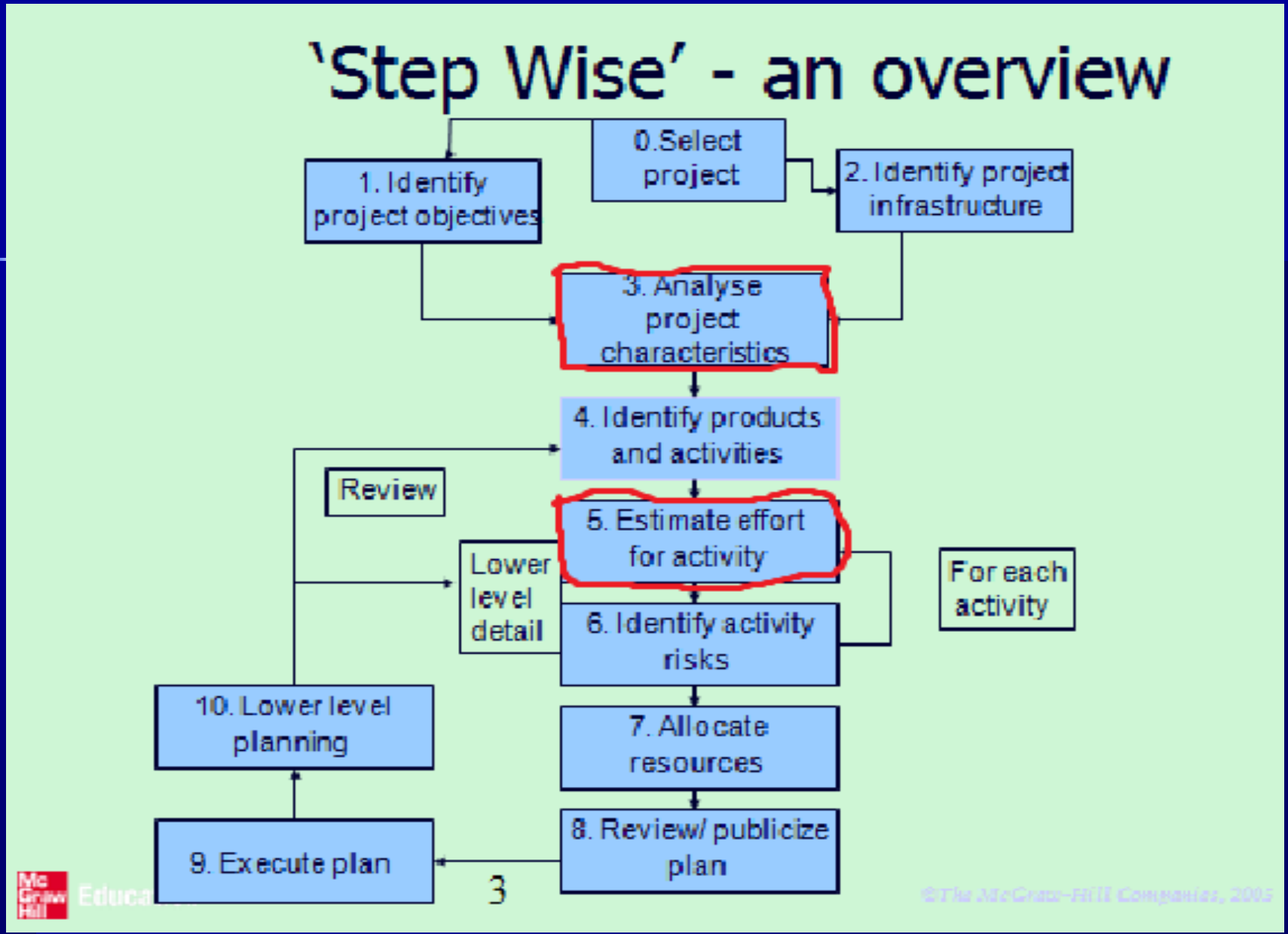
# Software Effort Estimation




Software estimation takes place in step 3 and step 5

# What makes a successful project?

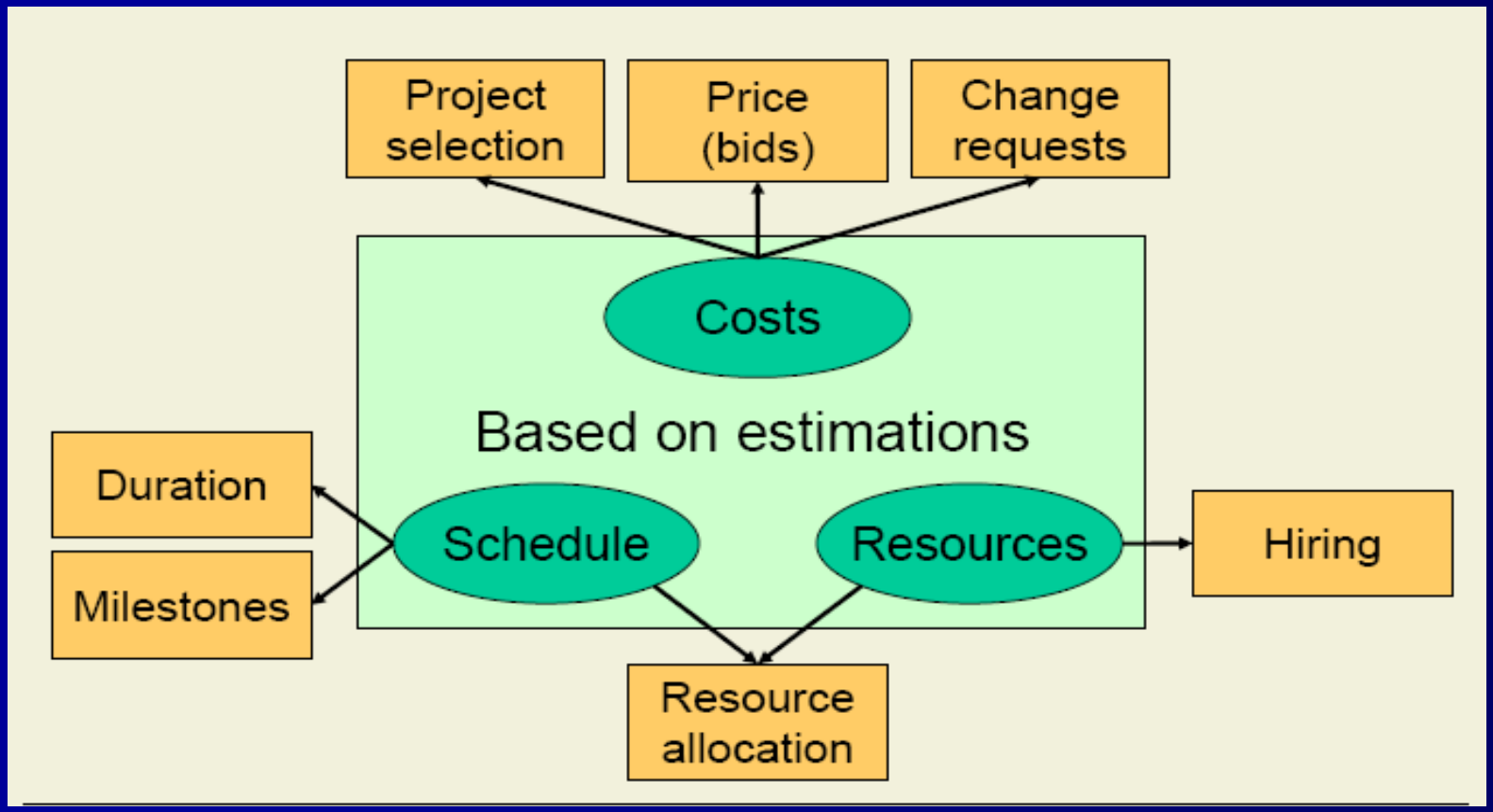
Delivering:

- agreed functionality
- on time
- at the agreed cost
- with the required quality

Stages:

1. set targets
2. Attempt to achieve targets

# Why estimations?



# What is estimation?

- The project manager must set expectations about the time required to complete the software among the stakeholders, the team, and the organization's management.
- If those expectations are not realistic from the beginning of the project, the stakeholders will not trust the team or the project manager.

# Software effort estimation

- Predicting the resources required for a software development process
- i.e. Effort estimation consists in predict how many hours of work and how many workers are needed to develop a project.

# Fundamental estimation questions

- How much effort is required to complete an activity?
- How much calendar time is needed to complete an activity?
- What is the total cost of an activity?

# Software cost components

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

# Where are estimation done?

- Strategic planning
- Feasibility Study
- System Specification
- Evaluation of supplier’s proposals
- Project Planning

# Measure of Work

- Time required vary with experience.
- Effort is measured using a measure called SLOC.
- Used to measure the size of a software program by counting the number of lines in the text of the program's source code
- SLOC is typically used to predict the amount of effort that will be required to develop a program.

# SLOC (Cont…)

- To estimate programming **productivity** or **effort** once the software is produced.
- Productivity= system size / Effort.
- Effort=system size/productivity

# KLOC

- KLOC stands for Thousands (kilo) lines of code.
- The size is determined by measuring the number of lines of source code a program has .

# Problems with over and under-estimates

- **Parkinson's Law**: 'work expands so as to fill the time available for its completion'.
- Implies when given an easy target staff will work less hard.
- Brook's Law: ' Putting more people on a late job makes it later'.
- Danger with the under estimate is the effect on quality.

# (cont..)

- Under-estimating
  - Under- Staffing resulting in staff burnout
  - Setting too short schedule results in loss of credibility as deadlines are missed
- Over-estimating
  - Projects cost more than they should resulting in unnecessary cost
  - Projects take longer to deliver resulting in lost opportunities

Calculate the productivity (that is, SLOC per work month) of each of the projects in Table 5.1 and also for the organization as a whole. If the project leaders for projects a and d had correctly estimated the source number of lines of code (SLOC) and then used the average productivity of the organization to calculate the effort needed to complete the projects, how far out would their estimates have been from the actual effort?

