Software Cost Estimation

Course Learning Outcomes

Discuss key objective of software cost estimation to accurately predict the resources, time, and budget required to complete a software project.

Key Takeaway Points

- **Software Cost Estimation**
- **Effort Estimation Based on Project Size**: COCOMO (Constructive Cost Model) estimates software development effort based on the size of the project in terms of lines of code (LOC).
- **Three Project Types**: COCOMO identifies three types of projects (organic, semi-detached, and embedded) with different effort multipliers, reflecting the complexity of the project.
- **Technology-Independent Measurement**: FPA measures software size based on user functions (inputs, outputs, inquiries, files), regardless of the technology or programming language used.
- **Functional Complexity Evaluation**: FPA assigns points to software features based on complexity (low, average, high) and uses these points to estimate effort and cost.

Why Cost Estimation?

- **Cost estimation is needed early for software pricing**

- **Software price = cost + profit**

Fundamental estimation questions

- How much effort is required to complete an activity?
- How much calendar time is needed to complete an activity?
- What is the total cost of an activity?
- How can project estimation and scheduling ensure accurate predictions for successful completion?

Introduction to Software Cost Estimation

- In software engineering, cost estimation is a crucial process for determining the resources required for software development, including time, effort, and budget.
- Two widely-used models for software cost estimation are:
  - **COCOMO (Constructive Cost Model)**
  - **Function Point Analysis (FPA).**

1. Basic COCOMO Model (Constructive Cost Model)

COCOMO, developed by Barry Boehm in the 1980s, is a regression-based model used to estimate the **cost, effort, and time** required to develop software projects. It uses **lines of code (LOC)** as the primary input.

Types of COCOMO Models:

1. **Basic COCOMO**: Suitable for small projects with relatively less complexity.
2. **Intermediate COCOMO**: Considers additional cost drivers such as software reliability, team capability, and product complexity.
3. **Detailed COCOMO**: The most comprehensive model that takes into account all the phases of software development and provides detailed cost estimation.

1. Basic COCOMO Model (Constructive Cost Model)

Types of COCOMO Models:

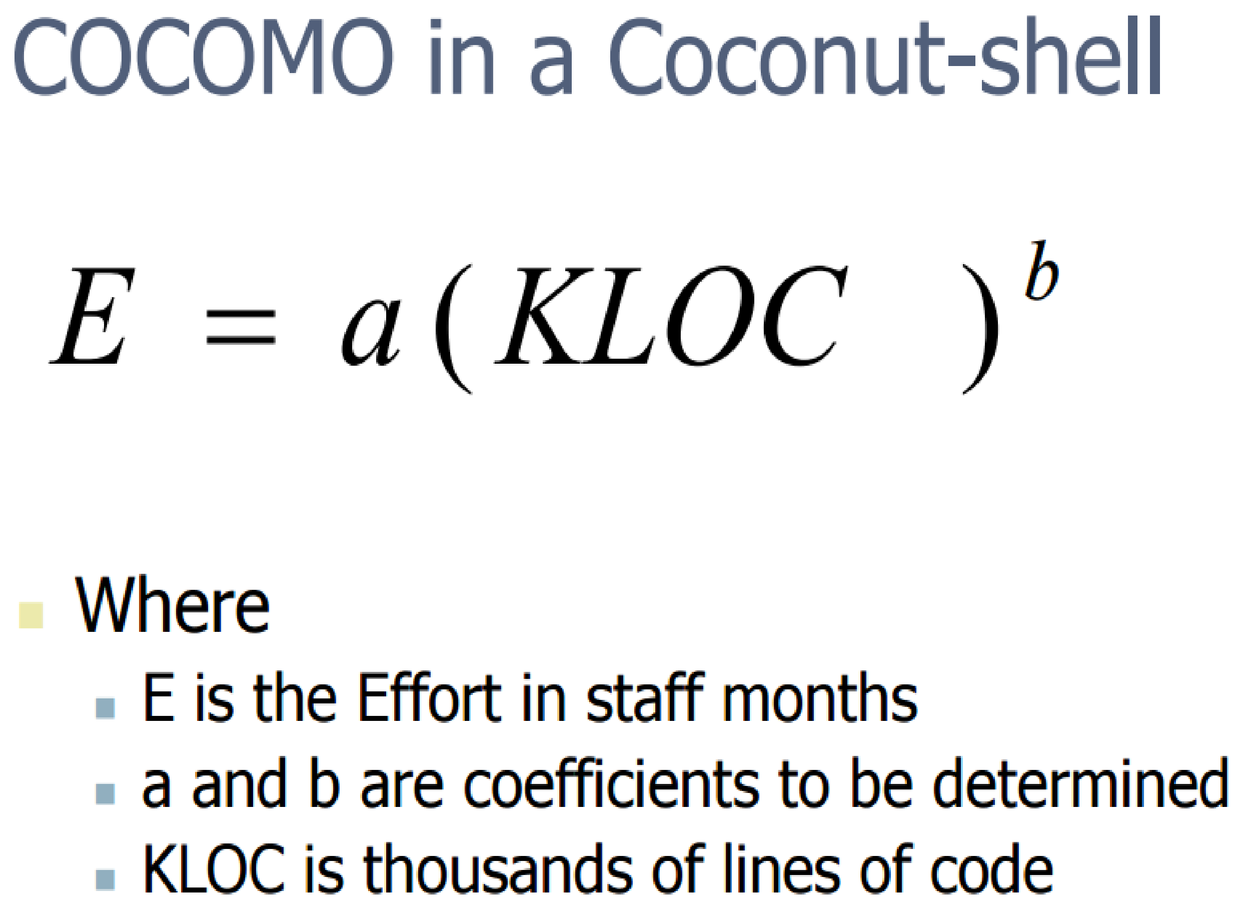
1. **Basic COCOMO**:
    1. Provides a rough estimate of the effort and cost.
    2. Suitable for small, straightforward projects.
    3. Does not consider detailed project characteristics.
2. **Intermediate COCOMO**:
    1. Takes into account additional project characteristics (15 cost drivers) like product reliability, complexity, and team capability.
    2. Provides a more accurate estimate than the Basic model.
3. **Detailed COCOMO**:
    1. Extends the Intermediate model by incorporating detailed information about each phase of the software development life cycle (SDLC).
    2. Most comprehensive and accurate.

Project types (COCOMO Model)

Project Category	Project size	Nature of project	Innovation	deadline of the project
Organic	2-50 KLOC	Small size, Experience developer, example, payroll system, inventory system, etc.	little	Not tight
Semi Detached	50-300 KLOC	Medium size project, medium size team, example DB large project, etc.	Medium	Medium
Embedded	Over 300 KLOC	Large project, Skills challenging, Big team required, real time system, example Airlines, ATMs etc.	Significance	Tight

1. Basic COCOMO Model (Constructive Cost Model)

Effort Applied



COCOMO in a Coconut-shell

$$E = a(KLOC)^b$$

- Where
  - E is the Effort in staff months
  - a and b are coefficients to be determined
  - KLOC is thousands of lines of code

1. Basic COCOMO Model (Constructive Cost Model)

The Constants

Mode	a	b
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embedded	3.6	1.20

1. Basic COCOMO Model (Constructive Cost Model)

- **COCOMO Project Effort Estimation**

  - **Example 1:** if a software product is **organic** and it is estimated to be **8,000 LOC**, the initial effort is calculated as:

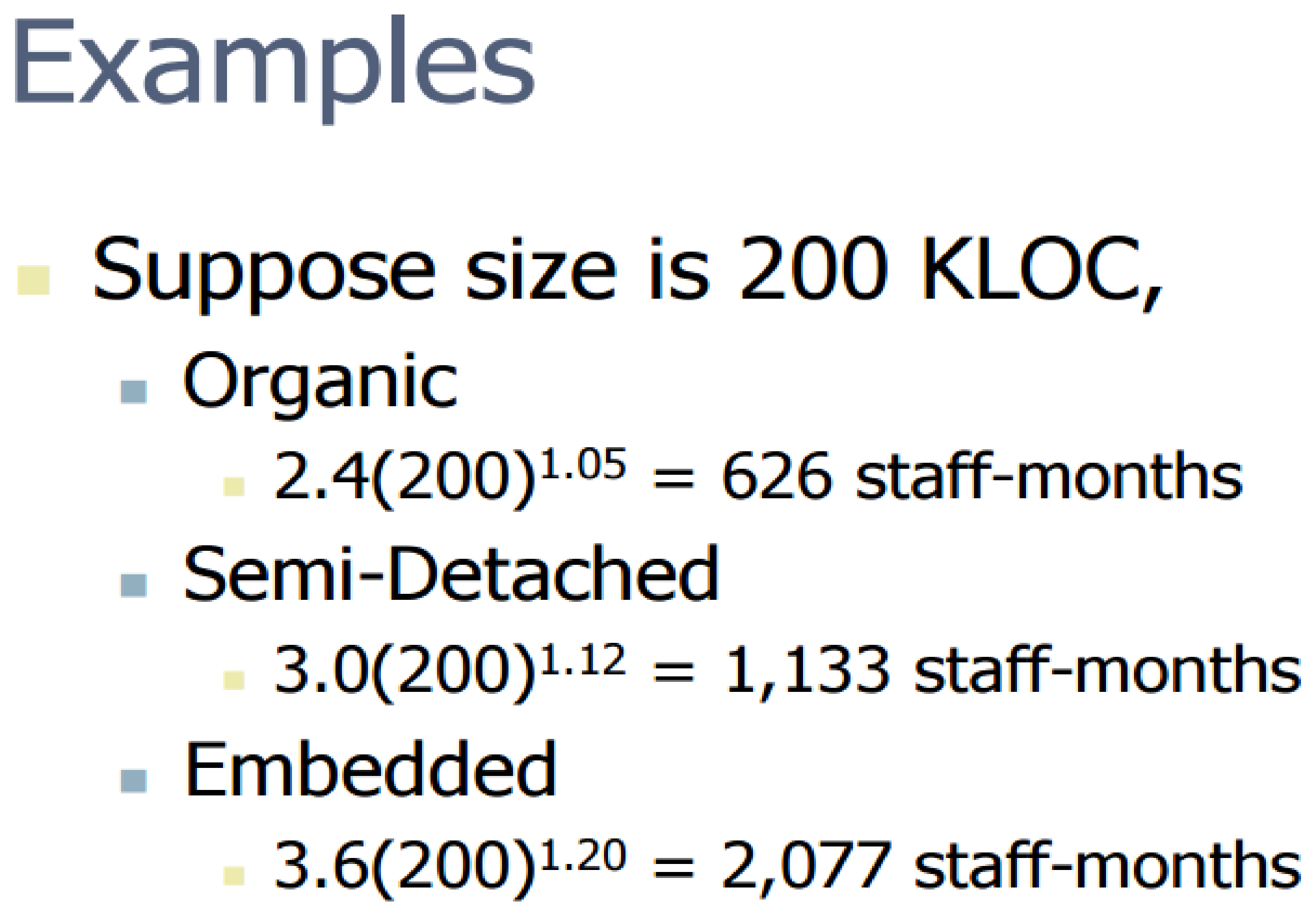
$$E_i = 2.4 \times 8^{1.05} = 25 \text{ person-months}$$

  - **Example 2:** if a software product is considered **embedded** and is estimated to be **10,000 LOC**, its nominal effort is as follows:

$$E_i = 3.6 \times 10^{1.20} = 58 \text{ person-months}$$

1. Basic COCOMO Model (Constructive Cost Model)

- **Effort Applied**



Examples

- Suppose size is 200 KLOC,
  - Organic
    - $2.4(200)^{1.05}$ = 626 staff-months
  - Semi-Detached
    - $3.0(200)^{1.12}$ = 1,133 staff-months
  - Embedded
    - $3.6(200)^{1.20}$ = 2,077 staff-months

1. Basic COCOMO Model (Constructive Cost Model)

Project Duration

$$TDEV = c(E)^d$$

- Where
  - TDEV is time for development
  - c and d are constants to be determined
  - E is the effort

1. Basic COCOMO Model (Constructive Cost Model)

Constants for TDEV

Mode	c	d
Organic	2.5	0.38
Semi-detached	2.5	0.35
Embedded	2.5	0.32

1. Basic COCOMO Model (Constructive Cost Model)

Example

- Picking up from the last example,
  - Organic
    - E = 626 staff months
    - TDEV = $2.5(626)^{0.38}$ = 29 months
  - Semi-detached
    - E = 1,133
    - TDEV = $2.5(1133)^{0.35}$ = 29 months
  - Embedded
    - E = 2077
    - TDEV = $2.5(2077)^{0.32}$ = 29 months

1. Basic COCOMO Model (Constructive Cost Model)

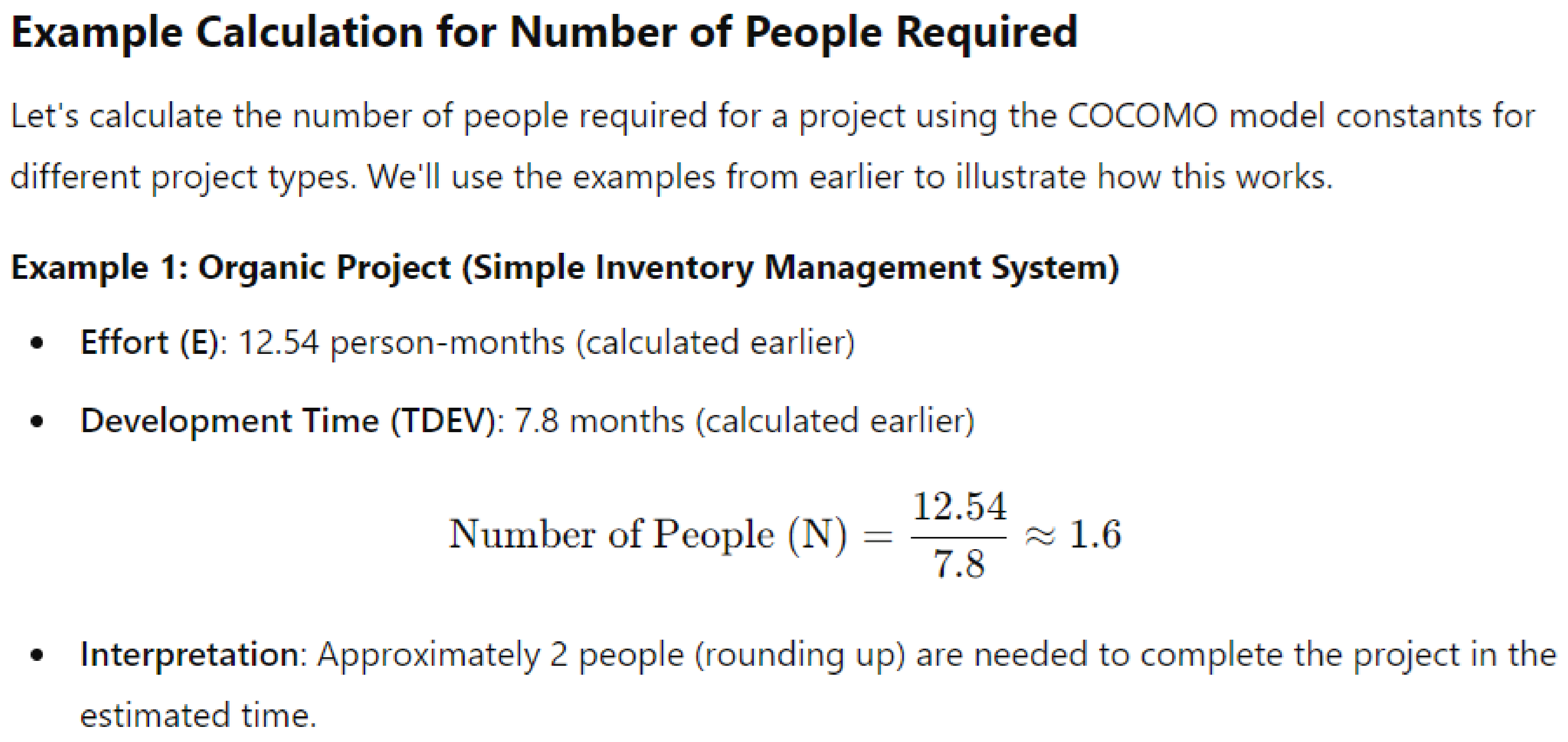
To determine the **number of people required** for a project using the COCOMO model, you can use the following relationship between **Effort (E)** and **Development Time (TDEV)**:

$$\text{Number of People (N)} = \frac{\text{Effort (E)}}{\text{Development Time (TDEV)}}$$

Where:

- **Effort (E)** is calculated in **person-months**.
- **Development Time (TDEV)** is the estimated project duration in **months**.

1. Basic COCOMO Model (Constructive Cost Model)



Example Calculation for Number of People Required

Let's calculate the number of people required for a project using the COCOMO model constants for different project types. We'll use the examples from earlier to illustrate how this works.

Example 1: Organic Project (Simple Inventory Management System)

- **Effort (E)**: 12.54 person-months (calculated earlier)
- **Development Time (TDEV)**: 7.8 months (calculated earlier)

$$\text{Number of People (N)} = \frac{12.54}{7.8} \approx 1.6$$

- **Interpretation**: Approximately 2 people (rounding up) are needed to complete the project in the estimated time.

1. Basic COCOMO Model (Constructive Cost Model)

Example 2: Semi-Detached Project (Healthcare Management System)

- **Effort (E)**: 195.3 person-months
- **Development Time (TDEV)**: 13.2 months

$$\text{Number of People (N)} = \frac{195.3}{13.2} \approx 14.8$$

- **Interpretation**: Approximately 15 people are needed for this project to complete it in the given time frame.

