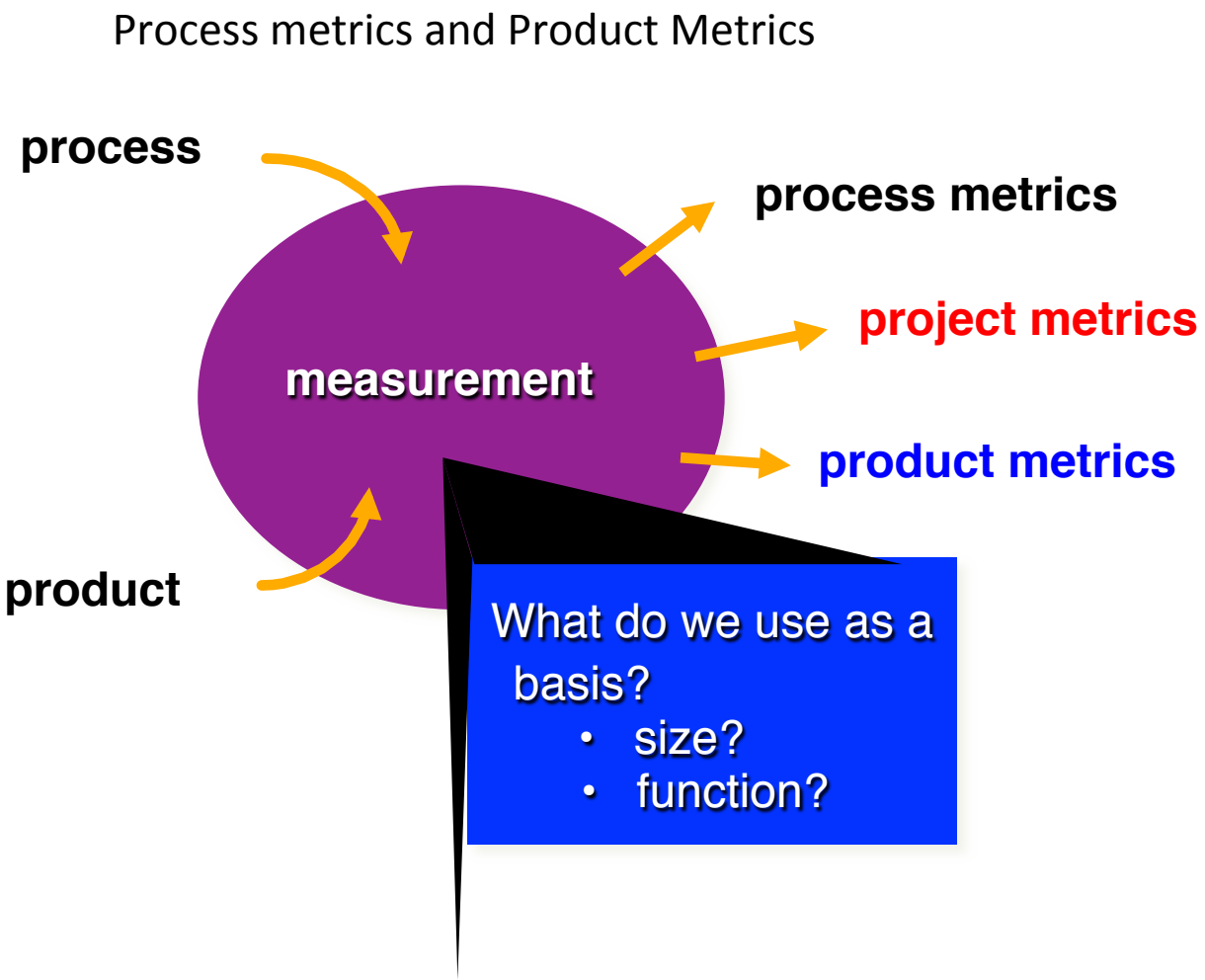
CMPS411 Software Engineering
Spring 2018

Lecture 16

Software Project Estimation:
Function Points
and
COCOMO

A Good Manager Measures



Estimation: Importance

- Cost and schedule estimation
- Estimating cost and schedule has to be done as early as possible during the project life cycle
- Type of Costs
  - Facilities: hardware, space, furniture, telephone, etc.
  - Software tools for designing and developing software
  - Staff (effort): the biggest component of cost.
- Type of Schedule
  - Final delivery
  - Milestones delivery.

Most Popular Software Size Metrics: Two approaches

- **Function based size metrics**
  - Software function is the main input
  - Function points (FP)
    - Based on estimated functions of a software
    - Five categories of function and their complexity
- **Source code (LOC) based size metrics**
  - Line of Code (LOC) is the main input
  - COCOMO 81
    - Basic COCOMO
    - Intermediate COCOMO with 15 cost drivers
  - COCOMO 2

Comparing Source Code with Software Function

Programming Language	Estimated Line of code (LOC) per Function point			
	avg.	median	low	high
Ada	**154**	-	104	205
Assembler	**337**	315	91	694
C	**162**	109	33	704
C++	**66**	53	29	178
COBOL	**77**	77	14	400
Java	**63**	53	77	-
JavaScript	**58**	63	42	75
Perl	**60**	-	-	-
PL/1	**78**	67	22	263
Powerbuilder	**32**	31	11	105
SAS	**40**	41	33	49
Smalltalk	**26**	19	10	55
SQL	**40**	37	7	110
Visual Basic	**47**	42	16	158

These data are just estimation, not an accurate representation.

Why Function Point?

