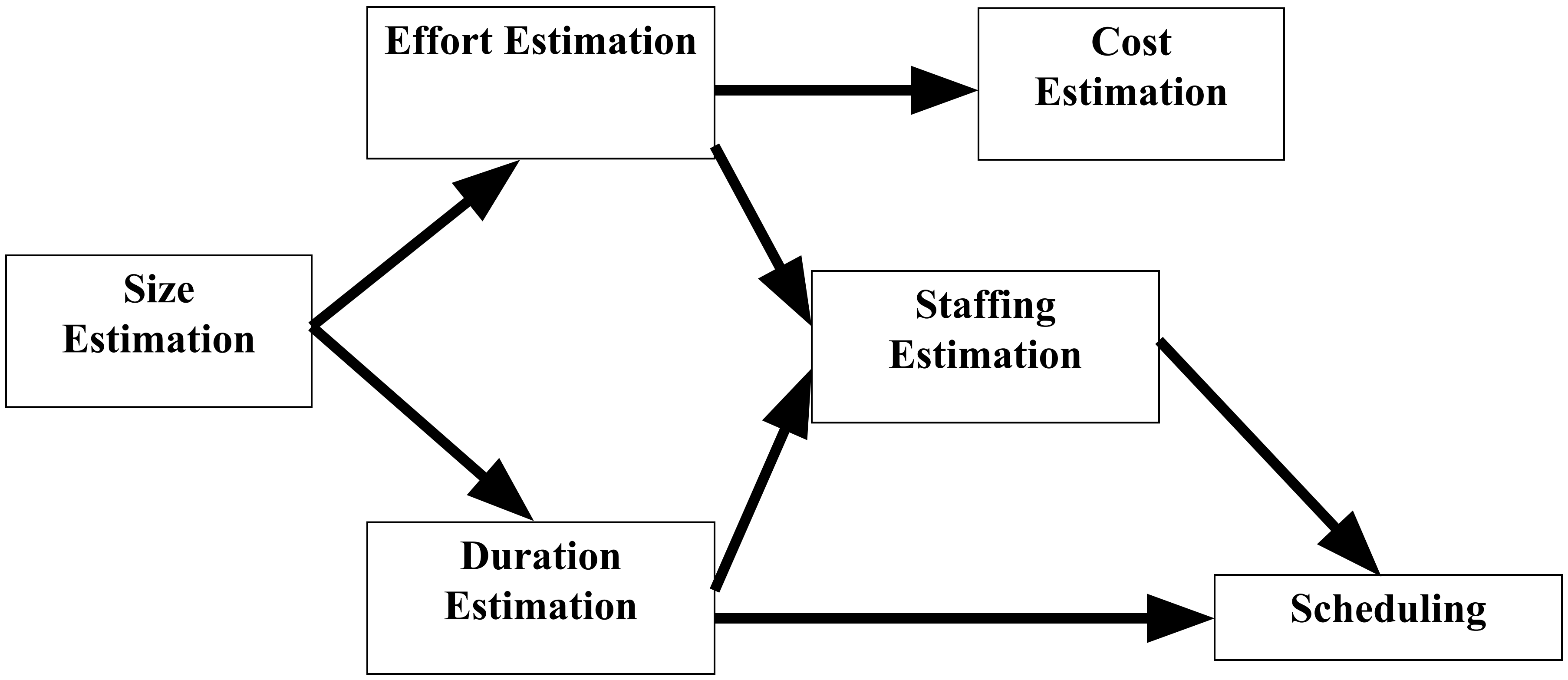
Estimation Techniques

Software Cost Estimation

- Determine size of the product.
- From the size estimate,
  - determine the effort needed.
- From the effort estimate,
  - determine project duration, and cost.

Software Cost Estimation



Software Size Metrics

- LOC (Lines of Code):
  - Simplest and most widely used metric.
  - Comments and blank lines should not be counted.

Disadvantages of Using LOC

- Size can vary with coding style.
- Focuses on coding activity alone.
- Correlates poorly with quality and efficiency of code.
- Penalizes higher level programming languages, code reuse, etc.