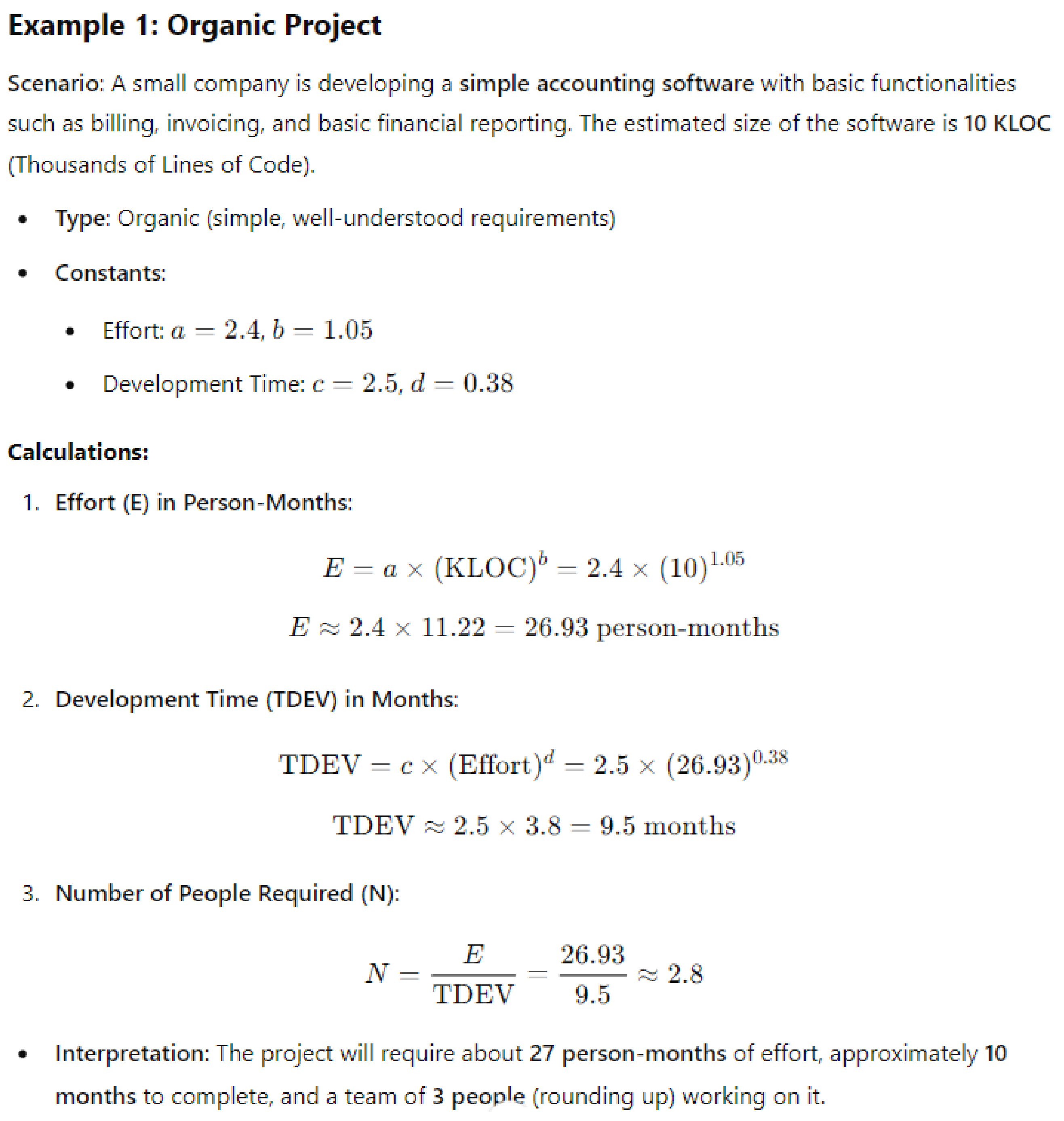
Example 3: Embedded Project (Real-Time Control System for Drones)

- **Effort (E)**: 1263.9 person-months
- **Development Time (TDEV)**: 27.8 months

$$\text{Number of People (N)} = \frac{1263.9}{27.8} \approx 45.5$$

- **Interpretation**: Approximately 46 people are required for the project to be completed within the estimated time.

1. Basic COCOMO Model (Constructive Cost Model)



Example 1: Organic Project

Scenario: A small company is developing a **simple accounting software** with basic functionalities such as billing, invoicing, and basic financial reporting. The estimated size of the software is **10 KLOC** (Thousands of Lines of Code).

- **Type:** Organic (simple, well-understood requirements)
- **Constants:**
  - Effort: $a = 2.4$, $b = 1.05$
  - Development Time: $c = 2.5$, $d = 0.38$

Calculations:

1. **Effort (E) in Person-Months:**

$$E = a \times (\text{KLOC})^b = 2.4 \times (10)^{1.05}$$

$$E \approx 2.4 \times 11.22 = 26.93 \text{ person-months}$$

2. **Development Time (TDEV) in Months:**

$$\text{TDEV} = c \times (\text{Effort})^d = 2.5 \times (26.93)^{0.38}$$

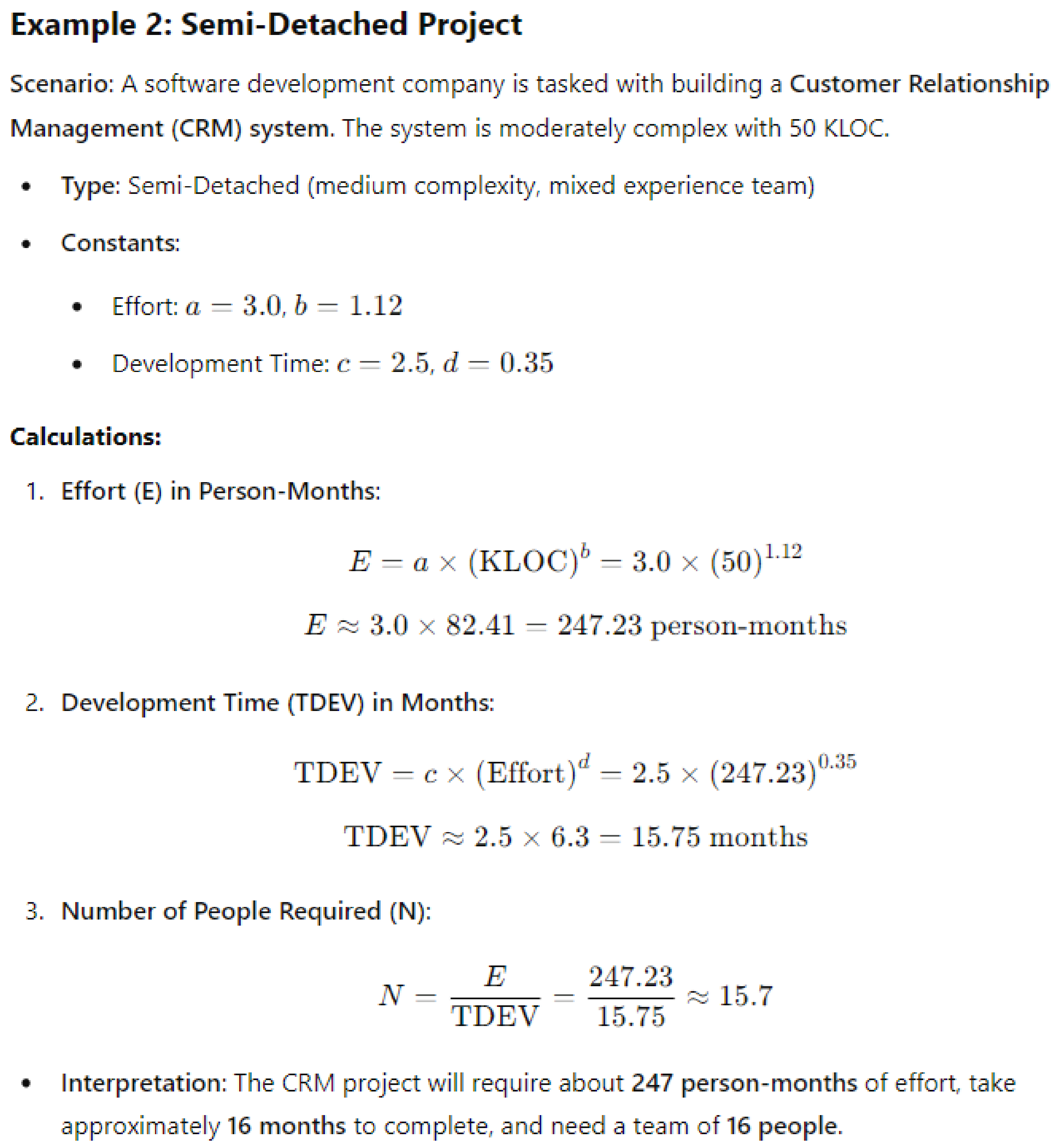
$$\text{TDEV} \approx 2.5 \times 3.8 = 9.5 \text{ months}$$

3. **Number of People Required (N):**

$$N = \frac{E}{\text{TDEV}} = \frac{26.93}{9.5} \approx 2.8$$

- **Interpretation:** The project will require about **27 person-months** of effort, approximately **10 months** to complete, and a team of **3 people** (rounding up) working on it.

1. Basic COCOMO Model (Constructive Cost Model)



Example 2: Semi-Detached Project

Scenario: A software development company is tasked with building a **Customer Relationship Management (CRM) system**. The system is moderately complex with 50 KLOC.

- **Type:** Semi-Detached (medium complexity, mixed experience team)
- **Constants:**
  - Effort: $a = 3.0$, $b = 1.12$
  - Development Time: $c = 2.5$, $d = 0.35$

Calculations:

1. **Effort (E) in Person-Months:**

$$E = a \times (\text{KLOC})^b = 3.0 \times (50)^{1.12}$$

$$E \approx 3.0 \times 82.41 = 247.23 \text{ person-months}$$

2. **Development Time (TDEV) in Months:**

$$\text{TDEV} = c \times (\text{Effort})^d = 2.5 \times (247.23)^{0.35}$$

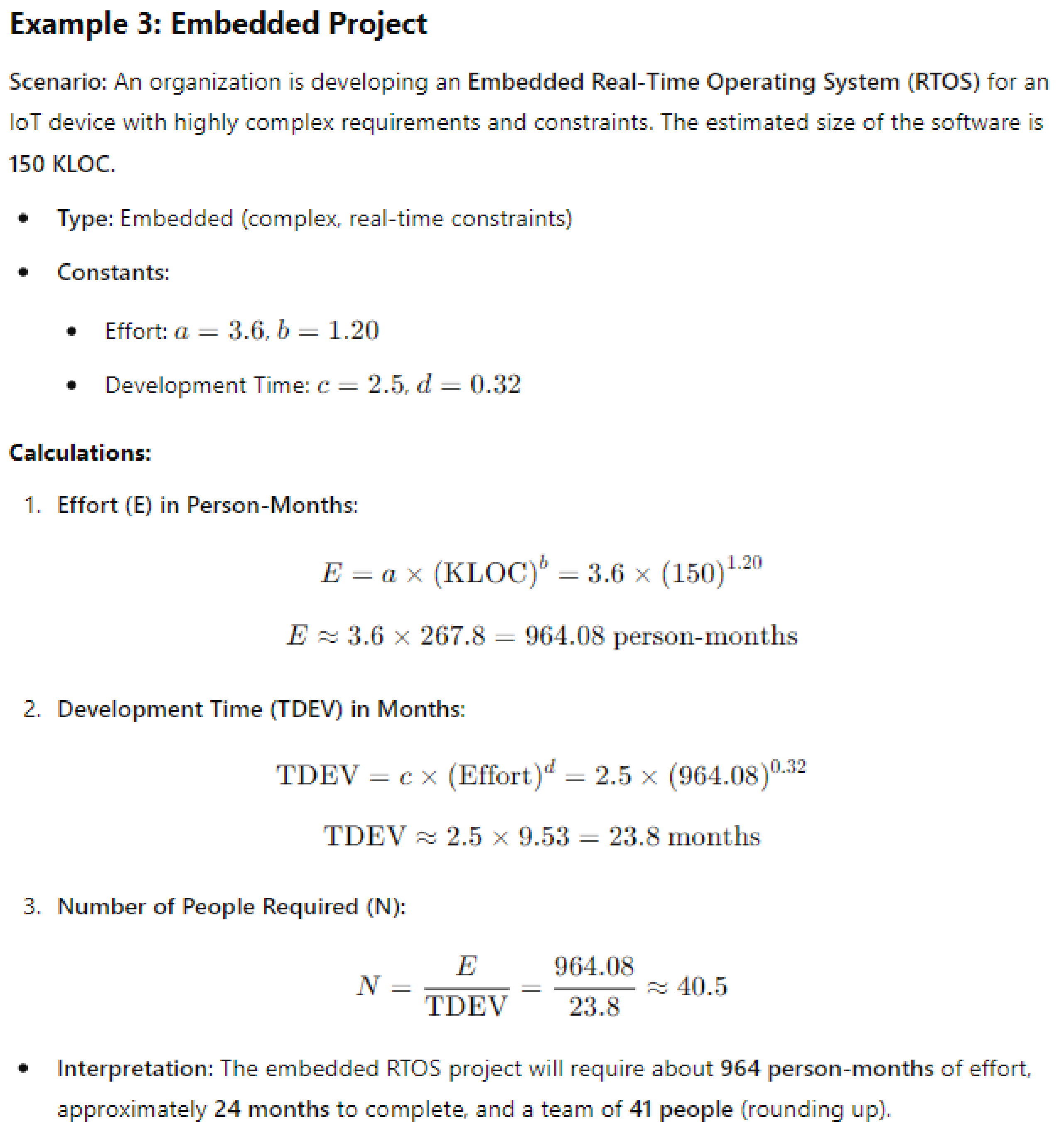
$$\text{TDEV} \approx 2.5 \times 6.3 = 15.75 \text{ months}$$

3. **Number of People Required (N):**

$$N = \frac{E}{\text{TDEV}} = \frac{247.23}{15.75} \approx 15.7$$

- **Interpretation:** The CRM project will require about **247 person-months** of effort, take approximately **16 months** to complete, and need a team of **16 people.**

1. Basic COCOMO Model (Constructive Cost Model)



Example 3: Embedded Project

Scenario: An organization is developing an **Embedded Real-Time Operating System (RTOS)** for an IoT device with highly complex requirements and constraints. The estimated size of the software is **150 KLOC**.

- **Type:** Embedded (complex, real-time constraints)
- **Constants:**
  - Effort: $a = 3.6$, $b = 1.20$
  - Development Time: $c = 2.5$, $d = 0.32$

Calculations:

1. **Effort (E) in Person-Months:**

$$E = a \times (\text{KLOC})^b = 3.6 \times (150)^{1.20}$$

$$E \approx 3.6 \times 267.8 = 964.08 \text{ person-months}$$

2. **Development Time (TDEV) in Months:**

$$\text{TDEV} = c \times (\text{Effort})^d = 2.5 \times (964.08)^{0.32}$$

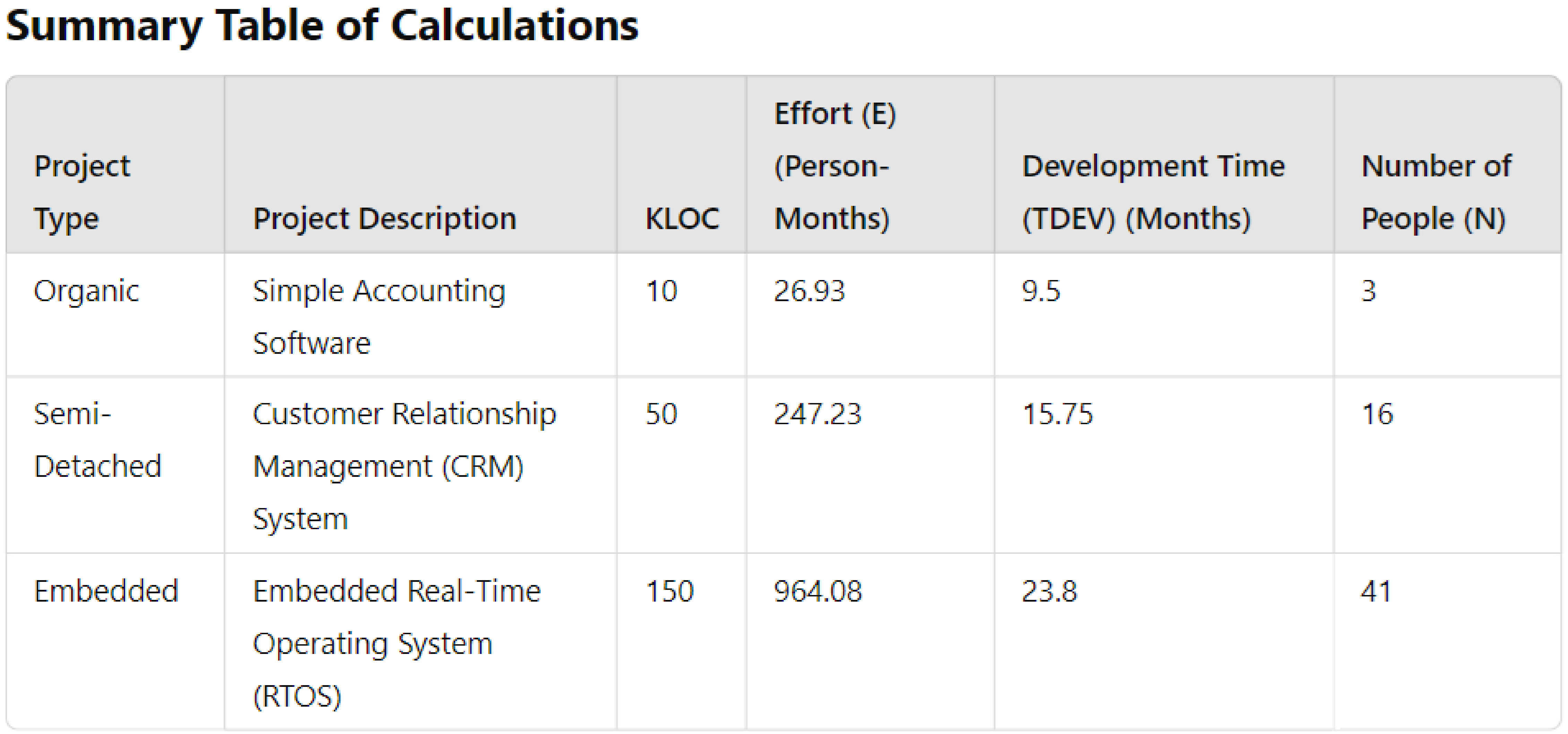
$$\text{TDEV} \approx 2.5 \times 9.53 = 23.8 \text{ months}$$

3. **Number of People Required (N):**

$$N = \frac{E}{\text{TDEV}} = \frac{964.08}{23.8} \approx 40.5$$

- **Interpretation:** The embedded RTOS project will require about **964 person-months** of effort, approximately **24 months** to complete, and a team of **41 people** (rounding up).

