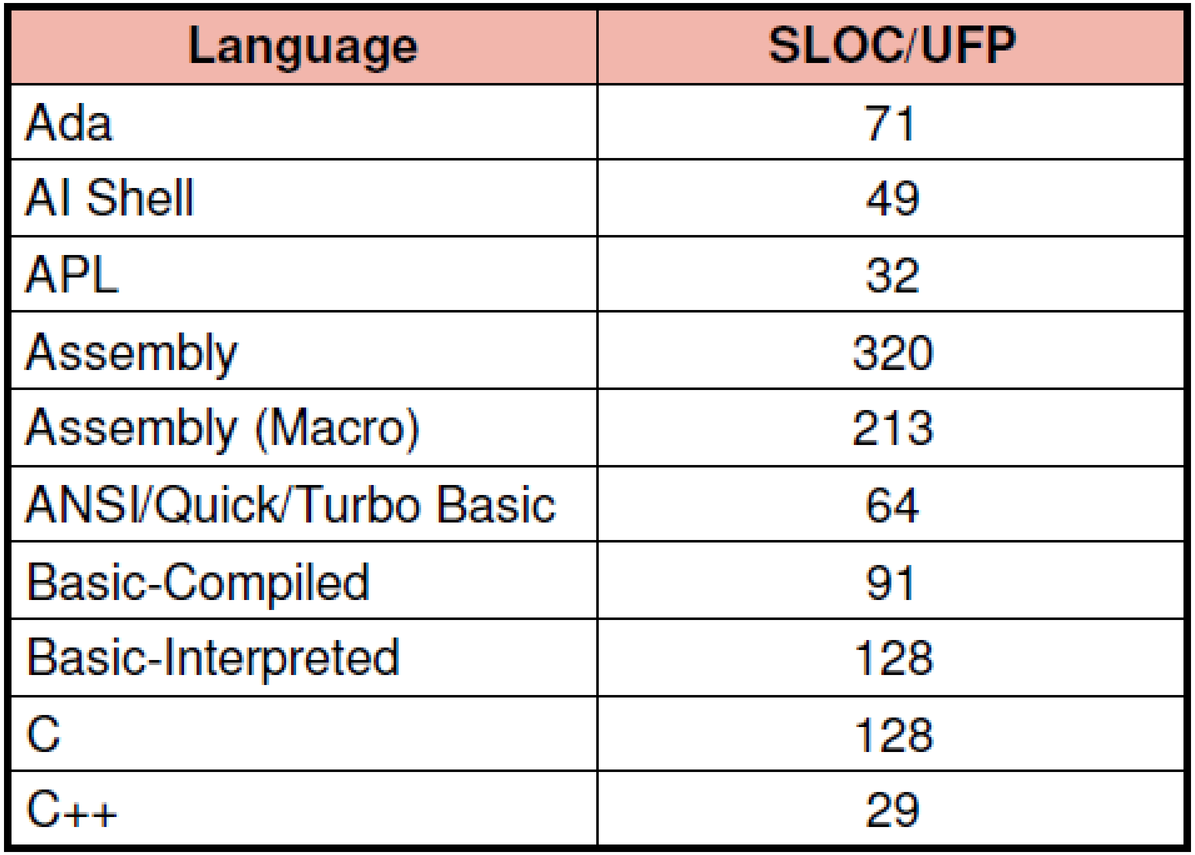
Size can be measured in any unit and the model can be calibrated accordingly. However, COCOMO II details are:

i. Application composition model uses the size in object points.

ii. The other two models use size in KLOC

Early design model uses unadjusted function points. These function points are converted into KLOC using Table 19. Post architecture model may compute KLOC after defining LOC counting rules. If function points are used, then use unadjusted function points and convert it into KLOC using Table 19.