**Table 5.1** *Some project data – effort in work months (as percentage of total effort in brackets)*

| | *Design* | | *Coding* | | *Testing* | | *Total* | |
|---|---|---|---|---|---|---|---|---|
| *Project* | *wm* | *(%)* | *wm* | *(%)* | *wm* | *(%)* | *wm* | *SLOC* |
| a | 3.9 | (23) | 5.3 | (32) | 7.4 | (44) | 16.7 | 6050 |
| b | 2.7 | (12) | 13.4 | (59) | 6.5 | (26) | 22.6 | 8363 |
| c | 3.5 | (11) | 26.8 | (83) | 1.9 | (6) | 32.2 | 13334 |
| d | 0.8 | (21) | 2.4 | (62) | 0.7 | (18) | 3.9 | 5942 |
| e | 1.8 | (10) | 7.7 | (44) | 7.8 | (45) | 17.3 | 3315 |
| f | 19.0 | (28) | 29.7 | (44) | 19.0 | (28) | 67.7 | 38988 |
| g | 2.1 | (21) | 7.4 | (74) | 0.5 | (5) | 10.1 | 38614 |
| h | 1.3 | (7) | 12.7 | (66) | 5.3 | (27) | 19.3 | 12762 |
| i | 8.5 | (14) | 22.7 | (38) | 28.2 | (47) | 59.5 | 26500 |

| *Project* | *Work-months* | *SLOC* | *Productivity (SLOC/month)* |
|---|---|---|---|
| a | 16.7 | 6050 | 362 |
| b | 22.6 | 8363 | 370 |
| c | 32.2 | 13334 | 414 |
| d | 3.9 | 5942 | 1524 |
| e | 17.3 | 3315 | 192 |
| f | 67.7 | 38988 | 576 |
| g | 10.1 | 38614 | 3823 |
| h | 19.3 | 12762 | 661 |
| i | 59.5 | 26500 | 445 |
| Overall | 249.3 | 153868 | 617 |

**Table F.5** *Estimated effort*

| *Project* | *Estimated work-months* | *Actual* | *Difference* |
|---|---|---|---|
| a | 6050/617 = 9.80 | 16.7 | 6.90 |
| d | 5942/617 = 9.63 | 3.9 | −5.73 |

There would be an under-estimate of 6.9 work-months for project *a* and an over-estimate of 5.7 for project *d*.

# Software Effort Estimation Techniques

- Expert Judgement
- Analogy
- Parkinson
- Price to win
- Top-down estimating
- Bottom-up estimating
- Algorithmic models.

# Expert Judgement

- Estimate is based on experience.
- Involve experts in
  - Development techniques
  - Application Domain
- Asking someone who is knowledgeable.
- Method used when estimating the effort needed to change an existing piece of software.

# Price to win

- The ‘estimate’ is a figure that appears to be sufficiently low to win a contract.

# Parkinson

- Identifies the staff effort available to do a project and use that as the estimate.

# Estimating by Analogy

- Analogy: where a similar completed project is identified and its actual effort forms the basis for new project.
- Use of analogy also called **case-based reasoning**.
- **Source cases** - project that already completed
- **Target case** – the new project.
- Calculate the base estimate (target case – source case)
- Can be done by software named ANGEL.
- Identifies the source that is nearest to target by measuring Euclidean Distance between cases.

# Estimating by Analogy (cont..)

- Distance=square-root of $[(\text{target\_parameter}_1 - \text{source\_parameter}_1)^2 + \ldots\ldots [(\text{target\_parameter}_n - \text{source\_parameter}_n)^2$
- The cost of a project is computed by comparing the project to a similar project in the same application domain
- Advantages: Accurate if project data available
- Disadvantages: Impossible if no comparable project has been tackled.

Say that the cases are being matched on the basis of two parameters, the number of inputs to and the number of outputs from the system to be built. The new project is known to require 7 inputs and 15 outputs. One of the past cases, Project A, has 8 inputs and 17 outputs. The Euclidean distance between the source and the target is therefore the square-root of $((7-8)^2+(17-15)^2)$, that is 2.24.

Project B has 5 inputs and 10 outputs. What would be the Euclidean distance between this project and the target new project being considered above? Is Project B a better analogy with the target than Project A?

The Euclidean distance between Project B and the target case

$$= \sqrt{(7-5)^2 + (15-10)^2}$$

$$= \sqrt{2^2 + 5^2}$$

$$= 5.39$$

Project A is therefore a closer analogy.

# Bottom up estimating

- The estimator breaks the projects into component task
- These component tasks are  sized.
- calculate the effort required for each task
- Bottom up estimation is suitable when a project is completely novel or no historical data is available.
- Otherwise, assumption have to be made about the characteristics of the final system like number and size of software modules.