1. Basic COCOMO Model (Constructive Cost Model)



Summary Table of Calculations

Project Type	Project Description	KLOC	Effort (E) (Person-Months)	Development Time (TDEV) (Months)	Number of People (N)
Organic	Simple Accounting Software	10	26.93	9.5	3
Semi-Detached	Customer Relationship Management (CRM) System	50	247.23	15.75	16
Embedded	Embedded Real-Time Operating System (RTOS)	150	964.08	23.8	41

2. Intermediate COCOMO

- It is used for medium sized projects.
- The cost drivers are intermediate to basic and advanced cocomo.
- Cost drivers depend upon product reliability, database size, execution and storage.
- Team size is medium.

3. Advanced COCOMO

- It is used for large sized projects.
- The cost drivers depend upon requirements, analysis, design, testing and maintenance.
- Team size is large.

COCOMO-II Model

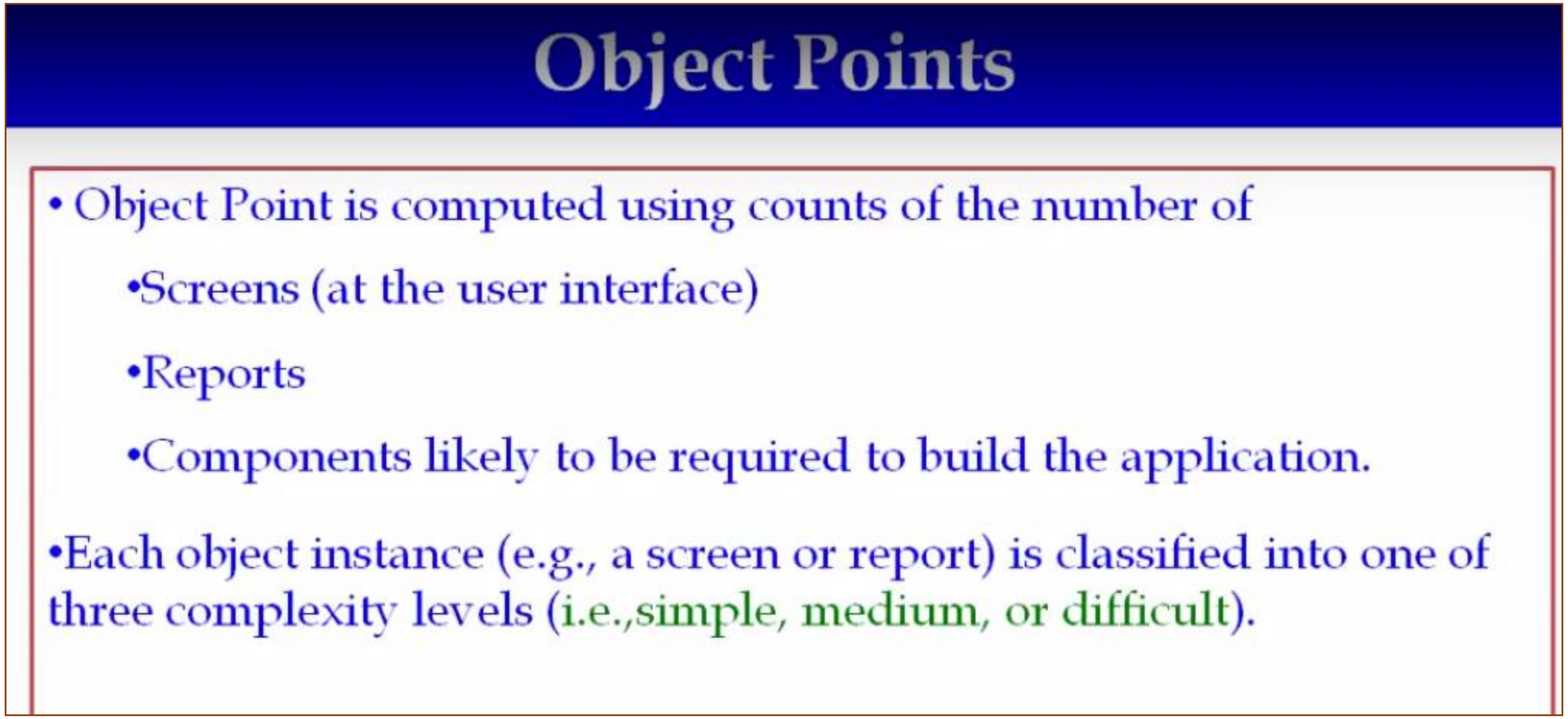
- COnstructive COst Model - II by Barry Boehm

- Application composition model - Used during the early stages of software engineering

- Early design stage model - used once requirements have been stabilized and basic software architecture has been established.

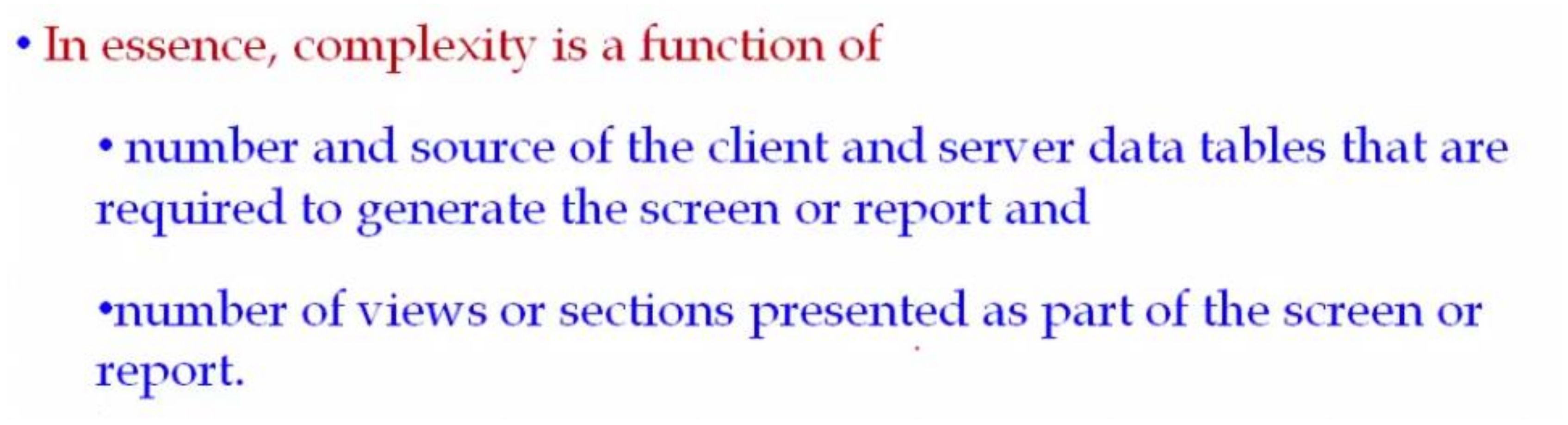
- Post-architecture-stage model - Used during the construction of the software

- The sizing information followed by this model is the indirect software measure *object points.*

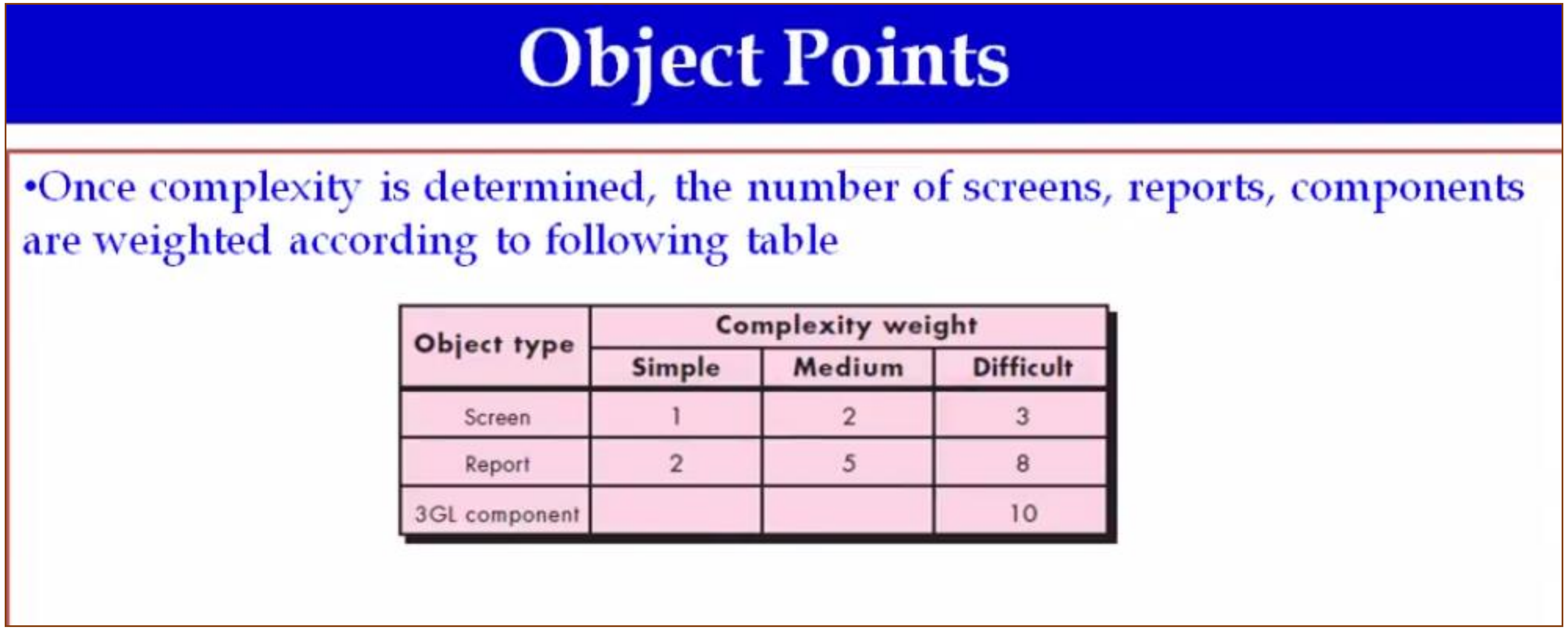


Object Points

- Object Point is computed using counts of the number of
  - Screens (at the user interface)
  - Reports
  - Components likely to be required to build the application.
- Each object instance (e.g., a screen or report) is classified into one of three complexity levels (i.e.,simple, medium, or difficult).



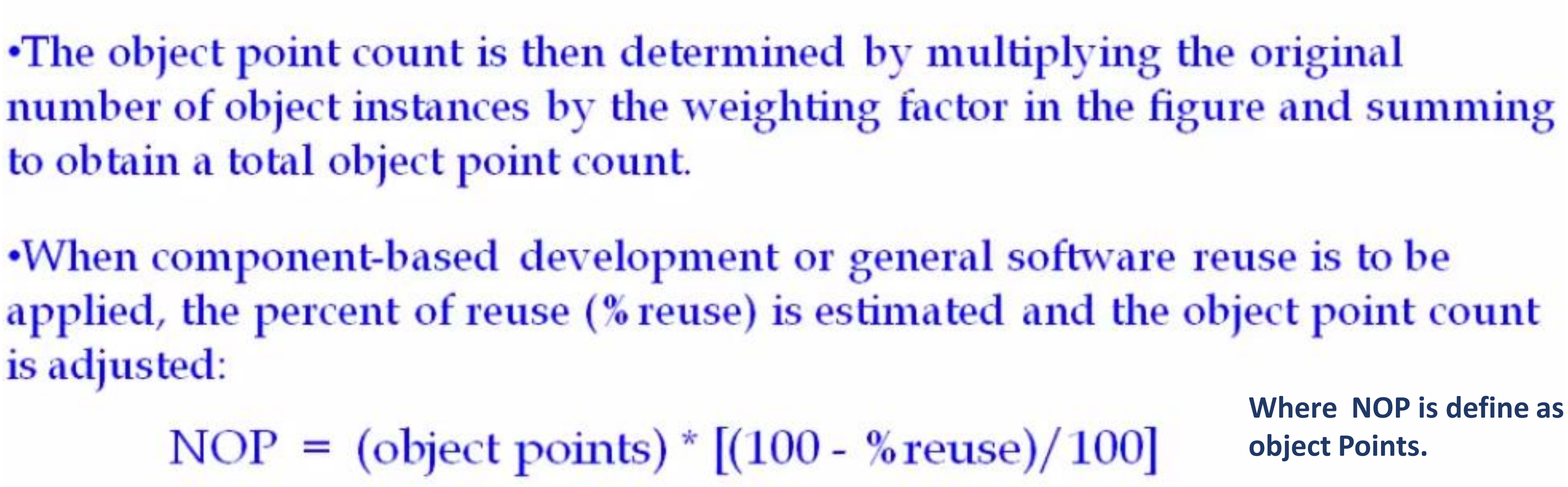
- In essence, complexity is a function of
  - number and source of the client and server data tables that are required to generate the screen or report and
  - number of views or sections presented as part of the screen or report.



Object Points

•Once complexity is determined, the number of screens, reports, components are weighted according to following table

Object type	Complexity weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component			10



•The object point count is then determined by multiplying the original number of object instances by the weighting factor in the figure and summing to obtain a total object point count.

•When component-based development or general software reuse is to be applied, the percent of reuse (%reuse) is estimated and the object point count is adjusted:

$$\text{NOP} = (\text{object points}) * [(100 - \%\text{reuse})/100]$$

Where NOP is define as new object Points.

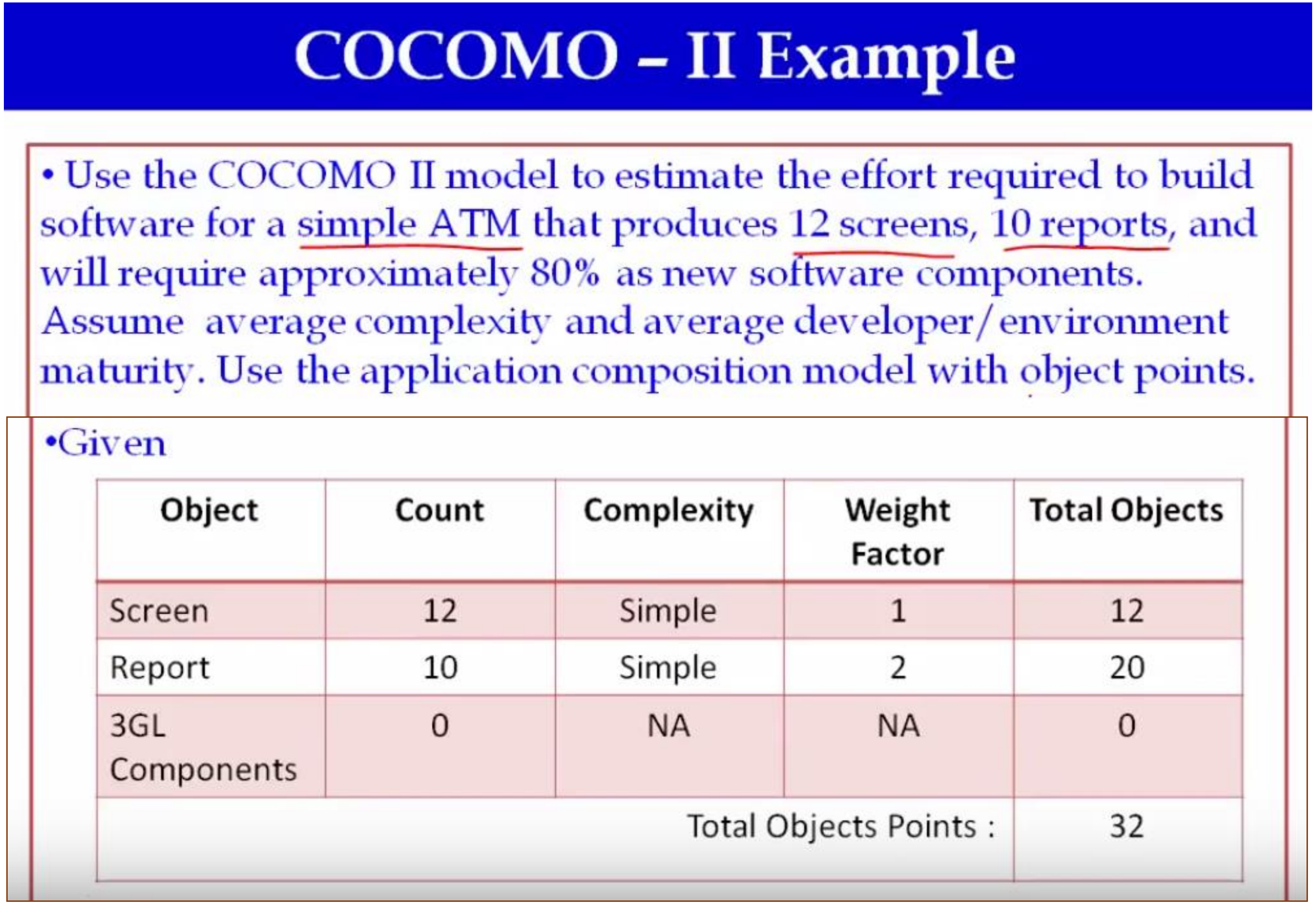
Effort Estimation

- Now the estimate of project effort is computed as follows.

$$\text{Estimated effort} = \frac{\text{NOP}}{\text{PROD}}$$

- Productivity rate can be derived from following table based on developer experience and organization maturity.