- ✧ Programming language independent
- ✧ Used readily countable characteristics that are determined early in the software process
- ✧ Does not “penalize” inventive (short) implementations that use fewer Line of Code (LOC) than other more clumsy versions with huge LOC
- ✧ Makes it easier to measure the impact of reusable components

Example: Line of Code (LOC) Approach

Function	Estimated LOC
user interface and control facilities (UICF)	2,300
two-dimensional geometric analysis (2DGA)	5,300
three-dimensional geometric analysis (3DGA)	6,800
database management (DBM)	3,350
computer graphics display facilities (CGDF)	4,950
peripheral control (PC)	2,100
design analysis modules (DAM)	8,400
estimated lines of code	**33,200**

- Average productivity for systems of this type = 620 LOC/pm.
- Cost =$8000 per month, the cost per line of code is approximately $13.
- Based on the estimated LOC and the historical productivity data, the total estimated project cost is **$431,600 and**
- The estimated effort is 54 person-months.

Function Points(FP)

- The approach is based on what the system offers (functions) to the users
- Measure what the users of a software provide to the system (*input),* request (*query*), receive (*output*), what the system stores (*files*), and how the system interacts with other systems(*interface*)
- Measure independently of technology used for implementation
- Provide a sizing metric to support quality and productivity
- Provide a vehicle for software estimation
- Provide a normalisation factor for software comparison
- Measuring functions is theoretically promising but realistically very difficult

Example: FP Approach

Information Domain Value	opt.	likely	pess.	est. count	weight	FP-count
number of inputs	20	24	30	24	4	97
number of outputs	12	15	22	16	5	78
number of inquiries	16	22	28	22	5	88
number of files	4	4	5	4	10	42
number of external interfaces	2	2	3	2	7	15
count-total						**321**

The estimated number of FP is derived:

$$FP_{estimated} = \text{count-total} \times [0.65 + 0.01 \times \sum (F_i)]$$

$$FP_{estimated} = 375$$

organizational average productivity = 6.5 FP/pm.

burdened labor rate = $8000 per month, approximately $1230/FP.

Based on the FP estimate and the historical productivity data, **total estimated project cost is $461,000 and estimated effort is 58 person-months.**

- **Opt. (optimistic) = Simple, low complexity ,**
- **Likely = Average complexity,**
- **Pess. (Pessimistic)= High complex**

Function Points: Main features

- ✧ FP weighting factor for complexity
- ✧ Unadjusted function count (UFC)
- ✧ Value adjusted factors (VAF)
- ✧ Technical complexity factor (TCF)
- ✧ Computing final function points

Function Point (FP) Components

- FP is a weighted total of five major components
  - **Number of external inputs** (transaction types)
    - Data required from the user to provide a function
    - selecting a menu is **not** an input!
  - **Number of external outputs** (report types)
    - data output to the user
  - **Number of logical files** (as the user conceive them)
    - internally maintained logical groups of data
  - **Number of external interface files**(accessed by the applications)
    - externally maintained logical groups of data by other system
  - **Number of inquiries** (types of on-line queries)
    - combination of input (request) and output (retrieval)

- *Specific user functionality is evaluated in terms of what is delivered, not how it is delivered*

FP Complexity: Schema

Functions/Components	Weighting Factor for complexity		
	Low	Average	High
	Optimistic	Likely	Most unlikely
External Input	3	4	6
External Output	4	5	7
Logical file	7	10	15
External Interface	5	7	10
Inquiry	3	4	6

- Each function/component type is categorised into one of three factors (low, average, high complexity) as above based on their relative complexity.
- These values are fixed.
- Find out how many of each component type are in high complexity, how many are in low, and how many are in average complexity.
- Then assign the above numbers to the components accordingly

Examples of Assigning Complexity Weights

- ***Example 1***. 5 external input are very complex, 12 input are average complex, and 24 input are low complex. We will assign the weights to these external inputs according to the table as:
  - External Input: (5 x 6), (12 x 4), (24 x 3) (we have used here the first row in the table for the external input)

- ***Example 2***: For 20 inquiries are too complex, 22 are average, and 70 are low complex. We get:
  - Inquiry: (20 x 6), (22 x 4), and (70 x 3). (we have used the weights from the last row in the table for the Inquiry)

Functions/Components	Low	Average	High
External Input	3	4	6
External Output	4	5	7
Logical file	7	10	15
External Interface	5	7	10
Inquiry	3	4	6

Unadjusted Function Count (UFC)

$$UFC = \sum_{i=1}^{5} \sum_{j=1}^{3} W_{ij} \times \mathrm{X}_{ij}$$

W_{ij} are the weighting factors of the 5 types of component by 3 complexity levels (*weighting factors*)

X_{ij} are the numbers of each component in the application.

In order to understand this formula, we have to see examples.

Example of UFC

Example: A system has 2 low, 5 average and 4 high complex external input. It has 5 low, 6 average, and 5 high complex external output. It has 4 low, 8 average and 2 high complex logical files.

The system also has 8 low, 4 average and 10 high complex inquiries.

Using the above formula and the table from the previous slide, we get the following:

External input: (2 x 3) + (5 X 4) + (4 x 6) = 50
External output: (5 x4) + (6 x 5) + (5 x 7) = 85
Logical files: (4 x7) + (8 x 10) + (2x15) =138
External interface: 0 + 0 + 0 = 0
Inquiry: (8 x 3) + (4 x 4) + (10 x6)=110

UFC = 50 + 85 + 138 + 0 + 110 = **383**

Value Adjusted Factor (VAF)

14 general system characteristics. These are fixed and applicable to any system.