# Bottom-up estimating

1. Break project into smaller and smaller components
2. Stop when you get to what one person can do in one/two weeks
3. Estimate costs for the lowest level activities
4. At each higher level calculate estimate by adding estimates for lower levels

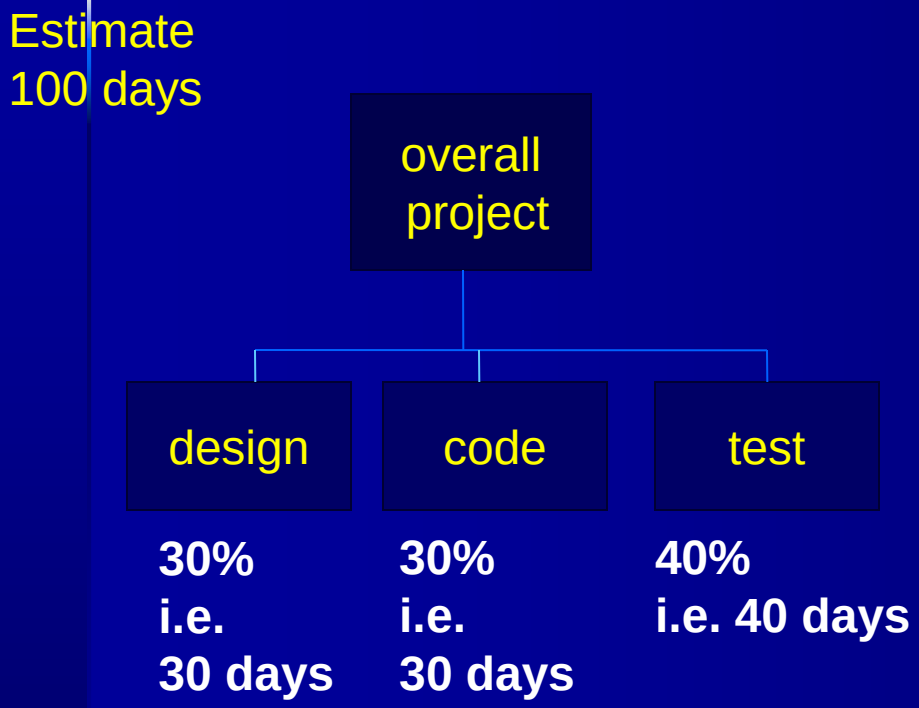
# Top-down approach and parametric models

- Associated with parametric models.
- E.g. House-owner who need to rebuild the house with insurance coverage.
- Insurance companies provide tables where owner can find an estimate based on parameters such as no. of storeys and the floor space.

# (cont...)

- Effort based on the characteristics of the final system.
- Effort = system size * productivity rate
- Once effort is calculated, allocate portion of that effort to various activities with in that project.

# Top-down estimates




- Produce overall estimate using effort driver (s)
- distribute proportions of overall estimate to components

# Bottom-up versus top-down

- Bottom-up
  - use when no past project data
  - identify all tasks that have to be done – so quite time-consuming
  - use when you have no data about similar past projects
- Top-down
  - produce overall estimate based on project cost drivers
  - based on past project data
  - divide overall estimate between jobs to be done

# Albrecht function point analysis

- Function point metric was devised by A.J. Albrecht.
- Means of measuring software size and *productivity.*
- FPA ,recognized method to measure the functional size of an information system.
- The unit of measurement is "function points".

# FP = count total [0.65 + 0.01 Σ(*Fi*)]

where count total is the sum of all FP entries

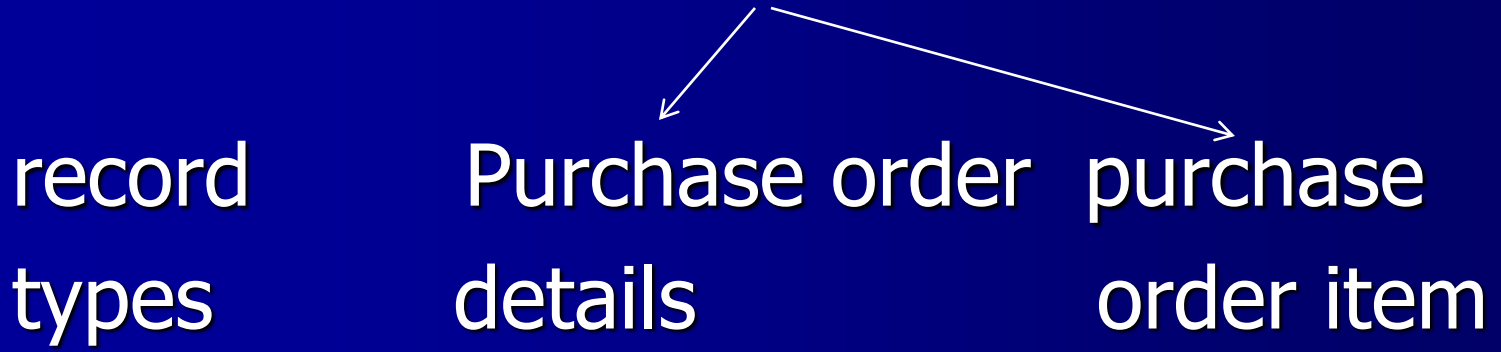
- The *Fi (i = 1 to 14) are "complexity adjustment values" based on responses to some* questions

# FPA's(Cont..)

- Basis of FPA is that computer based information system consists five major components or external user types.
- **External input types**: Input transactions that update internal computer files.
- **External output types**: Where data is output to the user.E.g. printed reports.