Developer's experience/capability	Very low	Low	Nominal	High	Very high
Environment maturity/capability	Very low	Low	Nominal	High	Very high
PROD	4	7	13	25	50



COCOMO – II Example

- Use the COCOMO II model to estimate the effort required to build software for a simple ATM that produces 12 screens, 10 reports, and will require approximately 80% as new software components. Assume average complexity and average developer/environment maturity. Use the application composition model with object points.

- Given

Object	Count	Complexity	Weight Factor	Total Objects
Screen	12	Simple	1	12
Report	10	Simple	2	20
3GL Components	0	NA	NA	0
			Total Objects Points :	32

COCOMO – II Example

• It is given that 80% of components have to be newly developed. So remaining 20% can be reused

•Now compute new object points as

NOP = (object points) * [(100 - %reuse)/100]

NOP = 32 * (100-20)/100 = 32*80 / 100

NOP = 25. 6 object points

• Since productivity is given average, we can assume PROD = 13

•Hence , effort = NOP/PROD

effort = 25.6/13

effort = 1.96 person months

COCOMO – II Example 2

An airline sales system is to be built in C. This is a new project and the back-end database server has been built. At the early stage, we need 3 screens and 1 report:

1. a booking screen to record a new advertising sale booking
2. a pricing screen showing the advertising rate for each day and each flight
3. an availability screen showing which flights are available
4. a sales report showing total sales for the month and year, and comparing them with previous months and years

The booking screen requires 3 data tables, namely, the table of customer contact details, the table that records the past history of the customer, and the table of available time slots. Only 1 view of the screen is enough. So, the booking screen is classified as simple. Similarly, the levels of difficulty of the pricing screen, the availability screen and the sales report are classified as simple, simple and medium, respectively. There is no 3GL component.

Name	**Objects**	**Complexity**	**Weight**
Booking	Screen	Simple	1
Pricing	Screen	Simple	1
Availability	Screen	Medium	2
Sales	Report	Medium	5
		Total	9

The assessment on the developers and the environment shows that the developers' experience is very low (4) and the CASE tool is low (7). So, we have a productivity rate of 5.5.

According to COCOMO II, the project requires approx. 1.64 (= 9/5.5) person-months.

Merits of the COCOMO Model

- **Provides Granularity**: Can offer estimations at different stages and levels (Basic, Intermediate, Detailed).
- **Adaptable to Different Project Types**: Suitable for a wide range of projects, from simple to complex.
- **Improves Planning and Scheduling**: Helps in effective project planning and allocation of resources.

Demerits of the COCOMO Model

- **Outdated for Modern Agile Practices**: Assumes a waterfall or sequential approach.
- **Dependent on LOC**: Line of code estimates can be highly variable and unreliable early in the project.
- **Limited Consideration of Non-Technical Factors**: Factors like market conditions, competitive pressures, and user acceptance are not covered.

Function Point Analysis Technique for Software Cost Estimation

Software Cost Components

- Hardware and software costs
- Travel and training costs
- Effort costs (the dominant factor in most projects)
  - salaries of engineers involved in the project
  - Social and insurance costs
- Effort costs must take overheads into account
  - costs of building, heating, lighting
  - costs of networking and communications
  - costs of shared facilities (e.g library, staff restaurant, etc.)

Introduction

- Increasingly important facet of software development is the ability to estimate the associated cost of development early in the development process.
- Estimating software size is a difficult problem that requires specific knowledge of the system functions in terms of
  - Scope
  - Complexity
  - Interactions
- Most frequently cited sources of software size metrics are
  - Lines of code
  - Function point analysis

Problems with Lines of Code

- Lack of a universally accepted definition for exactly what a line of code is
- Language dependence (high-level versus low-level programming languages)
- It is difficult to estimate the number of lines of code that will be needed to develop a system from information that is available in analysis and design phases
- Lines of code places all emphasis on coding, which is only part of the implementation phase of a software development project

What Are Function Points?

- Function Points measure software size by quantifying (counting) the functionality provided to the user based solely on logical design and functional specifications.
- Function point analysis is a method of quantifying the size and complexity of a software system in terms of the functions that the system delivers to the user
- It is independent of the computer language, development methodology, technology or capability of the project team used to develop the application.

What Are Function Points? …

- Function point analysis is designed to measure business applications (not scientific applications).

- Scientific applications generally deal with complex algorithms that the function point method is not designed to handle

What Are Function Points? …

- Simply stated, function points are a standard unit of measure that represent the functional size of a software application.
- In the same way that a house is measured by the square feet it provides, the size of an application can be measured by the number of function points it delivers to the users of the application.
- A good example is when I had my house built two years ago. I worked with a very straightforward home builder and he basically said "All, you have two choices here.
- First, how many square feet do you want to build? Second, what quality of materials do you want to use?" He continued "Let's say that you want to build a house that is 2,000 square feet.
- If you want to use cheap materials we can build it for $80 per square feet. That's $160,000. If you want to go top of the line then you're looking at more like $110 per square foot, and that's $220,000.

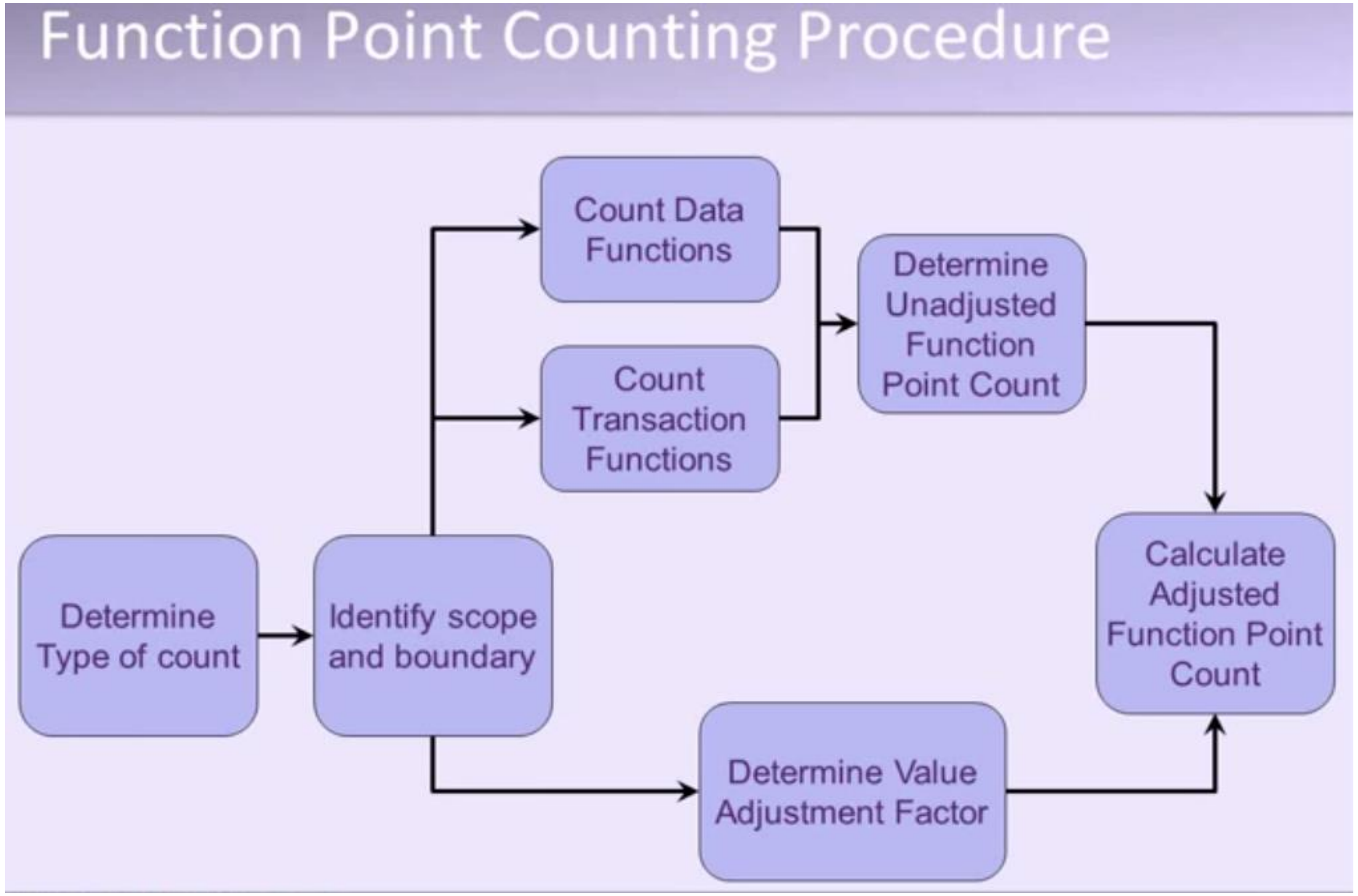
Important notes about the FPA process

- Measured from the user's perspective
  - The size of the application being measured is based on the user's view of the system.
  - It is based on what the user asked for, not what is delivered.
  - It's based on the way the user interacts with the system, including the screens that the user uses to enter input, and the reports the users receive as output.
  - Finally, it's also based on their understanding of the data that needs to be stored and processed by the system.
- Technology-independent
- Low cost
- Work well with use cases

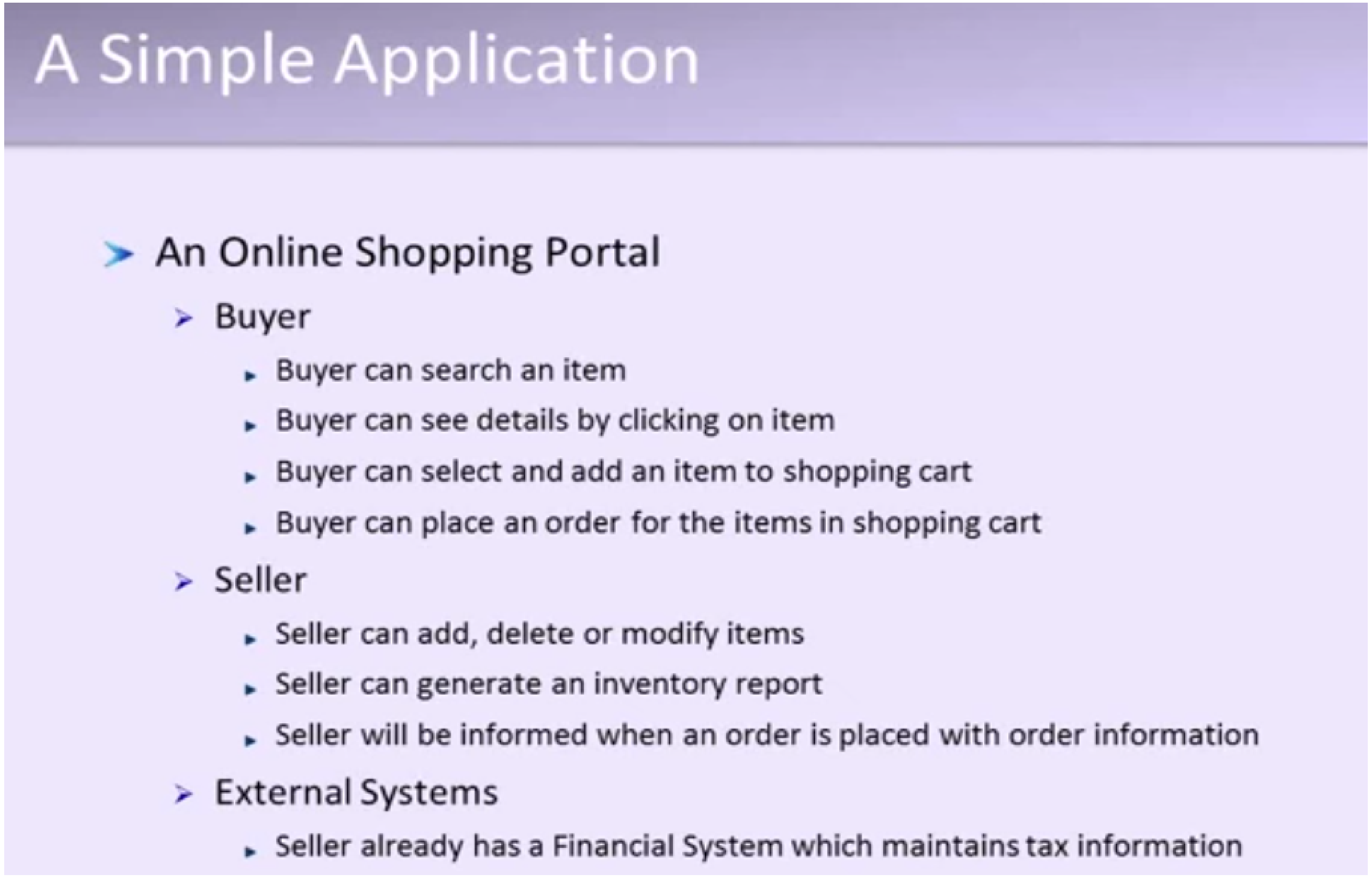
Function Point Counting Steps:

1. Determine the type of function point count
2. Identify the counting scope and application boundary
3. Determine the Unadjusted Function Point Count
4. Count Data Functions
5. Count Transactional Functions
6. Determine the Value Adjustment Factor
7. Calculate the Adjusted Function Point Count

Function Point Example


Function Point Counting Procedure
Determine Type of count
Identify scope and boundary
Count Data Functions
Count Transaction Functions
Determine Unadjusted Function Point Count
Determine Value Adjustment Factor
Calculate Adjusted Function Point Count

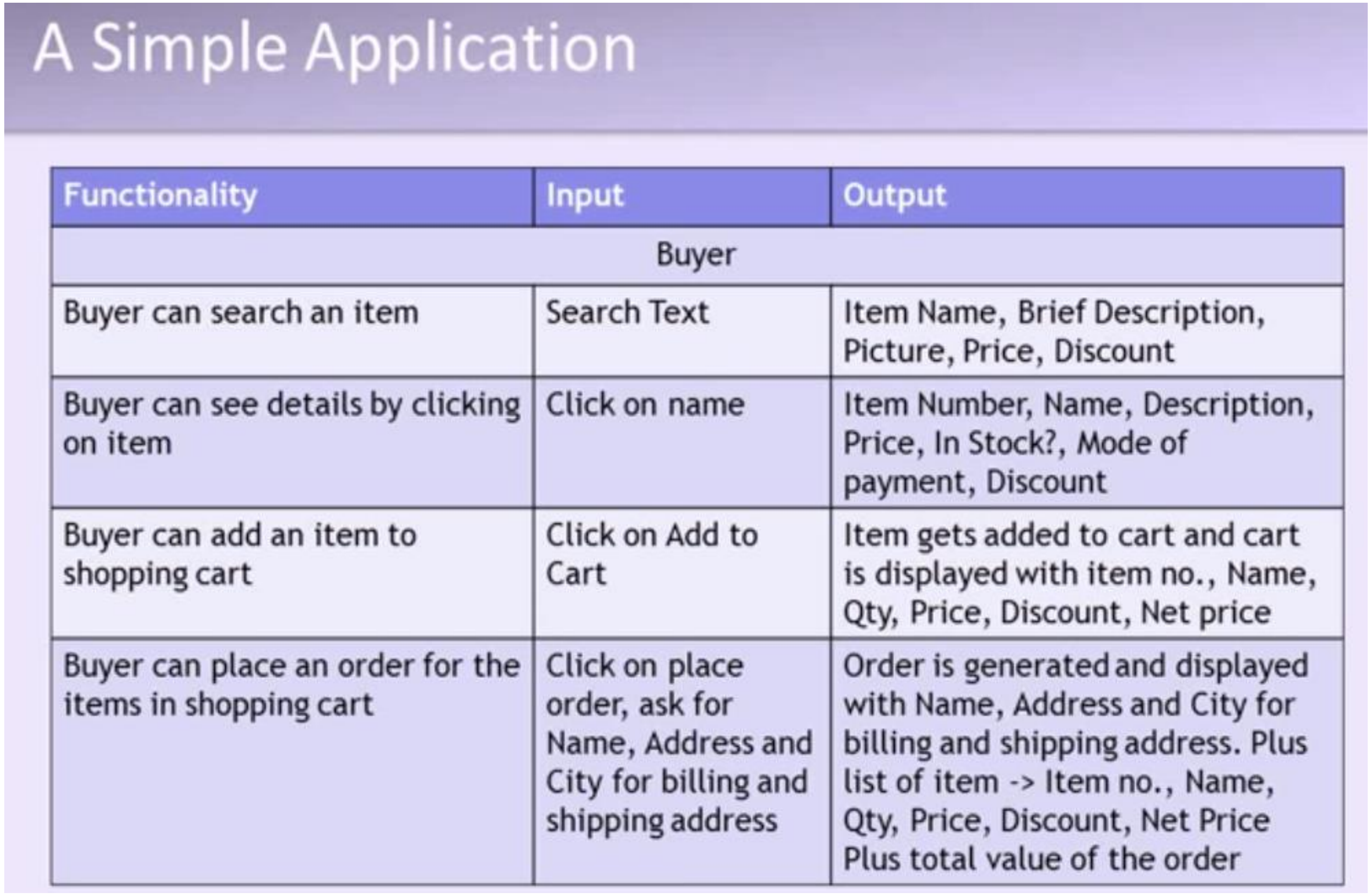
Function Point Example



A Simple Application

- An Online Shopping Portal
  - Buyer
    - Buyer can search an item
    - Buyer can see details by clicking on item
    - Buyer can select and add an item to shopping cart
    - Buyer can place an order for the items in shopping cart
  - Seller
    - Seller can add, delete or modify items
    - Seller can generate an inventory report
    - Seller will be informed when an order is placed with order information
  - External Systems
    - Seller already has a Financial System which maintains tax information

