Software Requirement Analysis

Course Outline

- TYPES OF REQUIREMENTS
- FEASIBILITY STUDY
- REQUIREMENT ELICITATION
- DFD, DATA DICTIONARY
- HIPO CHART, WARNIER ORR DIAGRAM

Requirement Engineering Outline

4 step Process

- Feasibility Study
- Requirement Gathering
- Software Requirement Specification
- Software Requirement validation

Requirement Engineering

- The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.

- **Requirement Engineering**
    - The process to gather the software requirements from client, analyze and document them is known as requirement engineering.

    - The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System Requirements Specification' document.

- Requirement Engineering Process
    - It is a four-step process, which includes –
    1. Feasibility Study
    2. Requirement Gathering
    3. Software Requirement Specification
    4. Software Requirement Validation

Feasibility Study

- When the client approaches the organization for getting the desired product developed, it comes up with rough idea about what all functions the software must perform and which all features are expected from the software.

- Referencing to this information, the analysts does a detailed study about whether the desired system and its functionality are feasible to develop.

- This feasibility study is focused towards goal of the organization. This study analyzes whether the software product can be practically materialized in terms of implementation, contribution of project to organization, cost constraints and as per values and objectives of the organization. It explores technical aspects of the project and product such as usability, maintainability, productivity and integration ability.

- The output of this phase should be a feasibility study report that should contain adequate comments and recommendations for management about whether or not the project should be undertaken.

Feasibility Study

PAGE 5

Requirement Engineering   Feasibility Study   Requirement gathering   Software Requirement Specification   Software Requirement Validation

Requirement Gathering

- If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user. Analysts and engineers communicate with the client and end-users to know their ideas on what the software should provide and which features they want the software to include.

Requirement gathering

Software Requirement Specification

- SRS is a document created by system analyst after the requirements are collected from various stakeholders.

- SRS defines how the intended software will interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations etc.

- The requirements received from client are written in natural language. It is the responsibility of system analyst to document the requirements in technical language so that they can be comprehended and useful by the software development team.

- SRS should come up with following features:
  - User Requirements are expressed in natural language.
  - Technical requirements are expressed in structured language, which is used inside the organization.
  - Design description should be written in Pseudo code.
  - Format of Forms and GUI screen prints.
  - Conditional and mathematical notations for DFDs etc.

Software Requirement Validation

- After requirement specifications are developed, the requirements mentioned in this document are validated. User might ask for illegal, impractical solution or experts may interpret the requirements incorrectly. This results in huge increase in cost if not nipped in the bud. Requirements can be checked against following conditions -

  - If they can be practically implemented
  - If they are valid and as per functionality and domain of software
  - If there are any ambiguities
  - If they are complete
  - If they can be demonstrated

Requirement Elicitation Process

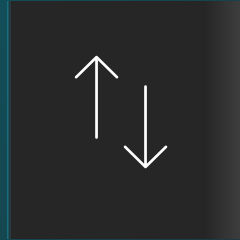**Requirement Gathering** → **Organizing Requirements** → **Negotiation & discussion** → **Requirement Specification**
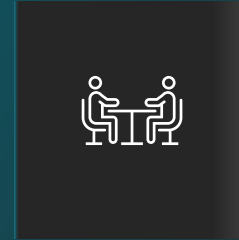
Requirement Gathering
The developers discuss with the client and end users and know their expectations from the software.
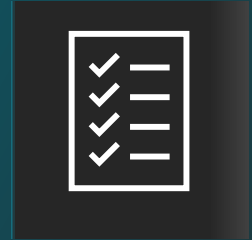
Organizing Requirements
The developers prioritize and arrange the requirements in order of importance, urgency and convenience.

Negotiation & Discussion
Requirements may then be prioritized and reasonably compromised. To remove the ambiguity and conflicts, they are discussed for clarity and correctness.

Documentation
All formal & informal, functional and non-functional requirements are documented and made available for next phase processing.

Elicitation Techniques

- Interviews
- Brainstorming Sessions
- Facilitated Application Specification Technique (FAST)
- Quality Function Deployment (QFD)
- Use Case Approach
- Surveys
- Questionnaires
- Task Analysis
- Domain Analysis
- Observation

Requirement Elicitation Techniques

Elicitation Techniques

- Interviews: Interviews are strong medium to collect requirements. Organization may conduct several types of interviews such as:
  - Structured (closed) interviews, where every single information to gather is decided in advance, they follow pattern and matter of discussion firmly.
  - Non-structured (open) interviews, where information to gather is not decided in advance, more flexible and less biased.
  - Oral interviews
  - Written interviews
  - One-to-one interviews which are held between two persons across the table.
  - Group interviews which are held between groups of participants. They help to uncover any missing requirement as numerous people are involved.

Elicitation Techniques

- **Brainstorming Sessions:**
  - It is a group technique
  - It is intended to generate lots of new ideas hence providing a platform to share views
  - A highly trained facilitator is required to handle group bias and group conflicts.
  - Every idea is documented so that everyone can see it.
  - Finally, a document is prepared which consists of the list of requirements and their priority if possible.
  - An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.

- **Facilitated Application Specification Technique:**
  - Its objective is to bridge the expectation gap – the difference between what the developers think they are supposed to build and what customers think they are going to get. A team-oriented approach is developed for requirements gathering. Each attendee is asked to make a list of objects that are-
    - Part of the environment that surrounds the system
    - Produced by the system
    - Used by the system
  - Each participant prepares his/her list, different lists are then combined, redundant entries are eliminated, team is divided into smaller sub-teams to develop mini-specifications and finally a draft of specifications is written down using all the inputs from the meeting.

- **Quality Function Deployment (QFD):**
  - In this technique customer satisfaction is of prime concern, hence it emphasizes on the requirements which are valuable to the customer.
  - 3 types of requirements are identified –
  - **Normal requirements** – In this the objective and goals of the proposed software are discussed with the customer. Example – normal requirements for a result management system may be entry of marks, calculation of results, etc
  - **Expected requirements** – These requirements are so obvious that the customer need not explicitly state them. Example – protection from unauthorized access.
  - **Exciting requirements** – It includes features that are beyond customer's expectations and prove to be very satisfying when present. Example – when unauthorized access is detected, it should backup and shutdown all processes.

Elicitation Techniques

- ## Quality Function Deployment (QFD):
  - The major steps involved in this procedure are –
    - Identify all the stakeholders, eg. Users, developers, customers etc
    - List out all requirements from customer.
    - A value indicating degree of importance is assigned to each requirement.
    - In the end the final list of requirements is categorized as –
      - ✓ It is possible to achieve
      - ✓ It should be deferred and the reason for it
      - ✓ It is impossible to achieve and should be dropped off

Elicitation Techniques

- **Use Case Approach:**
  - This technique combines text and pictures to provide a better understanding of the requirements.
  - The use cases describe the 'what', of a system and not 'how'. Hence, they only give a functional view of the system.
  - The components of the use case design includes three major things – Actor, Use cases, use case diagram.

