CSE412
Software Engineering

Nishat Tasnim Niloy

Lecturer

Department of Computer Science and Engineering

Faculty of Science and Engineering

Topic 9

Software Measurement and Estimation

(Project cost estimation techniques, Functional Point Analysis, FP counting method for determining software cost, COCOMO)

Project Cost Estimation

- Project cost estimation is the process of predicting the quantity, cost, and price of the resources required by the scope of a project.

- Since cost estimation is about the prediction of costs rather than counting the actual cost, a certain degree of uncertainty is involved.

- The software estimation process includes estimating the size of the software product to be produced, estimating the effort required, developing preliminary project schedules, and finally, estimating overall cost of the project

Direct Approach

LOC: Line of Code

- Source lines of code (SLOC) is a software metric used to measure the size of a software program by counting the number of lines in the text of the program's source code.

- There are two major types of SLOC measures: physical SLOC (LOC) and logical SLOC (LLOC).

- Physical SLOC is a count of lines in the text of the program's source code including comment lines. Blank lines are also included unless the lines of code in a section consists of more than 25% blank lines.

- Logical SLOC attempts to measure the number of executable "statements", but their specific definitions are tied to specific computer languages.

Direct Approach

COCOMO: Constructive Cost Model

- The COCOMO cost estimation model is used by thousands of software project managers and is based on a study of hundreds of software projects.

- The most fundamental calculation in the COCOMO model is the use of the Effort Equation to estimate the number of Person-Months required to develop a project

Indirect Approach

Function Point Analysis (FPA)

- FPA is used to make estimate of the software project, including its testing in terms of functionality or function size of the software product.

- The functional size of the product is measured in terms of the function point, which is a standard of measurement to measure the software application.

- The basic and primary purpose of the functional point analysis is to measure and provide the software application functional size to the client, customer, and the stakeholder on their request.

FPA Approach

Count the number of inputs, outputs, inquiries, master files, and interfaces required, then calculate the Unadjusted Function Points (UFP)
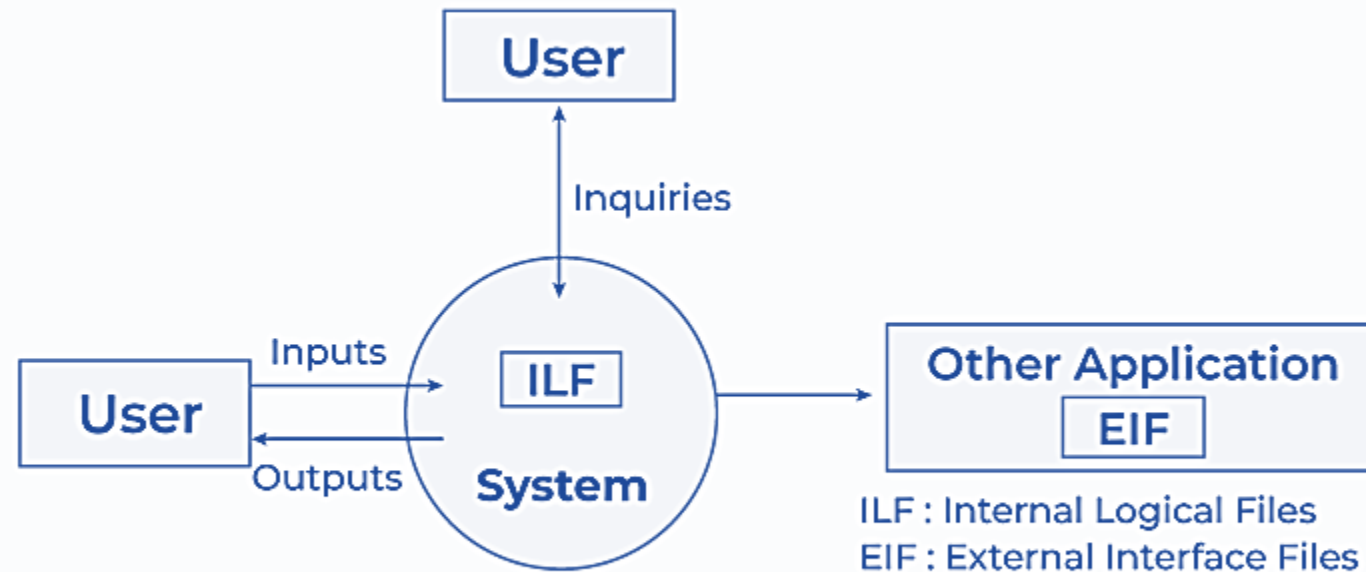
Calculate the adjusted function point (AFP) by multiplying these counts by an adjustment factor; the UFP and the product complexity adjustment.