# (cont…)

- **Logical internal file type**: standing files used by the system.
- E.g. Purchase order.



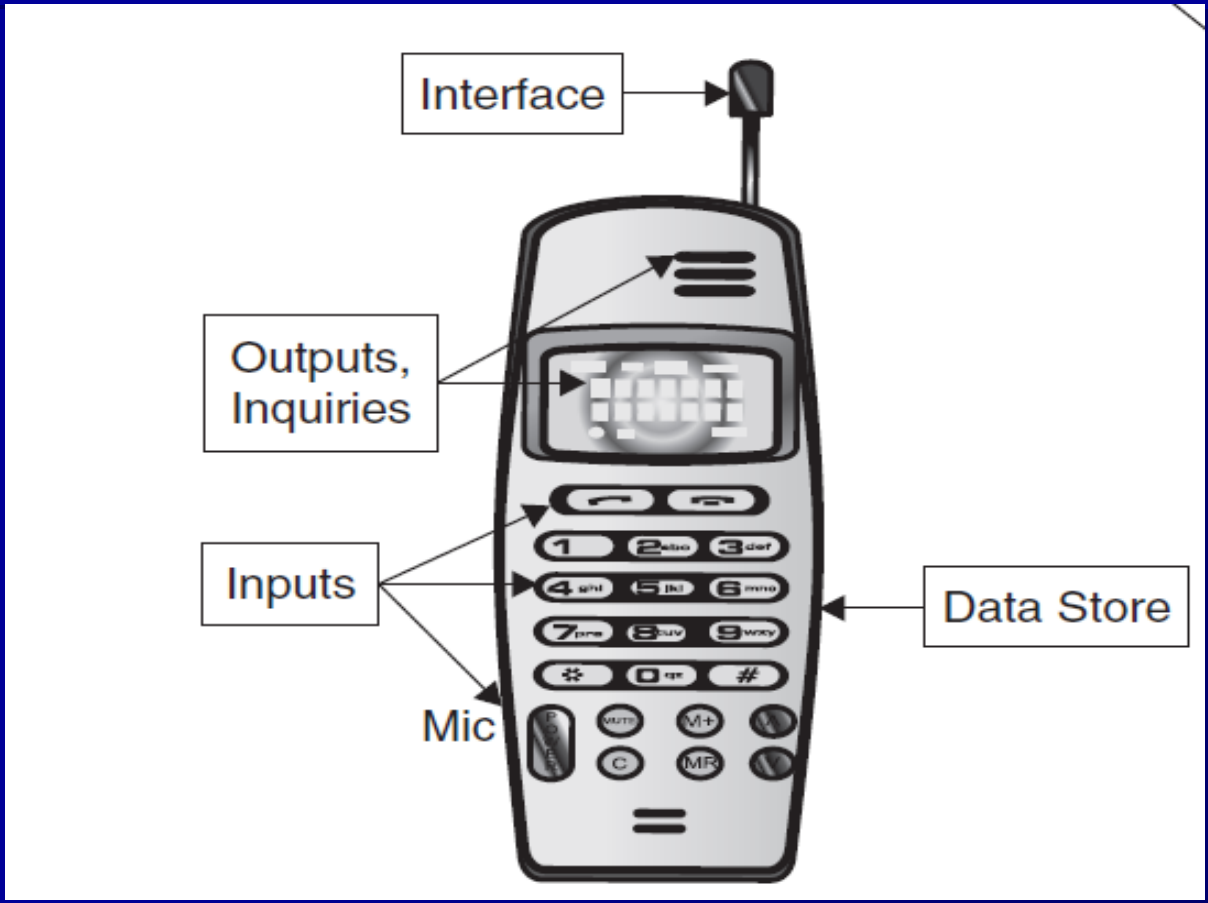
# (cont…)

- Data types in Purchase order details include: order number, supplier reference , order date.
- Data types in Purchase order items include product code, unit price and number ordered.
- **External interface file types**: Output and input that might pass to and from other computer application.

# (Cont..)

- **External Inquiry types**: transactions initiated by user that provide information but do not update the internal files.
- After identifying External user types each component is classified as either high, low or average complexity.

# Example.....



- Number of user inputs: Each user input that provides distinct application oriented data to the software is counted.
- Number of user outputs: Each user output that provides application oriented information to the user is counted. In this context "output" refers to reports, screens, error messages, etc. Individual data items within a report are not counted separately.
- Number of user inquiries: An inquiry is defined as an on-line input that results in the generation of some immediate software response in the form of an on-line output. Each distinct inquiry is counted.
- Number of files: Each logical master file is counted.
- Number of external interfaces: All machine-readable interfaces that are used to transmit information to another system are counted.

