Measuring Internal Product Attributes

Size

Internal Product Attributes

- In software engineering, internal product attributes refer to the inherent characteristics of the software codebase and architecture—those aspects that are not directly visible to the end user but impact maintainability, reliability, efficiency, and quality.

Software Size

- Size measures only indicate how much of an entity we have.
- Size alone cannot directly indicate external attributes such as effort, productivity, and cost
- It doesn’t measure the external attributes like “coding difficulties”, however, it is a good measure to predict software development time and resources. e.g.,
  - Productivity=size/effort
  - Defect density = Defect count/size

Properties of Valid Software Size Measurement

- Three properties for any valid measure of software size:
  - Nonnegativity: All systems have nonnegative size.
  - Null value: The size of a system with no elements is zero.
  - Additivity: The size of the union of two modules is the sum of the sizes of the two modules after subtracting the size of the intersection

Code Size

- The most commonly used measure of source code size is the number of lines of code (LOCs).
- We must explain how each of the following is handled:
    - Blank lines
    - Comment lines
    - Data declarations
    - Lines that contain several separate instructions
- One count can be as much as five times larger than another, simply because of the difference in counting technique

LOC Definitions

- NCLOC: Non-Commented Lines of Code (Comments and blank lines removed), sometimes called **effective lines of a code**.
- CLOC: Number of comment lines of program text (CLOC)
- Total size (LOC) = NCLOC + CLOC
- Density of comments = CLOC / LOC
- Is NCLOC a valid measure?
- The number of executable statements (ES): counts separate statements on the same physical line as distinct. It ignores comment lines, data declarations, and headings

Halstead’s Approach

- Halstead’s Software Science is a theoretical approach to measuring software complexity and predicting attributes of a program based on its code
- The approach is based on quantifying the properties of a program by examining its **operators** and **operands**, aiming to provide objective measures of complexity, size, and effort.
- Halstead’s theory is based on four fundamental measures derived from the source code:
    - μ_1 = Number of unique operators
    - μ_2 = Number of unique operands
    - N_1 = Total occurrences of operators
    - N_2 = Total occurrences of operands

From these basic measures, Halstead derived several important metrics

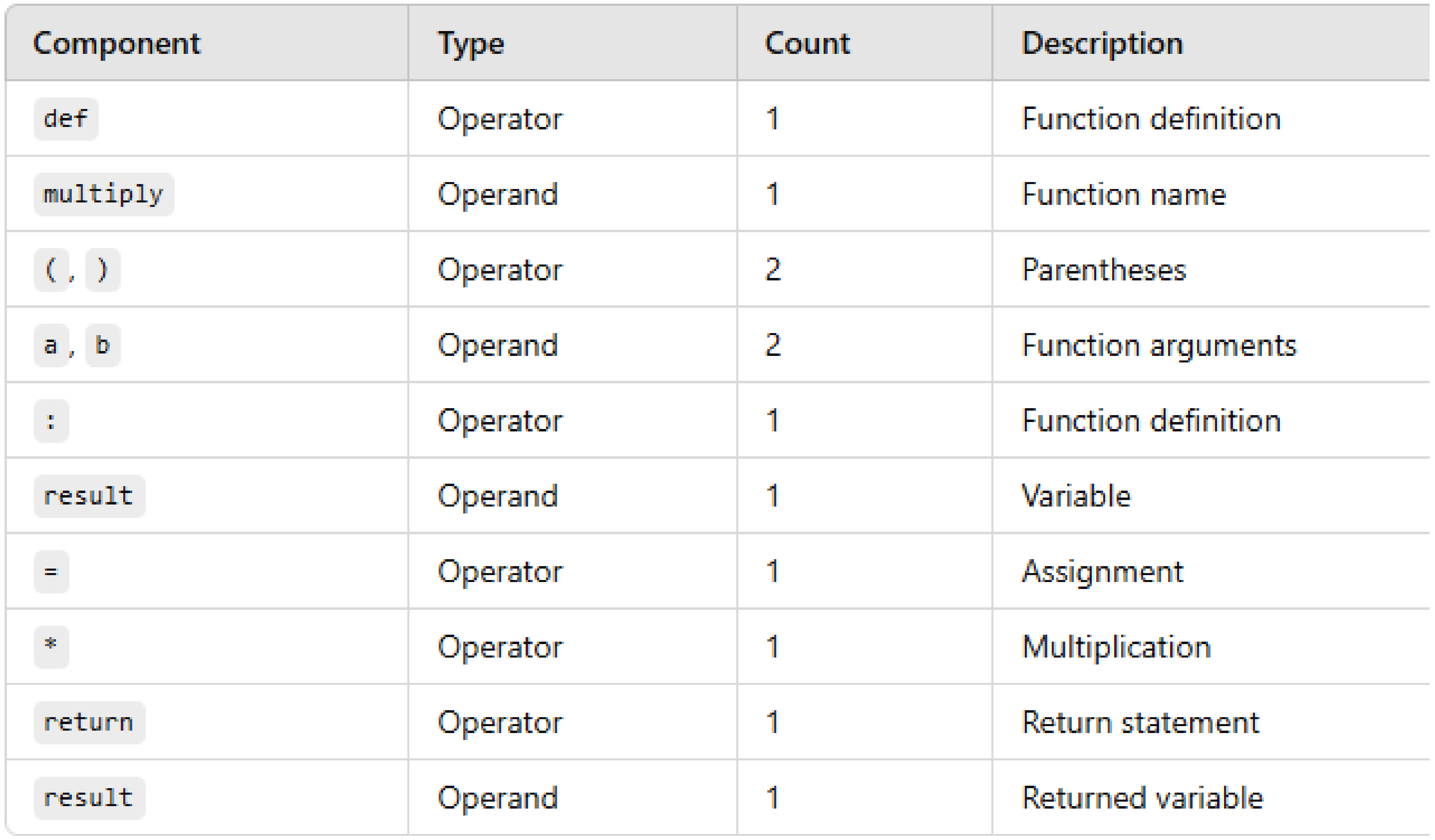
Halstead's Approach

- **Halstead Program Length:** The total number of operator occurrences and the total number of operand occurrences. N = $N_1 + N_2$

  Estimated program length is, $N^\wedge = \mu_1 \log_2 \mu_1 + \mu_2 \log_2 \mu_2$
- **Halstead Vocabulary:** $\mu = \mu_1 + \mu_2$
- **Program Volume:** V = Size * ($\log_2$ vocabulary) = N * $\log_2(\mu)$
- **Program Difficulty:** D = $(\mu_1 / 2) * (N_2 / \mu_2)$
- **Programming Effort:** E = D * V = Difficulty * Volume
- **Programming Time** (T) = E/18

Halstead’s Approach Example

```
def multiply(a, b):
    result = a * b
    return result;
```