Disadvantages of Using LOC (cont...)

- Measures lexical/textual complexity only.
  - does not address the issues of structural or logical complexity.
- Difficult to estimate LOC from problem description.
  - So not useful for project planning

Function Point Metric

- Overcomes some of the shortcomings of the LOC metric
- This metric is based on the idea that a software product supporting many features would certainly be of larger size than a product with less number of features.

Function Point Metric computation

- Proposed by Albrecht in early 80's:

  Step 1: UFP computation

  UFP= (No. of inputs)*4 + (No. of outputs)* 5+(No. of inquiries)*4 +(No. of files)*10 + (No. of interfaces)*1 0

- Input:
  - A set of related inputs is counted as one input.
- Output:
  - A set of related outputs is counted as one output.

Function Point Metric

- Inquiries:
  - Each user query type is counted.
- Files:
  - Files are logically related data and thus can be data structures or physical files.
- Interface:
  - Data transfer to other systems.

Software Project Planning

Function Count

Alan Albrecht while working for IBM, recognized the problem in size measurement in the 1970s, and developed a technique (which he called Function Point Analysis), which appeared to be a solution to the size measurement problem.

Software Project Planning

The principle of Albrecht’s function point analysis (FPA) is that a system is decomposed into functional units.

▪ Inputs	:	information entering the system
▪ Outputs	:	information leaving the system
▪ Enquiries	:	requests for instant access to information
▪ Internal logical files	:	information held within the system
▪ External interface files	:	information held by other system that is used by the system being analyzed.

Software Project Planning

The FPA functional units are shown in figure given below:

