Practical 3

Function Point Analysis and COCOMO Model

Functional Point (FP) Analysis

Allan J. Albrecht initially developed function Point Analysis in 1979 at IBM and it has been further modified by the International Function Point Users Group (IFPUG). FPA is used to make an estimate of the software project, including its testing in terms of functionality or function size of the software product. However, functional point analysis may be used for the test estimation of the product. The functional size of the product is measured in terms of the function point, which is a standard of measurement to measure the software application.

Objectives of FPA

The basic and primary purpose of the functional point analysis is to measure and provide the software application functional size to the client, customer, and the stakeholder on their request. Further, it is used to measure the software project development along with its maintenance, consistently throughout the project irrespective of the tools and the technologies.

Following are the points regarding FPs

1. FPs of an application is found out by counting the number and types of functions used in the applications. Various functions used in an application can be put under five types, as shown in Table:

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.

Weights of 5-FP Attributes

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.

Function Point (FP) is an element of software development which helps to approximate the cost of development early in the process. It may measure functionality from the user's point of view.

Counting Function Point (FP):

- **Step-1:**

  F = 14 * scale

  Scale varies from 0 to 5 according to character of Complexity Adjustment Factor (CAF). Below table shows scale:

  0 - No Influence
  1 - Incidental
  2 - Moderate
  3 - Average
  4 - Significant
  5 - Essential

  14 aspects of processing complexity questions are as follow:

  - Data Communication
  - Distributed Data Processing
  - Performance
  - Heavily Used Configuration
  - Transaction Role
  - Online Data Entry
  - End-User Efficiency
  - Online Update
  - Complex Processing
  - Reusability
  - Installation Ease
  - Operational Ease
  - Multiple Sites
  - Facilitate Change

- **Step-2:** Calculate Complexity Adjustment Factor (CAF).

  CAF = 0.65 + (0.01 * F)

- **Step-3:** Calculate Unadjusted Function Point (UFP).

  TABLE (Required)

Function Units	Low	Avg	High
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

  Multiply each individual function point to corresponding values in TABLE.

- **Step-4:** Calculate Function Point.

  FP = UFP * CAF

Example:
Given the following values, compute function points when all complexity adjustment factors (CAF) and weighting factors are average.

User Input = 50
User Output = 40
User Inquiries = 35
User Files = 6
External Interface = 4

Explanation:

- **Step-1:** As complexity adjustment factor is average (given in question), hence,
- scale = 3.
  F = 14 * 3 = 42

- **Step-2:**

  CAF = 0.65 + (0.01 * 42) = 1.07

- **Step-3:** As weighting factors are also average (given in question) hence we will multiply each individual function point to corresponding values in TABLE.

  UFP = (50*4) + (40*5) + (35*4) + (6*10) + (4*7) = 628

- **Step-4:**

  Function Point = 628 * 1.07 = 671.96

This is the required answer.

COCOMO Model – Introduction

COCOMO (Constructive Cost Estimation Model) model was proposed by Boehm (1981). According to Boehm, software cost estimation should be done through three stages: Basic COCOMO, Intermediate COCOMO, and Complete COCOMO.

Organic, Semidetached and Embedded Software Projects

According to Boehm (1981), any software development project can be classified into one of the following three categories based on the development complexity: organic, semi-detached, and embedded.

Organic: A development project can be considered of organic type, if the project deals with developing a well understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar types of projects.

Semi-detached: A development project can be considered of semi-detached type, if the development consists of a mixture of experienced and inexperienced staff. Team members may have limited experience on related systems but may be unfamiliar with some aspects of the system being developed.

Embedded: A development project is considered to be of embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational procedures exist.

Three stages of software cost estimation

Basic COCOMO Model

The **basic COCOMO model** gives an approximate estimate of the project parameters. The basic COCOMO estimation model is given by the following expressions:

$$\text{Effort} = a_1 \times (\text{KLOC})^{a_2}\ \text{PM}$$
$$\text{Tdev} = b_1 \times (\text{Effort})^{b_2}\ \text{Months}$$

where,

- **KLOC** is the estimated size of the software product expressed in Kilo Lines of Code,

- **a1**, **a2**, **b1**, **b2** are constants for each category of software products,
- **Tdev** 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).

The values of a1, a2, b1, b2 for different categories of products (i.e., organic, semi-detached, and embedded) as given by Boehm are summarized below.

Estimation of Development Effort: For the three classes of software products, the formulas for estimating the effort based on the code size are shown below:

Organic: $\text{Effort} = 2.4 \times (\text{KLOC})^{1.05}$ PM

Semi-detached: $\text{Effort} = 3.0 \times (\text{KLOC})^{1.12}$ PM

Embedded: $\text{Effort} = 3.6 \times (\text{KLOC})^{1.20}$ PM

Estimation of Development Time: For the three classes of software products, the formulas for estimating the development time based on the effort size are given below:

Organic: $\text{Tdev} = 2.5 \times (\text{Effort})^{0.38}$ Months

Semi-detached: $\text{Tdev} = 2.5 \times (\text{Effort})^{0.35}$ Months

Embedded: $\text{Tdev} = 2.5 \times (\text{Effort})^{0.32}$ Months

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, the nominal development time, and cost required to develop the product.

From the basic COCOMO estimation formula for organic software:

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

$$\text{Tdev} = 2.5 \times (91)^{0.38} = 14 \text{ Months}$$

$$\text{Cost} = 14 \times 15{,}000 = \text{Rs. } 2{,}10{,}000/-$$

Lab Work:

- Perform Estimation using Function Point Analysis and Basic COCOMO model for your system. [Make proper assumptions.]