- ## Use Case Approach:
  - Actor – It is the external agent that lies outside the system but interacts with it in some way. An actor maybe a person, machine etc. It is represented as a stick figure. Actors can be primary actors or secondary actors.
    - ✓ Primary actors – It requires assistance from the system to achieve a goal.
    - ✓ Secondary actor – It is an actor from which the system needs assistance.
  - Use cases – They describe the sequence of interactions between actors and the system. They capture who(actors) do what(interaction) with the system. A complete set of use cases specifies all possible ways to use the system.
  - Use case diagram – A use case diagram graphically represents what happens when an actor interacts with a system. It captures the functional aspect of the system.
    - ✓ A stick figure is used to represent an actor.
    - ✓ An oval is used to represent a use case.
    - ✓ A line is used to represent a relationship between an actor and a use case.

Elicitation Techniques

- Surveys : Organization may conduct surveys among various stakeholders by querying about their expectation and requirements from the upcoming system.

- Questionnaires : A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled. A shortcoming of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended.

- Task Analysis : Team of engineers and developers may analyze the operation for which the new system is required. If the client already has some software to perform certain operation, it is studied and requirements of proposed system are collected.

Elicitation Techniques

- Domain Analysis : Every software falls into some domain category. The expert people in the domain can be a great help to analyze general and specific requirements.

- Observation : Team of experts visit the client's organization or workplace. They observe the actual working of the existing installed systems. They observe the workflow at client's end and how execution problems are dealt. The team itself draws some conclusions which aid to form requirements expected from the software.

Software Design Tools

- Software analysis and design includes all activities, which help the transformation of requirement specification into implementation.
- Requirement specifications specify all functional and non-functional expectations from the software.
- These requirement specifications come in the shape of human readable and understandable documents, to which a computer has nothing to do.
- Software analysis and design is the intermediate stage, which helps human-readable requirements to be transformed into actual code.

Data Flow Diagram

- Data flow diagram is graphical representation of flow of data in an information system. It is capable of depicting incoming data flow, outgoing data flow and stored data. The DFD does not mention anything about how data flows through the system.

- There is a prominent difference between DFD and Flowchart. The flowchart depicts flow of control in program modules. DFDs depict flow of data in the system at various levels. DFD does not contain any control or branch elements.

- Types of DFD: Data Flow Diagrams are either Logical or Physical.

  - Logical DFD - This type of DFD concentrates on the system process, and flow of data in the system.For example in a Banking software system, how data is moved between different entities.

  - Physical DFD - This type of DFD shows how the data flow is actually implemented in the system. It is more specific and close to the implementation.

Data Flow Diagram - Components

➤ DFD can represent Source, destination, storage and flow of data using the following set of components -

Entity

Entities are source and destination of information data.
Entities are represented by a rectangles with their respective names.

Process

Activities and action taken on the data are represented by Circle or Round-edged rectangles.

Data Store

There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.

Data Flow

Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.
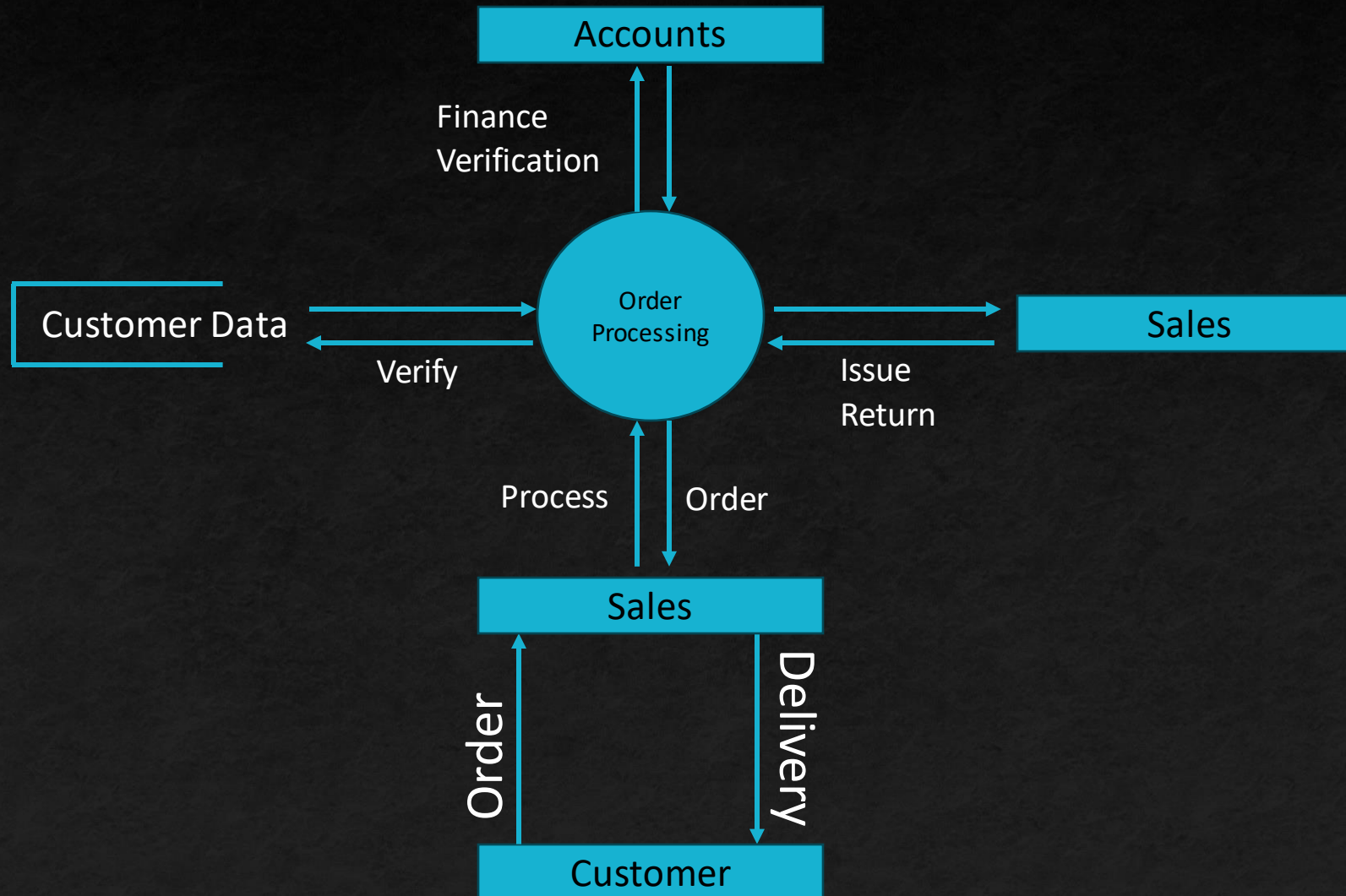
Data Flow Diagram - Levels

