Intro. To Software Engineering

SE-110

Software Cost Estimation

Lecture # 47
19 May

Rubab Jaffar
rubab.jaffar@nu.edu.pk

Today's Outline

- Fundamental Estimation Questions
- Fundamentals of software costing and pricing
- Software productivity
- Productivity Measures

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?
- Project estimation and scheduling are interleaved management activities.

Software Cost Components

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

Costing and Pricing

- Estimates are made to discover the cost, to the developer, of producing a software system.
- There is not a simple relationship between the development cost and the price charged to the customer.

Software Productivity

- A measure of the rate at which individual engineers involved in software development produce software and associated documentation.
- Not quality-oriented although quality assurance is a factor in productivity assessment.
- Essentially, we want to measure useful functionality produced per time unit.

Productivity Measures

- **Size related measures** based on some output from the software process. This may be lines of delivered source code, object code instructions, **etc.**

- **Function-related measures** based on an estimate of the functionality of the delivered software. Function-points are the best known of this type of measure.

Productivity Comparisons

- The lower level the language, the more productive the programmer
  - The same functionality takes more code to implement in a lower-level language than in a high-level language.
- The more verbose the programmer, the higher the productivity
  - Measures of productivity based on lines of code suggest that programmers who write verbose code are more productive than programmers who write compact code.

Function Points

- Based on a combination of program characteristics
  - external inputs and outputs;
  - user interactions;
  - external interfaces;
  - files used by the system.
- A weight is associated with each of these and the function point count is computed by multiplying each raw count by the weight and summing all values.

$$UFC = \sum(\text{number of elements of given type}) \times (\text{weight})$$

FP Information Domain

- Number of external inputs (EIs). Each external input originates from a user and provides distinct application-oriented data or control information. Inputs are often used to update internal logical files (ILFs). Inputs should be distinguished from inquiries, which are counted separately.

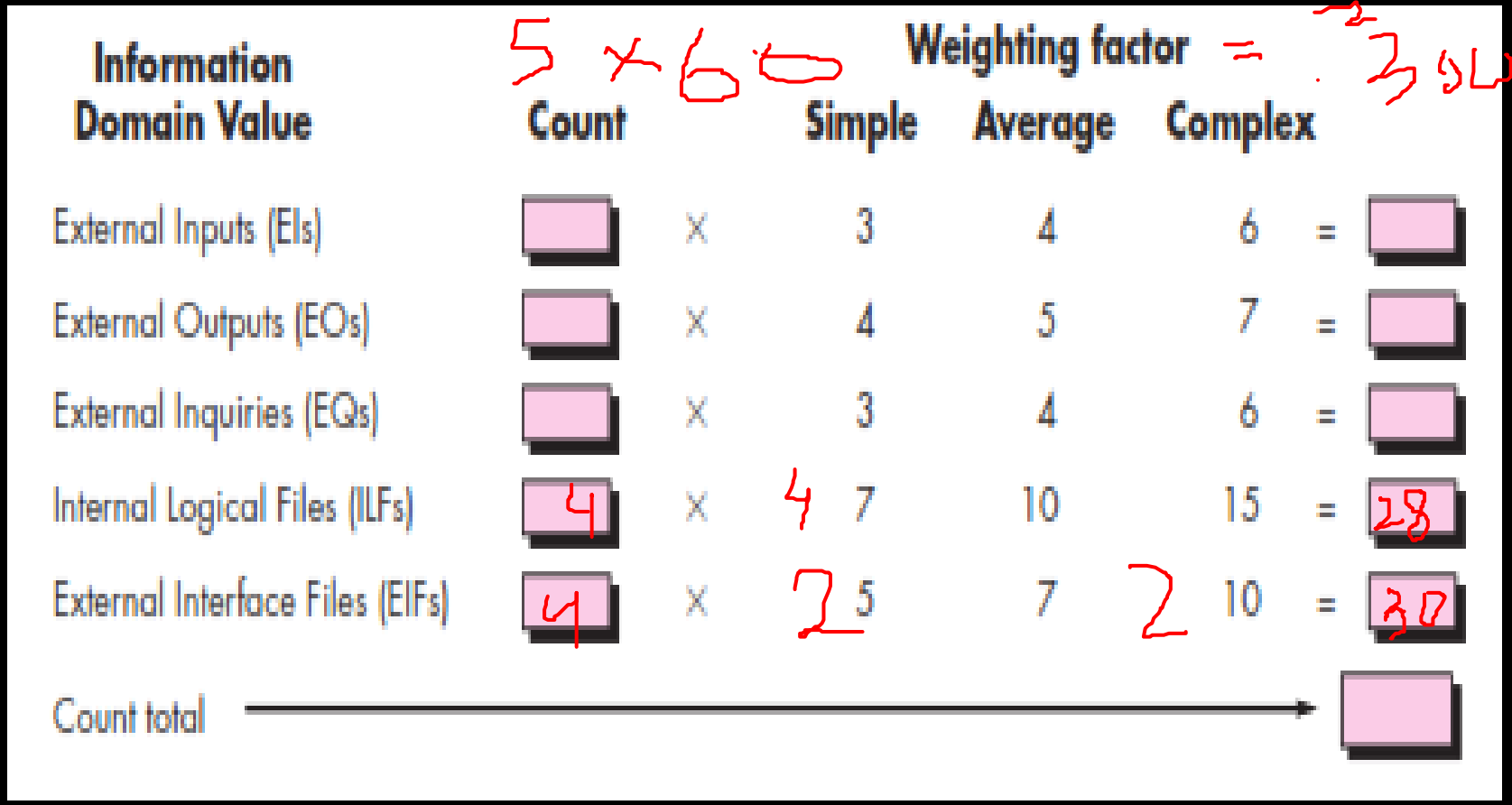
- Number of external outputs (EOs). Each external output is derived data within the application that provides information to the user.