Calculate the Source Lines of Code (SLOC) with the help of the AFP and the Language Factor (LF)

Function Point Analysis (FPA)

FPAs Functional Units System

Calculation of the unadjusted function points (UFP)

- The FPA measures functionality that the user requires. The specific user functionality is a measurement of the functionality delivered by the application as for user request.

- The 5 function types identified are:
  - external input which receives information from outside the application boundary,
  - external output which presents information of the information system,
  - external enquiries which is special kind of an external output. An external inquiry presents information of the information system based on a uniquely identifying search criterion, without applying additional processing (such as calculations).

Calculation of the unadjusted function points (UFP)

- **internal logical files** contains permanent data that is relevant to the user The information system references and maintains the data and
- **external interface files** also contains permanent data that is relevant to the user. The information system references the data, but the data is maintained by another information system

- For each function identified above the function is further classified as simple, average or complex and a weight is given to each.
- The sum of the weights quantifies the size of information processing and is referred to as the Unadjusted Function points.

Calculation of the unadjusted function points (UFP)

Function type	Simple	Average	Complex
External Input	3	4	6
External Inquiry	3	4	6
External Output	4	5	7
External Interface File	5	7	10
Internal Logical File	7	10	15

The table shows the function types and the weighting factors for the varying complexities.

Calculate Adjusted Function Point

- To calculate the Complexity adjustment value, several factors have to be considered, such as Backup and recovery, code design for reuse, etc.

- The adjusted function point denoted by FP is given by the formula:

  *FP = total UFP * (Complexity adjustment factor)*
  *= total UFP * (0.65 + (0.01 * Total complexity adjustment value))*

- Total complexity adjustment value is counted based on responses to questions called complexity weighting factors.

- Each complexity weighting factor is assigned a value (complexity adjustment value) that ranges between 0 (not important) to 5 (absolutely essential).

Calculate the Source Lines of Code (SLOC)

- Language Factor $= LF$
  - Varies from language to language (C, C++, java, python, Ruby etc.)
- Source Lines of Code (SLOC) $= FP * LF$

Example

		Weighting Factor			Count
		Simple	Average	Complex	
Input	Member Login	3			23
	Member Registration		4		
	Select research question for regression analysis		4		
	Select research question for correlation analysis		4		
	Select research question for hypothesis test analysis		4		
	Select research question for Chi Square test analysis		4		

Example

		Weighting Factor			Count
		Simple	Average	Complex	
Output	Member login confirmation	4			25
	Member Registration confirmation	4			
	Graph/Table of regression analysis		5		
	Graph/Table of correlation analysis	4			
	Graph/Table of hypothesis test analysis	4			
	Graph/Table of Chi square test analysis	4			

Example

		Weighting Factor			Count
		Simple	Average	Complex	
Inquiries	Validate member information		4		8
	View alumni list		4		
Files	Linear regression		10		40
	correlation		10		
	Hypothesis test		10		
	Chi square test		10		
Interfaces	Application server to database			10	20
	User to application server			10	
Total					116

Example

Number	Complexity Weighting Factor	Value
1	Backup and recovery	1
2	Data communications	2
3	Distributed processing	2
4	Performance critical	5
5	Existing operating environment	3
6	On-line data entry	3
7	Input transaction over multiple screens	1
8	Master files updated online	3
9	Information domain values complex	5
10	Internal processing complex	4
11	Code designed for reuse	5
12	Conversion/installation in design	4
13	Multiple installations	4
14	Application designed for change	4
	Total complexity adjustment value	46