➢ Level 0: Highest abstraction level DFD is known as Level 0 DFD, which depicts the entire information system as one diagram concealing all the underlying details. Level 0 DFDs are also known as context level DFDs.
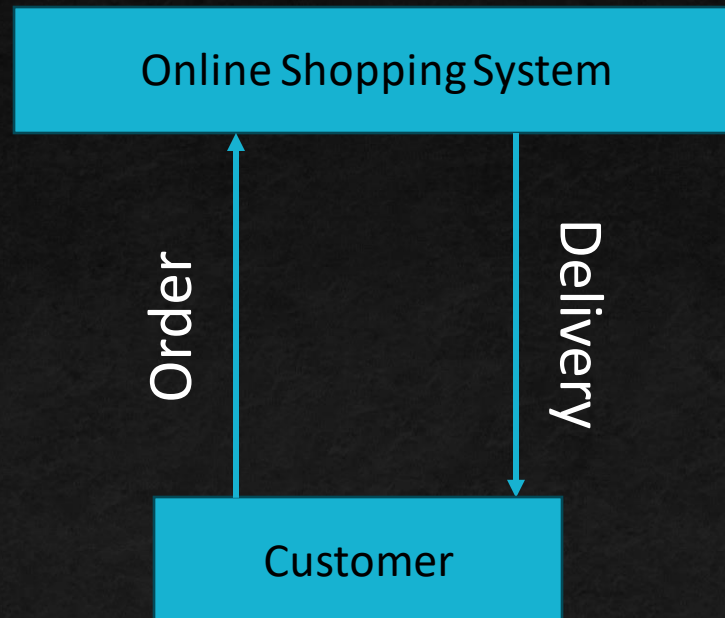
Data Flow Diagram - Levels

➢ Level 1: The Level 0 DFD is broken down into more specific, Level 1 DFD. Level 1 DFD depicts basic modules in the system and flow of data among various modules. Level 1 DFD also mentions basic processes and sources of information.

Data Flow Diagram - Levels

➤ Level 2: At this level, DFD shows how data flows inside the modules mentioned in Level 1.

➤ Higher level DFDs can be transformed into more specific lower level DFDs with deeper level of understanding unless the desired level of specification is achieved.

Structure Chart

➢ Structure chart is a chart derived from Data Flow Diagram. It represents the system in more detail than DFD. It breaks down the entire system into lowest functional modules, describes functions and sub-functions of each module of the system to a greater detail than DFD.

➢ Structure chart represents hierarchical structure of modules. At each layer a specific task is performed.

Structure Chart - Module

➢ It represents process or subroutine or task. A control module branches to more than one sub-module. Library Modules are re-usable and invokable from any module.

Module

Module   Module   Module

Sub - Module   Library Module
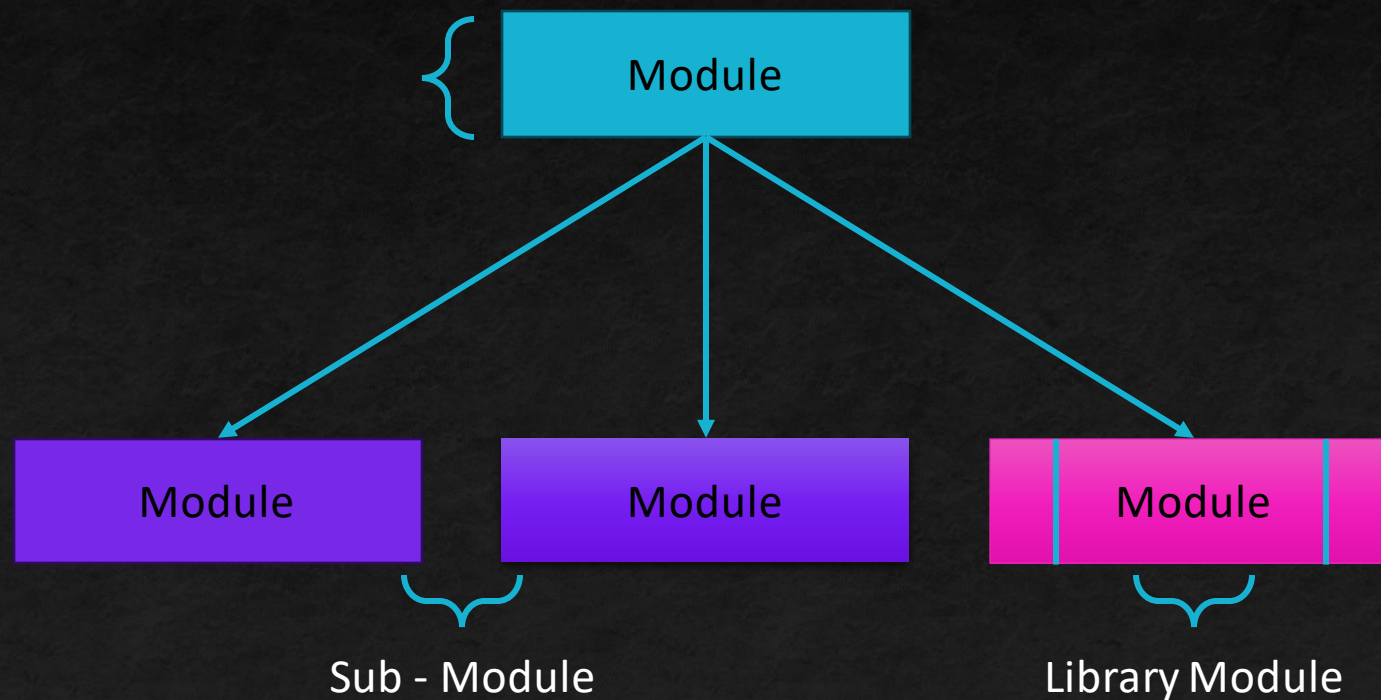
Structure Chart - Condition