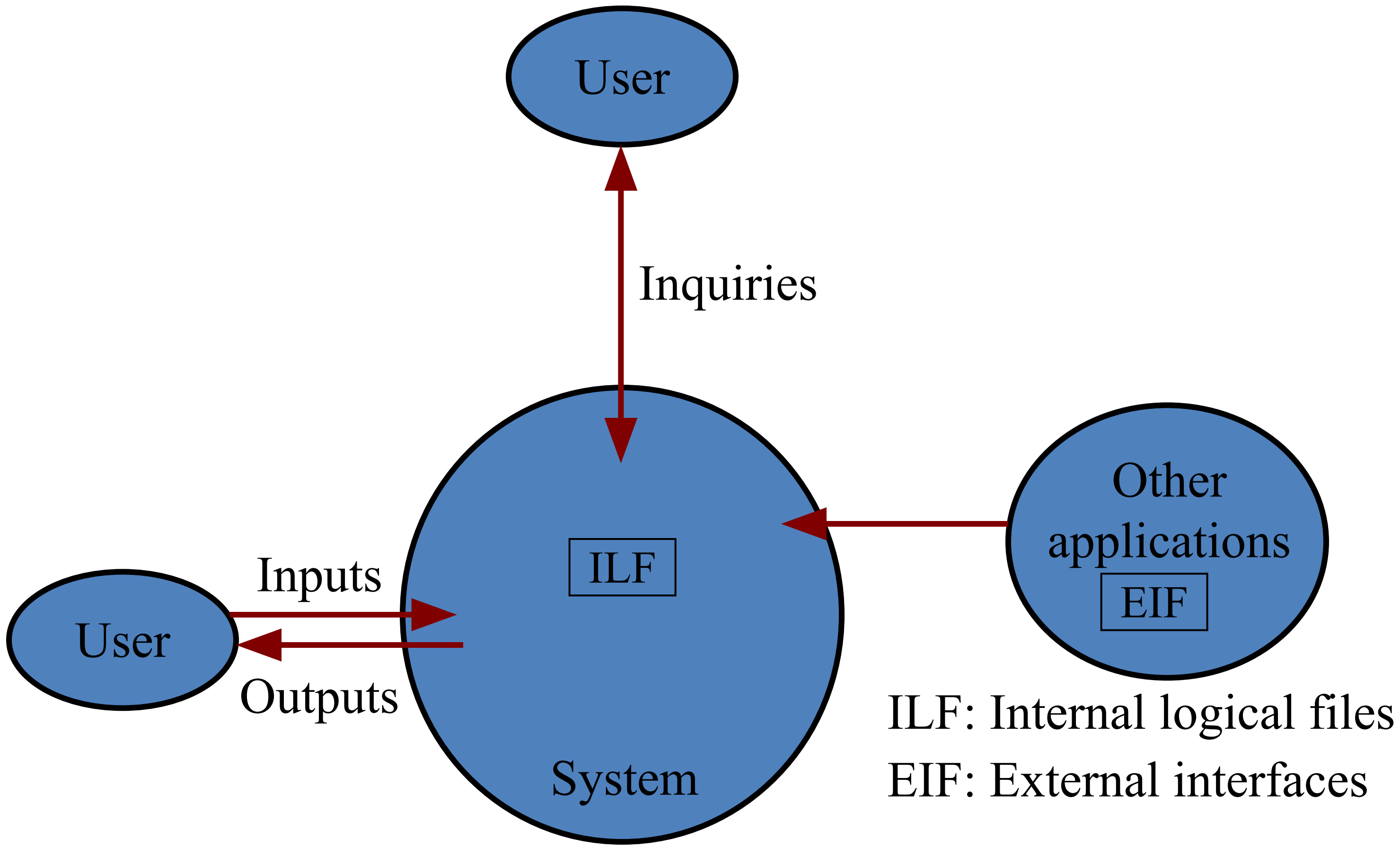


Fig. 3: FPAs functional units System

Software Project Planning

The five functional units are divided in two categories:

(i) Data function types

- Internal Logical Files (ILF): A user identifiable group of logical related data or control information maintained within the system.
- External Interface files (EIF): A user identifiable group of logically related data or control information referenced by the system, but maintained within another system. This means that EIF counted for one system, may be an ILF in another system.

Software Project Planning

(ii) Transactional function types

- External Input (EI): An EI processes data or control information that comes from outside the system. The EI is an elementary process, which is the smallest unit of activity that is meaningful to the end user in the business.
- External Output (EO): An EO is an elementary process that generate data or control information to be sent outside the system.
- External Inquiry (EQ): An EQ is an elementary process that is made up to an input-output combination that results in data retrieval.

Software Project Planning

Special features

- Function point approach is independent of the language, tools, or methodologies used for implementation; i.e. they do not take into consideration programming languages, data base management systems, processing hardware or any other data base technology.
- Function points can be estimated from requirement specification or design specification, thus making it possible to estimate development efforts in early phases of development.

Software Project Planning

- Function points are directly linked to the statement of requirements; any change of requirements can easily be followed by a re-estimate.
- Function points are based on the system user's external view of the system, non-technical users of the software system have a better understanding of what function points are measuring.

Software Project Planning

Counting function points

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

Software Project Planning

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 Project Planning

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.

Software Project Planning

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

Zij : 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*.

Software Project Planning

Organizations that use function point methods develop a criterion for determining whether a particular entry is Low, Average or High. Nonetheless, the determination of complexity is somewhat subjective.

$$FP = UFP * CAF$$

Where CAF is complexity adjustment factor and is equal to [0.65 + 0.01 x ΣF_i]. The F_i (i=1 to 14) are the degree of influence and are based on responses to questions noted in table 3.

Software Project Planning

Table 3 : Computing function points.

Rate each factor on a scale of 0 to 5.

0	*1*	*2*	*3*	*4*	*5*
No Influence	*Incidental*	*Moderate*	*Average*	*Significant*	*Essential*

Number of factors considered (F_i)

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 ?
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 ?

Software Project Planning

Functions points may compute the following important metrics:

Productivity	=	FP / persons-months
Quality	=	Defects / FP
Cost	=	Rupees / FP
Documentation	=	Pages of documentation per FP

These metrics are controversial and are not universally acceptable. There are standards issued by the International Functions Point User Group (IFPUG, covering the Albrecht method) and the United Kingdom Function Point User Group (UFPGU, covering the MK11 method). An ISO standard for function point method is also being developed.