1. Data communications
2. Distributed functions
3. Performance
4. Heavily used configuration
5. Transaction rate
6. On-line data entry
7. End-user efficiency
8. On-line update
9. Complex processing
10. Reusability
11. Installation ease
12. Operational ease
13. Multiple sites
14. Facilitation of change

• Scores ranging from 0 to 5 (called degree of influence) for each of these 14 general system characteristics are then summed to form the **value adjustment factor (VAF)**

Degree of Influence

- The **degree of influence on a scale of 0 – 5** is used to assess the impact of ***14 general system characteristics*** in terms of their likely effect for the application

    - 0 - Not present, or no influence
    - 1 - Incidental influence
    - 2 - Moderate influence
    - 3 - Average influence
    - 4 - Significant influence
    - 5 - Strong influence throughout

• 0 means it is irrelevant or not applicable to the software; and 5 means it is very essential.

Example of VAF

• **Example** : A system has data communication with incidental influence (less influence), and it has online data entry with strong influence throughout, and it operates moderate rate on multiple sites significant influence). We now assign VAF with degree of influence:

1. **Data communications - 1**	8. On-line update - 0
2. Distributed functions - 0	9. Complex processing - 0
3. Performance - 0	10. Reusability - 0
4. Heavily used configuration - 0	11. Installation ease - 0
5. Transaction rate - 0	12. Operational ease - 0
6. **On-line data entry - 5**	13. **Multiple sites - 5**
7. End-user efficiency - 0	14. Facilitation of change - 0

VAF of the above scenario is:

VAF= (1 x 1) + (2 x 5) = 11

Technical Complexity Factor (TCF)

$$TCF = 0.65 + 0.01 \sum_{i=1}^{14} F_i$$

Where F_i is the score for general system characteristics i. TCF varies from 0.65 to 1.35.

- TCF will be 0.65 if each $F_i = 0$.
- TCF will be 1.35 if each $F_i = 5$.

- We now calculate the TCF of our example from the previous slide:
  TCF = 0.65 + 0.01 (11) = 0.76
- From slide 17, we bring our **UFC = 383**.
- Finally, the number of function points is obtained by multiplying **UFC** and the **TCF**
- FP = UFC X TCF
- Apply this TCF to our UFC in our example. We get final FP:
  - **383 x 0.76 = 291.08 (FP)**
- If the estimated cost for one FP is QR. 10,000. The total cost to develop this system will be: QR 2,910,000. If it takes 1 day average to develop one FP, it will take 291 days to develop the entire system!

Summary of Function Point Calculation: How to do it?

- Identify each **functional component** (input, output, files, interface and inquiry) of your system
- Apply **complexity weight** (low, average or high) to each identified function
- Calculate points for each function depending on its complexity
- Add up all function points to give the (Raw) Unadjusted Function Point Count (**UFC**)
- Assign **degree of influence** score (0 to 5) to each of the 14 general system characteristics that your software might have, and get **VAF**.
- Apply an adjusted factor (± 35%) to fine tune the estimate and Find **Technical complexity factor (TCF)** or Value Adjustment Factor (VAF)
- TCF/VAF may range from 0.65 to 1.35, the adjustment to the original UFC is in the range ± 35%
- Calculate **FP.**

FP Calculation: Another Example

A system will likely have the following types and number of functions/components:

2 external inputs

3 external outputs

2 inquiries

2 external interface

7 logical files

Assume Average complexity in each case. See the table (slide 9). We used the average weight (the middle column) of the table.

UFC = (4 * 2) + (5 * 3) + (4 * 2) + (10 * 2) + (7 * 2)

UFC = 58

Assume the following 14 general system characteristics,

- **3, 5, 9, 11, 12, 13 are assigned weight 0 (*no influence*)**
- **1, 2, 6, 7, 8, 14 are assigned 3 (*average influence*) , and**
- **4, 10 are assigned 5 (*strong influence throughout*)** [See slide no. 20]

VAF = ((6 * 3) +(2 * 5)) = 28

TCF = 0.65 +0.01(28) = 0.93

FP = 58 * 0.93

We now see how to identify and count external input, external output, logical files, inquiry, and external interfaces.

External Input

- Definition
  - Data or control information which enters the application's external boundary
- Counting Method
  - Identify all functions that need input data from the user or other external systems
  - Count how many of them are complex, how many are average complex and how many are simple.
  - Examples of input:
    - Transactional data
    - Screen input, those are processed such as stored, transmitted further, or computed
    - Batch input
  - Menu screen or logon screens or menu selection data <u>are not</u> Input
  - Examples:
    - Customer name and account number (consider as one input) if entered at the same time or as a single screen format
    - Bank sending approval or rejection advice to the system one input)

External Output

- Definition
  - Data or control information which goes out of the application's external boundary
- Counting Method
  - Identify all processes that send output or control data external to the application's boundary
  - Count the number of complex, average complex and simple
  - Examples:
    - Exam result of a course
    - Weekly sale report
    - Flight information
  - Help screens, error messages are not external output

Logical Files: Definition

- User identifiable group of logically related data
  - Refers to data group of files as fulfilling specific user requirements
- Counting method
  - One group of data or a file is considered one logical file.
  - The ability to add, change, or delete data through the standarized process in application
  - Find the complex, average and simple logical files

Logical Files: Counting Method

- Identify all data that is stored internal to the application's boundary and required for the application
- Group the data logically based on the user's view
- Examples of logical files:
  - Master files such as Customer file
  - Student file
  - Course file
  - Application security data
  - Help message
- The followings are not logical files:
  - Temporary files,
  - Sort files
  - Back up files for corporate use, not for the user

