# Module 3

Functional Point Estimation and COCOMO

# Function Point Estimation:

- Function Point Analysis was initially developed by Allan J. Albercht in 1979 at IBM and it has been further modified by the International Function Point Users Group (IFPUG).

# The initial **Definition** is given **by Allan J. Albrecht:**

- *FPA gives a dimensionless number defined in function points which we have found to be an effective relative measure of function value delivered to our customer.*

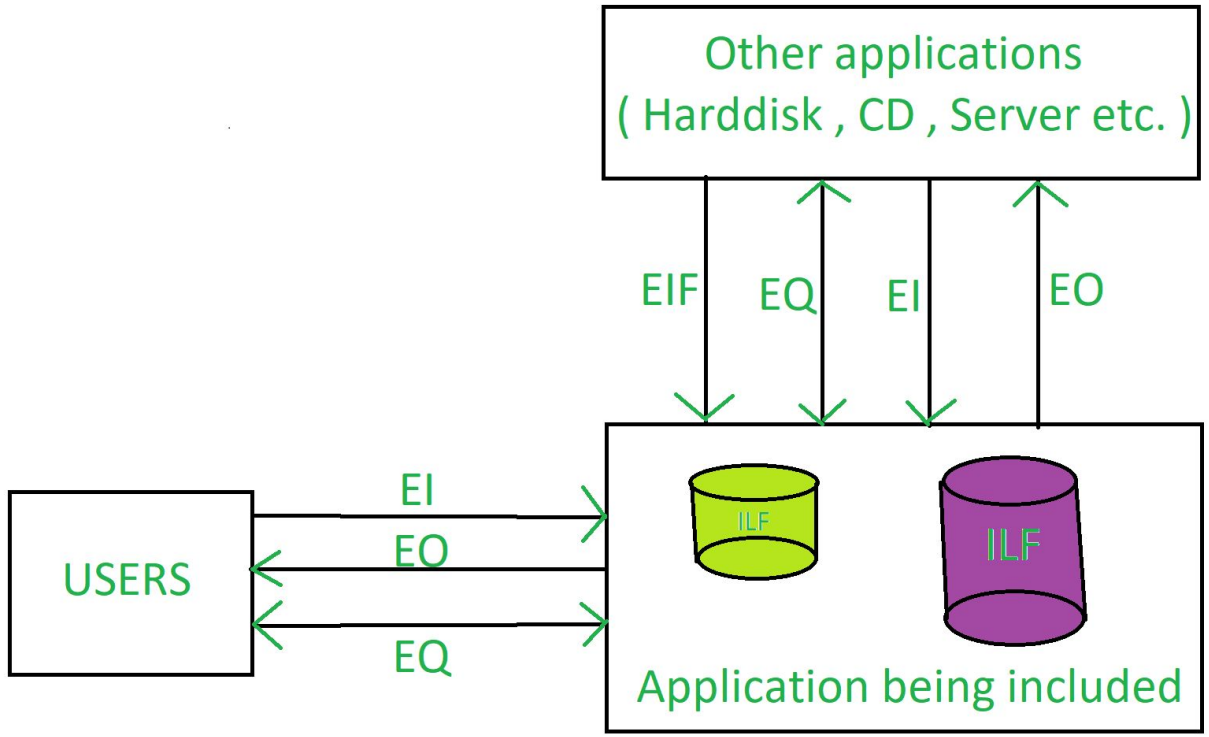
- FPA provides a standardized method to functionally size the software work product.
- This work product is the output of software new development and improvement projects for subsequent releases.
- It is the software that is relocated to the production application at project implementation.
- It measures functionality from the user's point of view i.e. on the basis of what the user requests and receives in return.

- Function Point Analysis (FPA) is a method or set of rules of Functional Size Measurement.
- It assesses the functionality delivered to its users, based on the user's external view of the functional requirements.
- It measures the logical view of an application, not the physically implemented view or the internal technical view.
- The Function Point Analysis technique is used to analyze the functionality delivered by software and Unadjusted Function Point (UFP) is the unit of measurement.

# Objectives of FPA:

- The objective of FPA is to measure the functionality that the user requests and receives.
- The objective of FPA is to measure software development and maintenance independently of the technology used for implementation.
- It should be simple enough to minimize the overhead of the measurement process.
- It should be a consistent measure among various projects and organizations.