Example

- Total Unadjusted Function Points (UFP) = 116
- Product Complexity Adjustment (PC) = 0.65 + (0.01 *46) = 1.11
- Total Adjusted Function Points (FP) = UFP * PC = 128.8 ≈ 129
- Language Factor (LF) for Java assumed as = 38
- Source Lines of Code (SLOC) = FP * LF = 4902

Exercise

Suppose you are a project manager of an e-commerce website. The analyst found that among the activities of external input and output function types, three are simple and three are average activities. There are in total two external inquiries all of which are simple, five internal logical file transactions all of which are average, and three external interface files- one is simple, and others are complex. As for the fourteen-complexity weighting factor, three are not essential, seven are moderately essential and the rest are very essential. If you want to develop this code in .Net language, use functional point analysis to **estimate** the **lines of code** for this project. You may need the provided information on the provided tables for your calculations.

Function type	Simple	Average	Complex
External Input	3	4	6
External Inquiry	3	4	6
External Output	4	5	7
External Interface File	5	7	10
Internal Logical File	7	10	15

complexity weighting factor
Very essential -5
Essential - 4
Moderately essential - 3
Slightly essential - 2
Not essential - 1

** avg language factor for .Net = 57

The Constructive Cost Model (COCOMO)

Constructive Cost model

(COCOMO)

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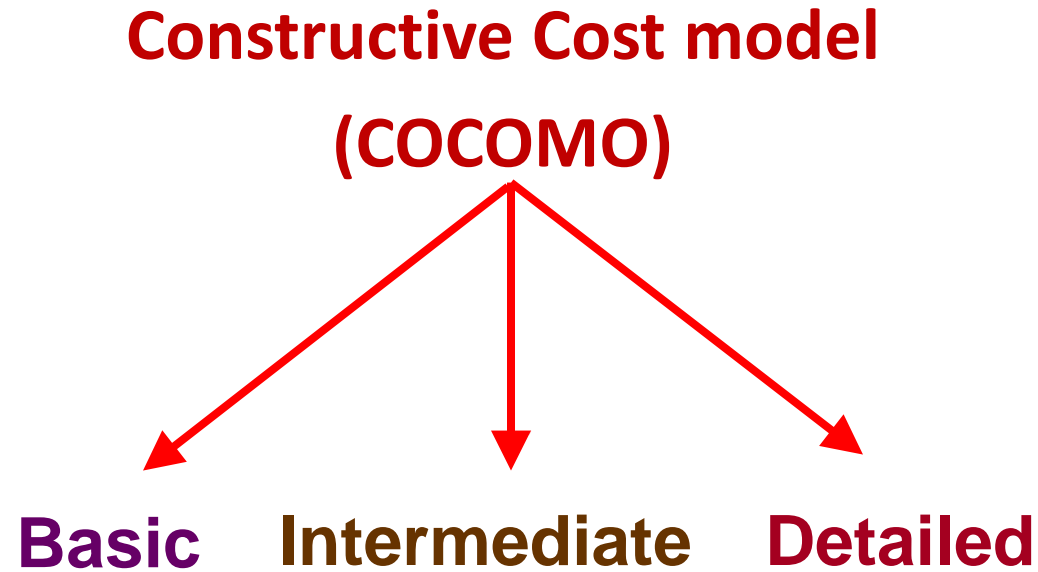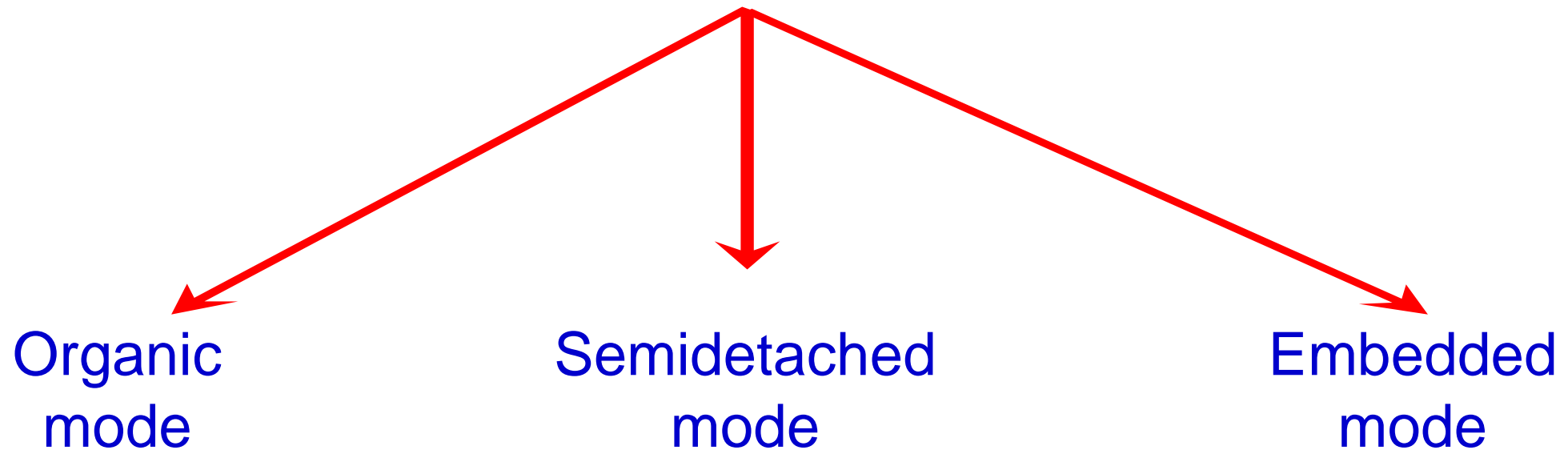
Model proposed by
B. W. Boehm's
through his book
Software Engineering Economics in 1981