- Number of external inquiries (EQs). An external inquiry is defined as an online input that results in the generation of some immediate software response in the form of an online output (often retrieved from an ILF).

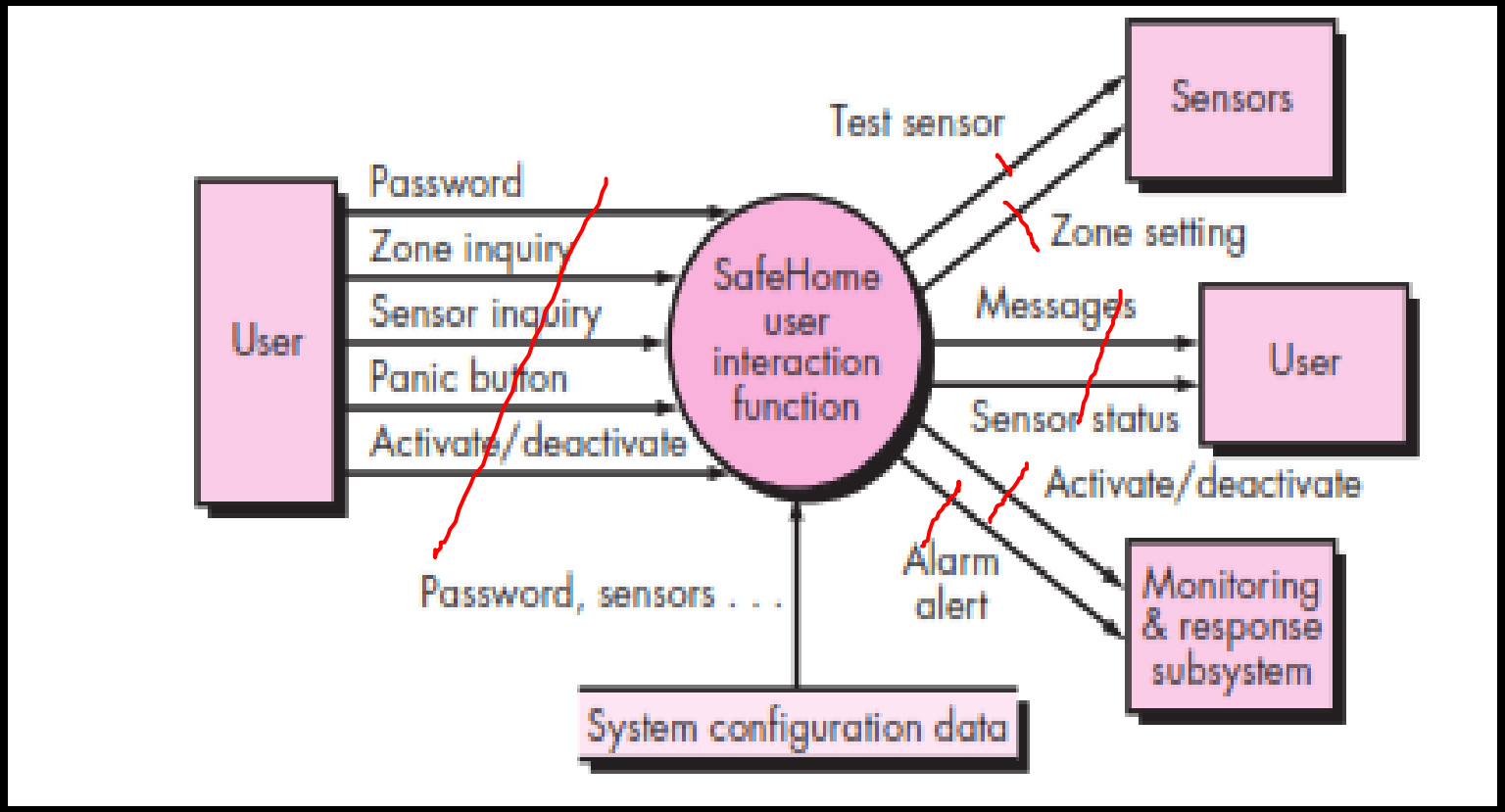
- Number of internal logical files (ILFs). Each internal logical file is a logical grouping of data that resides within the application's boundary.