Software Project Planning

Example: 4.1

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.

Software Project Planning

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

FP = UFP x CAF
= 628 x 1.07 = 672

Software Project Planning

Example:4.2

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 ?

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.

Software Project Planning

Example: 4.3

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:

(d) 6 with low complexity

(e) 13 with high complexity

(*iii*) External Inquiries:

(a) 3 with low complexity

(b) 4 with average complexity

(c) 2 high complexity

Software Project Planning

(*iv*) Internal logical files:

  (a) 2 with average complexity

  (b) 1 with high complexity

(v) External Interface files:

  (c) 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.

Software Project Planning

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 Project Planning

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

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

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

Software Cost Estimation

- Three main approaches to estimation:
  - Empirical
  - Heuristic
  - Analytical

Software Cost Estimation Techniques

- Empirical techniques:
  - an educated guess based on past experience.
- Heuristic techniques:
  - assume that the characteristics to be estimated can be expressed in terms of some mathematical expression.
- Analytical techniques:
  - derive the required results starting from certain simple assumptions.

Empirical Size Estimation Techniques

- Expert Judgement:
  - A guess made by an expert.
  - Suffers from individual bias.
- Delphi Estimation:
  - overcomes some of the problems of expert judgement.

Expert judgement

- Experts divide a software product into component units:
  - e.g. GUI, database module, data communication module, billing module, etc.
- Add up the guesses for each of the components.

Delphi Estimation:

- Team of Experts and a coordinator.
- Experts carry out estimation independently:
  - mention the rationale behind their estimation.
  - coordinator notes down any extraordinary rationale:
    - circulates among experts.

Delphi Estimation:

- Experts re-estimate.
- Experts never meet each other
  - to discuss their viewpoints.

Heuristic Estimation Techniques

- Single Variable Model:
  - Parameter to be Estimated=C1(Estimated Characteristic)^{d1}
- Multivariable Model:
  - Assumes that the parameter to be estimated depends on more than one characteristic.
  - Parameter to be Estimated=C1(Estimated Characteristic)^{d1}+ C2(Estimated Characteristic)^{d2}+…
  - Usually more accurate than single variable models.

COCOMO Model

- COCOMO (COnstructive COst MOdel) proposed by Boehm.
- Divides software product developments into 3 categories:
  - Organic
  - Semidetached
  - Embedded

COCOMO Product classes

- Roughly correspond to:
  - application, utility and system programs respectively.
    - Data processing and scientific programs are considered to be application programs.
    - Compilers, linkers, editors, etc., are utility programs.
    - Operating systems and real-time system programs, etc. are system programs.

Elaboration of Product classes

- Organic:
  - Relatively small groups
    - working to develop well-understood applications.
- Semidetached:
  - Project team consists of a mixture of experienced and inexperienced staff.
- Embedded:
  - The software is strongly coupled to complex hardware, or real-time systems.

COCOMO Model (CONT.)

- For each of the three product categories:
  - From size estimation (in KLOC), Boehm provides equations to predict:
    - project duration in months
    - effort in programmer-months
- Boehm obtained these equations:
  - examined historical data collected from a large number of actual projects.

COCOMO Model (CONT.)

- Software cost estimation is done through three stages:
  - Basic COCOMO,
  - Intermediate COCOMO,
  - Complete COCOMO.

Basic COCOMO Model (CONT.)

- Gives only an approximate estimation:
  - Effort = a1 $(KLOC)^{a2}$ PM
  - Tdev = b1 $(Effort)^{b2}$ Months
    - KLOC is the estimated kilo lines of source code,
    - a1,a2,b1,b2 are constants for different categories of software products,
    - Tdev is the estimated time to develop the software in months,
    - Effort estimation is obtained in terms of person months (PMs).

