

### Ques 1} Software Size Estimation

→ for the planned features (25 input/output modules, 10 user interfaces, 8 external files), standard industry values are used:

#### • Lines of Code (LOC):

→ Input/Output Modules:  $25 \times 125 = 3,125$  LOC

→ User Interfaces:  $10 \times 225 = 2250$  LOC

→ External files:  $8 \times 100 = 800$  LOC

→ Total LOC:  ~~$3125 + 2250 + 800 = 6175$  LOC~~

#### • Function Point Count:

→ External Inputs/Outputs:  $25 \times 4 = 100$

→ Internal Logical files:  $10 \times 7 = 70$

→ External Interface Files:  $8 \times 5 = 40$

→ Total function point:  $100 + 70 + 40 = 210$

#### • Halstead Metrics Calculation:

→ Using the provided data:

- Distinct operator ( $n_1$ ) = 15

- Distinct operand ( $n_2$ ) = 25

- Total Operator Occurrences ( $f_{N_1}$ ) = 80

- Total operand Occurrences ( $N_2$ ) = 100

→ The derived metrics are:

- Program Vocabulary:  $15 + 25 = 40$

- Program Length =  $80 + 100 = 180$

- Volume =  $180 \times \log_2(40) \approx 180 \times 5.32 = 957.6$

- Difficulty =  $(15/2) \times (100/25) = 7.5 \times 4 = 30$

- Effort =  $957.6 \times 30 = 28728$

## Basic COCOMO Model Estimation

→ for an organic project with 617.5 LOG :

- Effort applied (Person - Months) :

$$E = 2.4 \times (6.175)^{1.05} \approx 15.6 \text{ PP}$$

- Development Time (Months) :

$$D = 2.5 \times (15.6)^{0.38} \approx 6.3 \text{ months}$$

# This means project will require about 15.6 person-months of work and can be expected to finish it roughly 6.3 months if resourced acc. to COCOMO model.

## Ques 2) Online Food Delivery App

→ Top down vs Bottom-up Design

- Top down :

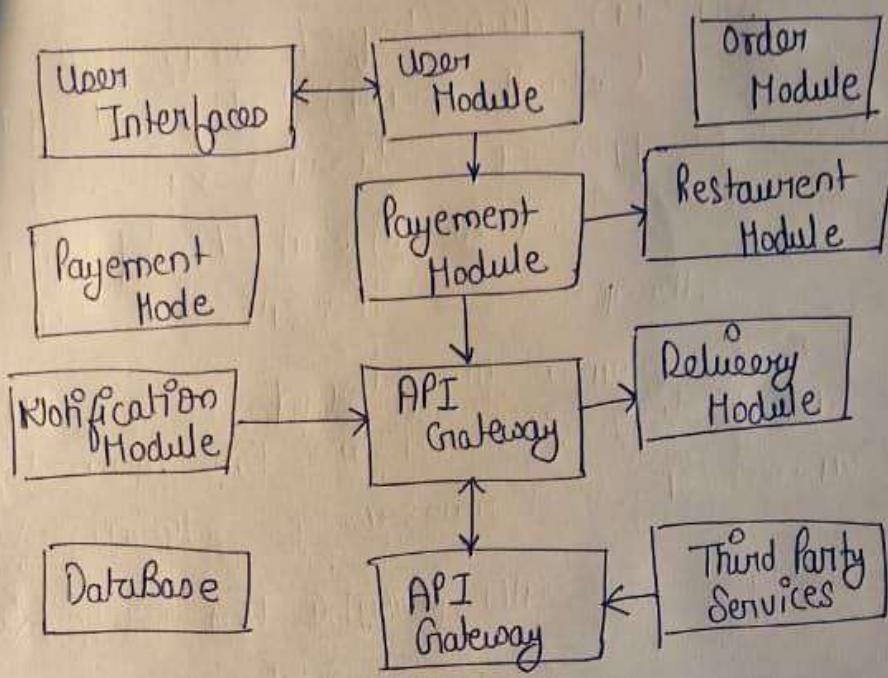
Start by breaking the whole system into main functions like User Management, Ordering, Payment and delivery, then divide each into smaller modules like Login, Cart, Payment, Gateway, and Delivery Tracking.

- Bottom-up :

Begin by creating and testing small reusable components such as Notifications Service or Payment Integration, then complete user workflows.

→ Design Principles:

- Cohesion: Keep related functionalities together, like all payment tasks in the payment module.
- Coupling: Ensure modules interact via well-defined interfaces or APIs reducing direct dependencies.
- Abstraction: Hide internal workings behind clean interfaces to enable easy updates and replacements.



### → Component Level Design Overview

The system consists of these key modules:

- User Interface : The mobile and web apps used by customers and restaurants.
- User Module : Manages user profile and authentication.
- Order Module : Handles order creation, updates and management.
- Payment Module : Manages payment processing and external payment gateway integrations.
- Restaurant Module : Manages restaurant and menu data.
- Delivery Module : Handles delivery assignment and tracking.
- Notification Module : Sends alert and updates to users.
- Database : Stores all application data including users, restaurants, and orders.
- API Gateway : Provides and secured communication among modules and external services.

### Q3) Functional Decomposition and Object Oriented Design (OOD) for a Library Management System:

→ Differences

- functional Decomposition: Breaks system into functions focusing on tasks; data and functions are separate; harder to extend and maintain as changes affect many functions.
- Object oriented Design: organizes system around objects combining data and behaviour; supports encapsulation, inheritance, and polymorphism; easier to extend and maintain.

→ UML Class Diagrams (OOD)

Key Classes:

- Library, Book, User (Member & Librarian), Transaction, Account

Relationships:

- Members borrow Books, librarians manage Books, Transaction links Users and Books with timestamps.

→ Better Design for Scalability and Maintenance.

Object-Oriented Design is preferred because its modularity, encapsulation, and reusability make it easier to extend, understand, and maintain as the system grows and changes.

~~Manu  
2/10~~