- Number of external interface files (EIFs). Each external interface file is a logical grouping of data that resides external to the application but provides information that may be of use to the application.

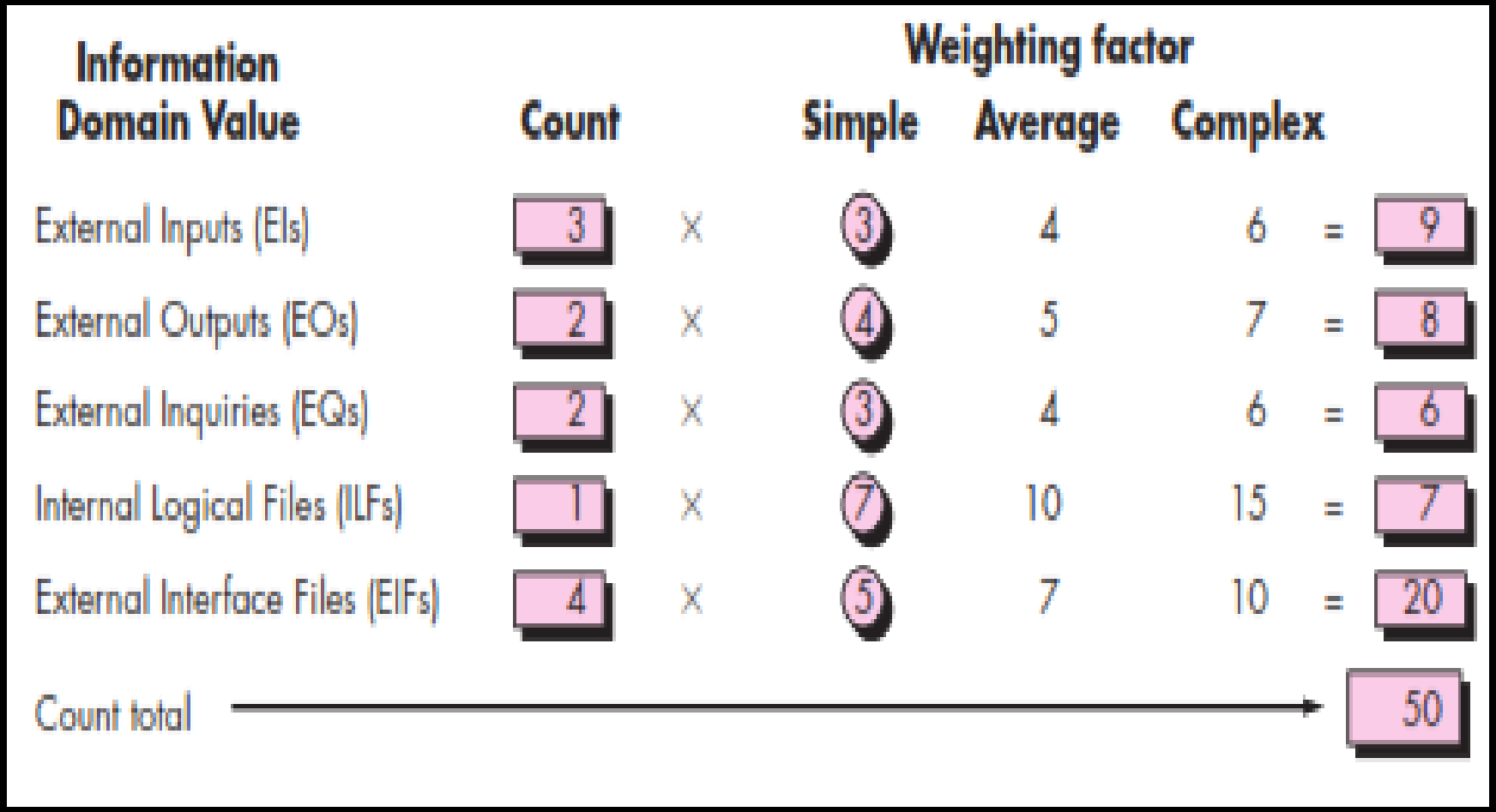
Computing FPs



Example; DFD for computing FP



Computing FP



Information Domain Value	Count		Weighting factor: Simple	Average	Complex		
External Inputs (EIs)	3	x	3	4	6	=	9
External Outputs (EOs)	2	x	4	5	7	=	8
External Inquiries (EQs)	2	x	3	4	6	=	6
Internal Logical Files (ILFs)	1	x	7	10	15	=	7
External Interface Files (EIFs)	4	x	5	7	10	=	20
Count total →							50

Function Points

- The function point count is modified by complexity of the project
- FPs are very subjective. They depend on the estimator
  - Automatic function-point counting is impossible.

Object Points

- Object points (alternatively named application points) are an alternative function-related measure to function points when 4GLs or similar languages are used for development.
- Object points are NOT the same as object classes.
- The number of object points in a program is a weighted estimate of
  - The number of separate screens that are displayed;
  - The number of reports that are produced by the system;
  - The number of program modules that must be developed to supplement the database code;

Object Point Estimation

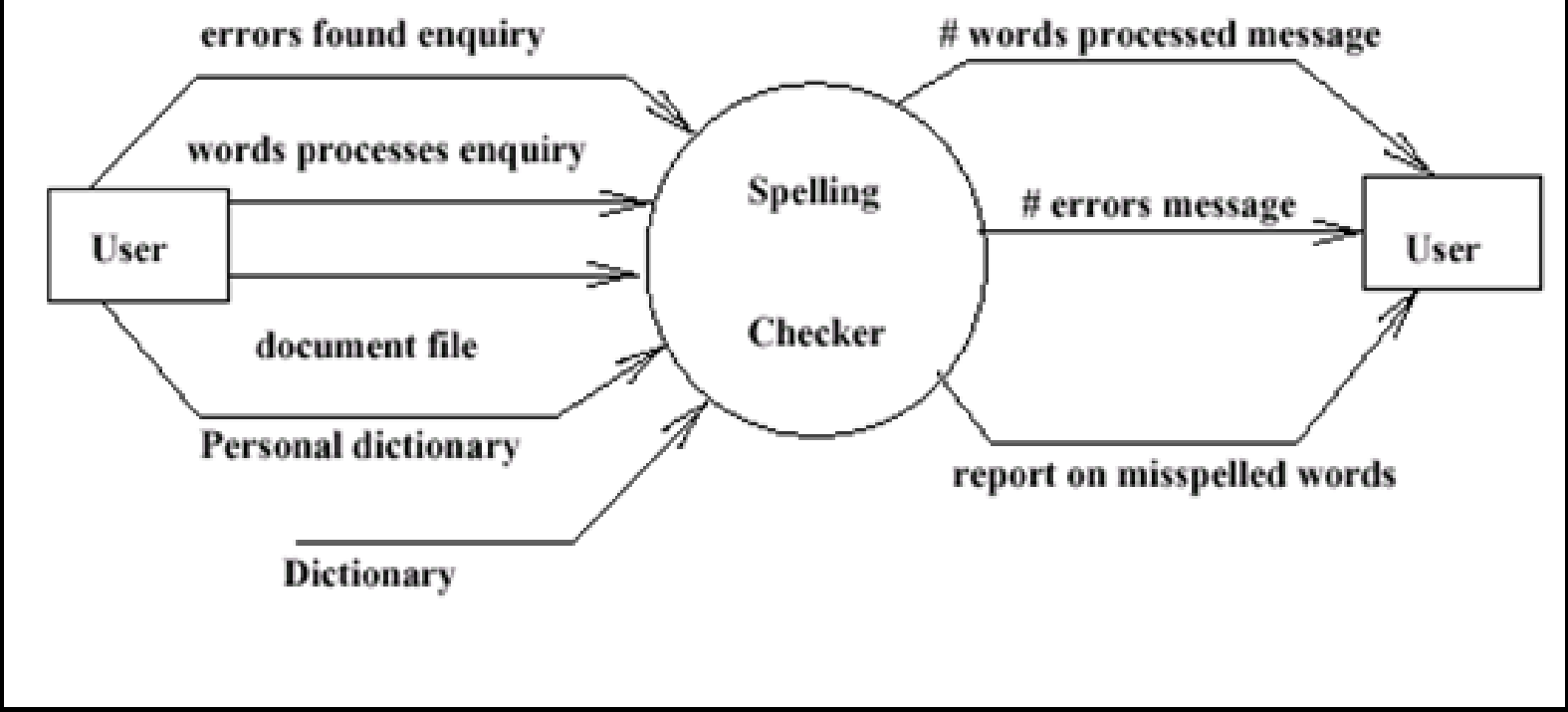
- Object points are easier to estimate from a specification than function points as they are simply concerned with screens, reports and programming language modules.
- They can therefore be estimated at a fairly early point in the development process.
- At this stage, it is very difficult to estimate the number of lines of code in a system.

