**Requirements Elicitation – Software Engineering**

**Requirements elicitation** is the process of gathering and defining the requirements for a software system. The goal of requirements elicitation is to ensure that the software development process is based on a clear and comprehensive understanding of the customer’s needs and requirements.

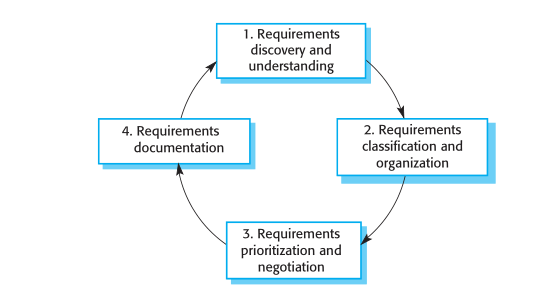
## **What is Requirement Elicitation?**

Requirements elicitation is perhaps the most difficult, most error-prone, and most communication-intensive software development.

1. It can be successful only through an effective customer-developer partnership. It is needed to know what the users require.
2. Requirements elicitation involves the identification, collection, analysis, and refinement of the requirements for a software system.
3. It is a critical part of the software development life cycle and is typically performed at the beginning of the project.
4. Requirements elicitation involves stakeholders from different areas of the organization, including business owners, end-users, and technical experts.
5. The output of the requirements elicitation process is a set of clear, concise, and well-defined requirements that serve as the basis for the design and development of the software system.

**The requirement elicitation techniques process involves:**

* Identifying stakeholders: Determining who will be involved in the project and who will be affected by its outcome.
* Needs and expectations: Understanding the needs and expectations of different stakeholders and prioritizing them.
* Documenting requirements: Creating a comprehensive and organized set of requirements used as a reference throughout the project.



A process model of the elicitation and analysis process is shown in Figure above. Each organization will have its own version or instantiation of this general model, depending on local factors such as the expertise of the staff, the type of system being developed, and the standards used.

The process activities are:

1. **Requirements discovery and understanding** This is the process of interacting with stakeholders of the system to discover their requirements. Domain requirements from stakeholders and documentation are also discovered during this activity.
2. **Requirements classification and organization** This activity takes the unstructured collection of requirements, groups related requirements and organizes them into coherent clusters.
3. **Requirements prioritization and negotiation** Inevitably, when multiple stakeholders are involved, requirements will conflict. This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiation. Usually, stakeholders have to meet to resolve differences and agree on compromise requirements.
4. **Requirements documentation** The requirements are documented and input into the next round of the spiral. An early draft of the software requirements documents may be produced at this stage, or the requirements may simply be maintained informally on whiteboards, wikis, or other shared spaces.

Various techniques can be used to elicit requirements. Here are some common ones:

**Interviews:**

Conduct one-on-one or group interviews with stakeholders to gather information about their needs, expectations, and concerns.

**Surveys and Questionnaires:**

Distribute surveys or questionnaires to a larger group of stakeholders to collect information in a structured format.

**Workshops:**

Organize workshops or brainstorming sessions with key stakeholders to encourage collaboration and generate ideas.

**Observation:**

Observe users in their natural environment to understand their behavior, tasks, and challenges. This can be particularly useful for system design.

Prototyping:

Develop prototypes or mockups to provide stakeholders with a tangible representation of the system, helping to clarify and refine requirements.

**Document Analysis:**

Review existing documentation, such as business plans, reports, and manuals, to extract relevant information about requirements.

**Focus Groups:**

Gather a diverse group of stakeholders to discuss and provide feedback on requirements. This technique promotes open discussion and can uncover varying perspectives.

**Use Cases and Scenarios:**

Develop use cases or scenarios to explore how users will interact with the system and to identify specific requirements in different situations.

**Ethnographic Studies:**

Conduct in-depth studies of the culture and practices within an organization to gain insights into the context in which the system will operate.

# Software Requirements Specification for Online Shopping System

## 1. Introduction

### 1.1 Purpose

The purpose of this document is to outline the requirements for the development of an online shopping system to facilitate easy and efficient purchasing of products for customers.

### 1.2 Scope

The online shopping system will include user registration, product browsing, shopping cart management, secure payment processing, and order tracking.

### 1.3 Intended Audience

This document is intended for developers, testers, project managers, and other stakeholders involved in the development and deployment of the online shopping system.

## 2. Overall Description

### 2.1 Product Perspective

The online shopping system will operate as a standalone web application interacting with a database for product storage and user information.