Component	Type	Count	Description
`def`	Operator	1	Function definition
`multiply`	Operand	1	Function name
`(` , `)`	Operator	2	Parentheses
`a` , `b`	Operand	2	Function arguments
`:`	Operator	1	Function definition
`result`	Operand	1	Variable
`=`	Operator	1	Assignment
`*`	Operator	1	Multiplication
`return`	Operator	1	Return statement
`result`	Operand	1	Returned variable

Halstead's Approach Example

Metric	Value	Formula / Explanation
μ_1 (Distinct operators)	5	def, (), :, =, *, return
μ_2 (Distinct operands)	4	multiply, a, b, result
N_1 (Total operators)	7	def, (), :, =, *, return, ()
N_2 (Total operands)	5	multiply, a, b, result, result

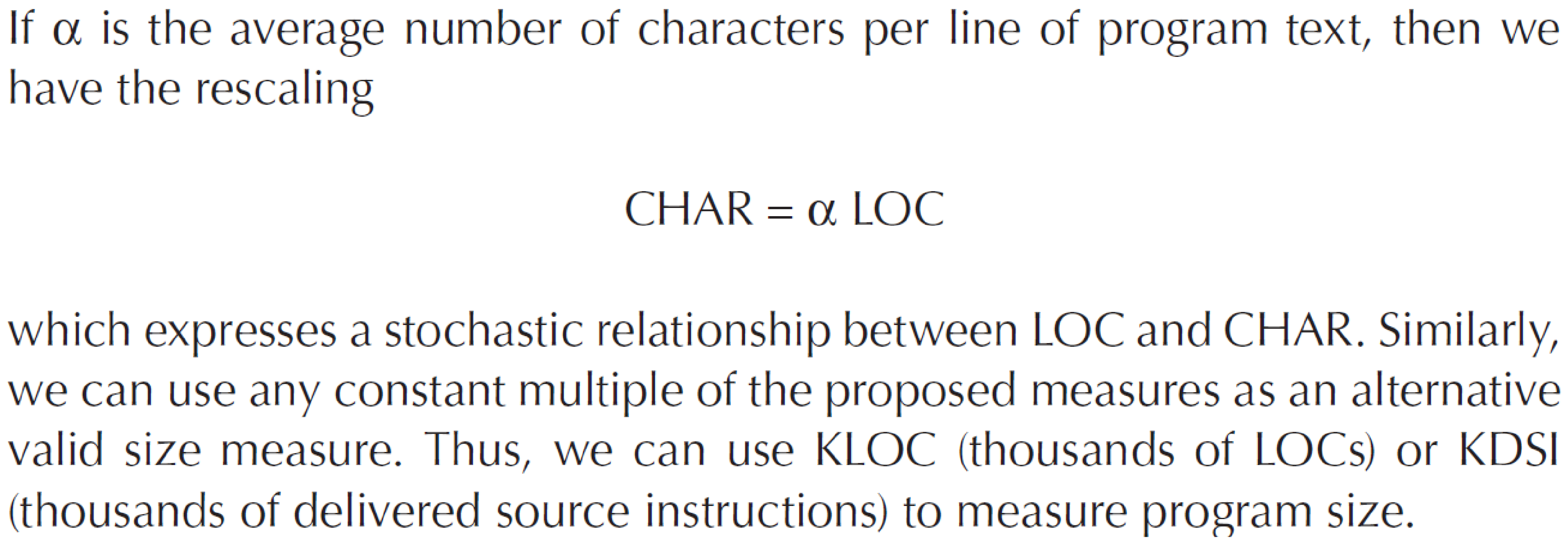
Insight

- **Low Program Difficulty (D)**: Indicates that the code is relatively easy to understand and modify.
- **Effort (E)** and **Programming Time (T)** are also low, reflecting the simplicity of the program.
- **Volume (V)** is moderate, as the program is small and straightforward.

- *Cyclomatic Complexity??*
- *Code Smells?*
- *Code Duplications?*

Alternative Size Measures

- We can measure the size in terms of
  - Number of bytes of computer storage
  - Number of characters in the program text



If α is the average number of characters per line of program text, then we have the rescaling

$$\text{CHAR} = \alpha\ \text{LOC}$$

which expresses a stochastic relationship between LOC and CHAR. Similarly, we can use any constant multiple of the proposed measures as an alternative valid size measure. Thus, we can use KLOC (thousands of LOCs) or KDSI (thousands of delivered source instructions) to measure program size.

Design Size

- Count design elements rather than LOCs
- Appropriate size measure depends on the design methodology, the artifacts developed, and the level of abstraction
- To measure the size of a procedural design, you can count the number of procedures and functions at the lowest level of abstraction
- At higher levels of abstraction, you can count the number of packages and subsystems. You can measure the size of a package or subsystem in terms of the number of functions and procedures in the package.

Design Size

- Object-oriented designs add new abstraction mechanisms: objects, classes, interfaces, operations, methods, associations, inheritance, etc.
- Thus, we will measure the size in terms of packages, design patterns, classes, interfaces, abstract classes, operations, and methods.
  - ***Packages*****:** Number of sub packages, number of classes, interfaces (Java), or abstract classes (C++)
  - ***Design patterns*****:**
    - Number of different design patterns used in a design
    - Number of design pattern realizations for each pattern type
    - Number of classes, interfaces, or abstract classes that play roles in each pattern realization
  - ***Classes, interfaces, or abstract classes*****:** Number of public methods or operations, number of attributes
  - ***Methods or operations*****:** Number of parameters, number of overloaded versions of a method or operation

Design Size

- **Weighted Methods per Class (WMC) measure:** measured by summing the weights of the methods in a class, where weights are unspecified complexity factors for each method
- Both the number of methods and the number of attributes can serve as class size measures
- One set of studies found that the number of methods is a better predictor of class change-proneness than the number of attributes

Requirement Analysis and Specification Size

- Requirement and specification documents combine text, graphs, and special mathematical diagrams and symbols.
- It may not be easy to generate a single-size measure because a requirement analysis often consists of a mix of document types.
- For example, a use case analysis may consist of a UML use case diagram along with a set of use case scenarios that may be expressed as either text or as UML activity diagrams

Cost of a Project

- The cost of a project is due to:
    - The requirements for software, hardware, and human resources
    - The cost of software development is due to the human resources needed
    - Most cost estimates are measured in ***person-months (PM)***