The Constructive Cost Model (COCOMO)

COCOMO applied to

Organic
mode

Semidetached
mode

Embedded
mode

Comparison of three COCOMO modes

Mode	Project size	Nature of Project	Innovation	Deadline of the project	Development Environment
Organic	Typically 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 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	Moderate
Embedded	Typically, over 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

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 are a_b, b_b, c_b and d_b.

Basic COCOMO coefficients

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

Staff Size and Productivity

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

Average staff size $(SS) = \dfrac{E}{D}$ Persons

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

Productivity $(P) = \dfrac{KLOC}{E}$ KLOC per PM

Example 1

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:

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

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

Estimated size of the project = 400 KLOC

(i) Organic mode

E = 2.4(400)$^{1.05}$ = 1295.31 person-month

D = 2.5(1295.31)$^{0.38}$ = 38.07 month

Solution

(ii) Semidetached mode

$$E = 3.0(400)^{1.12} = 2462.79 \text{ person-month}$$

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

(iii) Embedded mode

$$E = 3.6(400)^{1.20} = 4772.81 \text{ person-month}$$

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

Example 2

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**.

Solution

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

Hence $E = 3.0(200)^{1.12} = 1133.12$ person-month

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

Average staff size $(SS) = \dfrac{E}{D} \; person$

$$= \dfrac{1133.12}{29.3} = 38.67 \; person \approx 39 \; person$$

Solution

$$\text{Productivity} = \frac{KLOC}{E} = \frac{200}{1133.12}$$