# Alternatively the following questionnaire could be utilized

- Does the system require reliable backup and recovery?
- Are data communications required?
- Are there distributed processing functions?
- Is performance critical?
- Will the system run in an existing, heavily utilized operational environment?
- Does the system require on-line data entry?
- Does the on-line data entry require the input transaction to be build over multiple screens or operations?
- Are the master files updated online?
- Are the input, outputs, files or inquiries complex?
- Is the internal processing complex?
- Is the code designed to be reusable?
- Are conversions and installation included in the design?
- Is the system designed for multiple installations in different organizations?
- Is the applications designed to facilitate change and ease of use?

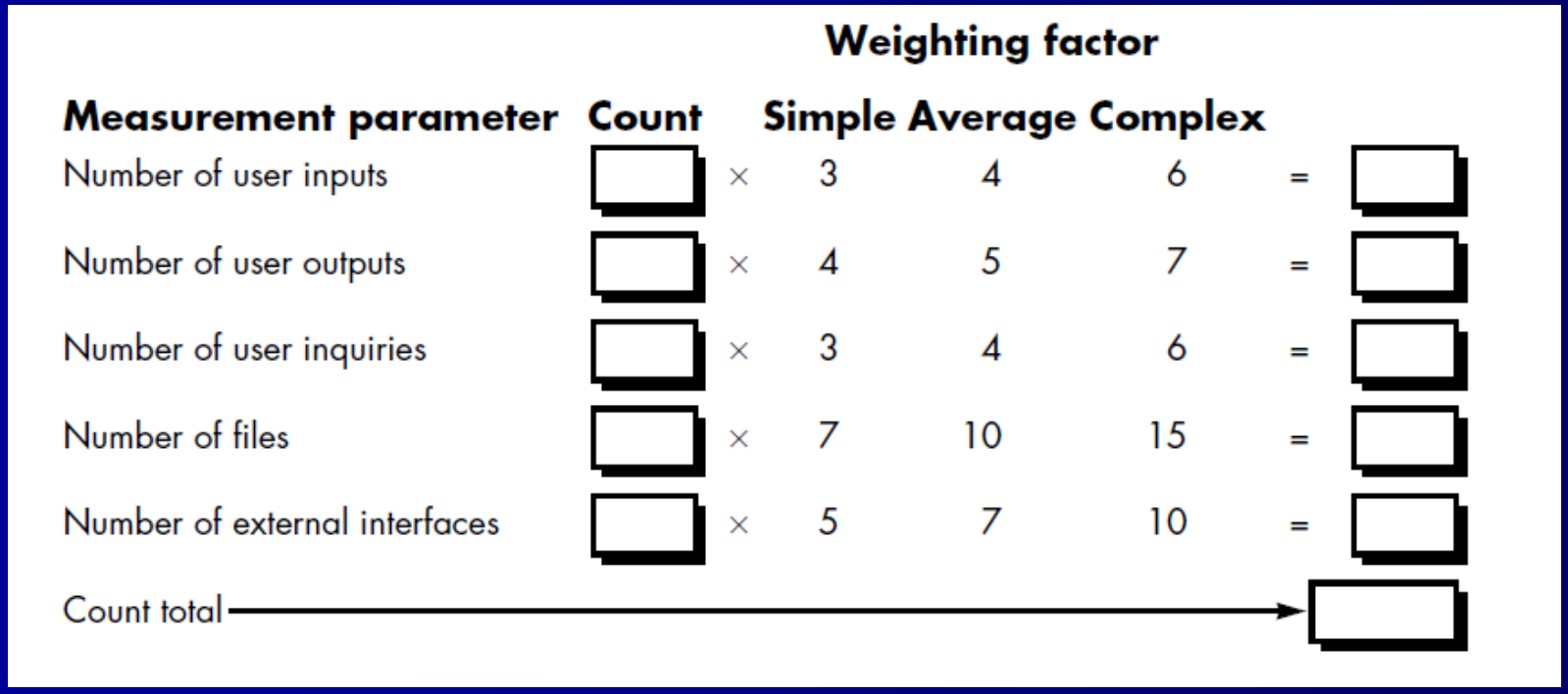
# Albrecht complexity multiplier

| External user type | low | average | high |
|---|---|---|---|
| External input type | 3 | 4 | 6 |
| External output type | 4 | 5 | 7 |
| Logical internal file | 7 | 10 | 15 |
| External interface file | 5 | 7 | 10 |
| External inquiry type | 3 | 4 | 6 |

Albrecht FP's now referred as IFPUG

# (Cont..)

- Count of external user type is multiplied by specified weights which are then summed to obtain overall FP count.
- Lines of code for languages:
- COBOL-91
- C-128
- QuickBasic- 64


Weighting factor
Measurement parameter
Count
Simple
Average
Complex
Number of user inputs
× 3 4 6 =
Number of user outputs
× 4 5 7 =
Number of user inquiries
× 3 4 6 =
Number of files
× 7 10 15 =
Number of external interfaces
× 5 7 10 =
Count total

# Example question

| External user types | Count | Complexity |
|---|---|---|
| Number of user inputs | 3 | simple |
| Number of user outputs | 10 | average |
| Number of user inquiries | 12 | simple |
| Number of files | 8 | complex |
| Number of external interfaces | 3 | average |