External Interface File

- Definition
  - User identifiable logically related data or control information maintained by another application, but utilised or accessed by the user's application software
- Counting Method
  - Identify all data that is stored external to the application boundary, and not maintained by the application software
  - Examples:
    - Reference data
    - Government regulations

External Inquiry

- Definition
  - A unique input/output combination that results in the retrieval of data, does not contain derived data and does not update an logical file
- Counting Method
  - Identify all processes where an input triggers a retrieval of data
  - Examples
    - Find the appointment of a patient with a doctor
    - Immediate retrieval of data (semester GPA of a semester)
    - Logon screen
    - Help
  - Menus, error messages, on-line documentation are not Inquiry

COCOMO: Topics covered

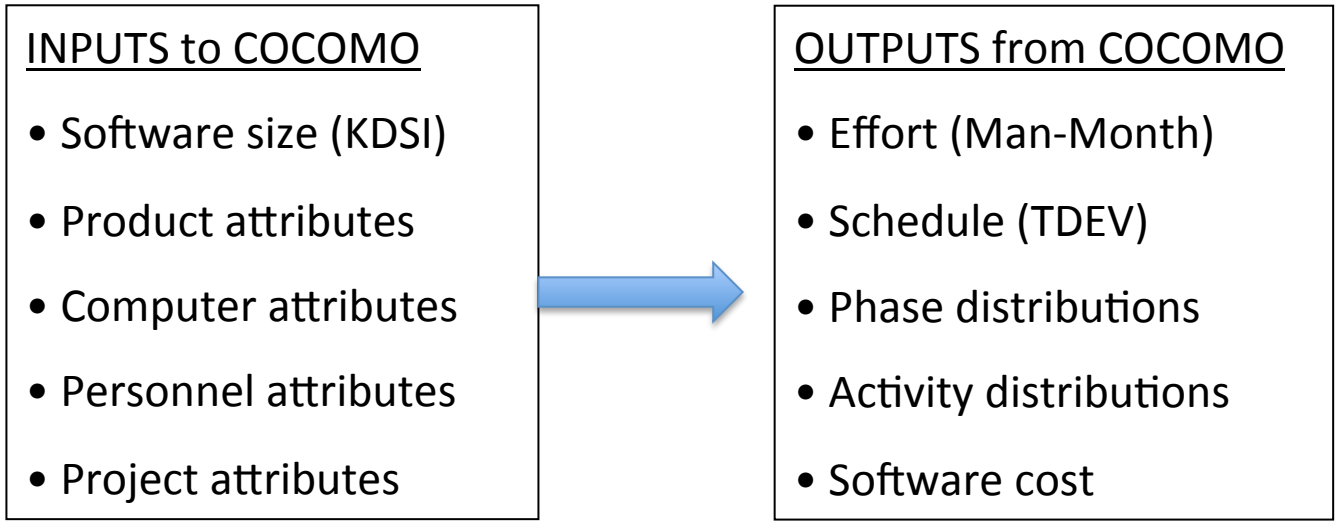
- ✧ LoC based Estimation techniques
- ✧ COCOMO81
  - Modes
  - Models
    - Basic COCOMO Model
    - Intermediate COCOMO Model
  - Maintenance efforts and annual traffic of LoC
- ✧ COCOMO II (COCOMO 2)

COCOMO History

- 1981 - The original COCOMO is introduced in Dr. Barry Boehm's textbook ***Software Engineering Economics***. This model is now generally called "COCOMO 81".
- 1987 - Ada COCOMO and Incremental COCOMO are introduced (proceedings, Third COCOMO Users Group Meeting, Software Engineering Institute).
- 1995, 1996 - Early papers describing COCOMO II published.
- 1997 - The first calibration of COCOMO II is released by Dr. Boehm, and named "COCOMO II. 1997".
- 1998 - The second calibration of COCOMO II is released. It's named "COCOMO II.1998".
- 1999 - COCOMO II.1998 is renamed to COCOMO II.1999 and then to COCOMO II.2000 (all three models are identical).
- 2000 - The book ***Software Cost Estimation with COCOMO II*** (Dr. Barry Boehm, et al) is published to document how to apply the latest estimation model. Most of the original ***Software Engineering Economics*** is still applicable to modern software projects.
- First published in 1981, the original COCOMO model has recently been superseded by COCOMO II, which reflects the improvements in professional software development practice, positioning COCOMO for continued relevancy into the 21st century.
- Nevertheless, COCOMO81 still remains the most popular cost estimation tool due to its easy to use methods.

COnstructive COst Model (COCOMO)

- Refers to parametric model used to estimate effort and schedule for software



(Boehm, B.: Software Economics, Prentice-Hall .)

Modes of COCOMO

- **Organic Mode**
  - Stable development environment without any need for innovative architecture
  - Highly familiar in-house environment with up to 50 KDSI size
  - Examples, Inventory system, university system.
- **Embedded Mode**
  - Operate under tight constraints of HW, SW, quality requirements, regulations, and operation procedures
  - Strict Validation and Verification procedures
  - Examples, air traffic control system, autopilot system
- **Semidetached mode**
  - A combination of organic and embedded mode characteristics
  - Teams will have intermediate level of expertise
  - Generally extends to 300 KDSI
  - Needs some innovative architectures and algorithms
  - Examples, global airlines reservation systems.

COCOMO Model

- **Basic COCOMO model**
  - Applicable to large majority of projects
  - Familiar in-house software development environment
  - Good for quick, early rough order of magnitude of costs
  - Estimates the effort using single predictor (KDSI)
- **Intermediate COCOMO model**
  - Incorporate 15 additional predictor variables
  - Cost drivers are same for the entire project lifecycle
  - Includes subjective assessments of product, process, tools **and personnel attributes**
- **Detailed/Advanced COCOMO model**
  - Estimates at module, sub-system and system level
  - Cost drivers which tend to vary at lowest level (analysis, design, coding and so on)

Terms Used in COCOMO

- The main cost driver is Delivered Source Instructions (DSI)
    - **DSI** excludes non-deliverable support such as test drivers, test cases, tools used
    - **DSI** excludes comments, but includes command language, format statements, data declarations
    - **1 KDSI** is one thousand delivered source instructions
- Excludes user training, installation planning, covers only direct charged labour
- COCOMO person-month/man-month (**MM**)consists of
    - **Man-Month (MM):** 152 hours of working time per month consistent with holidays, vacation, and sick leave
    - **Man-da**y : MM divided by 19 or 22
    - **Man-years**: MM multiplied by 12
    - **TDEV**: Development time in calendar month.
- COCOMO assumes that requirements specification is not changed significantly
- COCOMO excludes costs related to computer centre operators, personnel department, higher management etc.

Basic COCOMO Calculation: Effort and Schedule

- Provides basic effort and schedule estimates for the most **three common mode** of software development:

Mode	Effort	Schedule
Organic	$MM = 2.4(KDSI)^{1.05}$	$TDEV = 2.5(MM)^{0.38}$
Semidetached	$MM = 3.0(KDSI)^{1.12}$	$TDEV = 2.5(MM)^{0.35}$
Embedded	$MM = 3.6(KDSI)^{1.20}$	$TDEV = 2.5(MM)^{0.32}$

- **Productivity per month** = KDSI *divided by* MM
- **Average staffing** = MM *divided by* TDEV

Example: Basic COCOMO Estimates for Effort (MM)

Mode	Effort calculations
Organic	$MM = 2.4(KDSI)^{1.05}$
Semidetached	$MM = 3.0(KDSI)^{1.12}$
Embedded	$MM = 3.6(KDSI)^{1.20}$

Example of Effort (MM): We have used 5 different sizes (2, 8, 32, 128 and 512 KDSI) for each of these three modes as shown in the table below. We calculate here the MM (effort)

Effort (MM)	Small 2 KDSI	Intermediate 8 KDSI	Medium 32 KDSI	Large 128 KDSI	Very Large 512 KDSI
Organic	5.0	21.3	91	392	
Semidetached	6.5	31	146	687	3250
Embedded	8.3	44	230	1226	6420

Example: Basic COCOMO Estimates for Schedule (TDEV) in Month

Mode	Schedule
Organic	$TDEV = 2.5(MM)^{0.38}$
Semidetached	$TDEV = 2.5(MM)^{0.35}$
Embedded	$TDEV = 2.5(MM)^{0.32}$

Example of TDEV: We have used 5 different sizes (2, 8, 32, 128 and 512 KDSI) for each of these three modes as shown in the table below. We calculate here the TDEV(schedule)

Schedule (TDEV)	Small 2 KDSI	Intermediate 8 KDSI	Medium 32 KDSI	Large 128 KDSI	Very Large 512 KDSI
Organic	4.6	8	14	24	
Semidetached	4.8	8.3	14	24	42
Embedded	4.9	8.4	14	24	41

Example: Basic COCOMO Estimates for Productivity

Productivity (DSI per month by per MM) = $\frac{DSI}{MM}$

Example of productivity

Productivity	Small 2 KDSI	Intermediate 8 KDSI	Medium 32 KDSI	Large 128 KDSI	Very Large 512 KDSI
Organic	400	376	352	327	
Semidetached	308	258	219	186	158
Embedded	241	182	139	105	80

Example: Basic COCOMO Estimates for Average staffing (personnel)

$$\text{Average staffing} = \frac{MM}{TDEV}$$

Example of Staffing: Total number of staff required to finish the project within the TDEV. FSP = Full time equivalent software personnel

Schedule (TDEV)	Small 2 KDSI	Intermediate 8 KDSI	Medium 32 KDSI	Large 128 KDSI	Very Large 512 KDSI
Organic	1.1	2.7	6.5	16	
Semidetached	1.4	3.7	10	29	77
Embedded	1 .7	5.2	16	51	157

Intermediate COCOMO Calculation: Effort

- Intermediate COCOMO uses 15 cost drivers in addition to DSI.
- Intermediate COCOMO Nominal Effort estimation equations are shown below. The coefficients are a bit different than Basic COCOMO

Dev. Mode	*Nominal Effort estimate*
Organic	$MM = 3.2(KDSI)^{1.05}$
Semidetached	$MM = 3.0(KDSI)^{1.12}$
Embedded	$MM = 2.8(KDSI)^{1.20}$

Effort estimation = Nominal effort estimate * Effort adjustment factor

Note: *Schedule, staffing, productivity calculations are same as Basic COCOMO*

Intermediate COCOMO: Effort Adjustment Factors

Considers 15 additional cost drivers (CD) in addition to DSI for Effort adjustment

- **Product attributes**
  - RELY - Required SW reliability
  - DATA - Database size
  - CPLX - Product complexity
- **Computer attributes**
  - TIME - Execution time constraint
  - STOR - Main storage constraint
  - VIRT - Virtual machine volatility
  - TURN - Computer turnaround time

- **Personnel attributes**
  - ACAP - Analyst capability
  - AEXP - Applications experience
  - PCAP - Programmer capability
  - VEXP - Virtual machine experience
  - LEXP - Programming language exp.