➢ It is represented by small diamond at the base of module. It depicts that control module can select any of sub-routine based on some condition.
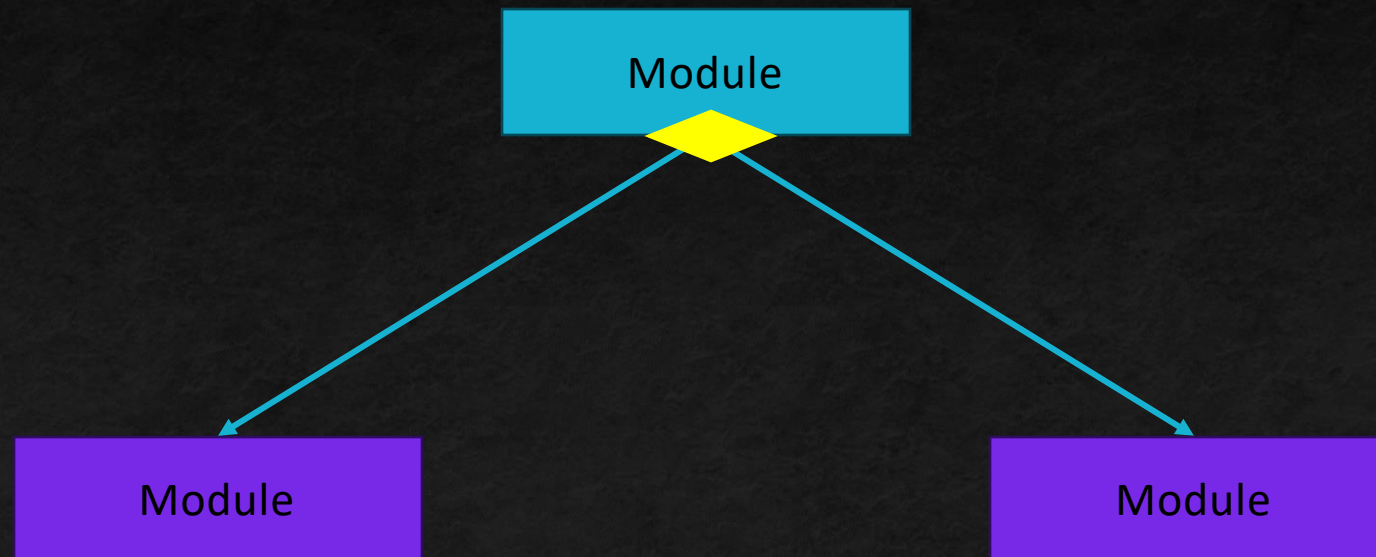
Structure Chart - Jump

➤ An arrow is shown pointing inside the module to depict that the control will jump in the middle of the sub-module.
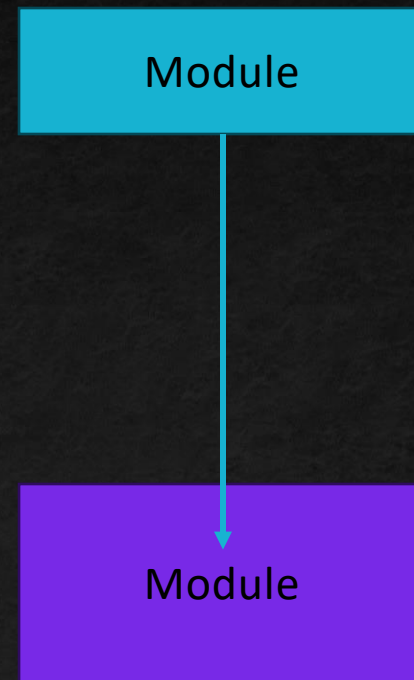
Module

Module

Structure Chart - Loop

➢ A curved arrow represents loop in the module. All sub-modules covered by loop repeat execution of module.
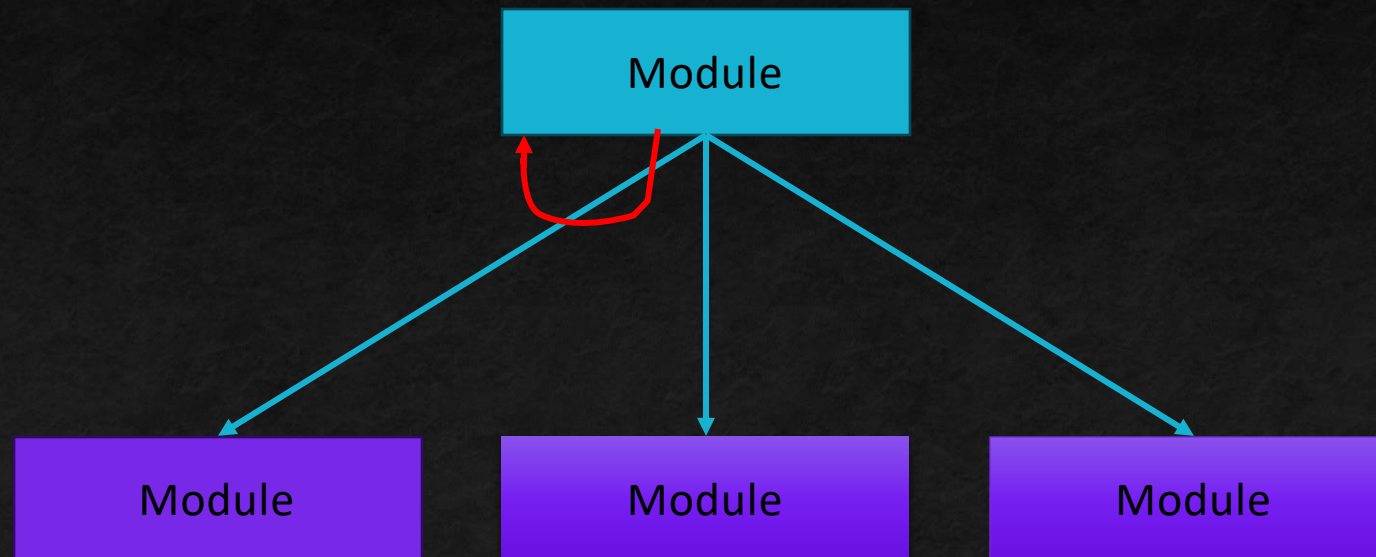
Structure Chart – Data flow

➢ A directed arrow with empty circle at the end represents data flow.

Structure Chart – Control flow

➢ A directed arrow with filled circle at the end represents control flow.
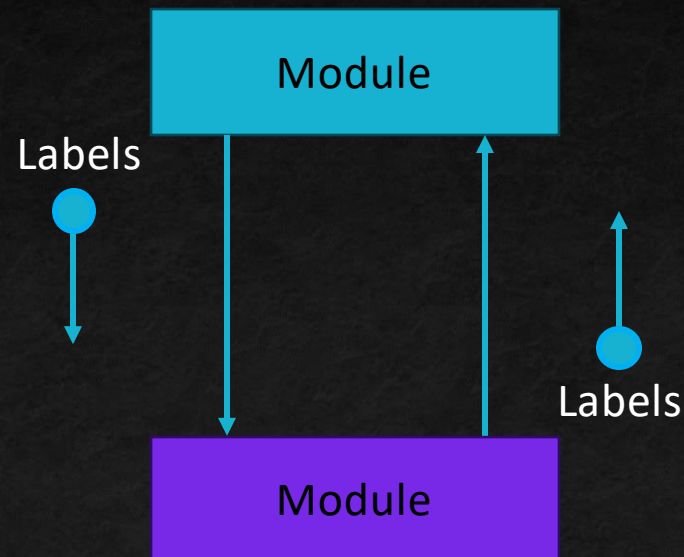
HIPO Diagram

➤ HIPO (Hierarchical Input Process Output) diagram is a combination of two organized method to analyze the system and provide the means of documentation. HIPO model was developed by IBM in year 1970.