# IFPUG file type complexity

| Number of record types | Number of data types | | |
|---|---|---|---|
| | <20 | 20-50 | >50 |
| 1 | low | low | average |
| 2-5 | low | average | high |
| >5 | average | high | high |

# IFPUG external input type complexity

| Number of file types | Number of data types accessed | | |
|---|---|---|---|
| | <5 | 5-15 | >15 |
| 0 or 1 | low | low | average |
| 2 | Low | average | high |
| >2 | average | high | high |

# IFPUG external output type complexity

| Number of file | Number of data types | | |
|---|---|---|---|
| | <6 | 6-19 | >19 |
| 0 or 1 | low | low | average |
| 2 or 3 | low | average | high |
| >3 | average | high | high |

# Function Point Mark II

- Mark II method recommended by CCTA(Central computer and Telecommunication Agency)
- Here assumption is that any transaction comprises the basic structure.

The task for which Brigette has been made responsible in Exercise 5.2 needs a program that will extract yearly salaries from the payroll file, and the details of courses and hours taught on each course by each member of staff from two files maintained by the time-tabling system. The program will calculate the staff costs for each course and put the results into a file that will then be read by the main accounting system. The program will also produce a report showing for each course the hours taught by each member of staff and the cost of those hours.

Using the method described above, calculate the Albrecht function points for this subsystem, assuming that the report is of high complexity, but that all the other elements are of average complexity.

The function types are as follows.

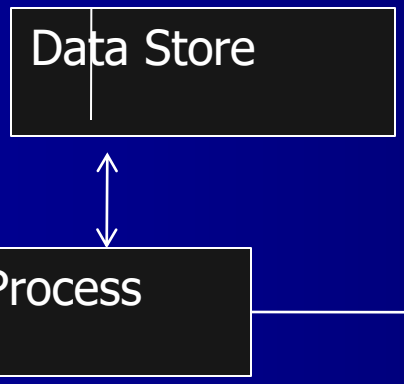
| | |
|---|---|
| External input types | none |
| External output types | the report, 1 |
| Logical internal file types | the accounting feeder file, 1 |
| External interface file types | payroll file, staff file (timetabling,) courses file (timetabling,) accounting feeder file, 4 |
| External inquiry types | none |

Because the accounting feeder file is outgoing, it is counted once as a logical internal file type and once as an external interface file type.

| | |
|---|---|
| External enquiry types | none |
| External output types | $1 \times 7 = 7$ |
| Logical internal file types | $1 \times 10 = 10$ |
| External interface types | $4 \times 7 = 28$ |
| External inquiry types | none |
| | |
| Total | 45 |

Estimated lines of Cobol = $45 \times 91 = 4095$

# (cont..)




- Input
  From user

Output
return to user

# (Cont..)

- **For each Transaction:**
- $W_i$ * (number of input data element types) +
- $W_e$ * (number of entity types referenced)+
- $W_o$ * (number of output data element types)

# Weightings

- $W_i$,$W_e$,$w_o$ are derived by asking developers what proportion of effort has been spend in previous projects.

  Industry averages are available i.e.

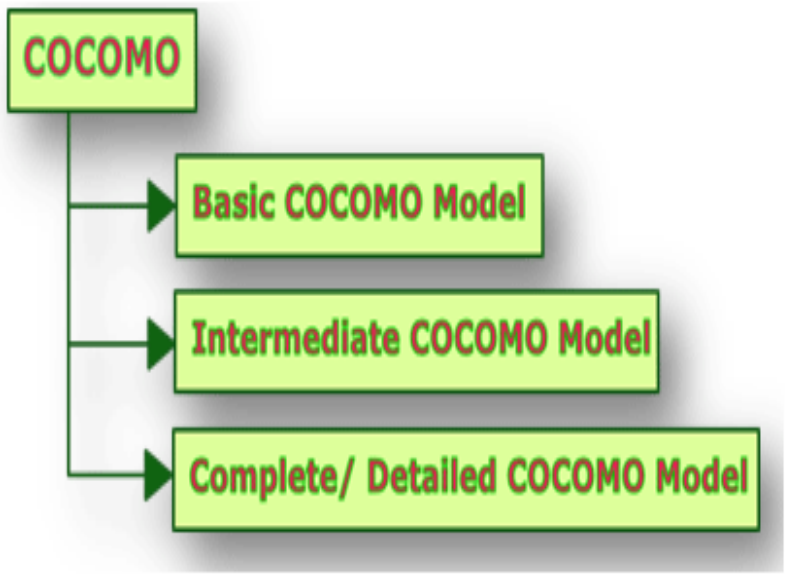
- 0.58 for $W_i$
- 1.66 for $W_e$
- 0.26 for $W_o$.

# COCOMO : a parametric model

- **Barry Boehm** designed **COCOMO**
- It give an estimate of the number of man-months it will take to develop a software product.

**COCOMO:** The COCOMO (Constructive Cost Estimation Model) is proposed by **DR. Berry Boehm in 1981** and that's why it is also known as COCOMO'81. It is a method for evaluating the cost of a software package. According to him software cost estimation should be done through three stages:

1. Basic COCOMO Model
2. Intermediate COCOMO Model
3. Complete/Detailed COCOMO Model



# The hierarchy of COCOMO models takes the following form:

*Model 1.* The Basic COCOMO model is a static, single-valued model that computes software development effort (and cost) as a function of program size expressed in estimated lines of code (LOC).

*Model 2.* The Intermediate COCOMO model computes software development effort as a function of program size and a set of "cost drivers" that include subjective assessments of product, hardware, personnel and project attributes.

*Model 3.* The Advanced COCOMO model incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step (analysis, design, etc.) of the software engineering process

# Basic COCOMO

- The basic COCOMO estimation model is given by the following expression:

  $$Effort = a * size^{b}$$

- Effort is measured in person-month.
- Size is measured in 'kdsi', thousands of delivered source code instruction.
- a and b are constants.
- Their value depends on whether the system is 'organic', 'semi-detached','embedded'.

# (cont..)

- Organic mode :when small teams develop software that is very familiar.
- Embedded mode: product being delivered within very tight constraints and unfamiliar.
- Semi-detached – combines both.

## Organic

The organic mode is typified by systems such as payroll, inventory, and scientific calculation. Other characterizations are that the project team is small, little innovation is required, constraints and deadlines are few, and the development environment is stable.

## Semidetached

The semidetached mode is typified by utility systems such as compilers, database systems, and editors. Other characterizations are that the project team is medium-size, some innovation is required, constraints and deadlines are moderate, and the development environment is somewhat fluid.

## Embedded

The embedded mode is typified by real-time systems such as those for air traffic control, ATMs, or weapon systems. Other characterizations are that the project team is large, a great deal of innovation is required, constraints and deadlines are tight, and the development environment consists of many complex interfaces, including those with hardware and with customers

- The basic COCOMO estimation model is given by the following expressions:

$$\textbf{Effort} = a \times (KLOC)^{b}\ PM$$

$$\textbf{Tdev} = 2.5 \times (Effort)^{c}\ Months$$

**People Required=effort/Development time**

- KLOC is the estimated size of the software product expressed in Kilo Lines of Code,
- a, b, c are constants for each category of software products,
- Tdev (nominal development time) is the estimated time to develop the software, expressed in months,
- Effort is the total effort required to develop the software product, expressed in person months (PMs).

# COCOMO constants

| System Type | a | b | c |
|---|---|---|---|
| Organic | 2.4 | 1.05 | 0.38 |
| Semi-detached | 3.0 | 1.12 | 0.35 |
| Embedded | 3.6 | 1.20 | 0.32 |

COCOMO cont…

- **Example:**

  Assume that the size of an organic type software product has been estimated to be 32,000 lines of source code. Assume that the average salary of software engineers be Rs. 15,000/- per month. Determine the effort required to develop the software product and the nominal development time.

From the basic COCOMO estimation formula for organic software:

Effort = **$2.4 \times (32)^{1.05} = 91$ PM**

Nominal development time = **$2.5 \times (91)^{0.38} =$ 14 months**

Cost required to develop the product = **14 x 15,000 = Rs. 210,000/-**

# COCOMO Intermediate

- Is an extension of the Basic COCOMO model
- Considers a set of four "cost driver attributes", each with a number of subsidiary attributes:
  - Product attributes
  - Computer attributes
  - Personnel attributes
  - Project attributes

# (cont..)

- **Product attributes**
  - Required software reliability
  - Size of application database
  - Complexity of the product
- **Hardware attributes**
  - Run-time performance constraints
  - Memory constraints
  - Volatility of the virtual machine environment
  - Required turnabout time

# Cont..

- **Personnel attributes**
  - Analyst capability
  - Software engineer capability
  - Applications experience
  - Virtual machine experience
  - Programming language experience
- **Project attributes**
  - Use of software tools
  - Application of software engineering methods
  - Required development schedule

# (cont..)

- In COCOMO Intermediate , a nominal effort estimate ($pm_{nom}$) is calculated same as the basic model.
- Finally it is adjusted by development effort multiplier(dem)

$$pm_{est} = pm_{nom} * dem$$

- *dem* is calculated by multiplying the multiplier selected for each cost drivers.

# COCOMO II

- COCOMO II addresses the following three phases of the spiral life cycle:
- applications development, early design and post architecture
- Most significant input to the COCOMO II model is size. Size is treated as a special cost driver in that it has an exponential factor, E. This exponent is an aggregation of five scale factors

- The five Scale Factors are:

- PREC Precedentedness (how novel the project is for the organization)
- FLEX Development Flexibility
- RESL Architecture / Risk Resolution
- TEAM Team Cohesion
- PMAT Process Maturity

Estimate of person months

$pm = A(size)^{(sf)} \times (em_1) \times (em_2) \times (em_3) \ldots (em_n)$

Scale factor is drived as

$sf = 1.01 + 0.01 \times \sum (\text{exponent driver ratings})$