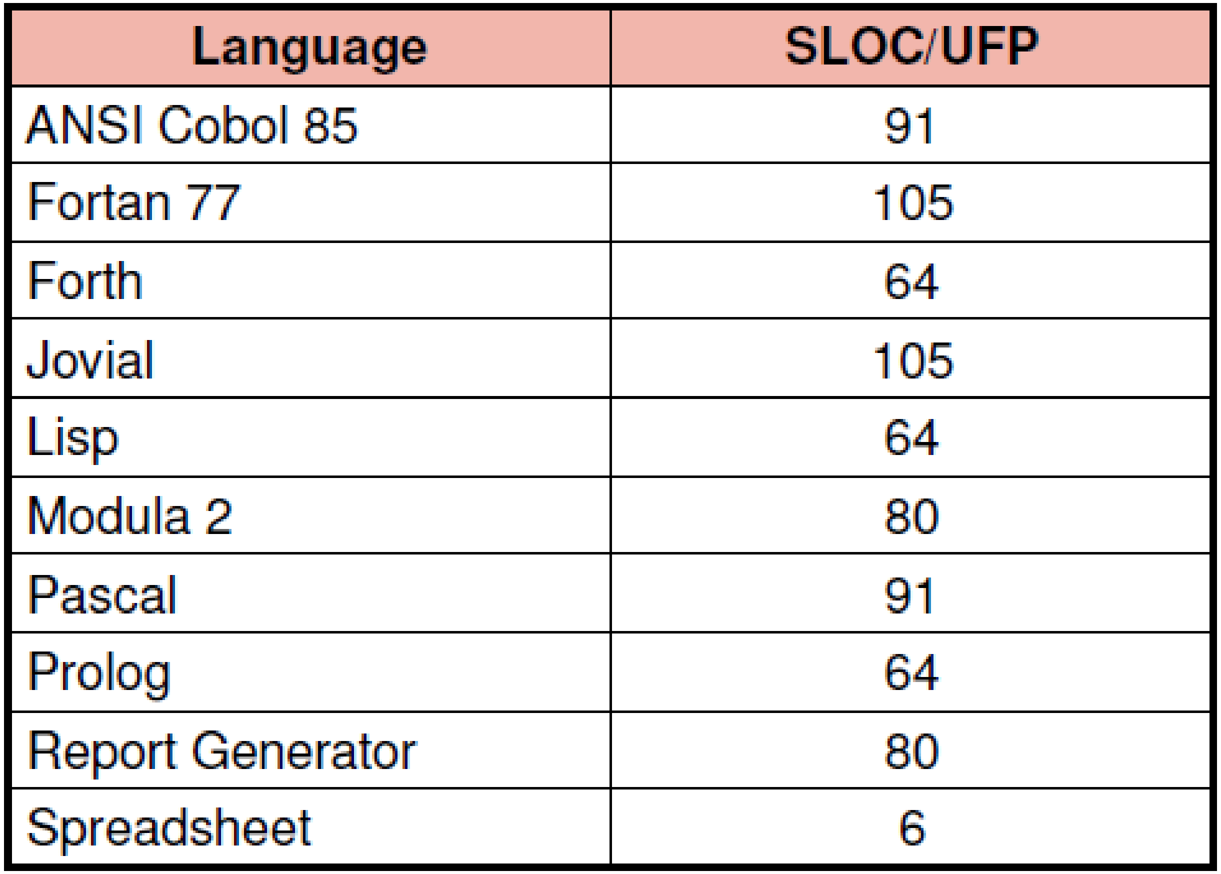
# COCOMO-II



| Language | SLOC/UFP |
|---|---|
| Ada | 71 |
| AI Shell | 49 |
| APL | 32 |
| Assembly | 320 |
| Assembly (Macro) | 213 |
| ANSI/Quick/Turbo Basic | 64 |
| Basic-Compiled | 91 |
| Basic-Interpreted | 128 |
| C | 128 |
| C++ | 29 |

**Table 19:** Converting function points to lines of code

# COCOMO-II



| Language | SLOC/UFP |
|---|---|
| ANSI Cobol 85 | 91 |
| Fortan 77 | 105 |
| Forth | 64 |
| Jovial | 105 |
| Lisp | 64 |
| Modula 2 | 80 |
| Pascal | 91 |
| Prolog | 64 |
| Report Generator | 80 |
| Spreadsheet | 6 |

**Table 19:** Converting function points to lines of code

# COCOMO-II

**Solution**

Here $B = 1.1129$

$PM_{\text{nominal}} = 194.41$ Person-months

$$PM_{adjusted} = PM_{\text{nominal}} \times \left[ \prod_{i=7}^{17} EM_i \right]$$

$= 194.41 \text{ x } (1.15 \text{ x } 1.19 \text{ x } 1.11 \text{ x } 0.67 \text{ x } 0.87)$

$= 194.41 \text{ x } 0.885$

$= 172.05$ Person-months

This image cannot currently be displayed.

# End of Chapter 5

Questions?