Development Effort Estimation

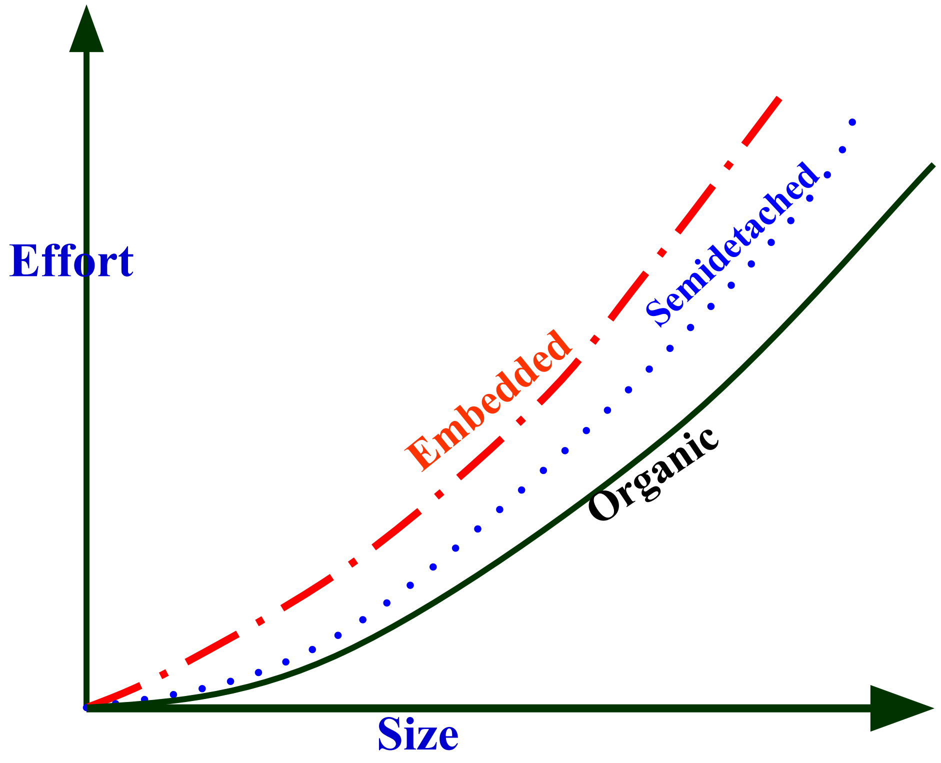
- Organic :
  - Effort = $2.4\ (\mathrm{KLOC})^{1.05}$ PM
- Semi-detached:
  - Effort = $3.0(\mathrm{KLOC})^{1.12}$ PM
- Embedded:
  - Effort = $3.6\ (\mathrm{KLOC})^{1.20}$PM

Development Time Estimation

- Organic:
  - $Tdev = 2.5\ (Effort)^{0.38}$ Months
- Semi-detached:
  - $Tdev = 2.5\ (Effort)^{0.35}$ Months
- Embedded:
  - $Tdev = 2.5\ (Effort)^{0.32}$ Months

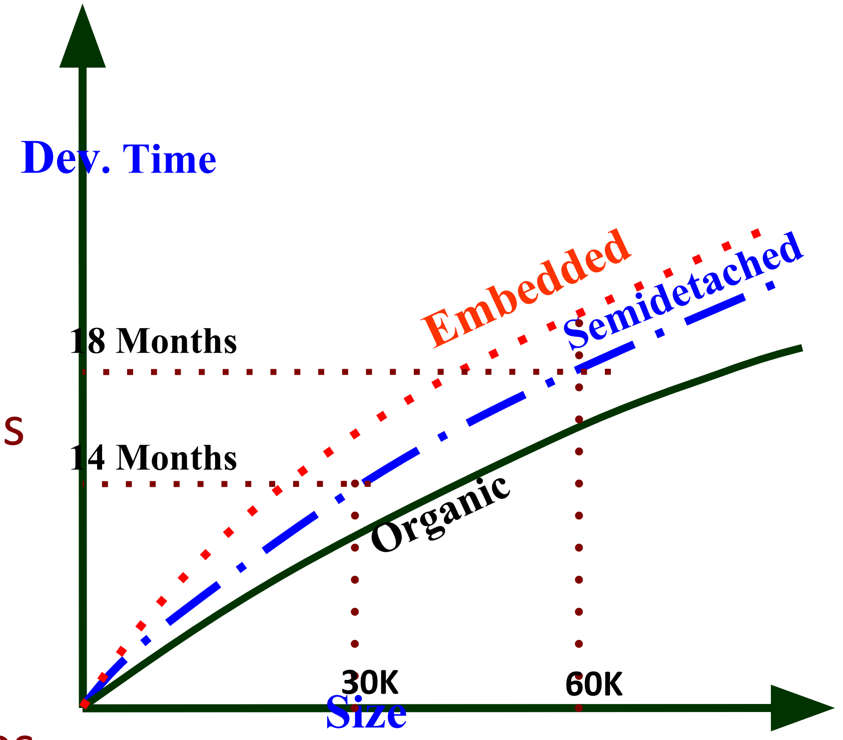
Basic COCOMO Model (CONT.)