# Types of FPA:


Other applications
( Harddisk , CD , Server etc. )
EIF
EQ
EI
EO
USERS
EI
EO
EQ
ILF
ILF
Application being included

# Types of FP Attributes

| Measurements Parameters | Examples |
|---|---|
| 1.Number of External Inputs(EI) | Input screen and tables |
| 2. Number of External Output (EO) | Output screens and reports |
| 3. Number of external inquiries (EQ) | Prompts and interrupts. |
| 4. Number of internal files (ILF) | Databases and directories |
| 5. Number of external interfaces (EIF) | Shared databases and shared routines. |

All these parameters are then individually assessed for complexity.

# Transactional Functional Type –

- **External Input (EI):** EI processes data or control information that comes from outside the application's boundary. The EI is an elementary process.
- **External Output (EO):** EO is an elementary process that generates data or control information sent outside the application's boundary.
- **External Inquiries (EQ):** EQ is an elementary process made up of an input-output combination that results in data retrieval.

# Data Functional Type –

- **Internal Logical File (ILF):** A user identifiable group of logically related data or control information maintained within the boundary of the application.

- **External Interface File (EIF):** A group of users recognizable logically related data allusion to the software but maintained within the boundary of another software.

# Benefits of FPA:

- FPA is a tool to determine the size of a purchased application package by counting all the functions included in the package.
- It is a tool to help users discover the benefit of an application package to their organization by counting functions that specifically match their requirements.
- It is a tool to measure the units of a software product to support quality and productivity analysis.
- It is a vehicle to estimate the cost and resources required for software development and maintenance.
- It is a normalization factor for software comparison.

# The drawback of FPA:

- It requires a subjective evaluation and involves many judgements.
- Many cost and effort models are based on LOC, so it is necessary to change the function points.
- Compared to LOC, there are less research data on function points.
- Run after creating the design spec.
- With subjective judgement, the accuracy rate of the assessment is low.
- Due to the long learning curve, it is not easy to gain proficiency.
- This is a very time-consuming method.

- All the parameters mentioned above are assigned some weights that have been experimentally determined and are shown in Table

| Measurement Parameter | Low | Average | High |
|---|---|---|---|
| 1. Number of external inputs (EI) | 7 | 10 | 15 |
| 2. Number of external outputs (EO) | 5 | 7 | 10 |
| 3. Number of external inquiries (EQ) | 3 | 4 | 6 |
| 4. Number of internal files (ILF) | 4 | 5 | 7 |
| 5. Number of external interfaces (EIF) | 3 | 4 | 6 |

The functional complexities are multiplied with the corresponding weights against each function, and the values are added up to determine the UFP (Unadjusted Function Point) of the subsystem.

**Computing FPs**

| Measurement Parameter | Count | | Weighing factor | | | |
|---|---|---|---|---|---|---|
| | | | Simple | Average | Complex | |
| 1. Number of external inputs (EI) | — | * | 3 | 4 | 6 = | — |
| 2. Number of external Output (EO) | — | * | 4 | 5 | 7 = | — |
| 3. Number of external Inquiries (EQ) | — | * | 3 | 4 | 6 = | — |
| 4. Number of internal Files (ILF) | — | * | 7 | 10 | 15 = | — |
| 5. Number of external interfaces(EIF) | — | * | 5 | 7 | 10 = | — |
| Count-total → | | | | | | |

# The Function Point (FP) is thus calculated with the following formula.

- **FP = Count-total * [0.65 + 0.01 * $\sum(f_i)$]**
  **= Count-total * CAF**
- Based on the FP measure of software many other metrics can be computed:

1. Errors/FP
2. $/FP.
3. Defects/FP
4. Pages of documentation/FP
5. Errors/PM.
6. Productivity = FP/PM (effort is measured in person-months).
7. $/Page of Documentation.

- LOCs of an application can be estimated from FPs. That is, they are interconvertible. **This process is known as backfiring**. For example, 1 FP is equal to about 100 lines of COBOL code.
- FP metrics is used mostly for measuring the size of Management Information System (MIS) software.

- **Example:** Compute the function point, productivity, documentation, cost per function for the following data:

1. Number of user inputs = 24
2. Number of user outputs = 46
3. Number of inquiries = 8
4. Number of files = 4
5. Number of external interfaces = 2
6. Effort = 36.9 p-m
7. Technical documents = 265 pages
8. User documents = 122 pages
9. Cost = $7744/ month

- Various processing complexity factors are: 4, 1, 0, 3, 3, 5, 4, 4, 3, 3, 2, 2, 4, 5.

# Solution:

| Measurement Parameter | Count | | Weighing factor |
|---|---|---|---|
| 1. Number of external inputs (EI) | 24 | * | 4 = 96 |
| 2. Number of external outputs (EO) | 46 | * | 4 = 184 |
| 3. Number of external inquiries (EQ) | 8 | * | 6 = 48 |
| 4. Number of internal files (ILF) | 4 | * | 10 = 40 |
| 5. Number of external interfaces (EIF)<br>Count-total → | 2 | * | 5=10<br>378 |

- So sum of all $f_i$ ($i \leftarrow 1$ to 14)
- $= 4 + 1 + 0 + 3 + 5 + 4 + 4 + 3 + 3 + 2 + 2 + 4 + 5 = 43$

- $FP = \text{Count-total} * [0.65 + 0.01 * \sum(f_i)]$
  $= 378 * [0.65 + 0.01 * 43]$
  $= 378 * [0.65 + 0.43]$
  $= 378 * 1.08 = 408$

$$\text{Productivity} = \frac{\text{FP}}{\text{Effort}} = \frac{408}{36.9} = 11.1$$

- Total pages of documentation

= technical document + user document

= 265 + 122

= 387pages

- Documentation

= Pages of documentation/FP

= 387/408

= 0.94

$$\text{Cost per function} = \frac{\text{cost}}{\text{productivity}} = \frac{7744}{11.1} = \$700$$

# Differentiate between FP and LOC

| FP | LOC |
| --- | --- |
| 1. FP is specification based. | 1. LOC is an analogy based. |
| 2. FP is language independent. | 2. LOC is language dependent. |
| 3. FP is user-oriented. | 3. LOC is design-oriented. |
| 4. It is extendible to LOC. | 4. It is convertible to FP (backfiring) |

# COCOMO and COCOMO II

- **COCOMO 1 Model**:
  The Constructive Cost Model was first developed by Barry W. Boehm. The model is for estimating effort, cost, and schedule for software projects. It is also called as Basic COCOMO. This model is used to give an approximate estimate of the various parameters of the project. Example of projects based on this model is business system, payroll management system and inventory management systems.
- **COCOMO 2 Model**:
  The COCOMO-II is the revised version of the original Cocomo (Constructive Cost Model) and is developed at the University of Southern California. This model calculates the development time and effort taken as the total of the estimates of all the individual subsystems. In this model, whole software is divided into different modules. Example of projects based on this model is Spreadsheets and report generator.

# COCOMO II

- COCOMO II is tuned to modern software life cycles. The original COCOMO model has been very successful, but it doesn't apply to newer software development practices as well as it does to traditional practices. COCOMO II targets modern software projects, and will continue to evolve over the next few years.
- COCOMO II is really three different models:
- The Application Composition Model.
- Suitable for projects built with modern GUI-builder tools. Based on new Object Points.
- The Early Design Model
- You can use this model to get rough estimates of a project's cost and duration before you've determined it's entire architecture.
- The Post-Architecture Model
- This is the most detailed COCOMO II model. You'll use it after you've developed your project's overall architecture. It has new cost drivers, new line counting rules, and new equations.
- Dr. Barry Boehm and his students are developing COCOMO II at USC.

| COCOMO I | COCOMO II |
| --- | --- |
| COCOMO I is useful in the waterfall models of the software development cycle. | COCOMO II is useful in non-sequential, rapid development and reuse models of software. |
| It provides estimates of effort and schedule. | It provides estimates that represent one standard deviation around the most likely estimate. |
| This model is based upon the linear reuse formula. | This model is based upon the non linear reuse formula |
| This model is also based upon the assumption of reasonably stable requirements. | This model is also based upon reuse model which looks at effort needed to understand and estimate. |
| Effort equation’s exponent is determined by 3 development modes. | Effort equation’s exponent is determined by 5 scale factors. |
| Development begins with the requirements assigned to the software. | It follows a spiral type of development. |
| Number of submodels in COCOMO I is 3 and 15 cost drivers are assigned | In COCOMO II, Number of submodel are 4 and 17 cost drivers are assigned |
| Size of software stated in terms of Lines of code | Size of software stated in terms of Object points, function points and lines of code |

- https://www.irjet.net/archives/V7/i5/IRJET-V7I5507.pdf