### 2.2 Product Features

* User registration and authentication
* Product catalog with search and filter options
* Shopping cart management
* Secure checkout process
* Order confirmation and tracking

### 2.3 User Classes and Characteristics

* Guest users: Users not logged in, can browse products.
* Registered users: Have accounts, can add items to the cart, and make purchases.
* Admin users: Manage product listings, user accounts, and order processing.

## 3. Specific Requirements

### 3.1 Functional Requirements

#### **3.1.1 User Registration**

* Users can create accounts by providing necessary information.
* User passwords must be securely stored using encryption.

#### **3.1.2 Product Browsing**

* Users can browse products by category, price range, and other filters.
* Product listings should include images, descriptions, and prices.

#### **3.1.3 Shopping Cart Management**

* Users can add/remove items from the cart.
* Cart contents should persist between user sessions.

#### **3.1.4 Secure Checkout**

* Secure payment processing using HTTPS.
* Support for multiple payment methods (credit card, PayPal, etc.).

#### **3.1.5 Order Tracking**

* Users can view the status of their orders.
* Email notifications for order confirmation and shipping updates.

### 3.2 Non-Functional Requirements

#### **3.2.1 Performance**

* System response time should be under 3 seconds.
* The system should handle at least 1000 concurrent users.

#### **3.2.2 Security**

* User data and payment information must be encrypted.
* Protection against common web application security vulnerabilities (SQL injection, cross-site scripting).

## 4. External Interface Requirements

### 4.1 User Interfaces

* Intuitive and responsive web interface for both desktop and mobile users.

### 4.2 Hardware Interfaces

* The system will run on standard web servers with sufficient processing power and memory.

### 4.3 Software Interfaces

* Integration with a relational database management system (e.g., MySQL).
* Payment gateway integration for secure transactions.

## 5. System Features

### 5.1 User Registration

* Users can register by providing a valid email address, password, and personal information.

### 5.2 Product Browsing

* Browse products by category, price, or search by keyword.
* View detailed product information, including images and descriptions.

### 5.3 Shopping Cart Management

* Add items to the cart, update quantities, and remove items.
* View the total cost of items in the cart.

### 5.4 Secure Checkout

* Enter shipping details.
* Choose a payment method and complete the transaction securely.

### 5.5 Order Tracking

* View order history and current order status.
* Receive email notifications for order updates.

## 6. Performance Requirements

### 6.1 Response Time

* The system should respond to user actions within 3 seconds.

### 6.2 Scalability

* The system should handle up to 1000 concurrent users without significant performance degradation.

## 7. Design Constraints

### 7.1 Technology Constraints

* The system will be developed using HTML, CSS, JavaScript, and a backend framework (e.g., Node.js).
* Database interactions will be through SQL queries.

### 7.2 Security Constraints

* Compliance with industry standards for secure payment processing.

## 8. Software Quality Attributes

### 8.1 Reliability

* The system should be available 99.9% of the time.

### 8.2 Maintainability

* Code should be well-documented, and modular for ease of maintenance.

## 9. Other Requirements

* The system should comply with relevant data protection regulations.
* The project timeline should adhere to the agreed-upon schedule.

## 10. Appendix

### 10.1 Glossary

* List of terms and definitions used throughout the document.

### 10.2 References

* Any external references or documents cited in the SRS.

**Requirement Models:**

Scenario-based models

Scenario-based models, such as Use Cases and User Stories, are effective tools for capturing and describing system requirements from a user's perspective. Here's an explanation and an example for each:

Use Cases:

Definition: Use cases are descriptions of interactions between a system and its external actors (users or other systems) to achieve a specific goal or complete a certain task.

Example: Online Shopping System

Use Case Name: Place Order

Actors: Customer, Inventory System, Payment System

Description: The customer selects items for purchase, adds them to the cart, provides shipping details, selects a payment method, and confirms the order. The Inventory System deducts the purchased items from the stock, and the Payment System processes the payment.

Use Case Name: View Order History

Actors: Registered Customer

Description: A registered customer can view a list of their previous orders, including details such as order date, items purchased, and current status.