Software Cost Estimation

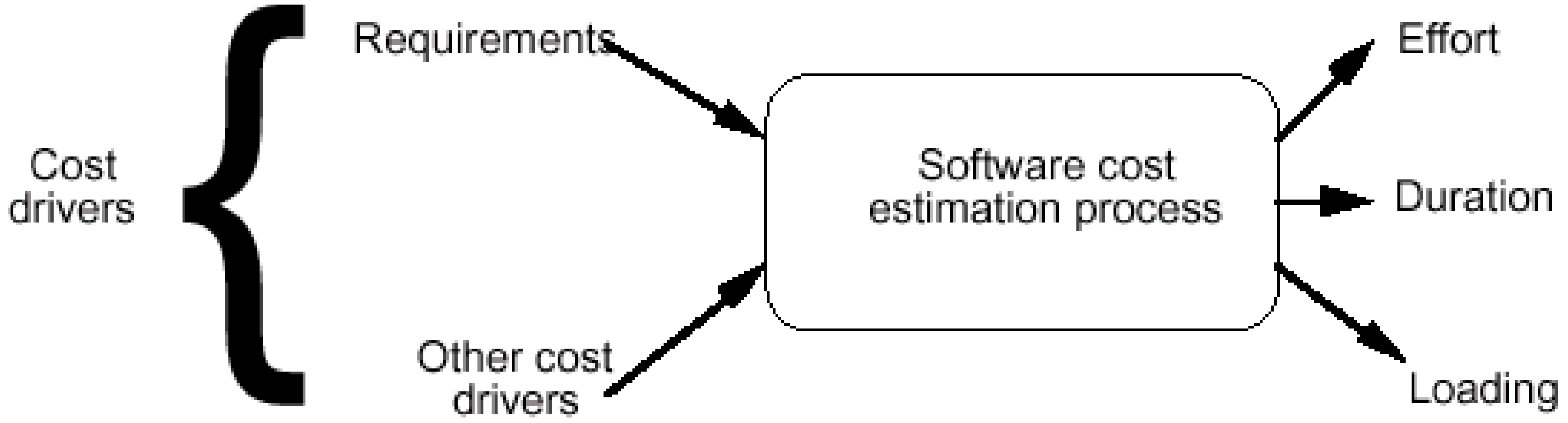
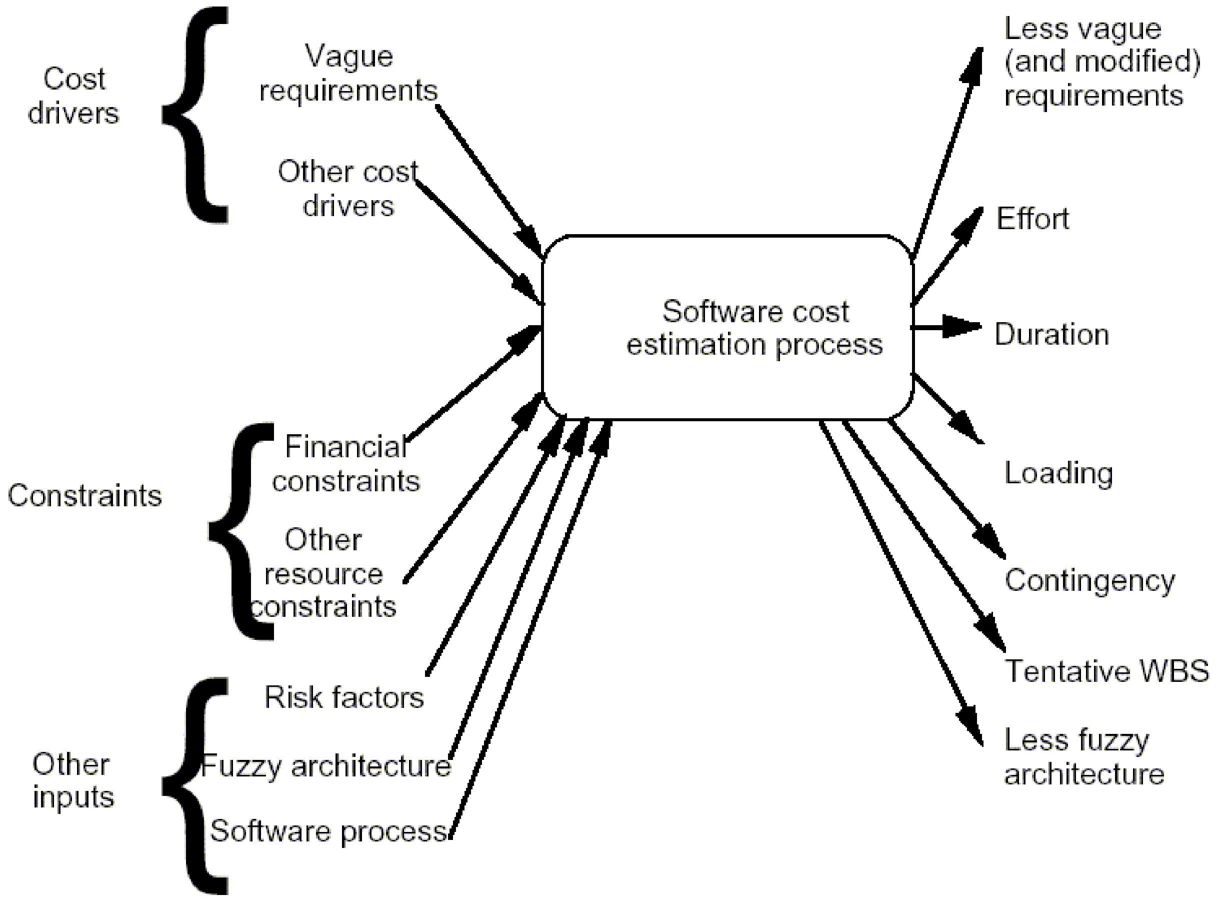


Figure 1. Classical view of software estimation process.

Actual View


Cost drivers
Vague requirements
Other cost drivers
Constraints
Financial constraints
Other resource constraints
Other inputs
Risk factors
Fuzzy architecture
Software process
Software cost estimation process
Less vague (and modified) requirements
Effort
Duration
Loading
Contingency
Tentative WBS
Less fuzzy architecture

Effort (using LOC)

- Effort Equation
  - **PM = a * $(\mathbf{KDSI})^{\mathbf{b}}$** (person-months)
    - where **PM** = number of person-month (=152 working hours),
    - **a** = a constant,
    - **KDSI** = thousands of "delivered source instructions" (DSI) and
    - **b** = a constant.

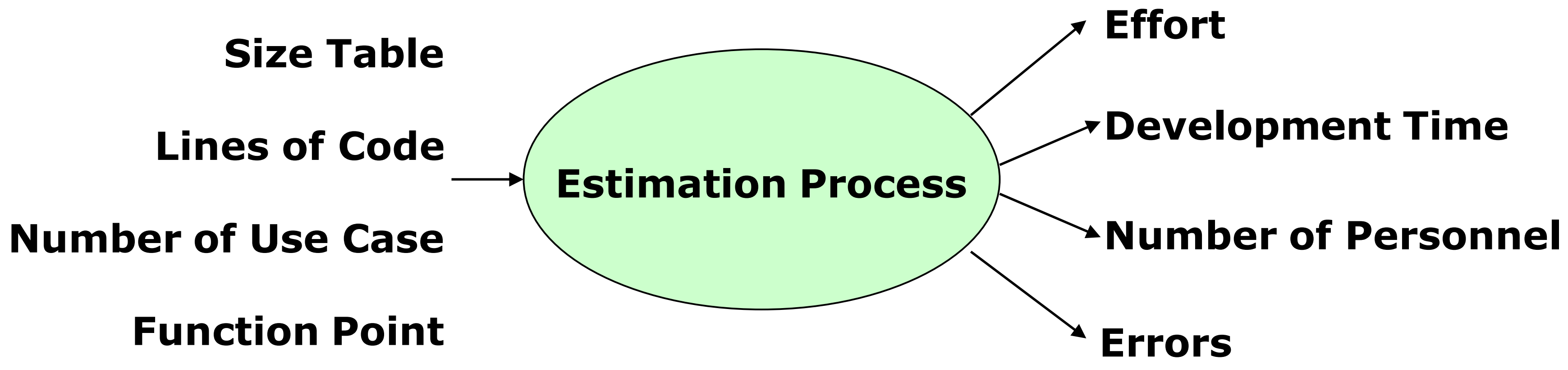
Productivity

- Productivity equation
  - **(DSI) / (PM)**
    - where **PM** = number of person-month (=152 working hours),
    - **DSI** = "delivered source instructions"