$$= 0.1765 \; KLOC \, per \, person\text{-}month$$

$$= (0.1765 * 1000) \; LOC \, per \, person\text{-}month$$

$$= 176 \; LOC \, per \, person\text{-}month$$

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

Intermediate Model

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

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	--

Multipliers of different cost drivers

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	--

Intermediate 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

Detailed COCOMO Model

Detailed COCOMO

Phase-Sensitive
effort multipliers

Three level
product hierarchy

Cost
drivers

design
& test

Modules
subsystem

Manpower allocation for
each phase

System level
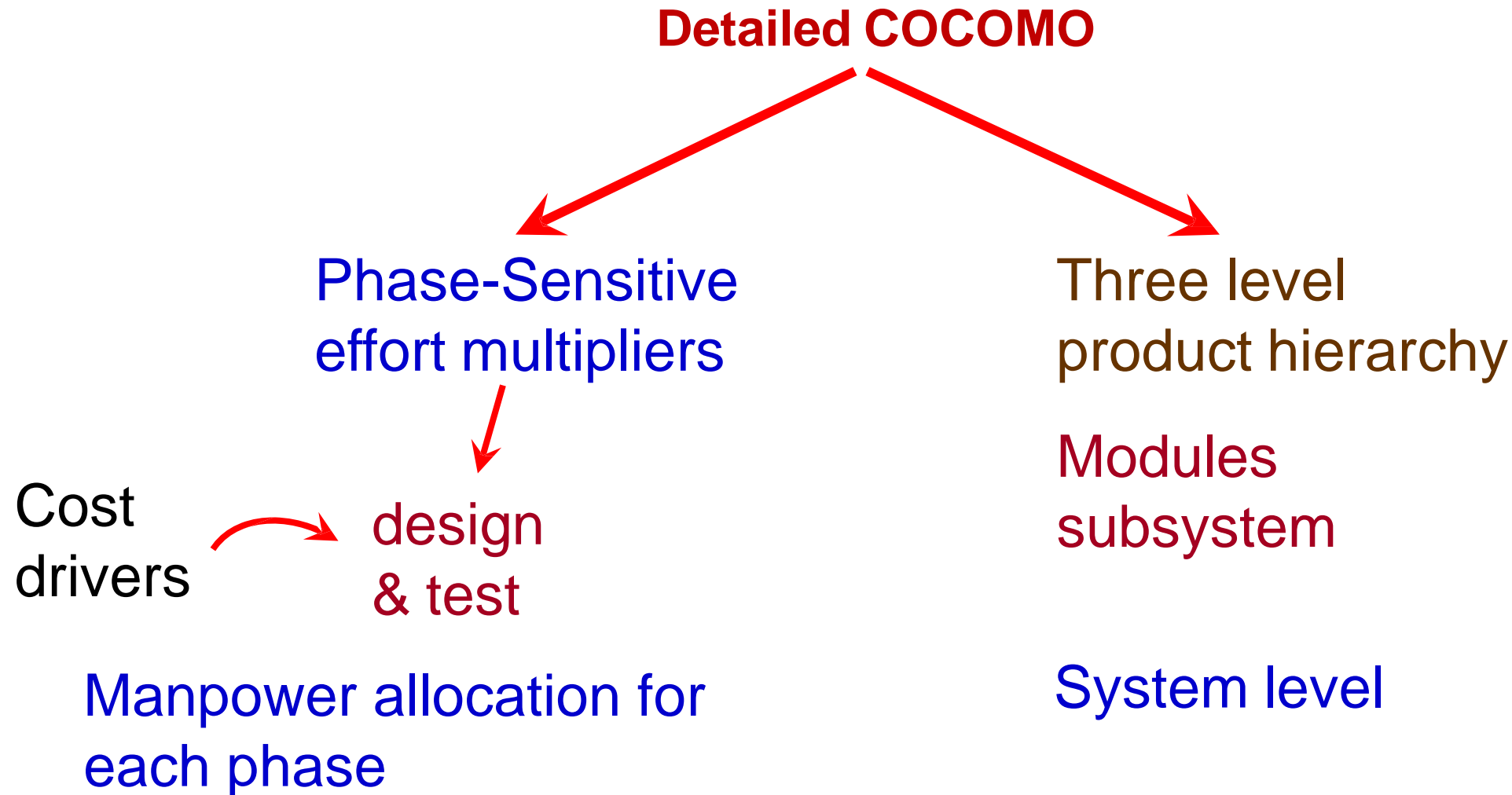
Detailed COCOMO: Development Phase

Plan / Requirements

EFFORT   :   6% to 8%

DEVELOPMENT TIME   :   10% to 40%

% depend on mode & size

Detailed COCOMO: Development Phase

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%

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

$$S\ (equivalent) = (S \times A) / 100$$

$$E_p = \mu_p E\ D_p$$

$$= \tau_p D$$

Lifecycle Phase Values of μ_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

Effort and schedule fractions occurring in each phase of the lifecycle

Lifecycle Phase Values of τ_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

Effort and schedule fractions occurring in each phase of the life cycle

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

Example

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 ?

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 x 1.14 = 0.9348

E   = 3712 x .9348 = 3470 PM

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

Solution

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 $= 3712 \times 1.22 = 4528$ PM

D $= 2.5 (4528)^{0.32} = 36.9$ 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 litter experience.