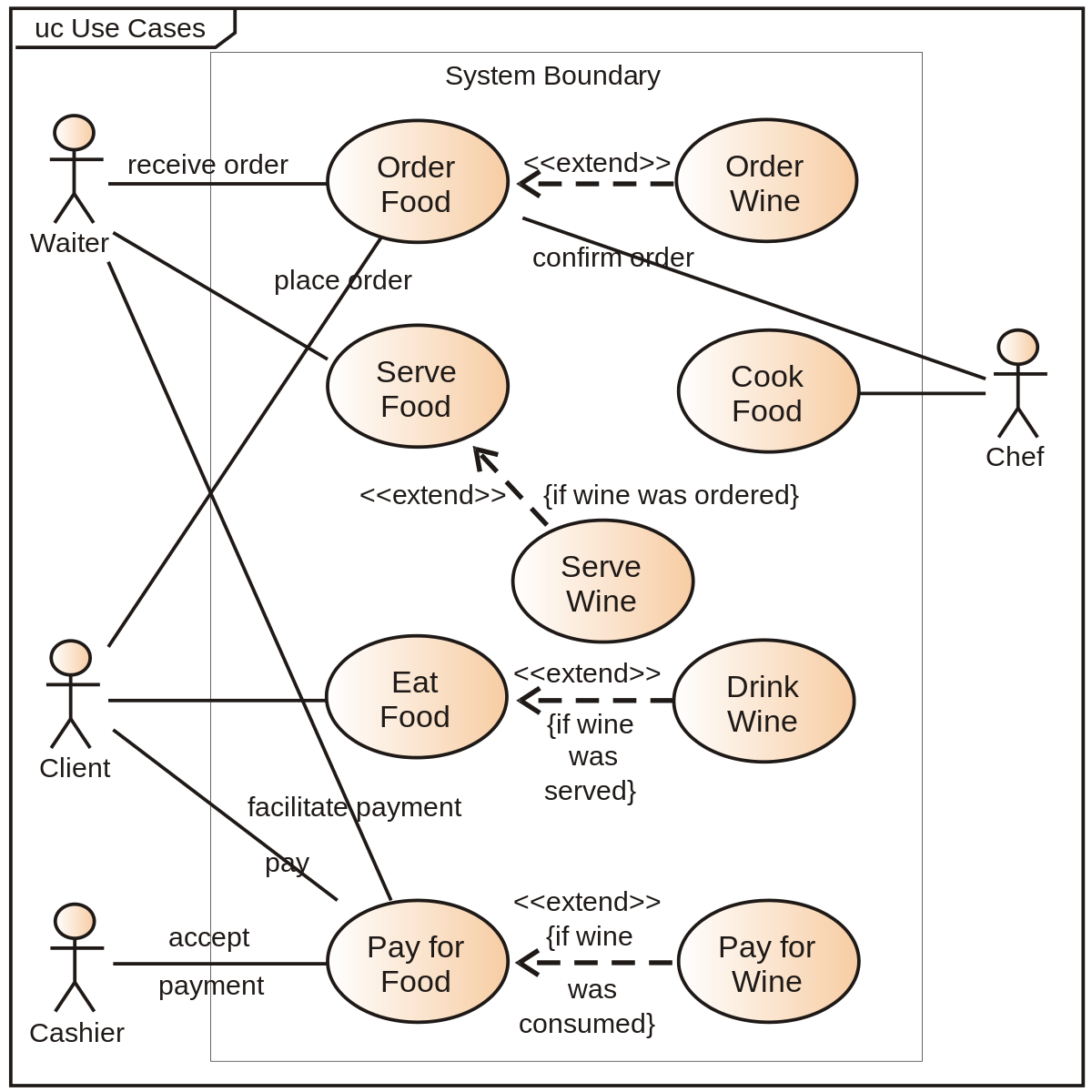
User Stories:

Definition: User stories are brief, natural language descriptions of features or functionalities from an end user's perspective, often written in the format: "As a [user type], I want [an action] so that [benefit or goal]."

Example: Online Banking System

User Story 1: As a customer, I want to be able to log in to my online banking account securely using two-factor authentication so that my financial information remains protected.

User Story 2: As a customer, I want to receive real-time notifications for any significant transactions on my account so that I can quickly identify and address any suspicious activity.