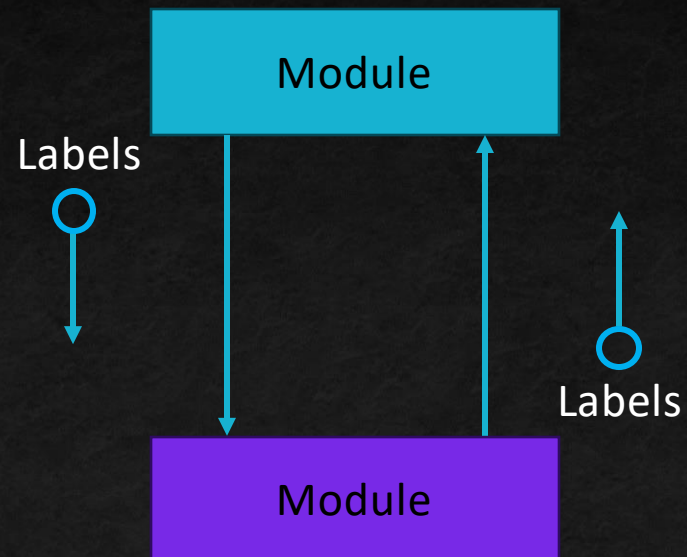➤ HIPO diagram represents the hierarchy of modules in the software system. Analyst uses HIPO diagram in order to obtain high-level view of system functions. It decomposes functions into sub-functions in a hierarchical manner. It depicts the functions performed by system.

➤ HIPO diagrams are good for documentation purpose. Their graphical representation makes it easier for designers and managers to get the pictorial idea of the system structure.

HIPO Diagram
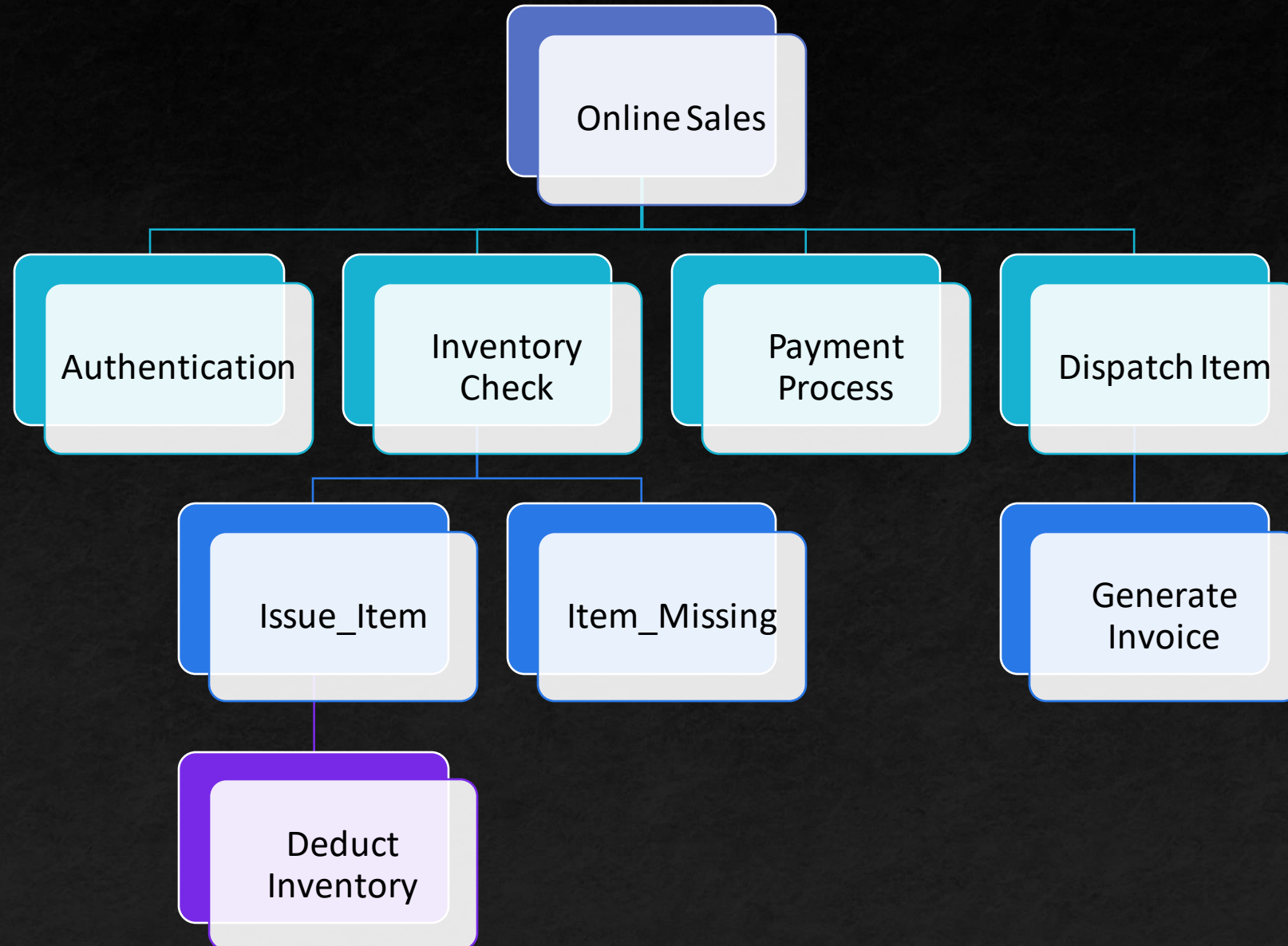
HIPO Diagram

➢ In contrast to IPO (Input Process Output) diagram, which depicts the flow of control and data in a module, HIPO does not provide any information about data flow or control flow.

Authentication

Input	Process	Output
User Screen	Take authentication credentials from user-screen Check Validity of credentials Call appropriate Module	User Screen
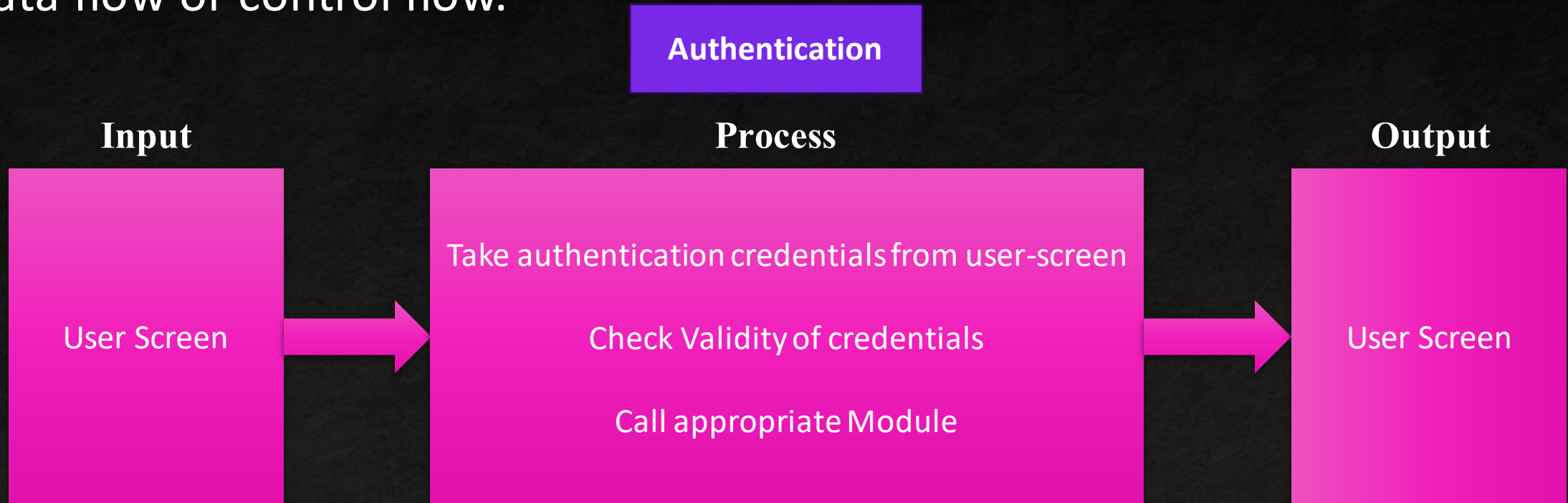
Data Dictionary