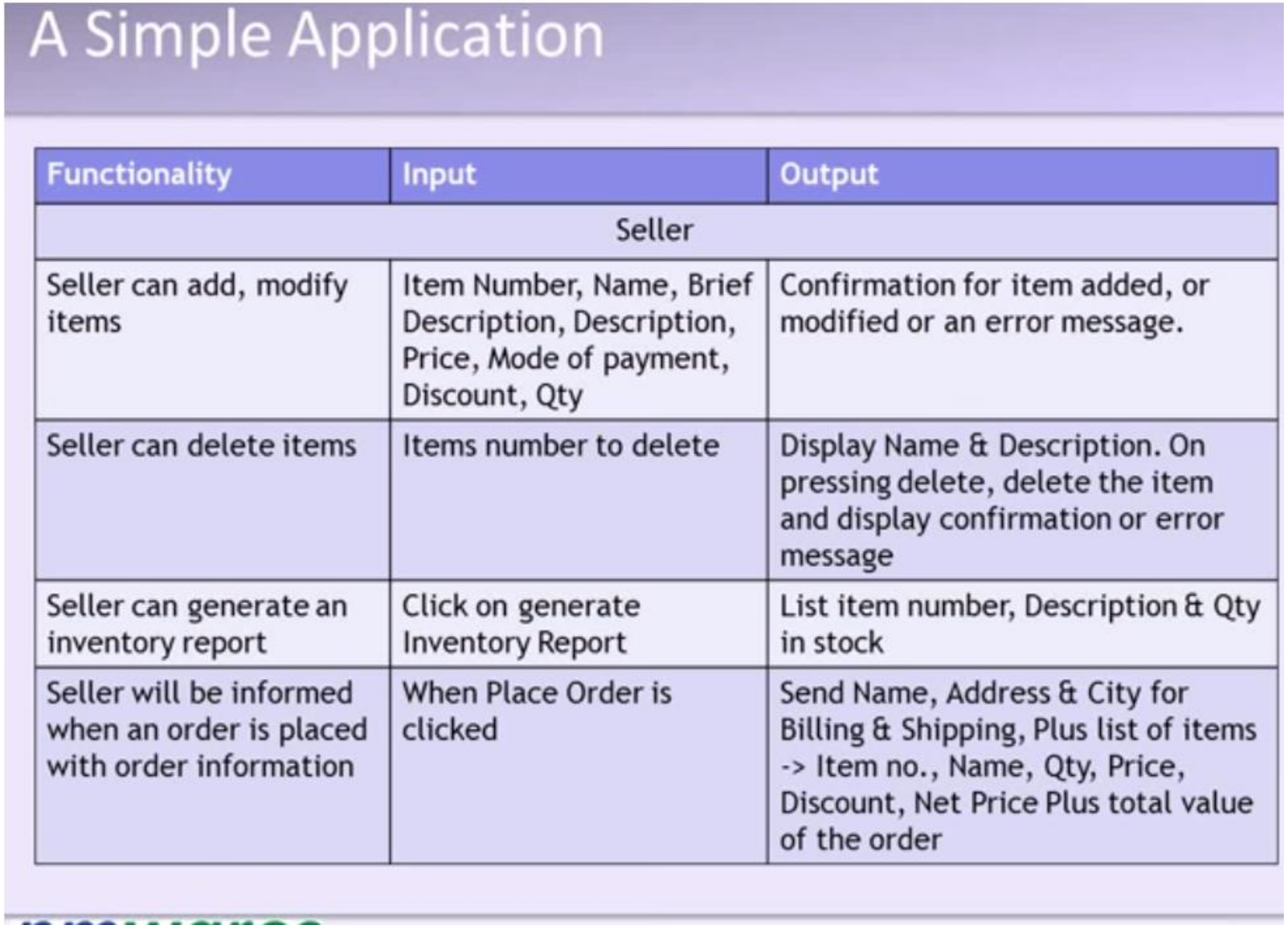
Function Point Example



A Simple Application

Functionality	Input	Output
Buyer		
Buyer can search an item	Search Text	Item Name, Brief Description, Picture, Price, Discount
Buyer can see details by clicking on item	Click on name	Item Number, Name, Description, Price, In Stock?, Mode of payment, Discount
Buyer can add an item to shopping cart	Click on Add to Cart	Item gets added to cart and cart is displayed with item no., Name, Qty, Price, Discount, Net price
Buyer can place an order for the items in shopping cart	Click on place order, ask for Name, Address and City for billing and shipping address	Order is generated and displayed with Name, Address and City for billing and shipping address. Plus list of item -> Item no., Name, Qty, Price, Discount, Net Price Plus total value of the order

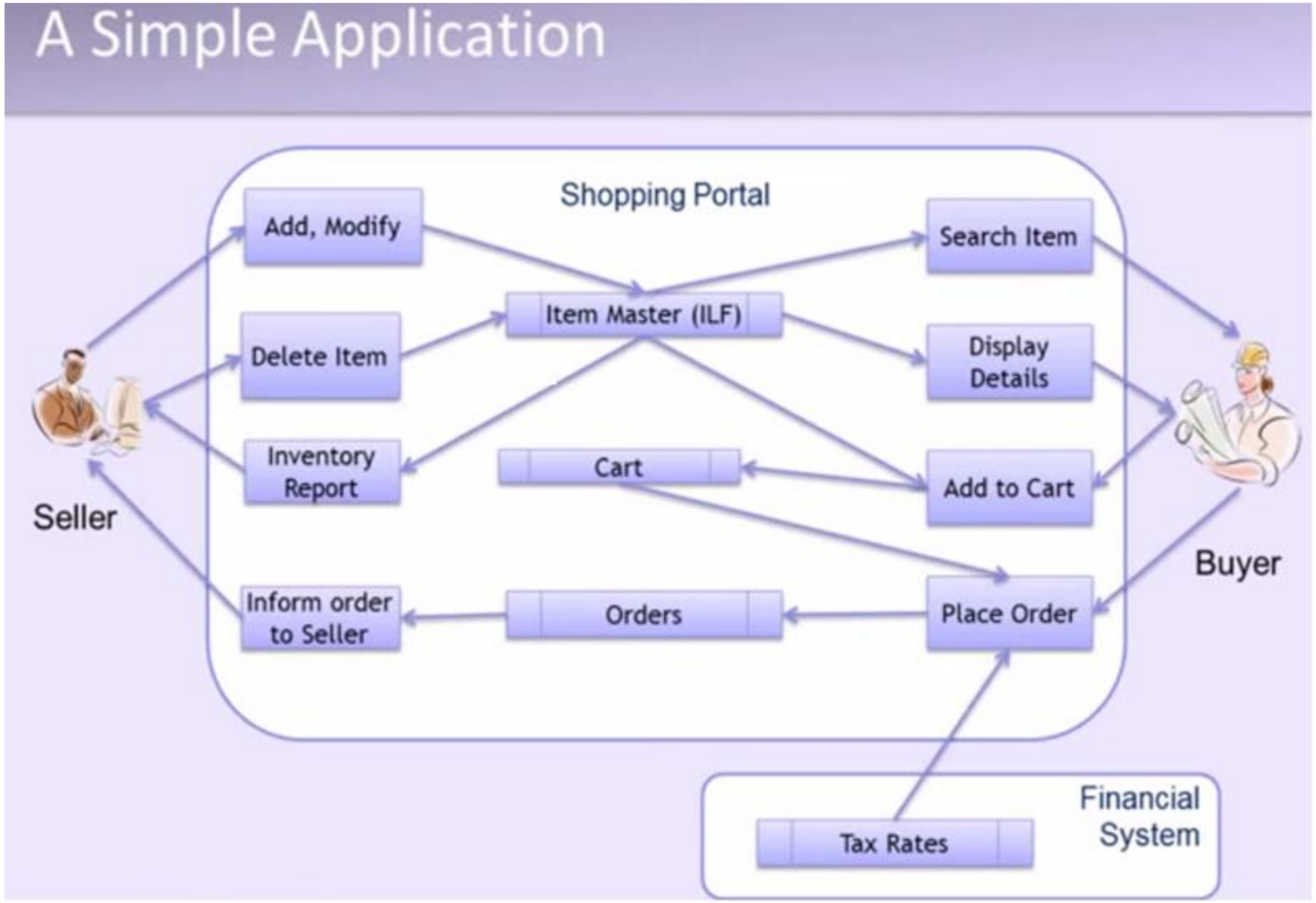
Function Point Example



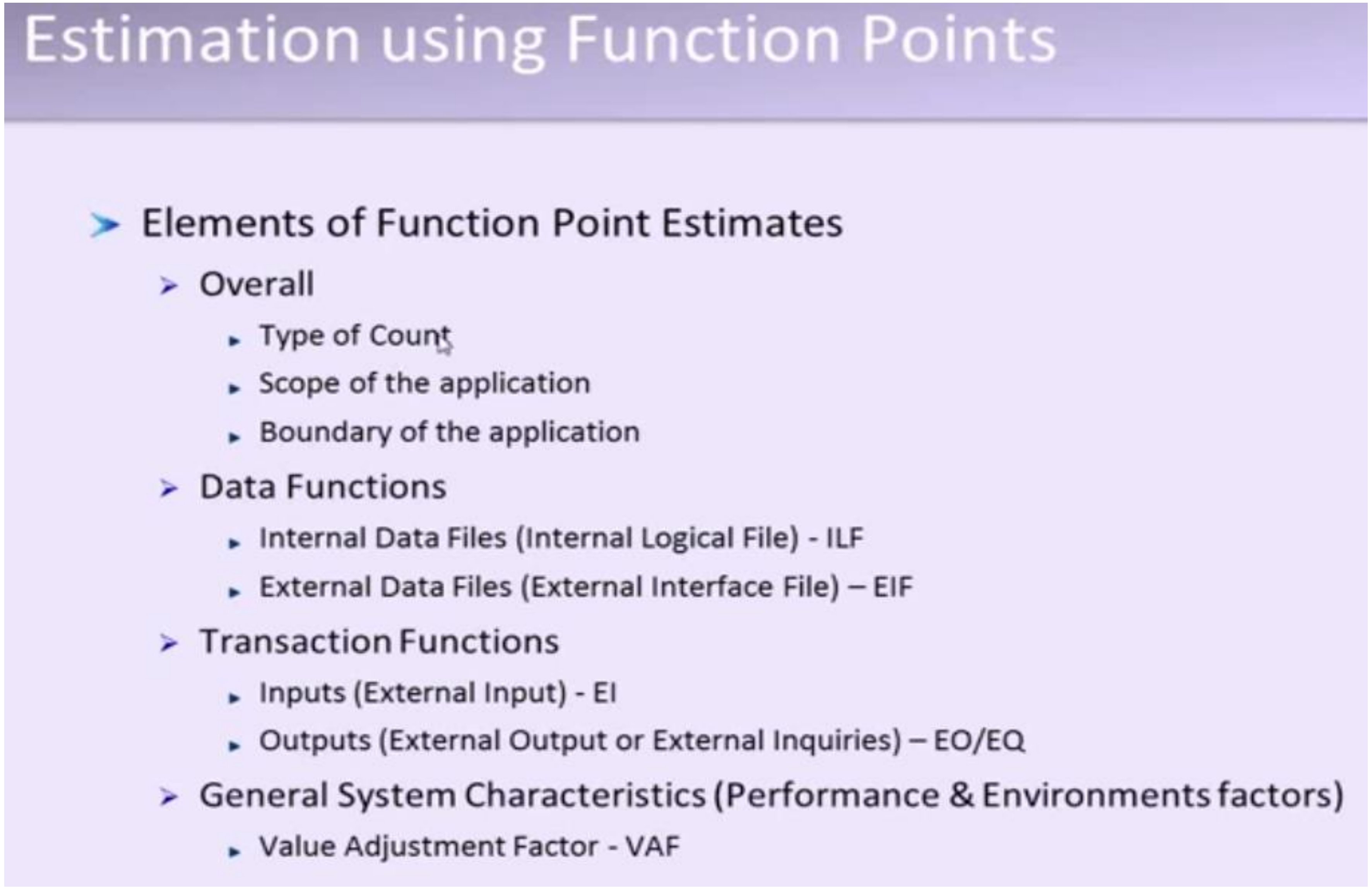
A Simple Application

Functionality	Input	Output
	Seller	
Seller can add, modify items	Item Number, Name, Brief Description, Description, Price, Mode of payment, Discount, Qty	Confirmation for item added, or modified or an error message.
Seller can delete items	Items number to delete	Display Name & Description. On pressing delete, delete the item and display confirmation or error message
Seller can generate an inventory report	Click on generate Inventory Report	List item number, Description & Qty in stock
Seller will be informed when an order is placed with order information	When Place Order is clicked	Send Name, Address & City for Billing & Shipping, Plus list of items -> Item no., Name, Qty, Price, Discount, Net Price Plus total value of the order