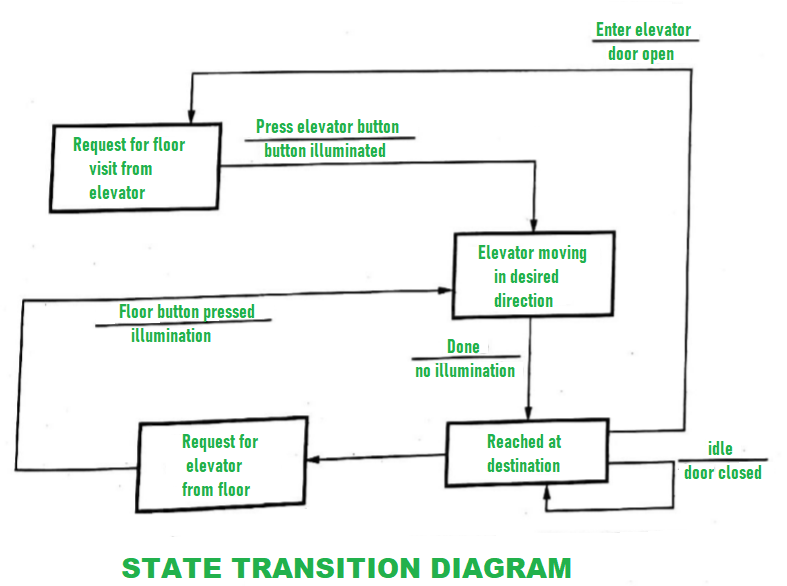
**Behavioural Model** 

**Behavioural Model** is specially designed to make us understand behaviour and factors that influence behaviour of a System. Behaviour of a system is explained and represented with the help of a diagram. This diagram is known as State Transition Diagram. It is a collection of states and events. It usually describes overall states that a system can have and events which are responsible for a change in state of a system.

So, on some occurrence of a particular event, an action is taken and what action needs to be taken is represented by State Transition Diagram.

**Example :**  
Consider an Elevator. This elevator is for n number of floors and has n number of buttons one for each floor.  
Elevator’s working can be explained as follows :

1. **Elevator buttons** are type of set of buttons which is there on elevator. For reaching a particular floor you want to visit, “elevator buttons” for that particular floor is pressed. Pressing, will cause illumination and elevator will start moving towards that particular floor for which you pressed “elevator buttons”. As soon as elevator reaches that particular floor,  
   illumination gets canceled.
2. **Floor buttons** are another type of set of buttons on elevator. If a person is on a particular floor and he wants to go on another floor, then elevator button for that floor is pressed. Then, process will be same as given above. Pressing, will cause illumination and elevator to start moving, and when it reaches on desired floor, illumination gets canceled.
3. When there is no request for elevator, it remains closed on current floor.



**Advantages :**

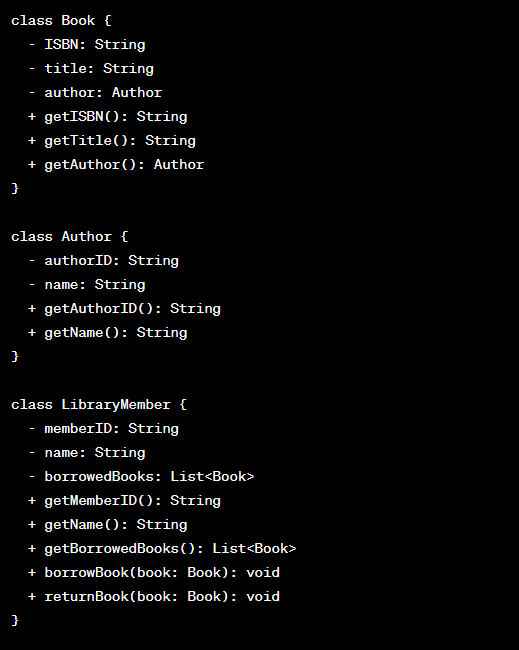
* Behavior and working of a system can easily be understood without any effort.
* Results are more accurate by using this model.
* This model requires less cost for development as cost of resources can be minimal.
* It focuses on behavior of a system rather than theories.

**Disadvantages :**

* This model does not have any theory, so trainee is not able to fully understand basic principle and major concept of modeling.
* This modeling cannot be fully automated.
* Sometimes, it’s not easy to understand overall result.
* Does not achieve maximum productivity due to some technical issues or any errors.

**Class-based model**

In software engineering, requirement models are used to represent and document the functional and non-functional requirements of a system. Class-based models, particularly class diagrams, can be employed to represent entities, their relationships, and their attributes in the context of requirements. Below is an example of how a class-based model can be used in the context of a simple requirement for a library management system.



**Explanation:**

**Book Class:**

* Represents a book with attributes such as ISBN, title, and an association with an Author.
* Provides methods to retrieve information about the book.