- **Project attributes**
  - MODP – Application of modern methods
  - Tool - Use of software tools
  - SCED- Required dev. schedule

Ratings of Effort Multipliers (cost drivers): Constant Values

Code	Cost Drivers	Ratings					
		Very Low	Low	Nominal	High	Very High	Extra High
	Product attributes						
RELY	Required software reliability	0.75	0.88	1.00	1.15	1.40	
DATA	Size of application database		0.94	1.00	1.08	1.16	
CPLX	Complexity of the product	0.70	0.85	1.00	1.15	1.30	1.65
	Hardware attributes						
TIME	Run-time performance constraints			1.00	1.11	1.30	1.66
STOR	Memory constraints			1.00	1.06	1.21	1.56
VIRT	Volatility of the virtual machine environment		0.87	1.00	1.15	1.30	
TURN	Required turnabout time		0.87	1.00	1.07	1.15	
	Personnel attributes						
ACAP	Analyst capability	1.46	1.19	1.00	0.86	0.71	
AEXP	Applications experience	1.29	1.13	1.00	0.91	0.82	
PCAP	Software engineer capability	1.42	1.17	1.00	0.86	0.70	
VEXP	Virtual machine experience	1.21	1.10	1.00	0.90		
LEXP	Programming language experience	1.14	1.07	1.00	0.95		
	Project attributes						
MODP	Application of software engineering methods	1.24	1.10	1.00	0.91	0.82	
TOOL	Use of software tools	1.24	1.10	1.00	0.91	0.83	
SCED	Required development schedule	1.23	1.08	1.00	1.04	1.10	

Calculating Ratings of Effort Multipliers: Some Guidelines

- **RELY** (required software reliability)
  - If the system involves recoverable loss if it does not work properly, the rating is nominal
  - If the system involves financial loss if it does not work properly, the rating is high
  - If the system involves risk to human life if it does not work properly, the rating is very high
- **DATA** (size of application databases)
  - If the database is very large, the effort rating is very high
- **CPLX** (complexity of the software)
  - If the system is moderate complex, the rating is nominal
- **MODP** (application of modern software engineering method)
  - If the system requires to use very sophisticated SW engineering method, the rating is very high
  - If the system requires to use some degree sophisticated SW engineering method, the rating is high
  - If the system requires to use any software engineering approach, the rating is nominal
- **TOOL** (Use of software tools
  - If the system requires to use very advanced SW tools, the rating is very high
  - If the system requires to use some advanced SW tool, the rating is high
  - If the system requires to use any SW tool, the rating is nominal

Intermediate COCOMO: An Example

- Example: a system with estimated 10 KDSI, Embedded mode
- Effort Multipliers (Cost drivers):

Cost Drivers	*Rating*	*Effort multiplier*
RELY	Nominal	1.00
DATA	Low	0.94
CPLX .	Very High	1.30
TIME	High	1.11
STOR	High	1.06
VIRT	Nominal	1.00
TURN	Nominal	1.00
ACAP	High	0.86
AEXP	Nominal	1.00
PCAP	High	0.86
VEXP	Low	1.10
LEXP	Nominal	1.00
MODP	High	0.91
TOOL	Low	1.10
SCED	Nominal	1.00
		1.17

- *These values come from the effort multipliers table (see previous slide)*
- *We then multiplied all these to each other (not average), and we get 1.17.*

- Nominal Effort = $2.8(10)^{1.2} = 44MM$
- Effort = 44MM * 1.17 = 51MM
- Cost = 51MM * ($6000 per MM) = $306,000

Another Example: Intermediate COCOMO Calculation

- Required: A software system for an office automation project. Project mode is organic.
- Following 4 modules in the system to implement:
  - data entry (0.6 KDSI)
  - data update (0.6 KDSI)
  - query (0.8 KDSI)
  - report generator (1.0 KDSI)

Total System SIZE = 3.0 KDSI

Efforts multipliers of the following 4 cost drivers are rated as follows (all other cost drivers are nominal, therefore, the value is 1.0):

- CPLX - Complexity : high (1.15)
- STOR - Storage: high (1.06)
- AEXP - Applications Experience: low (1.13)
- PCAP - Programmer capabilities: low (1.17)

Now we calculate the MM (effort):

MM = (1.15*1.06*1.13*1.17 effort multipliers) * 3.2 * (3.0 KDSI) $^{1.05}$

MM = 1.61* 3.2*3.17

MM = 16.33

Cost of the project = 16.33 x cost per MM

TDEV = 2.5*16.33 $^{0.38}$ = 7.23 (>7months to complete)

How many people should be hired? MM/TDEV = team members

Average staffing required= 16.33 / 7.23 = 2.26 (>2 team members)

Estimation Accuracy

- The size of a software system can only be known accurately when it is finished.
- Several factors influence the final size
  - Use of COTS and components;
  - Programming language;
  - Distribution of system.
- As the development process progresses, the size estimate gradually becomes more accurate.
- The estimates are subjective and vary according to the judgment of the estimator.

Causes of Inaccurate Estimates

- **Key causes**
  - Frequent request for change by users
  - Overlooked tasks
  - User's lack of understanding of the requirements
  - Insufficient analysis when developing estimates
  - Lack of coordination of system development, technical services, operations, data administration, and other functions during development
  - Lack of an adequate method or guidelines for estimating
  - Complexity of the proposed application system
  - Required integration with existing system

- Complexity of the program in the system
- Size of the system expressed as number of functions or programs
- Capabilities of the project team members
- Project team's experience with the application, the programming language, and hardware
- Capabilities of the project team members
- Database management system
- Number of project team member
- Extent of programming and documentation standards.

Estimating the Costs of Software Maintenance

- To change or enhance a system is sometimes determined by the costs and the benefits of the maintenance tasks
- Situations - no choice but to change the system:
  - New law requires changes
  - Operating environment is altered
  - Immediate closing of serious faults or defects
- Expected benefit is something the user has to come up with
  - At best, it should be a monetary value
  - At worst, a relative rating to other desired changes
- Maintenance estimation must be a quick and cheap method, no time for a long analysis

Approaches to Estimating Software Maintenance Costs

- Basic equation for computing Maintenance(modification) Efforts and costs:

  Maintenance (modification) Effort:= (((KCSI/KSSI) *MM(KSSI)) * CD) *CF

  KCSI = **Changed** (***new code + deleted code + modified code***) source LoC

  KSSI = Shipped source LoC of the running (current) software,

  MM = effort for the running (current) KSSI to develop,

  CD = Applicable 15 Cost drivers (CD) for the development of the changed code (KCSI)

  CF (complexity Factor) = Additional complexity (**MODP and RELY**) for the change. (See next slide for the values)

  After the maintenance (modification) process, the total size of the entire modified system can be calculated as

  Total KDSI = KSSI + KCSI – (deleted LoC + changed LoC)

CF (Complexity Factors) for Maintenance Efforts

- Two cost driver attributes are different for maintenance than for development: RELY and MODP

- For MODP, 3 categories.

		very low	low	Nominal	High	Very High
RELY		1.35	1.15	1.00	0.98	1.10
MODP	512K	1.45	1.20	1.00	0.84	0.70
	128K	1.40	1.18	1.00	0.85	0.72
	2K	1.25	1.12	1.00	0.90	0.81

Software Acquisitions

- **Customizing software**: Software may be acquired and then modified to customise the needs and adapted (reuse, adaptation)
- **In-house software development**: Software may be developed from the scratch in the organisation
- **Custom-build**: Software may be custom-build by an outside contractor to meet the purchaser's specification (contract)

Example: Modification of Software (customization)

- Organizations can buy ready-made software from the market
- Then they can change the software to customize for the organizational needs.
- For example:

    - A ready-made software costs QR. 2 million from the market
    - The original software has 10KDSI (KSSI)
    - The organization needs to change 3KDSI (KCSI) of 10KDSI (KSSI)

    - 3 KCSI = new 2KDSI, 0.7 deleted LoC, 0.3 modified LoC
    - The cost could be calculated as organic Intermediate COCOMO as:
        - Maintenance MM: (3/10) * (3.2(10)$^{1.05}$) * 1.10 (CD) * 1.12 (CF)
    - After the modification, the software will have the following size:
        - 12 KDSI = (10KDSI + 3KCSI) – (0.7 (deleted) + 0.3 (modified))

- CD (based on 15 cost drivers)
- CF is based on RELY (nominal = 1.00) and MODP is low (1.12)

- MODP = Application of modern SW Engineering/programming methods
- RELY = Required software reliability

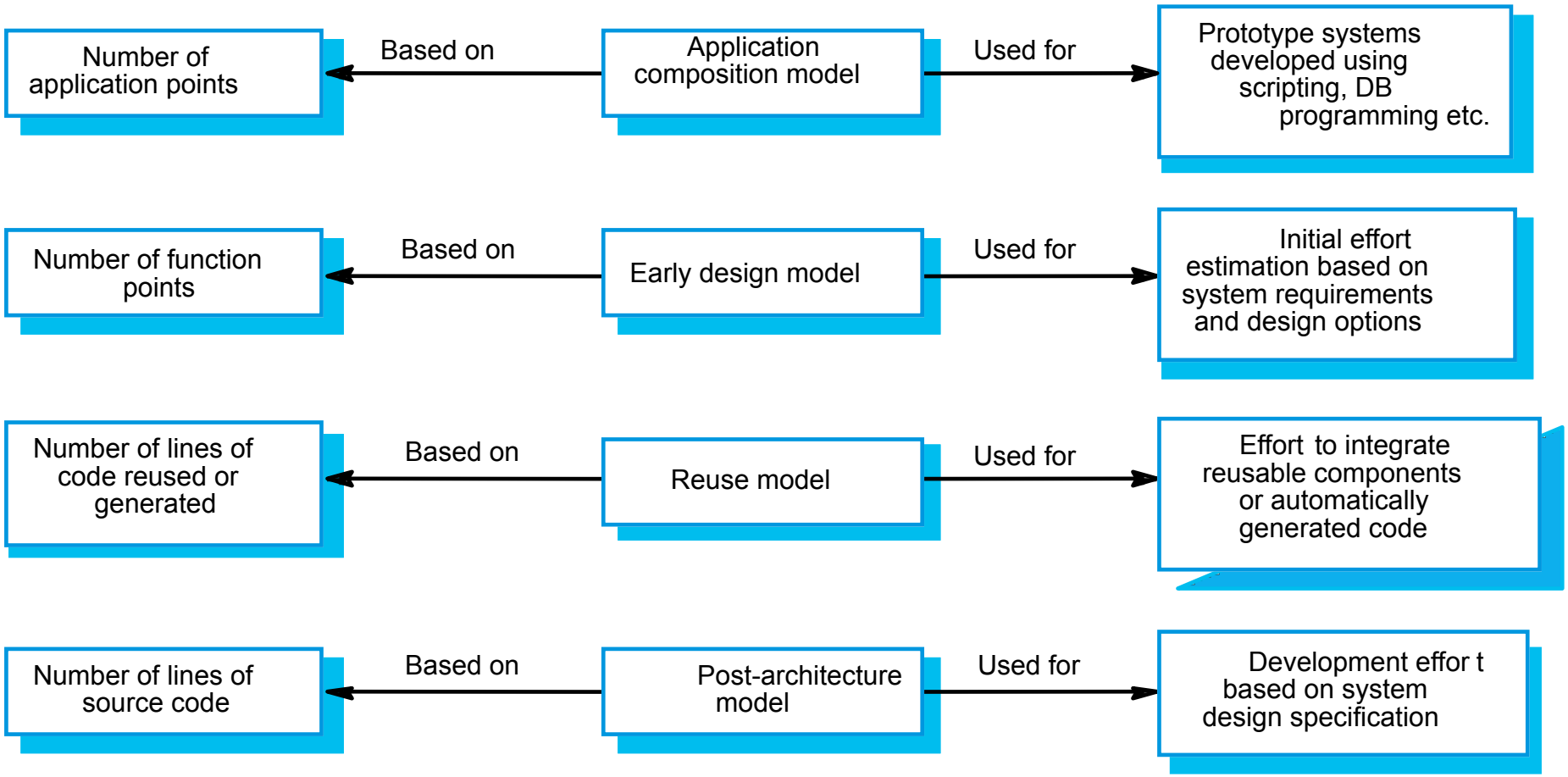
Example: Estimation for Annual Change Traffic

- Annual Change traffic of LoC:
  - Percentage of LoC changed/deleted/added to the running software annually.
- Example:
  - An organic mode software of size 10KDSI will change 30% annually. Of this changed traffic, 17% is the new code, 10% is the deleted code, and 3% is the modified code. CF and CD are constant 1.
    - What would be the total size of the software after 2 years?
    - What would be the MM after 2 years to modify these changes?
  - After Year 1, the size of the software would be:
    - 3 KCSI: (1700 + 1000 + 300) LoC
    - **MM** = $(3.2(10)^{1.05})$
    - $MM^{\text{year 1}} = (((3KCSI/10KDSI) * \mathbf{MM}) * 1 * 1)$
    - New KDSI after Year 1, : **11.7 (after modification in year 1)** = (10KDSI + 3KDSI) – (1,000 LoC + 300 LoC)
  - After year 2, the total size of the software would be:
    - 3.51 KCSI: (1989 + 1170 + 351) LoC
    - **MM** = $(3.2(11.7)^{1.05})$
    - $MM^{\text{year 2}} = (((3.51KCSI/11.7KDSI) * \mathbf{MM}) * 1 * 1)$
    - New KDSI after Year 2: **13.69 (after modification in year 2)** = (11.7KDSI + 3.51KDSI) - (1,170 LoC + 351 LoC)

COCOMOII (COCOMO 2)

- COCOMO 81 was developed with the assumption that a waterfall process would be used and that all software would be developed from scratch.
- Since its formulation, there have been many changes in software engineering practice and COCOMO II is designed to accommodate different approaches to software development.

Use of COCOMO 2 models



COCOMO 2 models

- COCOMO 2 incorporates a range of sub-models that produce increasingly detailed software estimates.
- An empirical model based on project experience.
- Well-documented, 'independent' model which is not tied to a specific software vendor.
- COCOMO 2 takes into account different approaches to software development, reuse, etc.
- The sub-models in COCOMO 2 are:
  - Application composition model. Used when software is composed from existing parts.
  - Early design model. Used when requirements are available but design has not yet started.
  - Reuse model. Used to compute the effort of integrating reusable components.
  - Post-architecture model. Used once the system architecture has been designed and more information about the system is available.

Differences Between COCOMO and COCOMO II

- ✧ Three modes are replaced by five Scale Factors (SF) in COCOMO II
- ✧ Requirements Volatility in COCOMo81 cost factor is replaced by Breakage (BRAK) in COCOMO II
- ✧ Added 7 new cost drivers: DOCU, RUSE, PVOL, PEXP, LTEX, PCON, SITE
- ✧ Removed these 5 cost drivers: VIRT, TURN, VEXP, LEXP, MODP

- COCOMO II is too complex for use without adequate training or tool with online-help
- Managers still prefer COCOMO81 over COCOMO II due to the simplicity of COCOMO81
- This course does not have much scope to go details of COCOMO II.

Problems with Lines of Code

- LOC (lines of code) used in programming languages is one of the input for estimation
- Problem with LOC: a statement can be spread out over several lines
- Problems with comments and data definitions
- Problems with general statements e.g., print, read.
- Difficult to estimate accurate LOC.
- Different programming languages may need different LOC for the implementation of the same program

References

- Bohem, B. W.:*Software Engineering Economics*. Prentice-Hall.
- Pressman, R.: Software Engineering - A practitioner's approach, McGraw Hill.
- http://en.wikipedia.org/wiki/COCOMO
- Function point calculator:
  http://groups.engin.umd.umich.edu/CIS/course.des/cis525/js/f00/artan/functionpoints.htm
- Pressman, R.: Software Engineering - A practitioner's approach, McGraw Hill.,
- Fenton, N. E., Pfleeger, S.L.: Software Metrics - A Rigorous and Practical Approach, International Thomson Computer Press
- Bohem, B. W.:*Software Engineering Economics*. Prentice-Hall,

Tools for COCOMO II (self prctice):

- http://www.softstarsystems.com/demo.htm
- http://csse.usc.edu/tools/COCOMOSuite.php