Function Point Example



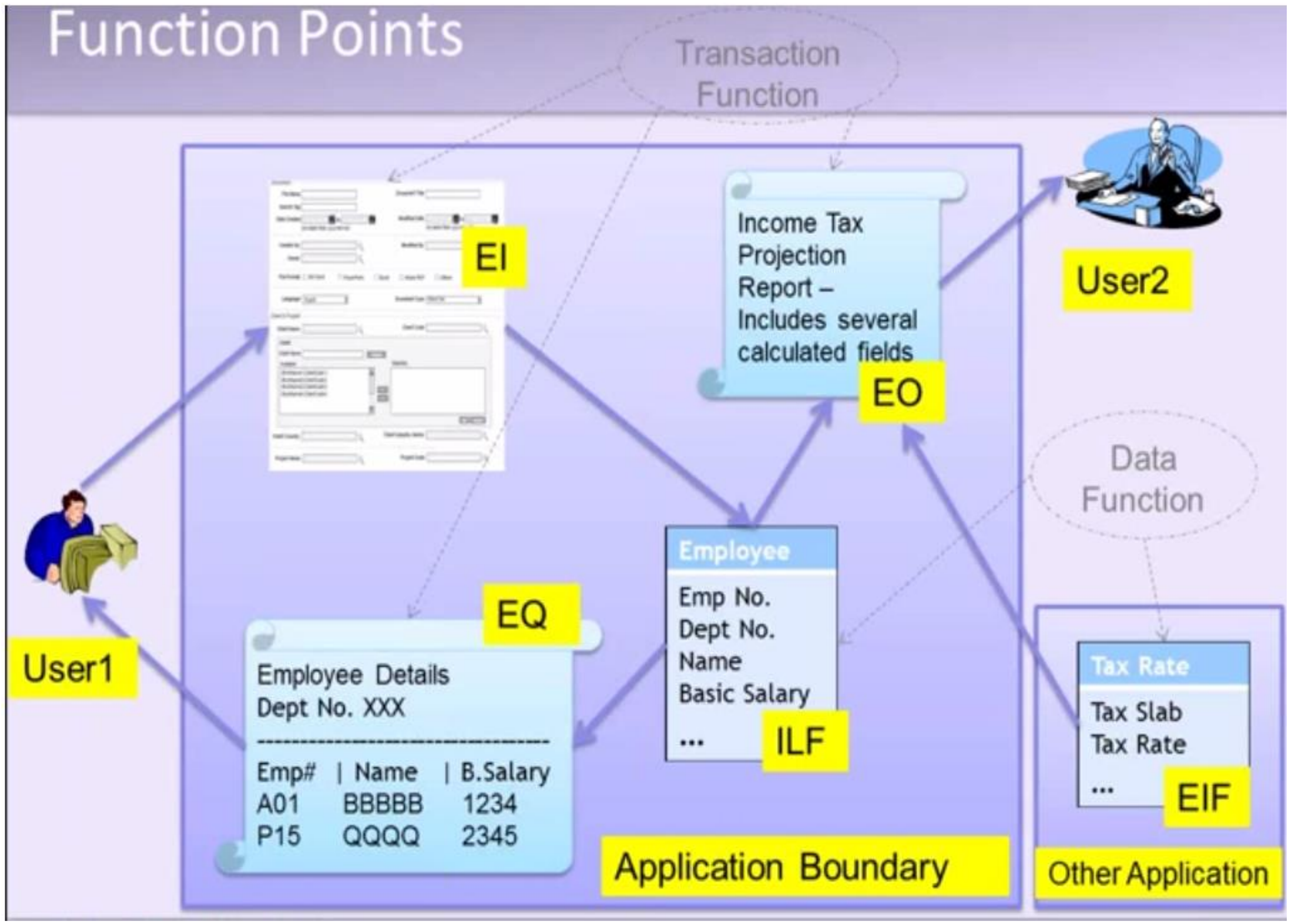
Function Point Example



Estimation using Function Points

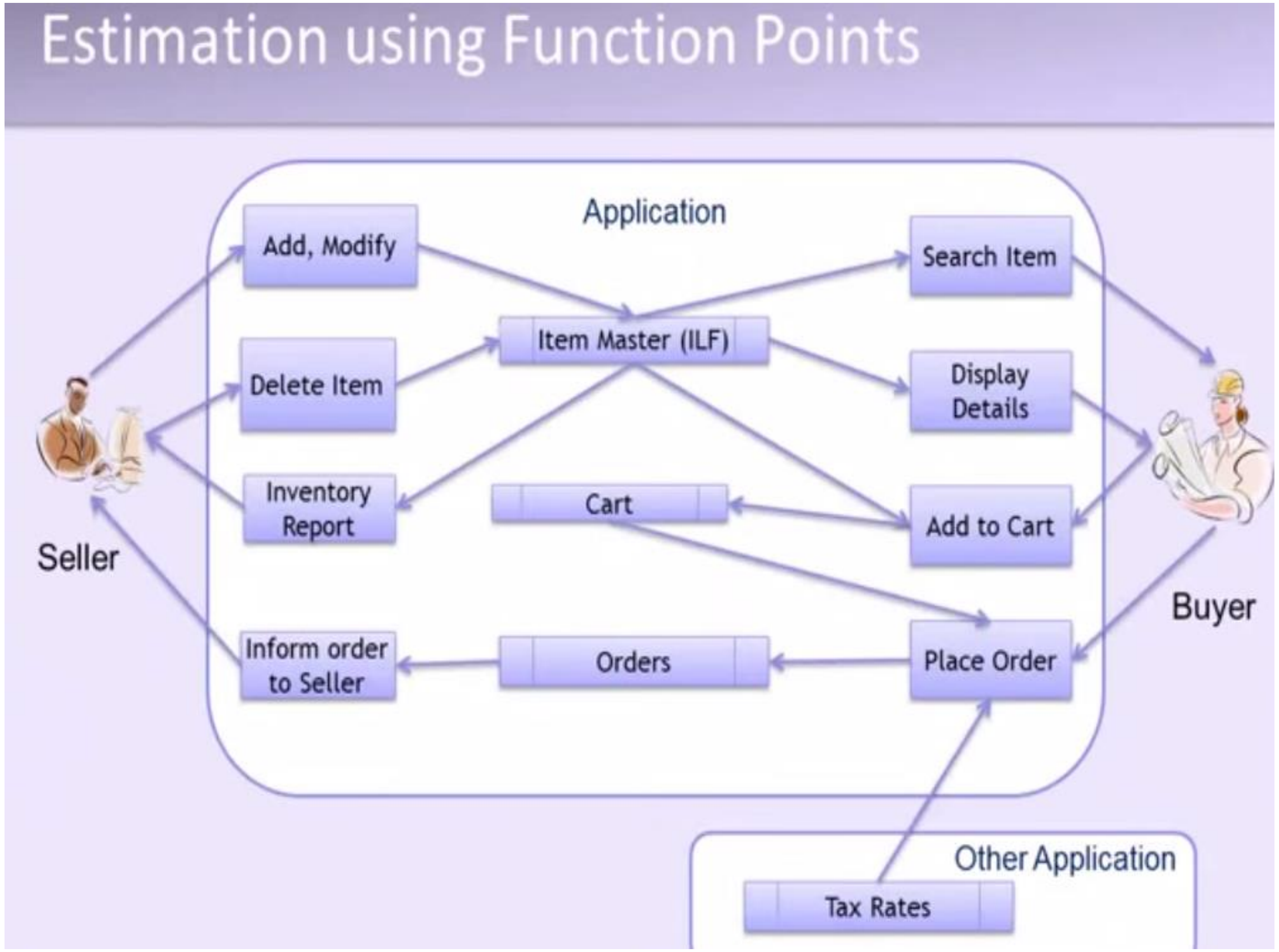
- Elements of Function Point Estimates
  - Overall
    - Type of Count
    - Scope of the application
    - Boundary of the application
  - Data Functions
    - Internal Data Files (Internal Logical File) - ILF
    - External Data Files (External Interface File) – EIF
  - Transaction Functions
    - Inputs (External Input) - EI
    - Outputs (External Output or External Inquiries) – EO/EQ
  - General System Characteristics (Performance & Environments factors)
    - Value Adjustment Factor - VAF

Function Point Example

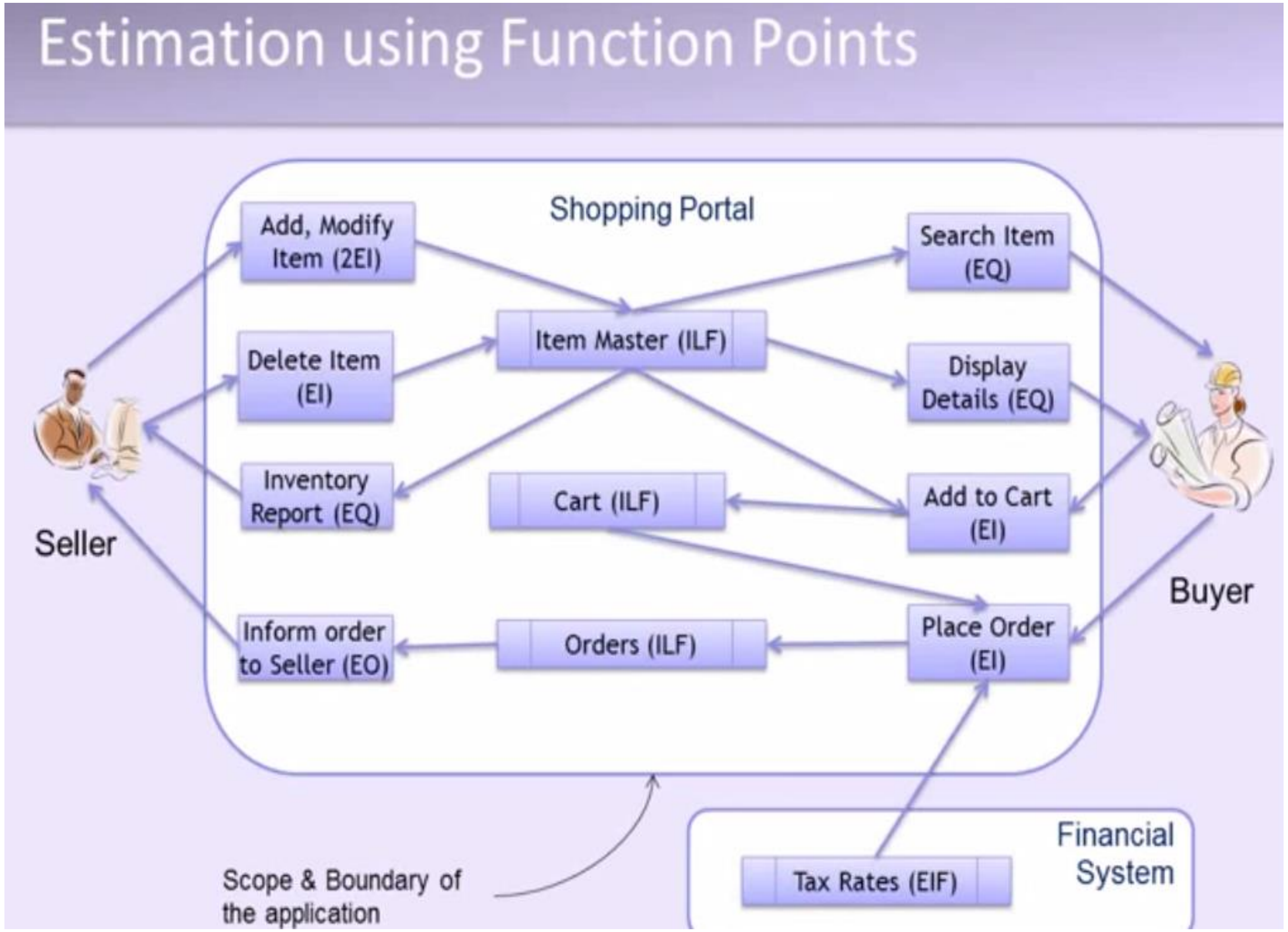


- **External Inputs (EI) :** Elementary process in which data crosses boundary from outside to inside.
- **External Outputs (EO) :** Processed information (calculation) result to user.
- **External Inquiries (EQ) :** Unprocessed information (no calculation) to user .
- **Internal Logical Files (ILF) :** Application can modify data. With in the boundary.
- **External Interface Files (EIF) :** Application can't modify data, it is just read only. Developed and maintain by others and use by us. Usually comes from other system.

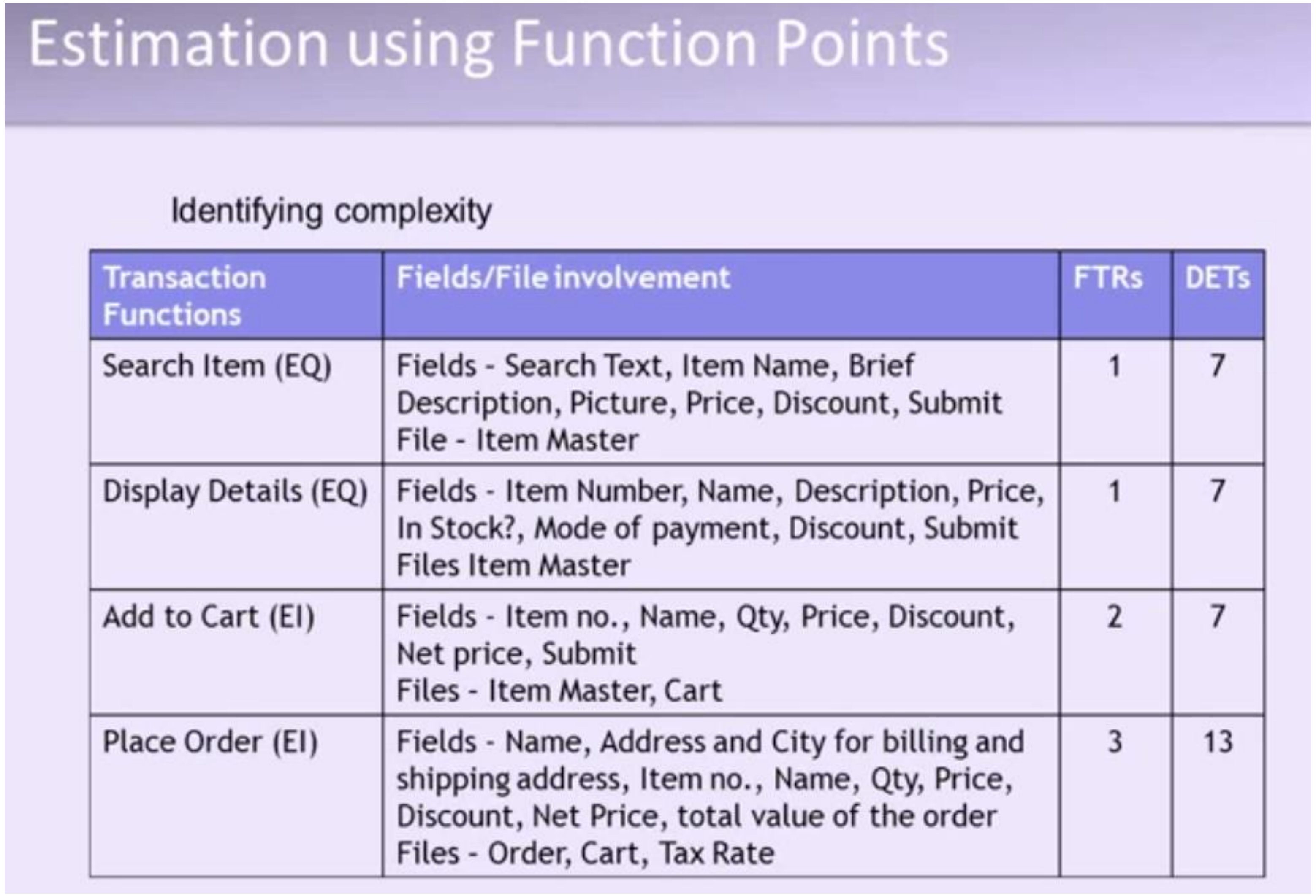
Function Point Example



Function Point Example



Function Point Example



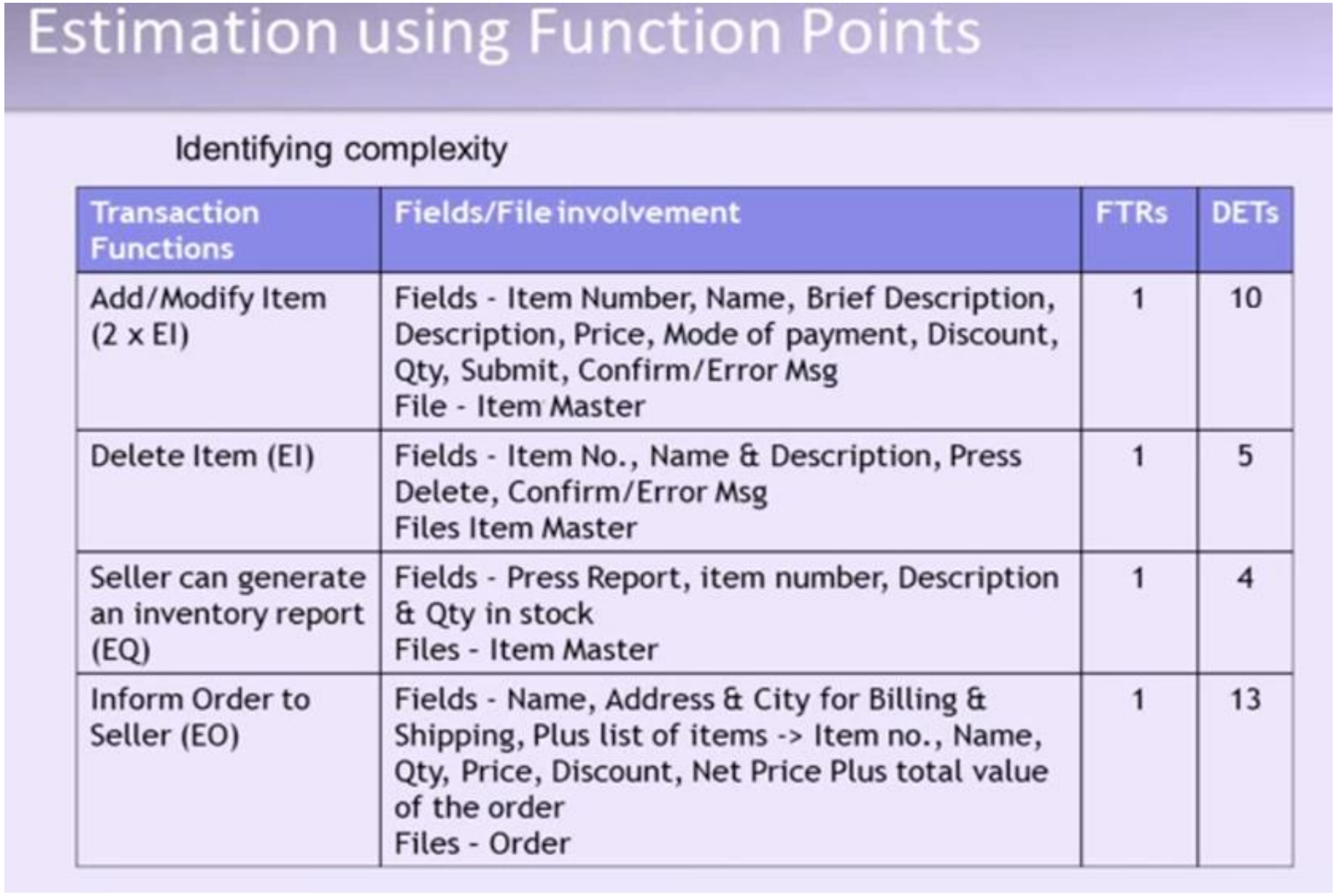
Estimation using Function Points

Identifying complexity

Transaction Functions	Fields/File involvement	FTRs	DETs
Search Item (EQ)	Fields - Search Text, Item Name, Brief Description, Picture, Price, Discount, Submit File - Item Master	1	7
Display Details (EQ)	Fields - Item Number, Name, Description, Price, In Stock?, Mode of payment, Discount, Submit Files Item Master	1	7
Add to Cart (EI)	Fields - Item no., Name, Qty, Price, Discount, Net price, Submit Files - Item Master, Cart	2	7
Place Order (EI)	Fields - Name, Address and City for billing and shipping address, Item no., Name, Qty, Price, Discount, Net Price, total value of the order Files - Order, Cart, Tax Rate	3	13

- File Type Reference (FTR), every file
- Data Element Type (DET) , every field

Function Point Example



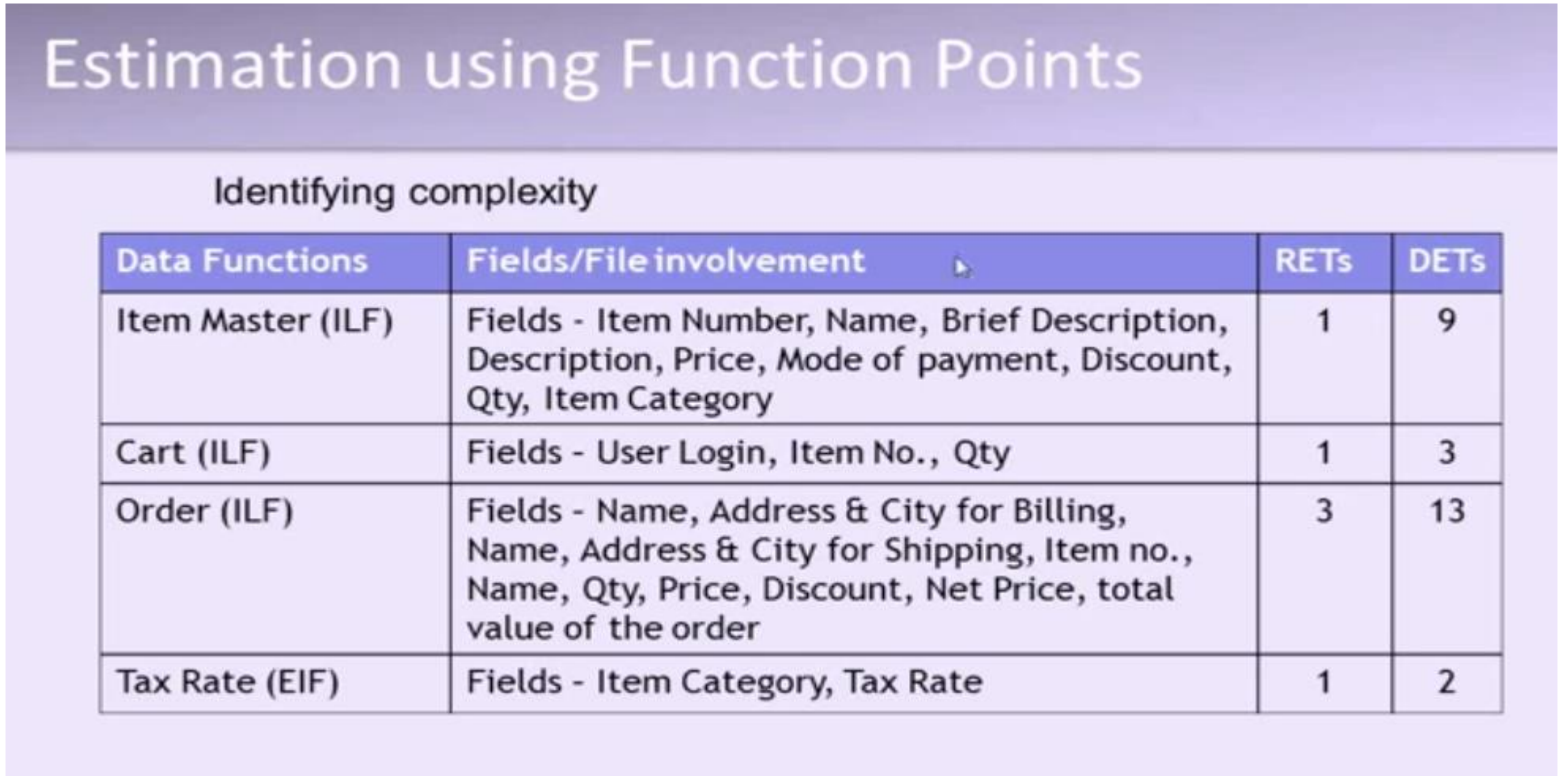
Estimation using Function Points

Identifying complexity

Transaction Functions	Fields/File involvement	FTRs	DETs
Add/Modify Item (2 x EI)	Fields - Item Number, Name, Brief Description, Description, Price, Mode of payment, Discount, Qty, Submit, Confirm/Error Msg File - Item Master	1	10
Delete Item (EI)	Fields - Item No., Name & Description, Press Delete, Confirm/Error Msg Files Item Master	1	5
Seller can generate an inventory report (EQ)	Fields - Press Report, item number, Description & Qty in stock Files - Item Master	1	4
Inform Order to Seller (EO)	Fields - Name, Address & City for Billing & Shipping, Plus list of items -> Item no., Name, Qty, Price, Discount, Net Price Plus total value of the order Files - Order	1	13

- File Type Reference (FTR), every file
- Data Element Type (DET) , every field

Function Point Example



Estimation using Function Points

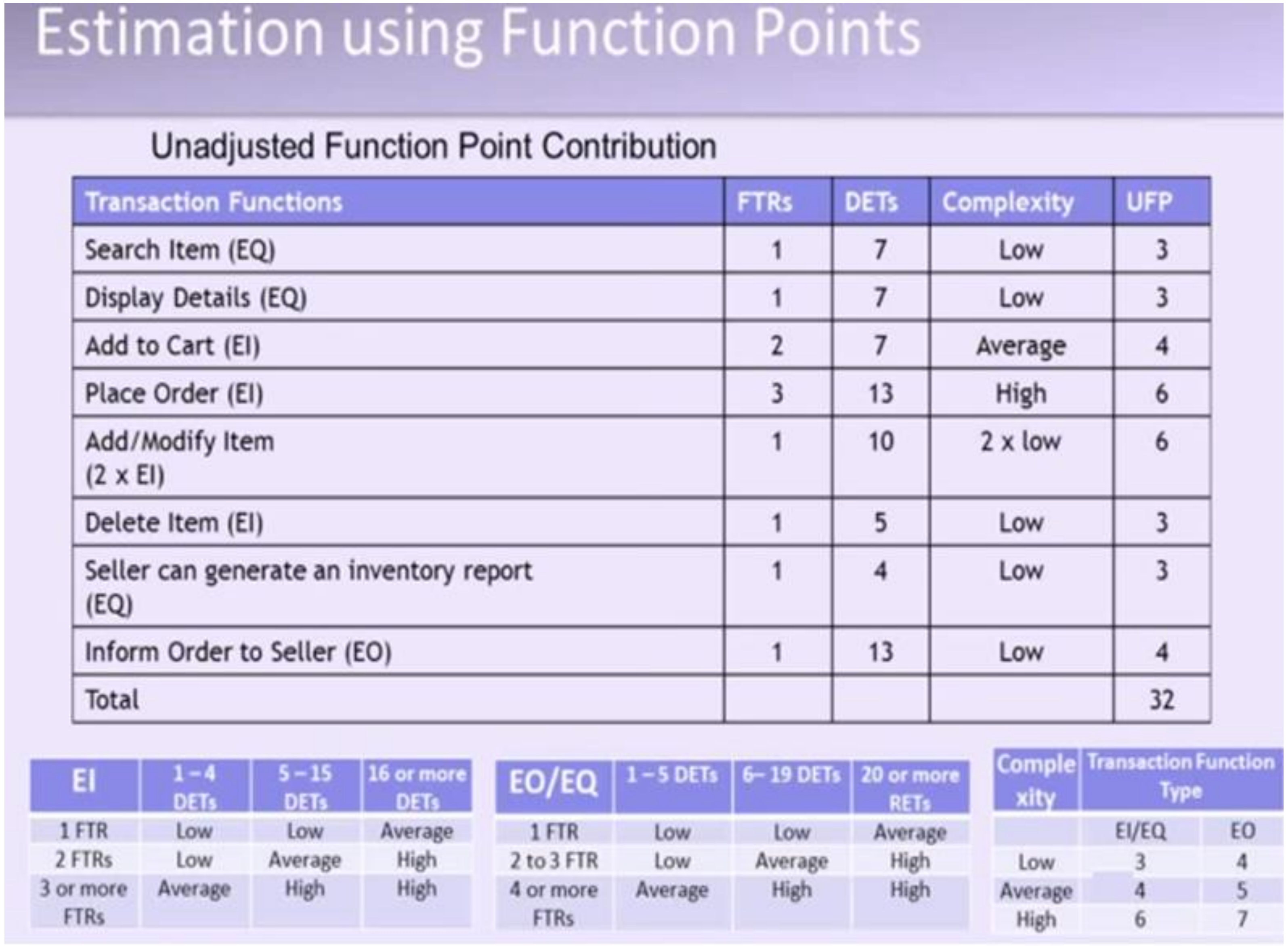
Identifying complexity

Data Functions	Fields/File involvement	RETs	DETs
Item Master (ILF)	Fields - Item Number, Name, Brief Description, Description, Price, Mode of payment, Discount, Qty, Item Category	1	9
Cart (ILF)	Fields - User Login, Item No., Qty	1	3
Order (ILF)	Fields - Name, Address & City for Billing, Name, Address & City for Shipping, Item no., Name, Qty, Price, Discount, Net Price, total value of the order	3	13
Tax Rate (EIF)	Fields - Item Category, Tax Rate	1	2

RETs: Record Element File .. sub group in the same file
DETs: Data Element Type

NOTE: Order (ILF) There are three sub grouping , billing and shipping are separate , some times shipping address is option may be customer use same billing address as shipping address, similarly , item numbers can be multiple .. item details another sub group.

Function Point Example



Estimation using Function Points

Unadjusted Function Point Contribution

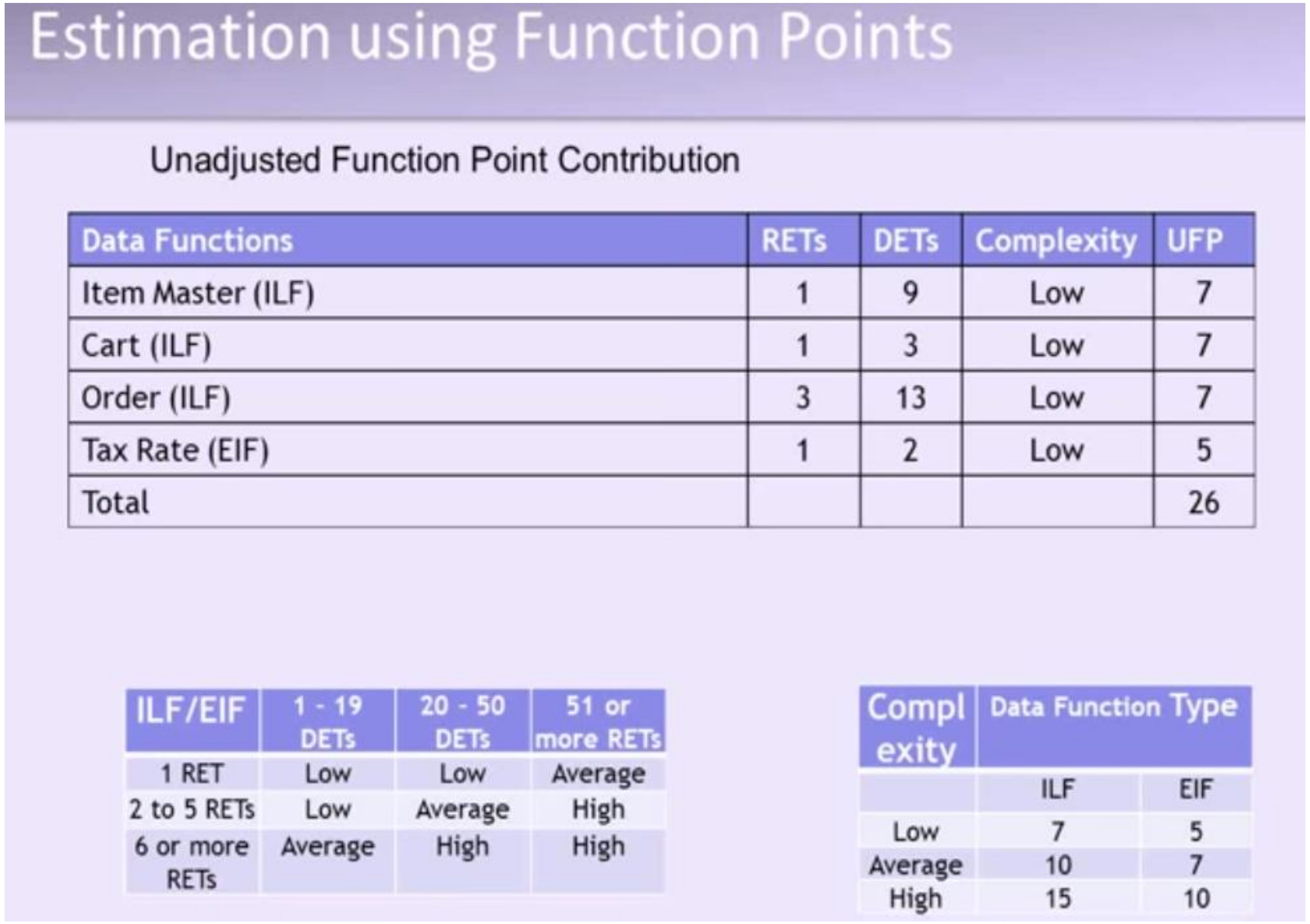
Transaction Functions	FTRs	DETs	Complexity	UFP
Search Item (EQ)	1	7	Low	3
Display Details (EQ)	1	7	Low	3
Add to Cart (EI)	2	7	Average	4
Place Order (EI)	3	13	High	6
Add/Modify Item (2 x EI)	1	10	2 x low	6
Delete Item (EI)	1	5	Low	3
Seller can generate an inventory report (EQ)	1	4	Low	3
Inform Order to Seller (EO)	1	13	Low	4
Total				32

EI	1 – 4 DETs	5 – 15 DETs	16 or more DETs
1 FTR	Low	Low	Average
2 FTRs	Low	Average	High
3 or more FTRs	Average	High	High

EO/EQ	1 – 5 DETs	6– 19 DETs	20 or more RETs
1 FTR	Low	Low	Average
2 to 3 FTR	Low	Average	High
4 or more FTRs	Average	High	High

Complexity	Transaction Function Type	
	EI/EQ	EO
Low	3	4
Average	4	5
High	6	7

Function Point Example



Estimation using Function Points

Unadjusted Function Point Contribution

Data Functions	RETs	DETs	Complexity	UFP
Item Master (ILF)	1	9	Low	7
Cart (ILF)	1	3	Low	7
Order (ILF)	3	13	Low	7
Tax Rate (EIF)	1	2	Low	5
Total				26

ILF/EIF	1 - 19 DETs	20 - 50 DETs	51 or more RETs
1 RET	Low	Low	Average
2 to 5 RETs	Low	Average	High
6 or more RETs	Average	High	High

Complexity	Data Function Type	
	ILF	EIF
Low	7	5
Average	10	7
High	15	10

General System Characteristics (GSCs)

GSCs are factors that describe the software system's environment and influence the complexity of the project.
There are **14 General System Characteristics**, and they are evaluated by assigning a rating from 0 to 5 based on how much **each characteristic influences** the system:

- **0** – No influence
- **1** – Insignificant influence
- **2** – Moderate influence
- **3** – Average influence
- **4** – Significant influence
- **5** – Strong influence

Here are the 14 General System Characteristics and examples of how to evaluate them:

- **(1) Data Communications**
  - Does the application communicate data between components or external systems?
  - Example: A web-based system with API integrations might be rated higher (e.g., 4 or 5).
- **(2) Distributed Data Processing**
  - Does the system operate on multiple machines or in a distributed environment?
  - Example: A cloud-based application working with distributed databases might get a higher score (e.g., 4).

General System Characteristics (GSCs):

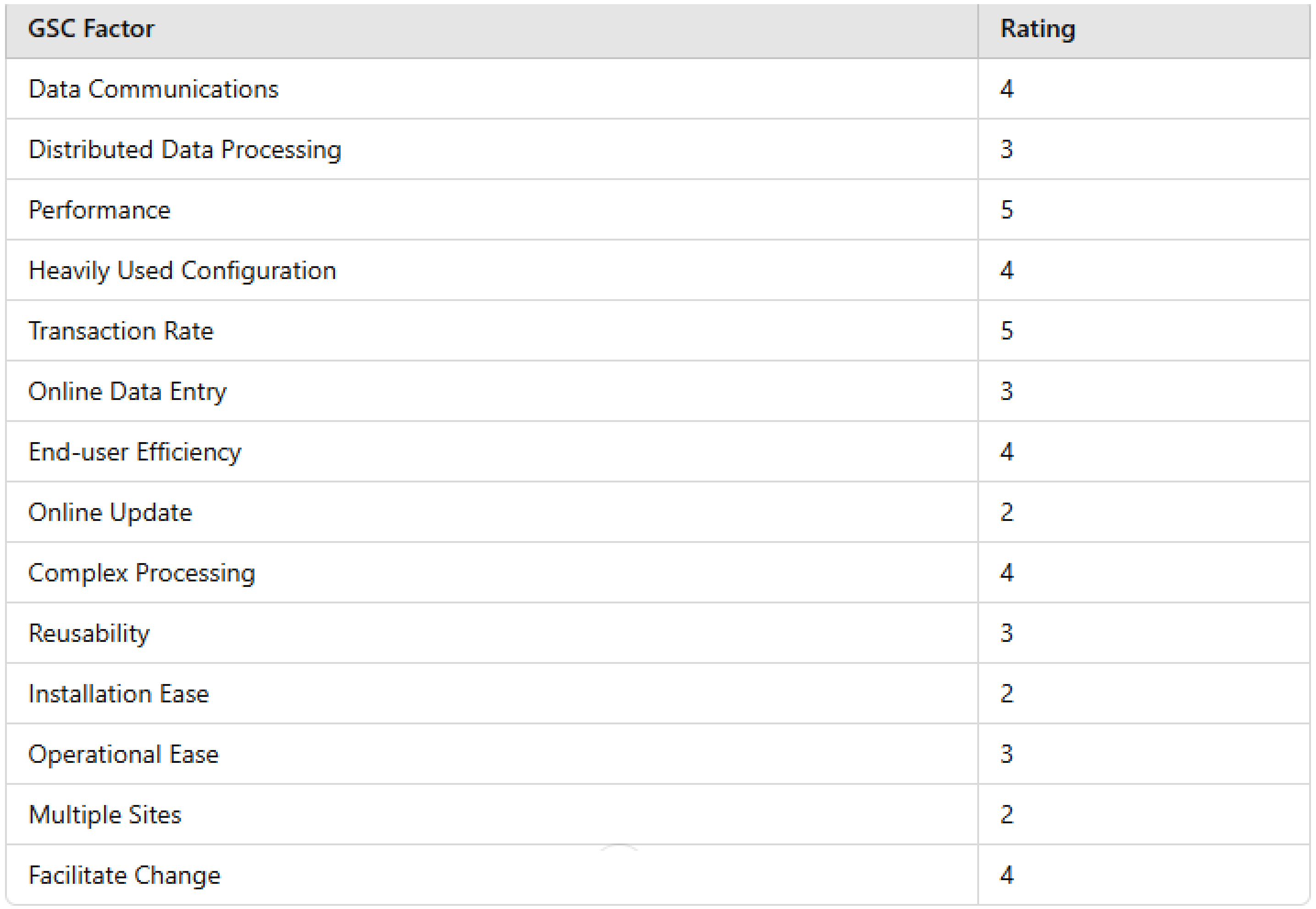
- **(3) Performance**
  - Are there performance requirements such as fast response time, high throughput, or resource optimization?
  - Example: A high-traffic website requiring minimal latency may score higher (e.g., 4 or 5).
- **(4) Heavily Used Configuration**
  - Does the system operate in a heavily used configuration, with many users accessing it concurrently?
  - Example: A system used by hundreds of concurrent users may have a rating of 4.
- **(5) Transaction Rate**
  - Are there high transaction processing demands in the system?
  - Example: A banking system that handles thousands of transactions daily might score high (e.g., 5).
- **(6) Online Data Entry**
  - Does the system require substantial online data entry?
  - Example: An e-commerce platform where users frequently input data can score around 4.
- **(7) End-user Efficiency**
  - Does the system emphasize user-friendly interfaces or end-user efficiency?
  - Example: A productivity tool with a user-friendly interface might score 4.

General System Characteristics (GSCs):

- **(8) Online Update**
  - Is online updating of data and files a critical feature?
  - Example: A system that allows real-time updates to a database may score high (e.g., 3 or 4).
- **(9) Complex Processing**
  - Does the system involve complex mathematical or logical processing?
  - Example: A scientific calculation system involving complex algorithms might score 5.
- **(10) Reusability**
  - Does the system involve reusable components or is there an emphasis on reusability?
  - Example: A system with reusable code components or modules may score higher (e.g., 3).
- **(11) Installation Ease**
  - Is ease of installation a priority?
  - Example: A web-based system that doesn’t require much installation effort might score lower (e.g., 2).
- **(12) Operational Ease**
  - Does the system need to be easy to operate by the end-users or the IT staff?
  - Example: A system designed for non-technical users may score higher (e.g., 4).
- **(13) Multiple Sites**
  - Does the system need to support multiple physical sites?
  - Example: A system used in various company branches might score higher (e.g., 3).
- **(14) Facilitate Change**
  - Is the system designed to easily accommodate future changes?
  - Example: A modular system that supports adding new features easily might score high (e.g., 5).

General System Characteristics (GSCs):

Assume the following scores for a system:

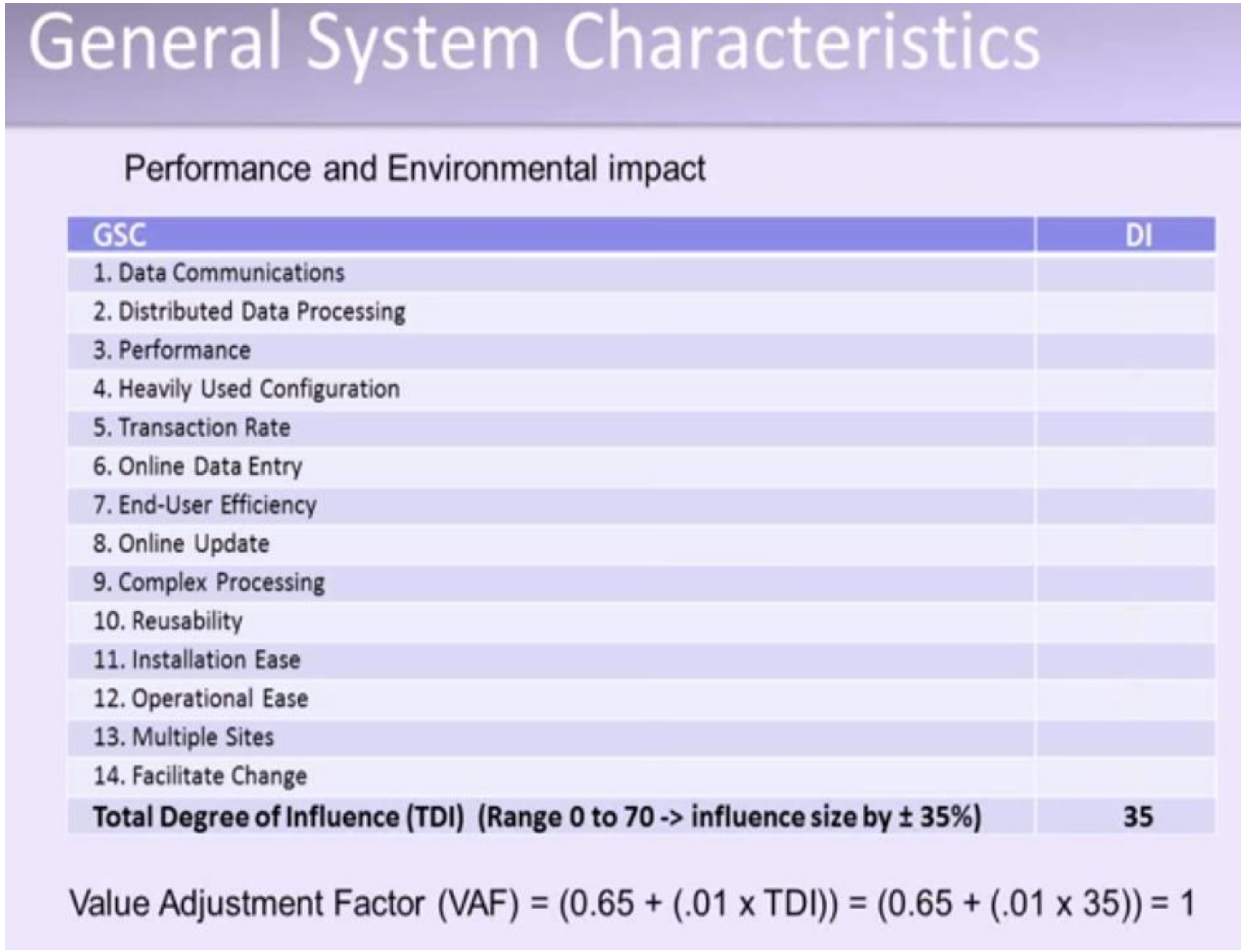


GSC Factor	Rating
Data Communications	4
Distributed Data Processing	3
Performance	5
Heavily Used Configuration	4
Transaction Rate	5
Online Data Entry	3
End-user Efficiency	4
Online Update	2
Complex Processing	4
Reusability	3
Installation Ease	2
Operational Ease	3
Multiple Sites	2
Facilitate Change	4

Total GSC Score **48**

Adjusted Function Point=Unadjusted Function Point×(0.65+0.01×Total GSC Score)

Function Point Example



General System Characteristics

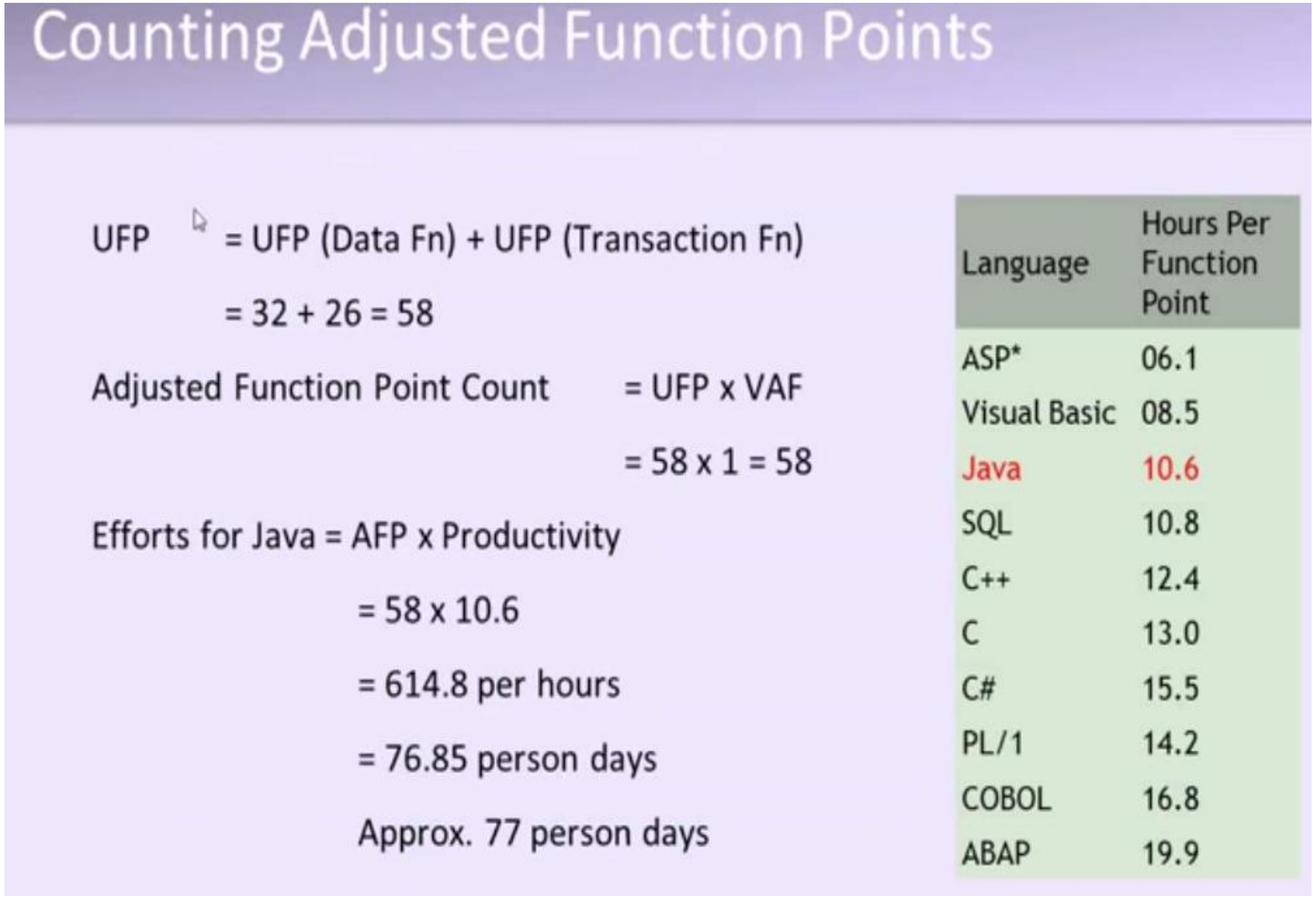
Performance and Environmental impact

GSC	DI
1. Data Communications	
2. Distributed Data Processing	
3. Performance	
4. Heavily Used Configuration	
5. Transaction Rate	
6. Online Data Entry	
7. End-User Efficiency	
8. Online Update	
9. Complex Processing	
10. Reusability	
11. Installation Ease	
12. Operational Ease	
13. Multiple Sites	
14. Facilitate Change	
Total Degree of Influence (TDI) (Range 0 to 70 -> influence size by ± 35%)	**35**

Value Adjustment Factor (VAF) = (0.65 + (.01 x TDI)) = (0.65 + (.01 x 35)) = 1

The degrees of influence range on a scale of **zero to five**, from no influence to strong influence

Function Point Example



Counting Adjusted Function Points

UFP = UFP (Data Fn) + UFP (Transaction Fn)

= 32 + 26 = 58

Adjusted Function Point Count = UFP x VAF

= 58 x 1 = 58

Efforts for Java = AFP x Productivity

= 58 x 10.6

= 614.8 per hours

= 76.85 person days

Approx. 77 person days

Language	Hours Per Function Point
ASP*	06.1
Visual Basic	08.5
Java	10.6
SQL	10.8
C++	12.4
C	13.0
C#	15.5
PL/1	14.2
COBOL	16.8
ABAP	19.9

Note: 614.8 person hours not 614.8 per hours

Example 1: Developing an Inventory Management System

Suppose you are tasked with developing an inventory management system for a company. Using Function Point Analysis, you would estimate the cost as follows:

1. **Identify User Functions**:
   1. **External Inputs (EI)**: Adding new items, updating stock information.
   2. **External Outputs (EO)**: Generating inventory reports.
   3. **External Inquiries (EQ)**: Querying item availability.
   4. **Internal Logical Files (ILF)**: The database of items.
   5. **External Interface Files (EIF)**: Integration with external supplier systems.
2. **Assign Complexity**: Each function is assigned a complexity level (low, average, or high) based on how many data points or fields are involved.
   1. EI: 3 EIs (add, update, delete) – all average complexity.
   2. EO: 2 reports (simple and detailed) – one low and one high complexity.
   3. EQ: 1 query interface – low complexity.
   4. ILF: Item database – average complexity.
   5. EIF: Supplier system – high complexity.

Example 1: Developing an Inventory Management System

3. **Calculate Unadjusted Function Points (UFP)**: Use predefined weights for each function based on its complexity and type. For example:
   1. EI (average) = 4 points × 3 inputs = 12 points.
   2. EO (low) = 4 points × 1 output = 4 points.
   3. EO (high) = 7 points × 1 output = 7 points.
   4. EQ (low) = 3 points × 1 query = 3 points.
   5. ILF (average) = 7 points × 1 file = 7 points.
   6. EIF (high) = 10 points × 1 file = 10 points.
   7. **UFP Total** = 12 + 4 + 7 + 3 + 7 + 10 = 43 points.
4. **Adjust for Technical Complexity Factors (TCF)**: Apply a multiplier based on system characteristics such as performance, security, usability, etc. For simplicity, assume the adjustment factor is 1.1.
5. **AFP (Adjusted Function Points)** = UFP × TCF = 43 × 1.1 = 47.3.
6. **Estimate Effort and Cost**: Based on historical data, each function point may require a certain number of person-hours (e.g., 10 hours per function point). Thus:
   1. **Total effort** = 47.3 FP × 10 hours/FP = 473 person-hours.
   2. If the hourly rate is $50, then the **total cost** is 473 hours × $50/hour = $23,650.

Example 2: Enhancing a Banking Application

Consider a scenario where you’re asked to add new features to an existing banking application.

1. **New User Functions**:
    1. Adding **External Inputs (EI)**: New transaction types like international payments.
    2. New **External Outputs (EO)**: Monthly statements for international accounts.
    3. Modifying the **Internal Logical Files (ILF)**: Adding international transactions to the database.
2. **Assign Complexity**:
    1. EI: 1 new input for international payment – high complexity.
    2. EO: 1 new statement output – average complexity.
    3. ILF: 1 file update – average complexity.
3. **Calculate UFP**:
    1. EI (high) = 6 points × 1 = 6 points.
    2. EO (average) = 5 points × 1 = 5 points.
    3. ILF (average) = 7 points × 1 = 7 points.

Example 2: Enhancing a Banking Application

Consider a scenario where you’re asked to add new features to an existing banking application.

4. **UFP Total** = 6 + 5 + 7 = 18 points.
5. **Adjust for TCF**: Assume a technical complexity factor of 1.2 due to security and compliance requirements.
6. **AFP** = 18 × 1.2 = 21.6.
7. **Estimate Effort and Cost**: If each function point requires 12 hours:
    1. **Total effort** = 21.6 FP × 12 hours/FP = 259.2 person-hours.
    2. At an hourly rate of $60, the **total cost** would be 259.2 hours × $60/hour = $15,552.

Questions:

- How do the effort estimations differ between COCOMO (which uses LOC) and FPA (which uses function points)?
- Which method (COCOMO or FPA) provides a more accurate estimation for your project, and why?
- Between COCOMO and FPA, which model do you find more applicable to modern software development methodologies (e.g., Agile, DevOps)?
- How well does each model handle changes in requirements and evolving software complexity?
- Which method (COCOMO or FPA) do you think captures system complexity more effectively?
- An application has the following characteristics:
    - 14 Low External Inputs
    - 25 High External Outputs
    - 10 Low Internal Logical Files
    - 18 High External Interface Files
    - 14 Average External Inquiries

  Additionally, the sum of the General System Characteristics (GSCs) ratings is 41.

Calculate the following:

a) What is the **unadjusted function point count (UFP)**?

b) what is the **adjusted function point count (AFP)**?