**Author Class:**

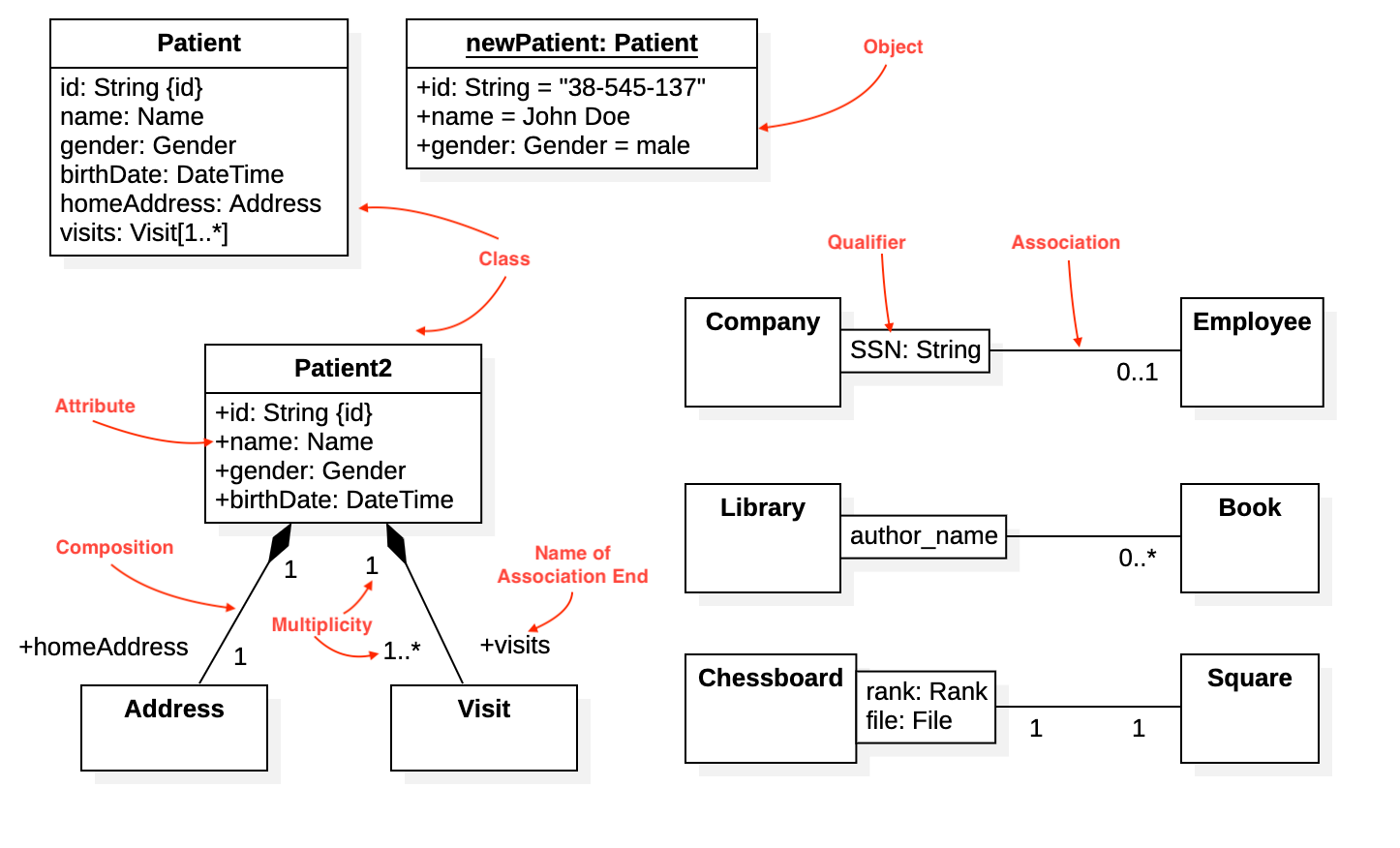
* Represents an author with attributes like authorID and name.
* Provides methods to retrieve information about the author.

**LibraryMember Class:**

* Represents a library member with attributes like memberID, name, and a list of borrowedBooks.
* Provides methods to retrieve information about the member, get the list of borrowed books, borrow a book, and return a book.

**Associations:**

* The association between Book and Author represents that each book is written by one author.
* The association between LibraryMember and Book represents that each member can borrow multiple books.



Software Project Estimation

**LOC**

A **line of code (LOC)** is any line of text in a code that is not a comment or blank line, and also header lines, in any case of the number of statements or fragments of statements on the line. LOC clearly consists of all lines containing the declaration of any variable, and executable and non-executable statements. As Lines of Code (LOC) only counts the volume of code, you can only use it to compare or estimate projects that use the same language and are coded using the same coding standards.