➢ Data dictionary is the centralized collection of information about data. It stores meaning and origin of data, its relationship with other data, data format for usage etc. Data dictionary has rigorous definitions of all names in order to facilitate user and software designers.

➢ Data dictionary is often referenced as meta-data (data about data) repository. It is created along with DFD (Data Flow Diagram) model of software program and is expected to be updated whenever DFD is changed or updated.

Data Dictionary - Requirement

➢ The data is referenced via data dictionary while designing and implementing software. Data dictionary removes any chances of ambiguity. It helps keeping work of programmers and designers synchronized while using same object reference everywhere in the program.

➢ Data dictionary provides a way of documentation for the complete database system in one place. Validation of DFD is carried out using data dictionary.

Data Dictionary - Contents

➢ **Data dictionary should contain information about the following**
- • Data Flow
- • Data Structure
- • Data Elements
- • Data Stores
- • Data Processing

Data Flow is described by means of DFDs as studied earlier and represented in algebraic form as described.

Sign	Description
=	Composed of
{}	Repetition
()	Optional
+	And
[/]	Or

Examples:
Address = House No + (Street / Area) + City + State

Course ID = Course Number + Course Name + Course Level + Course Grades

Data Dictionary - Elements

➢ Data elements consist of Name and descriptions of Data and Control Items, Internal or External data stores etc. with the following details:

- ➢ Primary Name
- ➢ Secondary Name (Alias)
- ➢ Use-case (How and where to use)
- ➢ Content Description (Notation etc.)
- ➢ Supplementary Information (preset values, constraints etc.)

Data Dictionary – Store, Processing

- It stores the information from where the data enters into the system and exists out of the system. The Data Store may include :
  - Files
    - Internal to software.
    - External to software but on the same machine.
    - External to software and system, located on different machine.
  - Tables
    - Naming convention
    - Indexing property

- Data Processing
- There are two types of Data Processing:

  - Logical: As user sees it
  - Physical: As software sees it

Entity – Relationship Model

➢ Entity-Relationship model is a type of database model based on the notion of real world entities and relationship among them. We can map real world scenario onto ER database model. ER Model creates a set of entities with their attributes, a set of constraints and relation among them.

➢ ER Model is best used for the conceptual design of database. ER Model can be represented as follows :

```
( Process )   ( Process )                          ( Process )   ( Process )
        \      /                                          \      /
    [  Entity  ] ───────── ◆ Relationship ◆ ───────── [  Entity  ]
```
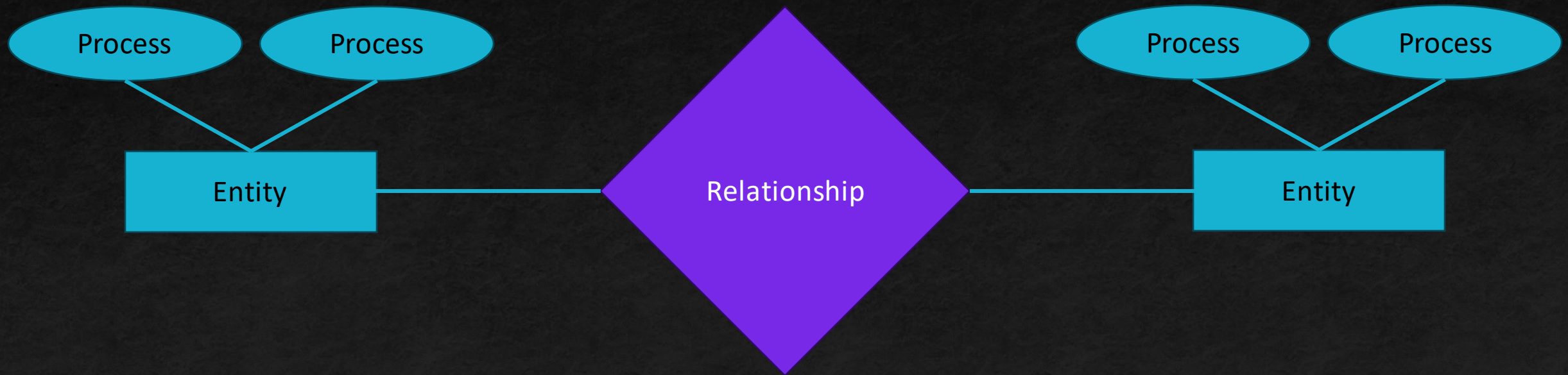
Entity – Relationship Model

- **Entity** - An entity in ER Model is a real world being, which has some properties called attributes. Every attribute is defined by its corresponding set of values, called domain. For example, Consider a school database. Here, a student is an entity. Student has various attributes like name, id, age and class etc.
- **Relationship** - The logical association among entities is called relationship. Relationships are mapped with entities in various ways. Mapping cardinalities define the number of associations between two entities.
- **Mapping cardinalities:**
  - one to one
  - one to many
  - many to one
  - many to many

Function Point Analysis

➤ Function Point Analysis was initially developed by Allan J. Albrecht 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.

➤ Functional Point Analysis 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.

Function Point Analysis - Objectives

➢ 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.

Function Point Analysis - Types

- There are basically two types of Functional Point Analysis, that are listed below.
- **Transactional Functional Type**
- Data Functional Type

- **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.

Function Point Analysis - Types

➢ There are basically two types of Functional Point Analysis, that are listed below.

➢ Transactional Functional Type

➢ **<u>Data Functional Type</u>**

➢ **<u>Data Functional Type</u>**

➢ 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.

Function Point Analysis - Benefits

➢ 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.