Schedule and Average Staffing

- Schedule equation
  - **TDEV = c * $(PM)^d$ (**months)
    - where TDEV = number of months estimated for software development.
- Average Staffing Equation
  - **(PM) / (TDEV)** (FSP)
    - where FSP means Full-time-equivalent Software Personnel.

Cost Estimation Process



Project Size - Metrics

1. **Number of functional requirements**
2. **Cumulative number of functional and non-functional requirements**
3. **Number of Customer Test Cases**
4. **Number of ‘typical sized’ use cases**
5. **Number of inquiries**
6. **Number of files accessed (external, internal, master)**
7. **Total number of components (subsystems, modules, procedures, routines, classes, methods)**
8. **Total number of interfaces**
9. **Number of System Integration Test Cases**
10. **Number of input and output parameters (summed over each interface)**
11. **Number of Designer Unit Test Cases**
12. **Number of decisions (if, case statements) summed over each routine or method**
13. **Lines of Code, summed over each routine or method**

Project Size - Metrics

Availability of Size Estimation Metrics:

	Development Phase	Available Metrics
a	Requirements Gathering	1, 2, 3
b	Requirements Analysis	4, 5
d	High Level Design	6, 7, 8, 9
e	Detailed Design	10, 11, 12
f	Implementation	12, 13

Function Points

- Function Point (FP) is a software size estimation technique used to measure the functionality delivered by a system, independent of the programming language, tools, or technology used.
- A function point calculates software size with the help of logical design and performance of functions as per user requirements.
- Function Point (FP) is an element of software development that helps to **approximate the cost of development early in the process**

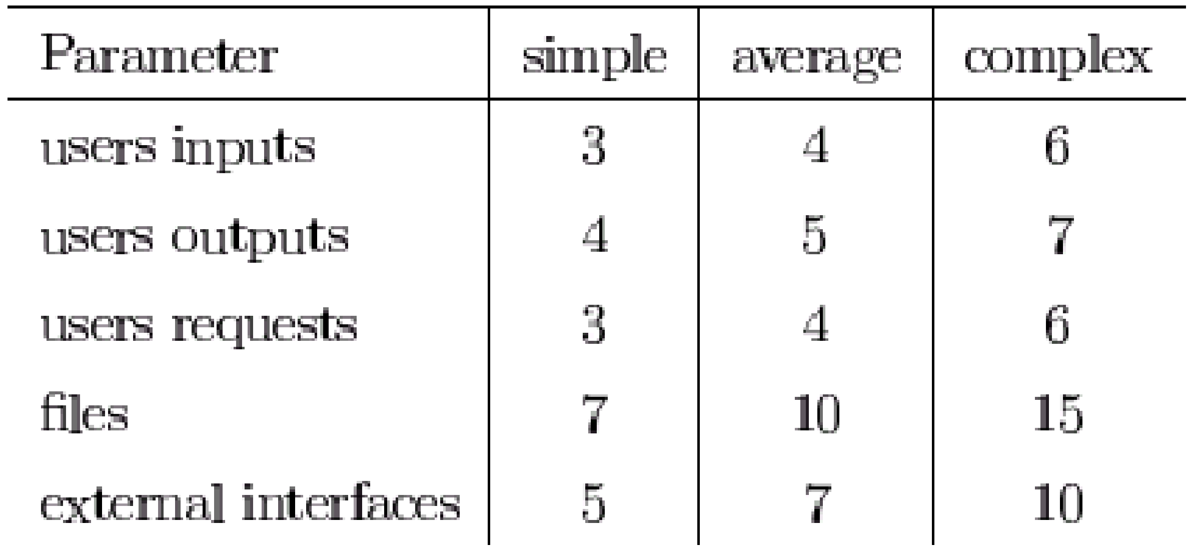
Function Points Calculation

STEP 1: Measure size in terms of the amount of functionality in a system. Function points are computed by first calculating an ***unadjusted function point count*** **(UFC).** Counts are made for the following categories

- ❑*External inputs* – those items provided by the user that describe distinct application-oriented data (such as file names and menu selections)
- ❑*External outputs* – those items provided to the user that generate distinct application-oriented data (such as reports and messages, rather than the individual components of these)
- ❑*External inquiries* – interactive inputs requiring a response
- ❑*External files* – machine-readable interfaces to other systems
- ❑*Internal files* – logical master files in the system

Function Points (cont.)

- **STEP 2:** Multiply each number by a weight factor, according to complexity (**simple**, **average** or **complex**) of the parameter, associated with that number. The value is given by a table:



Parameter	simple	average	complex
users inputs	3	4	6
users outputs	4	5	7
users requests	3	4	6
files	7	10	15
external interfaces	5	7	10

Function Points (cont.)

- **<u>STEP 3</u>:** Calculate the total **UFP** (Unadjusted Function Points)
- **<u>STEP 4</u>:** Calculate the total **TCF** (Technical Complexity Factor) by giving a value between 0 and 5 according to the importance of the following points (next slide):

TCF=0.65+0.01*DI

Function Points (cont.)