- Effort is somewhat super-linear in problem size.



Basic COCOMO Model (CONT.)

- Development time
  - sublinear function of product size.
- When product size increases two times,
  - development time does not double.
- Time taken:
  - almost same for all the three product categories.



Basic COCOMO Model (CONT.)

- Development time does not increase linearly with product size:
  - For larger products more parallel activities can be identified:
    - can be carried out simultaneously by a number of engineers.

Basic COCOMO Model (CONT.)

- Development time is roughly the same for all the three categories of products:
  - For example, a 60 KLOC program can be developed in approximately 18 months
    - regardless of whether it is of organic, semi-detached, or embedded type.
  - There is more scope for parallel activities for system and application programs,
    - than utility programs.

Example

- The size of an organic software product has been estimated to be 32,000 lines of source code.

- Effort = $2.4*(32)^{1.05}$ = 91 PM
- Nominal development time = $2.5*(91)^{0.38}$ = 14 months

Intermediate COCOMO

- Basic COCOMO model assumes
  - effort and development time depend on product size alone.
- However, several parameters affect effort and development time:
    - Reliability requirements
    - Availability of CASE tools and modern facilities to the developers
    - Size of data to be handled

Intermediate COCOMO

- For accurate estimation,
  - the effect of all relevant parameters must be considered:
  - Intermediate COCOMO model recognizes this fact:
    - refines the initial estimate obtained by the basic COCOMO by using a set of 15 cost drivers (multipliers).

Intermediate COCOMO (CONT.)

- If modern programming practices are used,
  - initial estimates are scaled downwards.
- If there are stringent reliability requirements on the product :
  - initial estimate is scaled upwards.

Intermediate COCOMO (CONT.)

- Rate different parameters on a scale of one to three:
  - Depending on these ratings,
    - multiply cost driver values with the estimate obtained using the basic COCOMO.

Intermediate COCOMO (CONT.)

- Cost driver classes:
  - Product: Inherent complexity of the product, reliability requirements of the product, etc.
  - Computer: Execution time, storage requirements, etc.
  - Personnel: Experience of personnel, etc.
  - Development Environment: Sophistication of the tools used for software development.

Shortcoming of basic and intermediate COCOMO models

- Both models:
  - consider a software product as a single homogeneous entity:
  - However, most large systems are made up of several smaller sub-systems.
    - Some sub-systems may be considered as organic type, some may be considered embedded, etc.
    - for some the reliability requirements may be high, and so on.