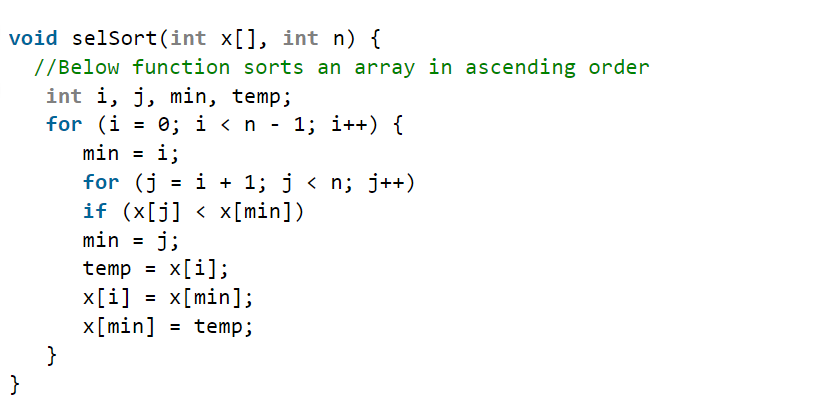
## **Advantages of Lines of Code (LOC):**

* Effort Estimation: LOC is occasionally used to estimate development efforts and project deadlines at a high level. Although caution is necessary, project planning can begin with this.
* Comparative Analysis: High-level productivity comparisons between several projects or development teams can be made using LOC. It might provide an approximate figure of the volume of code generated over a specific time frame.
* Benchmarking Tool: When comparing various iterations of the same program, LOC can be used as a benchmarking tool. It may bring information on how modifications affect the codebase’s total size.

## **Disadvantages of Lines of Code (LOC):**

## The biggest problem with LOC is that we can use it to estimate projects that plan to use one programming language with a fixed syntax and agreed-upon coding standard. This is so because LOC counts lines of the code as per the programming language syntax and semantics.

* **The second problem with LOC is that it skips documentation lines.** For example, we often make small changes in the production environment but add extensive comments or annotations to document them. However, comments and other hints aren’t covered by LOC.
* Another problem is that **LOC doesn’t consider the complexity of the underlying code statements.** So, it doesn’t correctly match the quality and efficiency of the code since **not all lines are equally important, complex, or easy to write.** Sometimes, a few lines of code involving complex logic can be harder to come by than very large but straightforward programs.

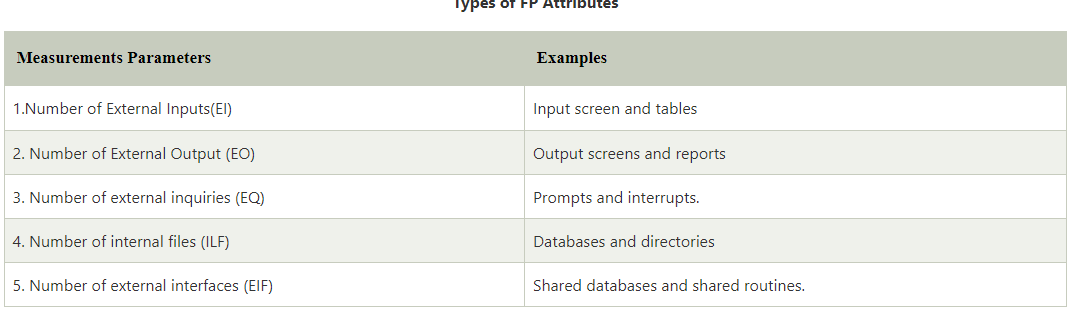


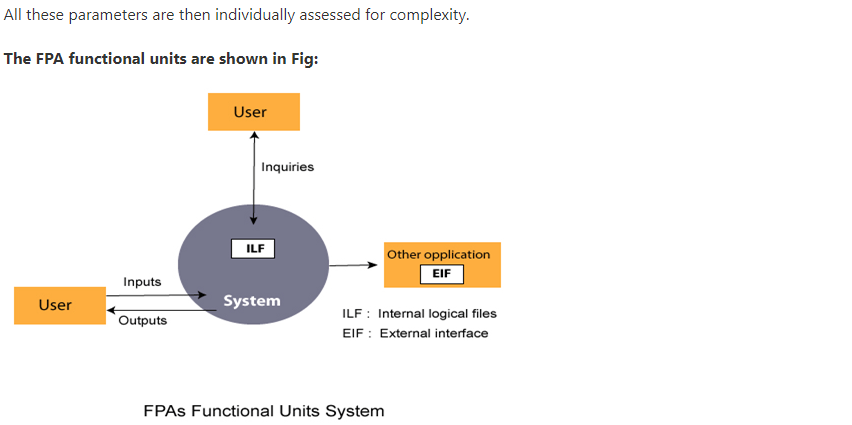
So, now If LOC is simply a count of the number of lines then the above function shown contains **13 lines of code (LOC).**But when comments and blank lines are ignored, the function shown above contains **12 lines of code (LOC)**.

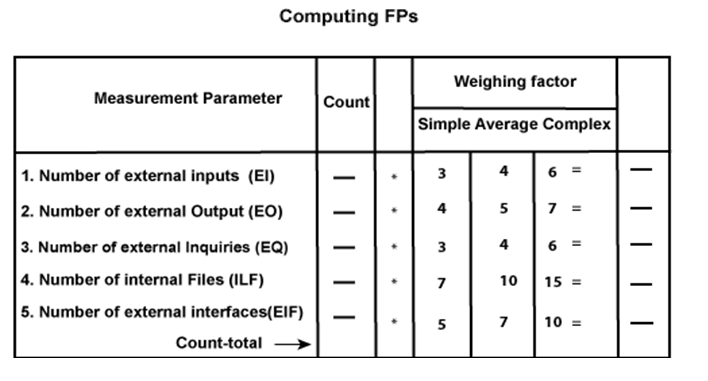
Function Points (FP):

Function points are a unit of measurement used to quantify the functionality provided by a software application. They are independent of the programming language and technology used. Function points are calculated based on the user inputs, outputs, queries, files, and interfaces in a system.

The process of determining function points involves assigning weights to different components of the software, such as inputs, outputs, inquiries, files, and interfaces. These weights are then multiplied by the number of occurrences of each component, and the results are summed to obtain the total function points.



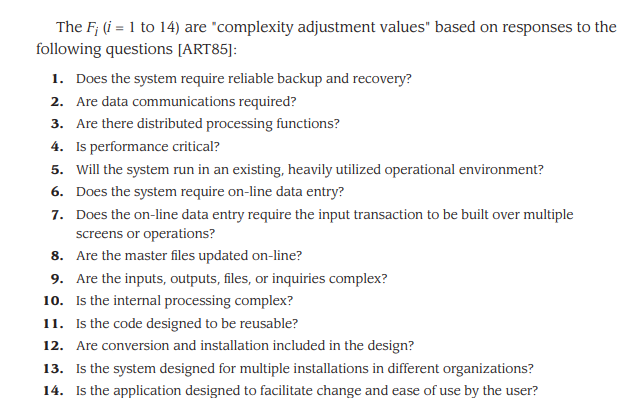




Compute the function point, productivity, documentation, cost per function for the following data:

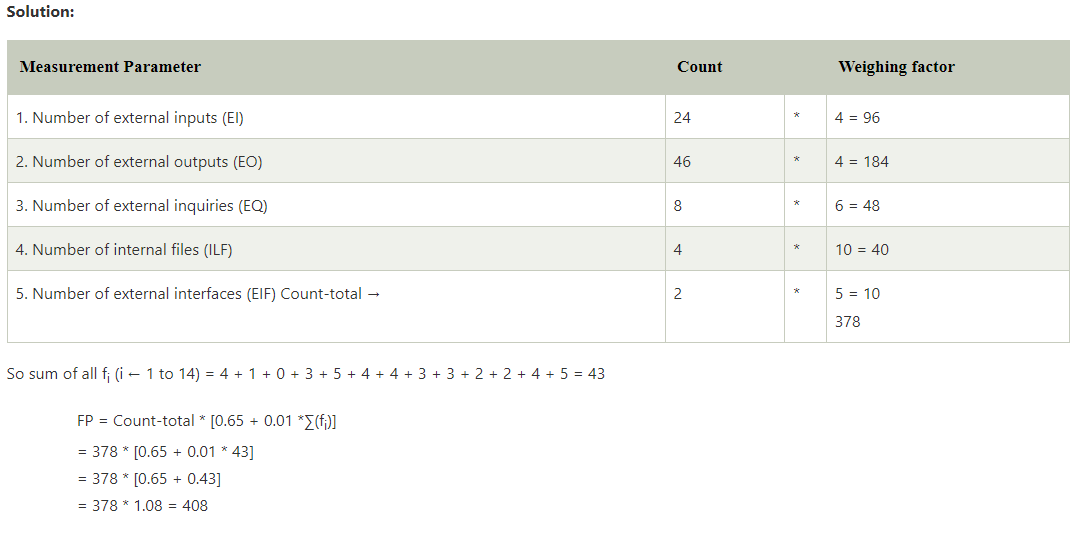
1. Number of user inputs = 24
2. Number of user outputs = 46
3. Number of inquiries = 8
4. Number of files = 4
5. Number of external interfaces = 2

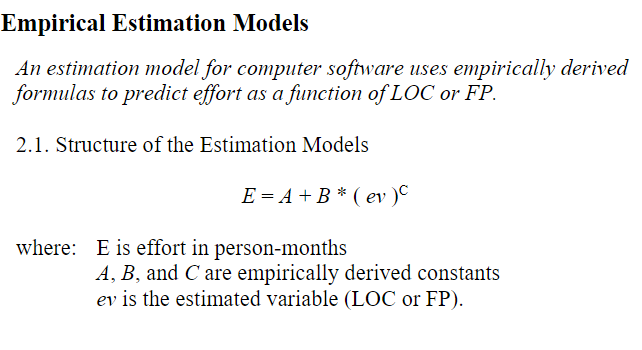
Various processing complexity factors are: 4, 1, 0, 3, 3, 5, 4, 4, 3, 3, 2, 2, 4, 5.

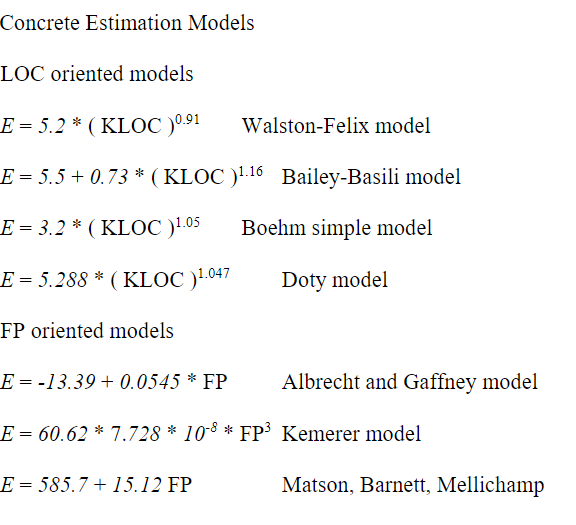


*FP = Count-Total \* [0.65 + 0.01 \* ⅀****(fi)****]*

*= Count \* CAF*







# COCOMO Model

Boehm proposed COCOMO (Constructive Cost Estimation Model) in 1981.COCOMO is one of the most generally used software estimation models in the world. COCOMO predicts the efforts and schedule of a software product based on the size of the software.

**The necessary steps in this model are:**

1. Get an initial estimate of the development effort from evaluation of thousands of delivered lines of source code (KDLOC).
2. Determine a set of 15 multiplying factors from various attributes of the project.
3. Calculate the effort estimate by multiplying the initial estimate with all the multiplying factors i.e., multiply the values in step1 and step2.

The initial estimate (also called nominal estimate) is determined by an equation of the form used in the static single variable models, using KDLOC as the measure of the size. To determine the initial effort Ei in person-months the equation used is of the type is shown below

**Ei=a\*(KDLOC)b**

**In COCOMO, projects are categorized into three types:**

1. Organic
2. Semidetached
3. Embedded

**1.Organic:** A development project can be treated of the 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 methods of projects. **Examples of this type of projects are simple business systems, simple inventory management systems, and data processing systems.**

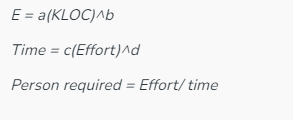
**2. Semidetached:** A development project can be treated with semidetached type if the development consists of a mixture of experienced and inexperienced staff. Team members may have finite experience in related systems but may be unfamiliar with some aspects of the order being developed. **Example of Semidetached system includes developing a new operating system (OS), a Database Management System (DBMS), and complex inventory management system.**

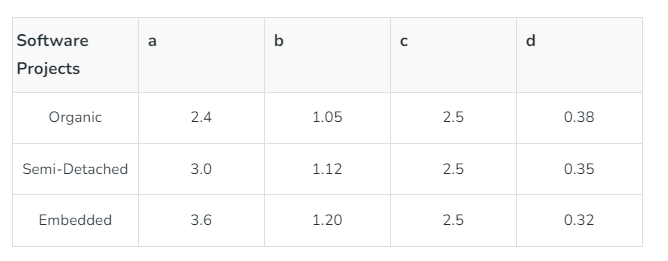
**3. Embedded:** A development project is treated to be of an embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational method exist. **For Example:** ATM, Air Traffic control.

According to Boehm, software cost estimation should be done through three stages:

1. Basic Model
2. Intermediate Model
3. Detailed Model

### 1. Basic Model





Suppose a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three model i.e., organic, semi-detached & embedded.