- **<u>Technical Complexity Factors:</u>**

1. Data Communication
2. Distributed Functions
3. Performance
4. Heavily Utilized Hardware
5. High Transaction Rates
6. Online Data Entry
7. Online Updating
8. End-user Efficiency
9. Complex Computations
10. Reusability
11. Ease of Installation
12. Ease of Operation
13. Portability
14. Maintainability/Facility change

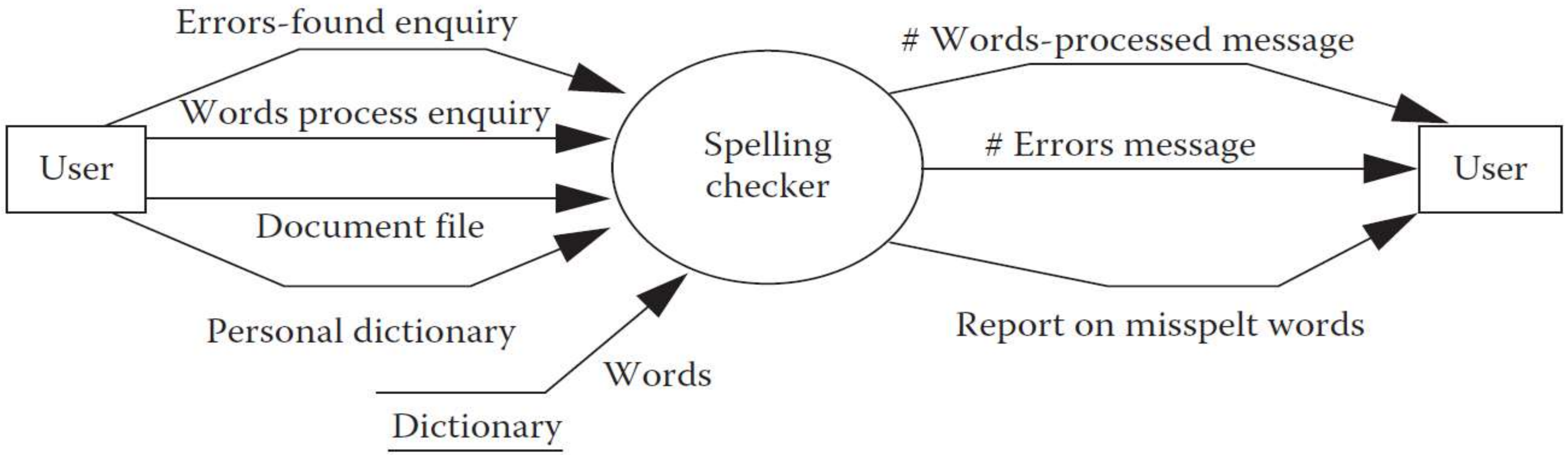
Each component or subfactor is rated from 0 to 5, where 0 means the subfactor is irrelevant, 3 means it is average, and 5 means it is essential to the system being built

Function Points (cont.)

- **STEP 5:** Sum the resulting numbers to obtain **DI** (degree of influence)
- **STEP 6: TCF** (Technical Complexity Factor) by given by the formula
  - **TCF=0.65+0.01*DI**
- **STEP 6:** Function Points are given by the formula
  - **FP=UFP*TCF**

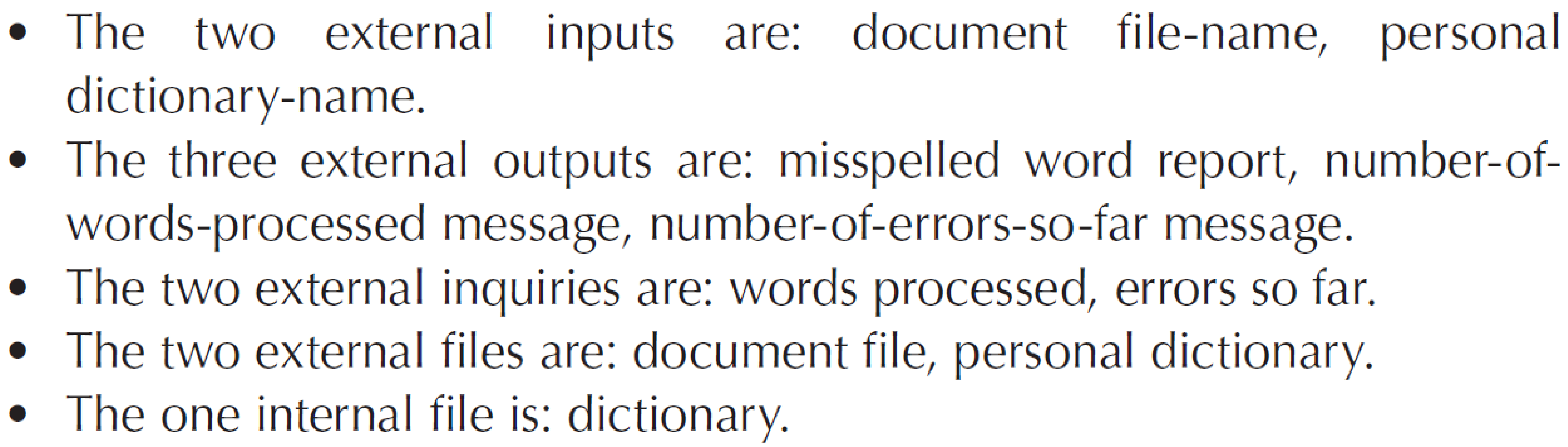
Function Points Example

Spell-checker spec: The 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 of spelling errors found at any stage during processing.



A = # external inputs = 2, B = # external outputs = 3, C = # inquiries = 2,
D = # external files = 2, and E = # internal files = 1

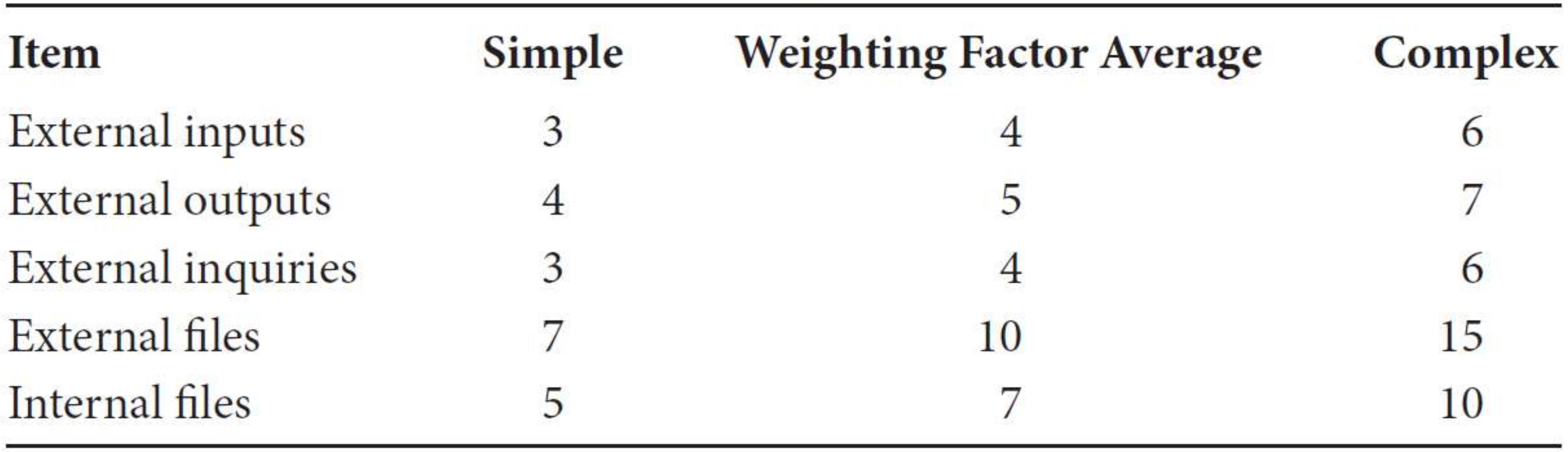
Function Points Example



- The two external inputs are: document file-name, personal dictionary-name.
- The three external outputs are: misspelled word report, number-of-words-processed message, number-of-errors-so-far message.
- The two external inquiries are: words processed, errors so far.
- The two external files are: document file, personal dictionary.
- The one internal file is: dictionary.