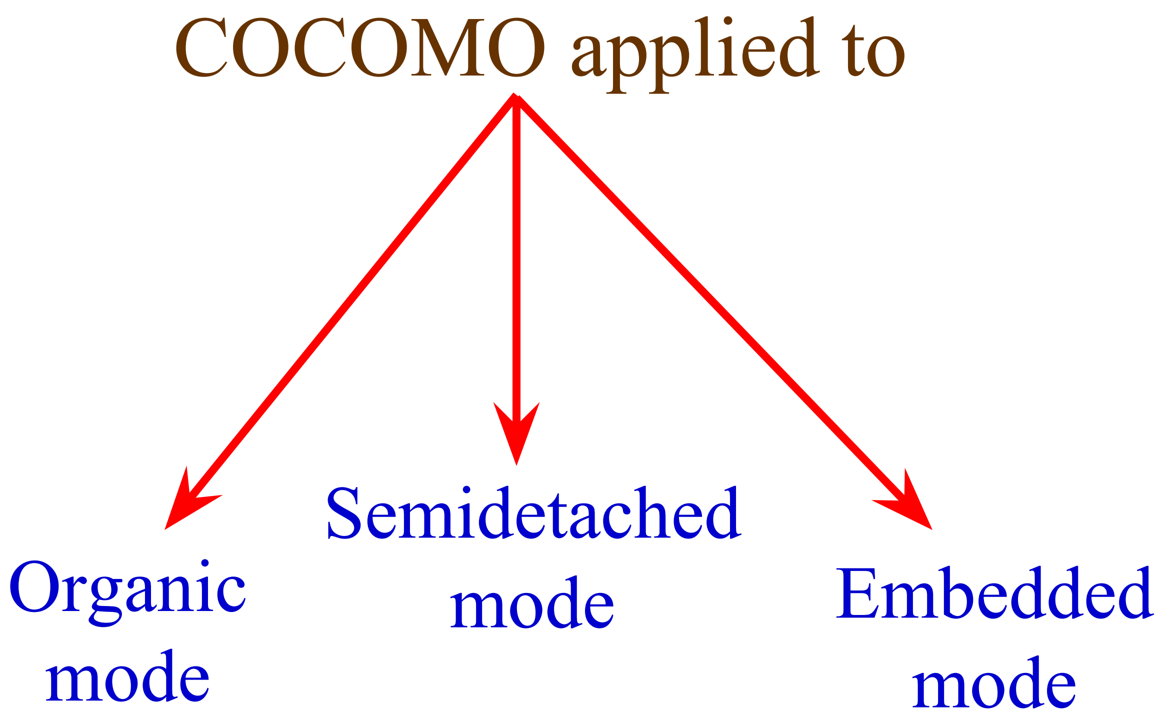
Complete COCOMO

- Cost of each sub-system is estimated separately.
- Costs of the sub-systems are added to obtain total cost.
- Reduces the margin of error in the final estimate.

Complete COCOMO Example

- 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)
- Costs of the components are estimated separately:
  - summed up to give the overall cost of the system.

Software Project Planning



Software Project Planning

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	Medium
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

Table 4: The comparison of three COCOMO modes

Software Project Planning

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).

Software Project Planning

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

Software Project Planning

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:

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

Software Project Planning

Example: 4.5

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.

Software Project Planning

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

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

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

Software Project Planning

(ii) Semidetached mode

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

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

(iii) Embedded mode

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

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

Software Project Planning

Example: 4.6

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.

Software Project Planning

Solution

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

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

$D = 2.5(1133.12)^{0.35} = 29.3 \text{ PM}$

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

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

Software Project Planning

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

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

Software Project Planning

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

Software Project Planning

(*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

Software Project Planning

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	--

Software Project Planning

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

Software Project Planning

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

Software Project Planning

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 ?

Software Project Planning

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

Software Project Planning

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 litter experience.

Software Project Planning

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

Software Project Planning

Development Phase

Plan / Requirements

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

% depend on mode & size

Software Project Planning

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%

Software Project Planning

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

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

$$E_p = \mu_p E$$

$$D_p = \tau_p D$$

Software Project Planning

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

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

Software Project Planning

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

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

Software Project Planning

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

Software Project Planning

Example: 4.8

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.

Software Project Planning

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**

Software Project Planning

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

$$EAF = 1.15 \times 1.15 \times 0.86 \times 1.07 = 1.2169$$

Software Project Planning

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

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

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

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

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

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

$$E_p = \mu_p E$$

$$D_p = \tau_p D$$

Software Project Planning

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