Function Point Analysis - Characteristics

➢ We can calculate the function point with the help of the number of functions and types of functions used in application. These are classified into five types:

Measurement Parameters	Examples
Number of External Inputs (EI)	Input Screen and Tables
Number of External Output (EO)	Output Screens and Reports
Number of External Inquiries (EQ)	Prompts and Interrupts
Number of Internal Files (ILF)	Databases and Directories
Number of External Interfaces (EIF)	Shared databases and shared routines

➢ Functional Point helps in describing system complexity and also shows project timelines. It is majorly used for business systems like information systems.

Function Point Analysis – Weights

➢ Weights of 5 Functional Points Attributes: Functional Complexities help us in finding the corresponding weights, which results in finding the Unadjusted Functional point (UFp) of the Subsystem.

Measurement Parameters	Low	Average	High
Number of External Inputs (EI)	3	4	6
Number of External Output (EO)	4	5	7
Number of External Inquiries (EQ)	3	4	6
Number of Internal Files (ILF)	7	10	15
Number of External Interfaces (EIF)	5	7	10

Function Point Analysis – 14 Questions

1. reliable backup and recovery required ?
2. data communication required ?
3. are there distributed processing functions ?
4. is performance critical ?
5. will the system run in an existing heavily utilized operational environment ?,
6. on line data entry required ?
7. does the on line data entry require the input transaction to be built over multiple screens or operations ?

Function Point Analysis – 14 Questions

8. are the master files updated on line ?
9. is the inputs, outputs, files or inquiries complex ?
10. is the internal processing complex ?
11. is the code designed to be reusable ?
12. are the conversion and installation included in the design ?
13. is the system designed for multiple installations in different organizations ?
14. is the application designed to facilitate change and ease of use by the user ?

Function Point Analysis – 14 Questions Scale rating

- 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

- Calculate Complexity Adjustment Factor (CAF).
- CAF = 0.65 + (0.01 * F)

- **FP = UFP * CAF**

Function Point Analysis – Numerical 1

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

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

and Scale factor for all the 14 questions sets to an average
Calculate CAF, UFP and FP

Function Point Analysis – Numerical 2

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

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

and Scale factor for all the 14 questions sets to an Significant
Calculate CAF, UFP and FP

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

User Input = 30
User Output = 60
User Inquiries = 23
User Files = 08
External Interface = 02

It is given that the complexity weighting factors for I, O, E, F, and N are 4, 5, 4, 10, and 7, respectively. It is also given that, out of fourteen value adjustment factors that influence the development effort, four factors are not applicable, each of the other four factors has value 3, and each of the remaining factors has value 4. Calculate CAF, UFP and FP

Function Point Analysis – Numerical 4

Given the following values, compute function point when all weighting factors are High.

User Input = 60
User Output = 50
User Inquiries = 25
User Files = 8
External Interface = 5

and Scale factor for all the 14 questions sets to an Essential
Calculate CAF, UFP and FP

Given the following values, compute function point when all complexity adjustment factor (CAF) and weighting factors are given as follows.

User Input = 60
User Output = 50
User Inquiries = 25
User Files = 8
External Interface = 5

It is given that the complexity weighting factors for I, O, E, F, and N are 6, 7, 6, 15, and 10, respectively. It is also given that, out of fourteen value adjustment factors that influence the development effort, four factors are Incidental, each of the other five factors has value 3, and each of the remaining factors has value 5. Calculate CAF, UFP and FP

LOC – Lines of Code

- 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.

LOC – Lines of Code – Features, Advantages, Disadvantages

- Variations such as "source lines of code", are used to set out a codebase.
- LOC is frequently used in some kinds of arguments.
- They are used in assessing a project's performance or efficiency.

- Most used metric in cost estimation.
- Its alternates have many problems as compared to this metric.
- It is very easy in estimating the efforts.

- Very difficult to estimate the LOC of the final program from the problem specification.
- It correlates poorly with quality and efficiency of code.
- It doesn't consider complexity.

LOC – Lines of Code – Example 1

```
void selSort(int x[], int n) {
//Below function sorts an array in ascending order
int i, j, min, temp;
for (i = 0; i < n - 1; i++) {
        min = i;
        for (j = i + 1; j < n; j++)
        if (x[j] < x[min])
        min = j;
        temp = x[i];
        x[i] = x[min];
        x[min] = temp;
}
}
```

Calculate the LOC of the above code

LOC – Lines of Code – Example 2

```
void main()
{
        int fN, sN, tN;
        cout << "Enter the 2 integers: ";
        cin >> fN >> sN;
        // sum of two numbers in stored in variable sum
        sum = fN + sN;
        // Prints sum
        cout << fN << " + " << sN << " = " << sum;
        return 0;
}
```

Calculate the LOC of the above code

COCOMO

- Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e number of Lines of Code.
- Cocomo (Constructive Cost Model) is one of the model for cost estimation and project planning. If you want to do any project planning, you need to know the size of the product and cost estimation of the product.
- Used to calculate the effort, development time, Average staff size and productivity
- It is a procedural cost estimate model for software projects and is often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality.
- It was proposed by Barry Boehm in 1981 and is based on the study of 63 projects, which makes it one of the best-documented models.

COCOMO

- The key parameters which define the quality of any software products, which are also an outcome of the Cocomo are primarily Effort & Schedule:
    - **Effort:** Amount of labor that will be required to complete a task. It is measured in person-months units.
    - **Schedule:** Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put in. It is measured in the units of time such as weeks, and months.

COCOMO - Models

- COCOMO includes / divided into three types
  - Basic COCOMO model
  - Intermediate COCOMO model
  - Complete / detailed COCOMO model
- Whatever model is been taken from the above they are applied on three classes of software projects / modes.
- The three classes are as follows:
  - Organic Mode
  - Semi – detached Mode
  - Embedded Mode

COCOMO - Models

- 1. Organic – A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.

- 2. Semi-detached – A software project is said to be a Semi-detached type if the vital characteristics such as team size, experience, and knowledge of the various programming environment lie in between that of organic and Embedded. The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience and better guidance and creativity. Eg: Compilers or different Embedded Systems can be considered Semi-Detached types.

- 3. Embedded – A software project requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

COCOMO - Models

Class	Project Size	Nature of Project	Innovation	Deadline of Project
Organic	2 to 50 KLOC	Small size, Experience developers Eg: Payroll, Inventory Projects	Little	Not tight
Semi – Detached	50 to 300 KLOC	Medium Size, Medium Size team Eg: DB Systems Etc.	Medium	Medium
Embedded	Above 300 KLOC	Large Projects, Real time systems Eg: ATMs, Air traffic Controls	Significant	Tight

COCOMO - Models

- Basic COCOMO Model – Formula for Cost Estimation

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

$$time = c(Effort)^d$$

$$Person required = \text{Effort / time}$$

E = Effort

a,b,c,d are co - efficients

Software Projects	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semi – detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

- The effort is measured in Person-Months and as evident from the formula is dependent on Kilo-Lines of code. The development time is measured in months.

COCOMO – Numerical 1

Suppose that a project was estimated to be 400 KLOC, Calculate the effort and development time for each of the three modes/classes i.e. Organic, Semi – Detached and embedded. Consider the basic COCOMO model.