Function Points Example

TABLE 8.2 Function Point Complexity Weights



Item	Simple	Weighting Factor Average	Complex
External inputs	3	4	6
External outputs	4	5	7
External inquiries	3	4	6
External files	7	10	15
Internal files	5	7	10

Function Points Example

- 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).

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

Function Points Example

Technical Complexity Factors:

- *F*1. Data Communication
- *F*2. Distributed Data Processing
- *F*3. Performance Criteria
- *F*4. Heavily Utilized Hardware
- *F*5. High Transaction Rates
- *F*6. Online Data Entry
- F7. Online Updating
- *F*8. End-user Efficiency
- *F*9. Complex Computations
- *F*10. Reusability
- *F*11. Ease of Installation
- *F*12. Ease of Operation
- *F*13. Portability
- F14. Maintainability

- Each component or subfactor is rated from 0 to 5, where 0 means the subfactor is irrelevant, 3 means it is average, and 5 means it is essential to the system being built.
- Sum the resulting numbers to obtain **DI (degree of influence)**

Function Points Example

- The following formula combines the 14 ratings into a final technical complexity factor

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

i.e.,

$$\mathrm{TCF} = 0.65 + 0.01 * \mathrm{DI}$$

- The final calculation of FPs multiplies the UFC by the technical complexity factor TCF

$$\mathrm{FP} = \mathrm{UFC} \times \mathrm{TCF}$$

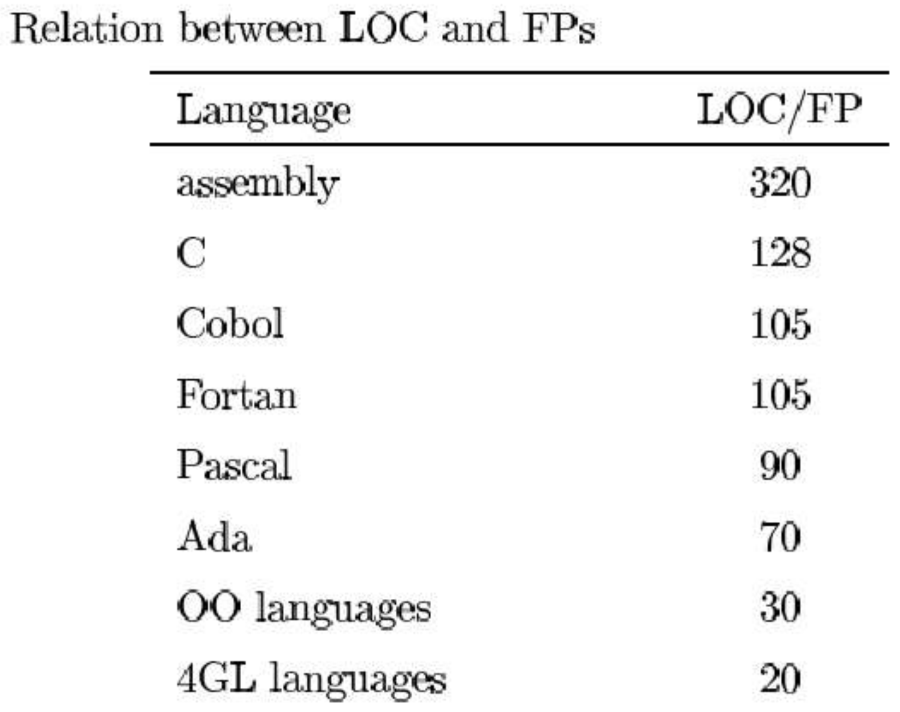
Function Points Example

- Function Points
    - FP=UFP*(0.65+0.01*DI)= 55*(0.65+0.01*30)=52.25

    - That means **FP=52.25**
- ** *considering* DI *or* TCF *= 30*

Relation between LOC and FP

- Relationship:
  - ***LOC = Language Factor * FP***
  - where
    - **LOC** (Lines of Code)
    - **FP** (Function Points)



Relation between LOC and FPs

Language	LOC/FP
assembly	320
C	128
Cobol	105
Fortan	105
Pascal	90
Ada	70
OO languages	30
4GL languages	20

Relation between LOC and FP

Assuming LOC’s per FP for:

Java = 53,

C++ = 64

aKLOC = FP * LOC_per_FP / 1000

It means for the SpellChekcer Example: (Java)

LOC=52.25*53=2769.25 LOC or 2.76 KLOC

IFPUG

- International Function Point User Group meets regularly to discuss FPs and their applications, and they publish guidelines with counting rules.
- Please visit https://ifpug.org/

COCOMO I

- **COnstructive COst MOdel** was introduced by **Dr. Barry Boehm**
- Software projects under COCOMO model strategies are classified into 3 categories, **organic, semi-detached, and embedded**
- **Organic**: A software project is said to be an organic type if**-**
  - Project is small and simple.
  - Project team is small with prior experience.
  - The problem is well understood and has been solved in the past.
  - Requirements of projects are not rigid, such a mode example is payroll processing system.

COCOMO I

- **Semi-Detached Mode:** A software project is said to be a Semi-Detached type if-
  - Project has complexity.
  - Project team requires more experience, better guidance, and creativity.
  - The project has an intermediate size and has mixed rigid requirements such a mode example is a transaction processing system which has fixed requirements.
  - It also includes the elements of organic mode and embedded mode.
  - Few such projects are- Database Management System(DBMS), new unknown operating system, difficult inventory management system

COCOMO I

- **Embedded Mode:** A software project is said to be an Embedded mode type if-
  - A software project has fixed requirements of resources .
  - Product is developed within very tight constraints.
  - A software project requiring the highest level of complexity, creativity, and experience requirement fall under this category.
  - Such mode software requires a larger team size than the other two models.

Types of COCOMO Models

- COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. Any of the three forms can be adapted according to our requirements. These are types of COCOMO model:
  - Basic COCOMO Model
  - Intermediate COCOMO Model
  - Detailed COCOMO Model

Basic COCOMO Model

- The first level, Basic COCOMO can be used for quick and slightly rough calculations of Software Costs.
- This is because the model solely considers based on lines of source code together with constant values obtained from software project types rather than other factors which have major influences on the Software development process as a whole.
- The basic COCOMO estimation model is given by the following expressions:
    - $Effort = a * (KLOC)^b$
    - $Duration = c * (Effort)^d$
    - $Person\ Required = Effort/Duration$

Basic COCOMO Model

- The values for a, b, c, and d differ depending on which mode you are using

Mode	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Basic COCOMO model

- **Example –** Consider a software project using semi-detached mode with 300 Kloc .find out effort estimation, development time, and person estimation.
- **Solution –**
- **Effort (E) =** a*(KLOC)^{b} = 3.0*(300)$^{\mathbf{1.12}}$ = 1784.42 PM
  Development Time (D) = c**(E)^{d}* = *2.5*(1784.42)$^{0.35}$ = 34.35 Months(M)
  Person Required (P) = E/D = 1784.42/34.35 = 51.9481 Persons ~52 Persons

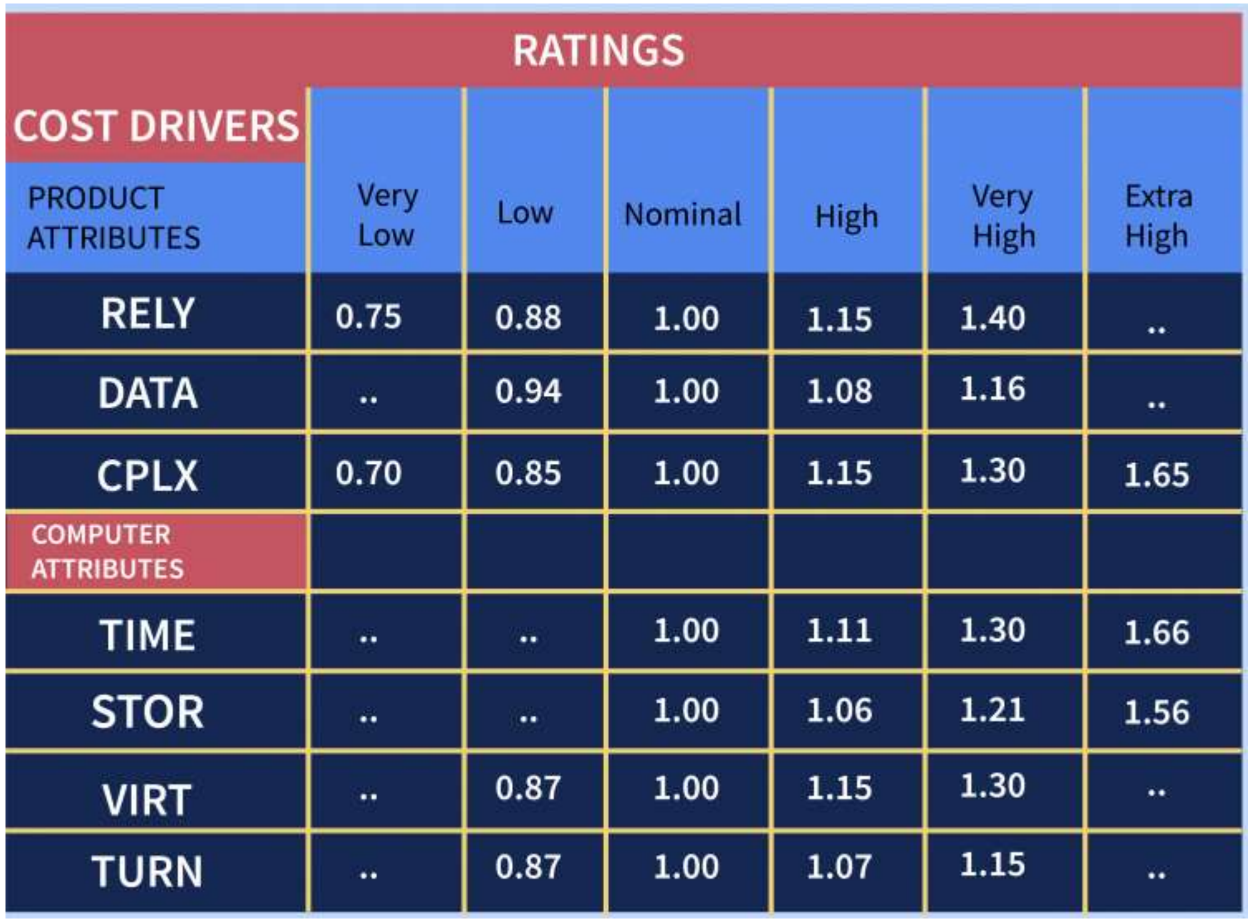
Intermediate COCOMO Model

- Intermediate COCOMO model is an extension of the Basic COCOMO model which includes a set of cost drivers into account in order to enhance more accuracy to the cost estimation model as a result.
- The estimation model makes use of a set of "cost driver attributes" to compute the cost of the software.
- The estimated effort and scheduled time are given by the relationship:
  - $Effort(E) = a * (KLOC)^b * EAF$
  - $Duration = c * (Effort)^d$
  - $Person\ Required = Effort/Duration$
- EAF = It is an Effort Adjustment Factor, which is calculated by multiplying the parameter values of different cost driver parameters. For ideal, the value is 1.

Classification of Cost Drivers and their Attributes

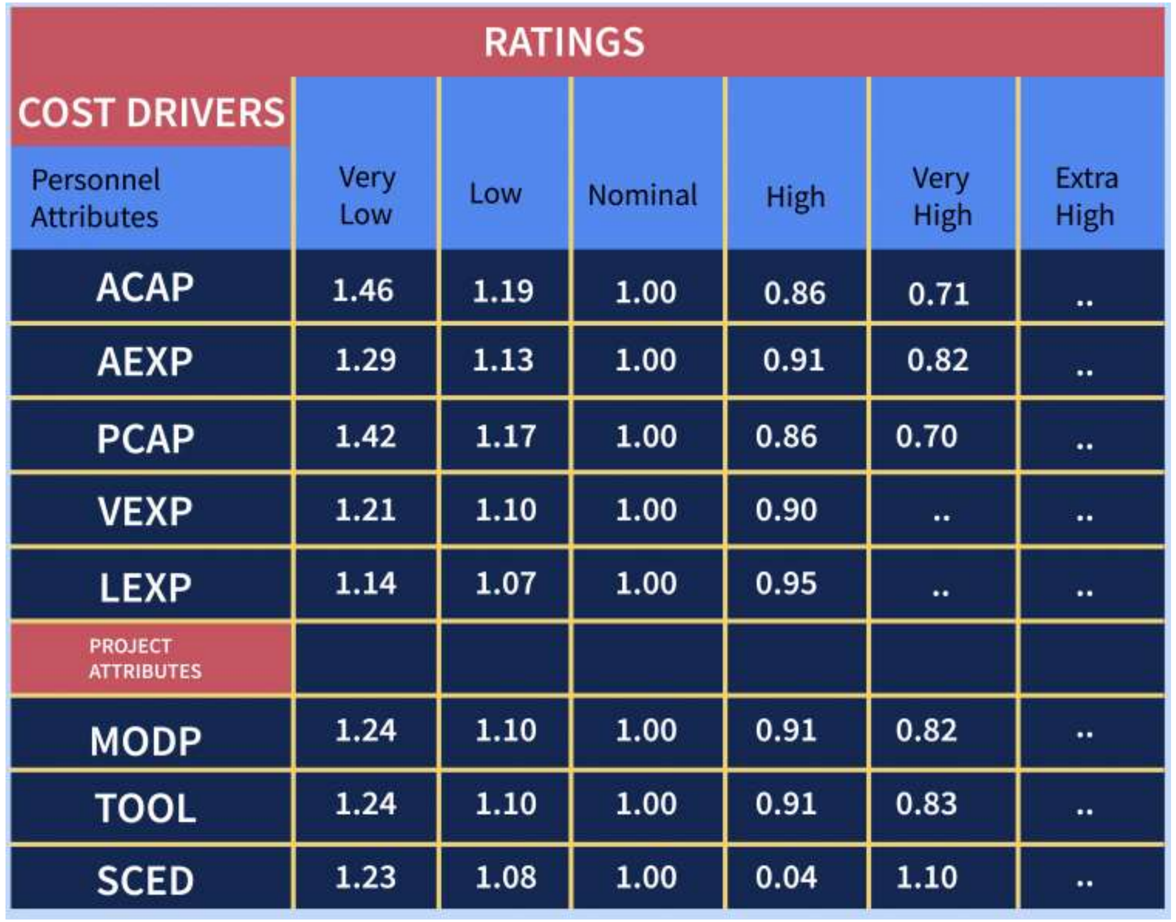
- **Product attributes**
  - Required software reliability extent
  - Size of the application database
  - The complexity of the product
- **Hardware attributes**
  - Run-time performance constraints
  - Memory constraints
  - The volatility of the virtual machine environment
  - Required turnabout time
- **Personal attributes –**
  - Analyst capability
  - Software engineering capability
  - Applications experience
  - Virtual machine experience
  - Programming language experience
- **Project attributes**
  - Use of software tools
  - Application of software engineering methods
  - Required development schedule

Cost Drivers and Ratings



COST DRIVERS	RATINGS					
PRODUCT ATTRIBUTES	Very Low	Low	Nominal	High	Very High	Extra High
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	..

Cost Drivers and Ratings



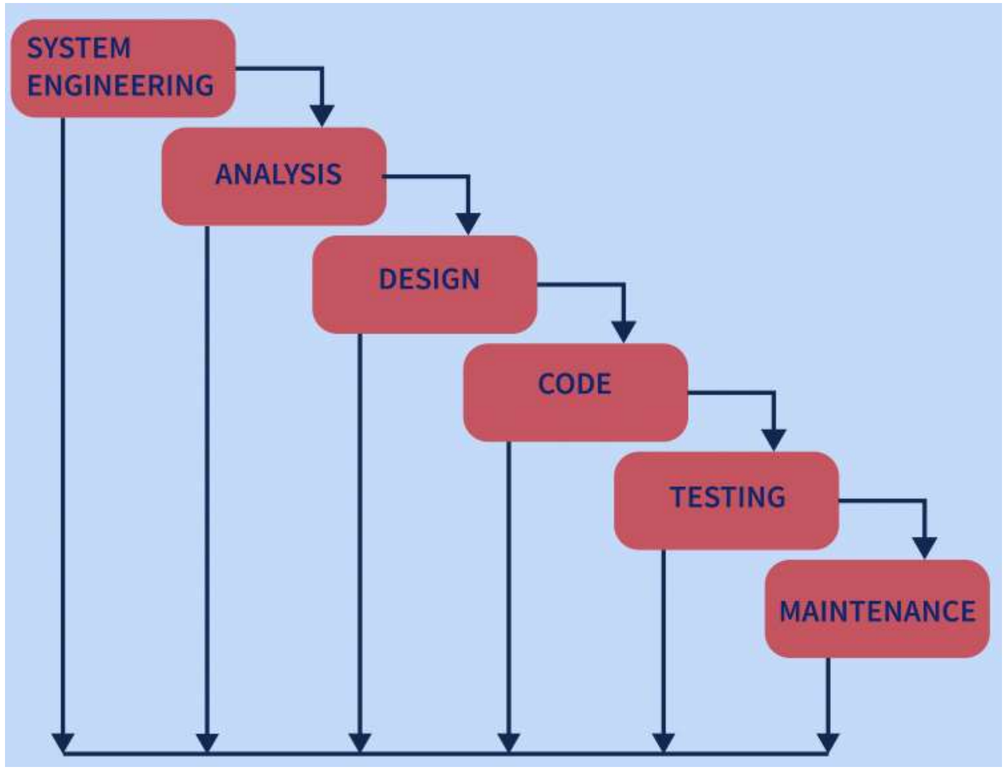
COST DRIVERS	RATINGS					
Personnel Attributes	Very Low	Low	Nominal	High	Very High	Extra High
ACAP	1.46	1.19	1.00	0.86	0.71	..
AEXP	1.29	1.13	1.00	0.91	0.82	..
PCAP	1.42	1.17	1.00	0.86	0.70	..
VEXP	1.21	1.10	1.00	0.90	..	..
LEXP	1.14	1.07	1.00	0.95	..	..
PROJECT ATTRIBUTES						
MODP	1.24	1.10	1.00	0.91	0.82	..
TOOL	1.24	1.10	1.00	0.91	0.83	..
SCED	1.23	1.08	1.00	0.04	1.10	..

Example

- **Example:** For a given project was estimated with a size of 300 KLOC. Calculate the Effort, Scheduled time for development by considering developer having high application experience and very low experience in programming.
- **Ans:**
  - Given the estimated size of the project is: 300 KLOC
    Developer having highly application experience: 0.82 (as per above table)
    Developer having very low experience in programming: 1.14(as per above table)
  - EAF = 0.82*1.14 = 0.9348
    Effort (E) = a*(KLOC)^{b} *EAF = 3.0*(300)$^{1.12}$ *0.9348 = 1668.07 MM
    Scheduled Time (D) = c*(E)^{d} = 2.5*(1668.07)$^{0.35}$ = 33.55 Months(M)

Detailed/Advanced COCOMO Model

- The model accounts for the influence of the individual development phase (analysis, design, etc.) of the project.
- It uses multipliers for each phase of the project. It gives more accurate estimations
- The whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.
- For instance, detailed COCOMO will perform cost estimation in each development phase of the Waterfall Model



Detailed/Advanced COCOMO Model

- In the Detailed COCOMO Model, the cost of each subsystem is estimated separately. This approach reduces the margin of error in the final estimate.
- **Example:** A distributed Management Information System (MIS) product for an organization having offices at several places across the country can have the following sub-components:
  - Database part
  - Graphical User Interface (GUI) part
  - Communication part
- Of these, the communication part can be considered as Embedded software. The database part could be Semi-detached software, and the GUI part Organic software.
- The costs for these three components can be estimated separately, and summed up to give the overall cost of the system