Reconciling LOC and FP Metrics /Productivity Estimates

- FPs can be used to estimate LOC depending on the average number of LOC per FP for a given language
  - LOC = AVC * number of function points;
  - AVC is a language-dependent factor varying from 200-300 for assemble language to 2-40 for a 4GL;
  - The relationship between lines of code and function points depends upon the programming language that is used to implement the software and the quality of the design.
  - A number of studies have attempted to relate FP and LOC measures. The following table 4 [QSM02] provides rough estimates of the average number of lines of code required to build one function point in various programming languages:

Programming Language	LOC per Function Point			
	Average	Median	Low	High
Access	35	38	15	47
Ada	154	—	104	205
APS	86	83	20	184
ASP 69	62	—	32	127
Assembler	337	315	91	694
C	162	109	33	704
C++	66	53	29	178
Clipper	38	39	27	70
COBOL	77	77	14	400
CoolGen/IEF	38	31	10	180
Culprit	51	—	—	—
DBase IV	52	—	—	—
Easytrieve+	33	34	25	41
Excel47	48	—	31	63
Focus	43	42	32	56
FORTRAN	—	—	—	—
FoxPro	32	35	25	35
Ideal	66	52	34	203
IEF/CoolGen	38	31	10	180
Informix	42	31	24	57
Java	63	53	77	—
JavaScript	58	63	42	75
JCL	91	123	26	130
JSP	59	—	—	—
Lotus Notes	21	22	15	25
Mantis	71	27	22	250
Mapper	118	81	16	245
Natural	60	52	22	141
Oracle	30	35	4	217
PeopleSoft	33	32	30	40
Perl	60	—	—	—
PL/1	78	67	22	263
Powerbuilder	32	31	11	105
REXX	67	—	—	—
RPG II/III	61	49	24	155
SAS	40	41	33	49
Smalltalk	26	19	10	55
SQL	40	37	7	110
VBScript30	34	27	30	—
Visual Basic	47	42	16	158

Example: Calculate the function points for the following system.
The of Spell-Checker accepts as input a document file and an optional personal dictionary file. The checker lists all words not contained in either of these files. The user can query the number of words processed and the number spelling errors found at any stage during processing.



The weight factor is given by following table

Description	Low	Medium	High	Total
Inputs	__x 3	__x 4	__x 6	____
Outputs	__x 4	__x 5	__x 7	____
Queries	__x 3	__x 4	__x 6	____
Files	__x 7	__x 10	__x 15	____
Program Interfaces	__x 5	__x 7	__x 10	____

Solution

- 2 users inputs: document file name, personal dictionary name (average)
- 3 users outputs: fault report, word count, misspelled error count (average)
- 2 users requests: #treated words?, #found errors? (average)
- 1 internal file: dictionary (average)
- 2 external files: document file, personal dictionary (av).

$$UFP = 4 \times 2 + 5 \times 3 + 4 \times 2 + 10 \times 1 + 7 \times 2 = 55$$

That is all