Mode / Class	Effort	Development Time
Organic	1295.31	38.07
Semi – Detached	2462.79	38.45
Embedded	4772.81	38

COCOMO – Numerical 2

Assume that the size of an organic type software product is estimated to be 32000 delivery source of instructions. Assume that the average salary of a software developer is $2000 per month. Determine the effort required to develop the software product, the nominal development time and the staff cost of developing the product

COCOMO – Intermediate

- Basic model is quick but inaccurate and phase insensitive
- Intermediate model includes a set of 15 additional Predictors (Cost Drivers)
- It also takes development environment into consideration during cost estimation
- Cost Drivers are used to adjust NOMINAL cost of project to actual project environment
- All 15 Cost drivers are divided into 4 categories

COCOMO – Intermediate – 15 COST Drivers

- **Product Attributes**
  1. Required Software Reliability (RELY)
  2. Database Size (DATA)
  3. Product Complexity (CPLX)
- **Computer Attributes**
  1. Execution time Constraints (TIME)
  2. Main Storage Constraints (STOR)
  3. Virtual Machine volatility (VIRT)
  4. Computer turnaround time (TURN)

COCOMO – Intermediate – 15 COST Drivers

- **Personnel Attributes**
  1. Analyst Capability (ACAP)
  2. Application Experience (AEXP)
  3. Programmer Capability (PCAP)
  4. Virtual Machine Experience (VEXP)
  5. Programming Language Experience (LEXP)
- **Project Attributes**
  1. Modern Programming Practices (MODP)
  2. Use of Software tools (TOOL)
  3. Required development schedule (SCED)

COCOMO – Intermediate – 15 COST Drivers

- **<u>Each Cost driver is divided into 6 categories i.e. Very Low, Low, Nominal, High, Very High, Extra High. Applied to what extent the Cost driver applies to the project</u>**

Product Attributes	Very Low	Low	Nominal	High	Very High	Extra High
RELY	0.75	0.88	1.00	1.15	1.40	----
DATA	----	0.94	1.00	1.05	1.16	----
CPLX	0.70	0.85	1.00	1.15	1.30	1.65

Computer Attributes	Very Low	Low	Nominal	High	Very High	Extra High
TIME	----	----	1.00	1.11	1.30	1.66
STOR	----	----	1.00	1.06	1.21	1.56
VIRT	----	0.87	1.00	1.15	1.30	----
TURN	----	0.87	1.00	1.07	1.15	----

COCOMO – Intermediate – 15 COST Drivers

Personnel Attributes	Very Low	Low	Nominal	High	Very High	Extra High
ACAP	1.46	1.19	1.00	0.86	0.71	----
AEXP	1.29	1.13	1.00	0.91	0.85	----
PCAP	1.42	1.17	1.00	0.86	0.70	----
VEXP	1.21	1.10	1.00	0.90	----	----
LEXP	1.14	1.07	1.00	0.95	----	----

Project Attributes	Very Low	Low	Nominal	High	Very High	Extra High
MODP	1.24	1.10	1.00	0.91	0.82	----
TOOL	1.24	1.10	1.00	0.91	0.83	----
SCED	1.23	1.08	1.00	1.04	1.10	----

COCOMO – Intermediate – EAF

- EAF also known as Effort Adjustment Factory
- Is Calculated by multiplying all the values that have been obtained after categorizing each cost driver

COCOMO – Intermediate – Equation

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

$$time = c(Effort)^d$$

$$Person required = \text{Effort / time}$$

Software Projects	a	b	c	d
Organic	3.2	1.05	2.5	0.38
Semi – detached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

COCOMO – Intermediate – Numerical 1

A new project with estimated 400 KLOC embedded system has to be developed. Project Manager has a choice of hiring from 2 pools of developers: Very highly capable (with apps) with very little experience in programming language **OR** developers of low quality but a lor of programming language experience. Which is better choice in terms of 2 pools

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

$$time = c(Effort)^d$$

Case 1: Effort Adjustment Factor
1) Application Experience – Very High
2) Language Experience – Very Low

0.82*0.14 = 0.934

Case 2: Effort Adjustment Factor
1) Application Experience – Very Low
2) Language Experience – Very High

1.29*0.95 = 1.22

COCOMO – Detailed Mode

- Detailed Mode is Phase Sensitive
- It calculates the effect of cost drivers on each phase of SDLC
- It uses phase – sensitive effort multipliers for each cost driver to determine the amount of effort and time required to complete each phase of SDLC.
- It establishes Module – Subsystem – System Hierarchy. It means that the rating that is given to the cost driver is done at that level only where the cost driver is most sensitive or most susceptible to variable.

COCOMO – Detailed Mode

- **ADJUSTMENT FACTOR: A = 0.4(DD) + 0.3 (C) + 0.3 (I)** where
  - DD: - Design Document, C:- Code, I : - testing / integration
- **Depending on the above Factor the Size (KLOC) would also be adjusted. Formula is (S * A)/100**

$$Effort = \mu_p E$$

$$Development\ time = \tau_p D$$

COCOMO – Detailed Mode

Personnel Attributes	Plan & Reqmt	System Design	Detail Design	Code & Test	Integration & Test
Life Cycle phase value of **µp**					
Organic Small S = 2	0.06	0.16	0.26	0.42	0.16
Organic Medium S = 32	0.06	0.16	0.24	0.38	0.22
Semi Detached Medium S = 32	0.07	0.17	0.25	0.33	0.25
Semi Detached Large S = 128	0.07	0.17	0.24	0.31	0.28
Embedded Large S = 128	0.08	0.18	0.25	0.26	0.31
Embedded Extra Large S = 320	0.08	0.18	0.24	0.24	0.34

Personnel Attributes	Plan & Reqmt	System Design	Detail Design	Code & Test	Integration & Test
Life Cycle phase value of **Tp**					
Organic Small S = 2	0.10	0.19	0.24	0.39	0.18
Organic Medium S = 32	0.12	0.19	0.21	0.34	0.26
Semi Detached Medium S = 32	0.20	0.26	0.21	0.27	0.26
Semi Detached Large S = 128	0.22	0.27	0.19	0.25	0.29
Embedded Large S = 128	0.36	0.36	0.18	0.18	0.28
Embedded Extra Large S = 320	0.40	0.38	0.16	0.16	0.30

Consider a project with the following main components 1) Screen Edit – 4KLOC, 2) CLI – 2 KLOC, 3) File I/p and O/p – 1 KLOC, 4) Cursor Movement – 2 KLOC, 5) Screen Movement – 3 KLOC. Using COCOMO determine:

a) Overall Cost and Schedule estimates where SW Reliability is High, Language Experience is low, Product Complexity is High, and Analyst Capability is High

b) Cost and Schedule estimate for different phases

First Lesson Summary

Here is what we learned

- **First skill:**
Lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum

- **Second skill:**
Lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum

- **Third skill:**
Lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum