

• Software Engineering •



CHAPTER-4

Structured System Design

Design Concepts

Software design is the **process of transforming user requirements into a suitable form**, which helps the programmer in software coding and implementation.

During the software design phase, the design document is produced, based on the customer requirements as documented in the SRS document. Hence, this **phase aims to transform the SRS document into a design document**.

Design Concepts

Objectives of Software Design

Correctness: it should correctly implement all the functionalities of the system.

Efficiency: A good software design should address the resources, time and cost optimization issues.

Flexibility: The ability to adapt and changes easily (modifications, enhancements, and scalability).

Understandability: understandable & all the modules are arranged in layers.

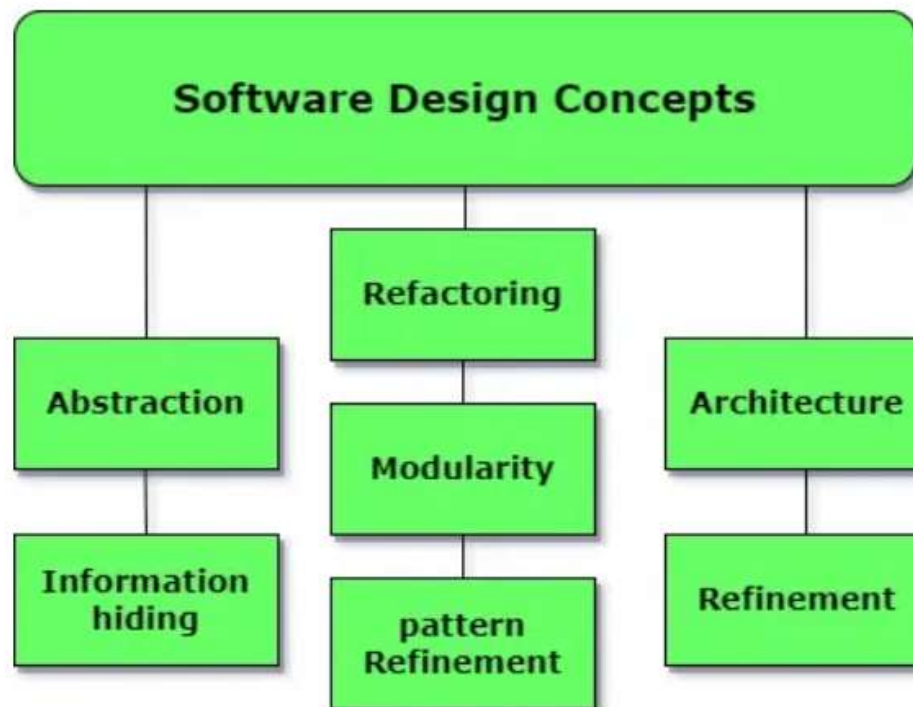


Design Concepts

Completeness: The design should have all the components like data structures, modules, external interfaces etc.

Maintainability: A good software design aims to create a system that is easy to understand, modify, and maintain over time.

Design Concepts



Software Design Concepts

Design Concepts

❖ **Abstraction (Hide Irrelevant data):** Abstraction simply means to **hide the details** to reduce complexity and increase efficiency or quality.

❖ **Modularity (subdivide the system):** Modularity simply means **dividing the system or project into smaller parts** to reduce the complexity of the system or project. In the same way, modularity in design means **subdividing a system into smaller parts so that these parts can be created independently** and then use these parts in different systems to perform different functions.



Design Concepts

❖ **Architecture (design a structure of something):** Architecture simply means a technique to design a structure of something. Architecture in designing software is a concept that focuses on various elements and the data of the structure. These components interact with each other and use the data of the structure in architecture.

❖ **Refinement (removes impurities):** Refinement simply means to refine something to remove any impurities if present and increase the quality.

Design Concepts

The refinement concept of software design is a process of developing or presenting the software or system in a detailed manner which means elaborating a system or software. Refinement is very necessary to find out any error if present and then to reduce it.

❖ **Pattern (a Repeated form):** A pattern simply means a repeated form or design in which the same shape is repeated several times to form a pattern. The pattern in the design process means the repetition of a solution to a common recurring problem within a certain context.

Design Concepts

- ❖ **Information Hiding (Hide the Information):** Information hiding simply means to **hide the information** so that it cannot be accessed by an unwanted party. In software design, information hiding is achieved by **designing the modules in a manner that the information gathered or contained in one module is hidden** and can't be accessed by any other modules.
- ❖ **Refactoring (Reconstruct something):** Refactoring simply means **reconstructing something** in such a way that it does not affect the behavior of any other features.

Design Concepts

Refactoring in software design means reconstructing the design to reduce complexity and simplify it without impacting the behavior or its functions. Fowler has defined refactoring as “the process of changing a software system in a way that it won’t impact the behavior of the design and improves the internal structure”.



Design Model

The design model is a representation of the system that **bridges the gap between requirements and implementation**. It serves as a blueprint for the software system and focuses on **how the system will work, detailing its architecture, components, interfaces, and interactions**.



Design Model

Importance of Design Model

- Provides a **clear understanding** of the system.
- Helps in **identifying** potential issues **early in the development process**.
- Facilitates **communication among stakeholders**.
- **Serves as a guide for developers** during implementation.
- Ensures **consistency and maintainability in the system**.



Design Model

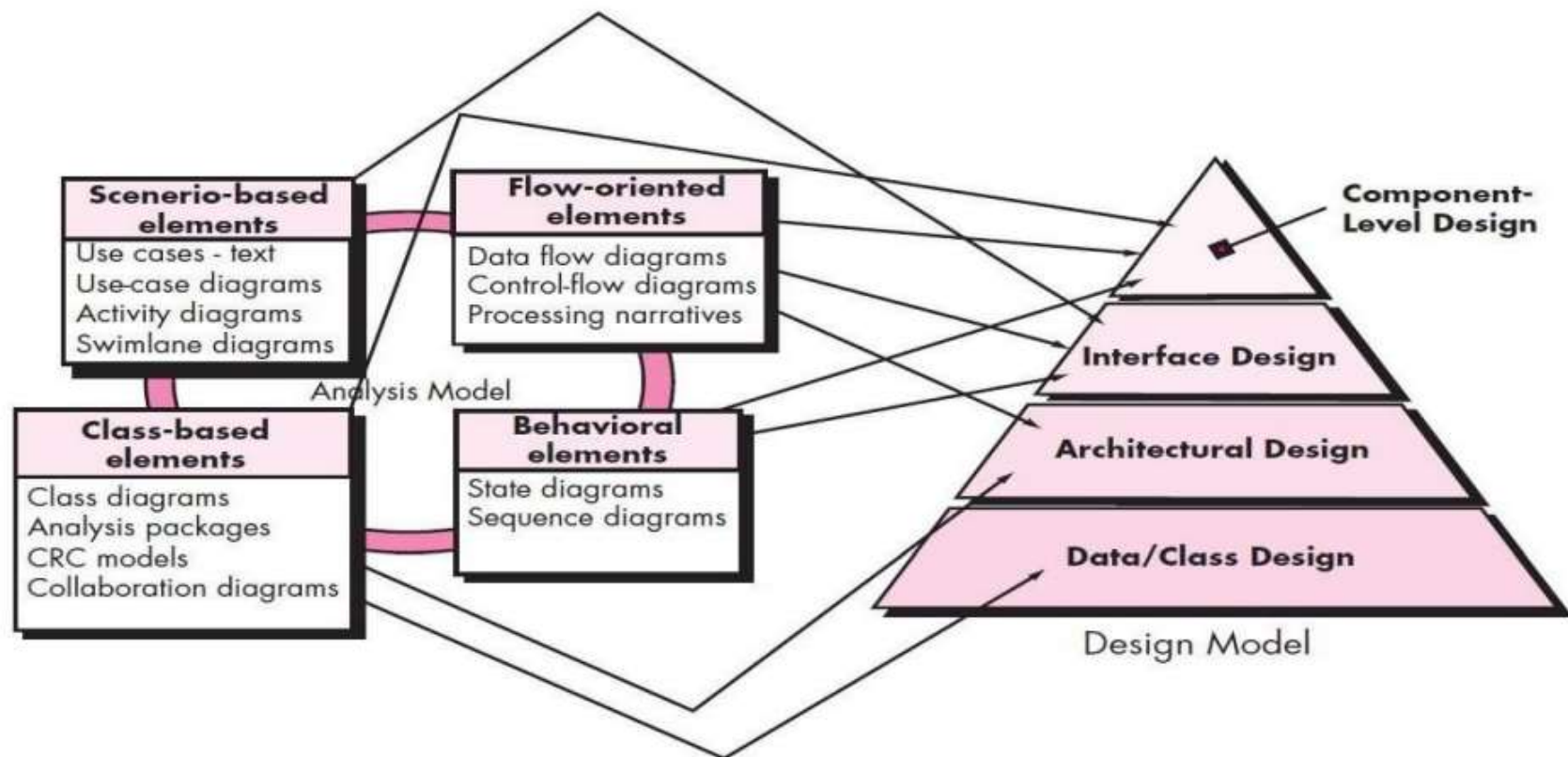


Fig : Translating the requirements model into the design model



Design Model

Data Design

Sometimes referred to as data architecting. It creates a model of data or information that is **represented at a high level of abstraction** (the customer/user's view of data).

- **Customer's/ User's View:**
 - Data Architecting (Creates a model of data that is represented at a high level of abstraction). (Build Architecture of Data)
- **Program Component Level:** The design of Data structure & algorithms.
- **Application Level:** Translate Data Model into a database.
- **Business Level:** Data warehouse(Reporting & Analysis of DB) & Data mining(Analysis).

Design Model

Architectural Design

Provides an **overall view of the software product** (Similar like Floor Plan of house).

The architectural model is derived from **three sources**:

- (1) **Information about the application domain** for the software to be built.
- (2) **Specific requirements model elements** such as - data flow diagrams or analysis classes, their relationships and collaborations for the problem at hand.
- (3) **The availability of architectural styles and patterns.**

Design Model

Interface Design

The interface design elements for software represent **information flows into and out of the system** and how it is communicated among the components defined as part of the architecture.

There are three important elements of interface design:

- (1) The **user interface** (UI);
- (2) **External interfaces** to other systems, devices, networks, or other producers or consumers of information;
- (3) **Internal interfaces** between various design components.



Design Model

These interface design elements allow the software to communicate externally and enable internal communication and collaboration among the components that populate the software architecture.

Design Model

Component-Level Design

The component-level design for software **fully describes the internal detail of each software component.**

To accomplish this, the component-level design defines data structures for all local data objects and algorithmic detail for all processing that occurs within a component and an interface that allows access to all component operations.



Design Model

Deployment-Level Design

Elements **indicate how software functionality and subsystems will be allocated within the physical computing environment that will support the software.**

Software Architecture

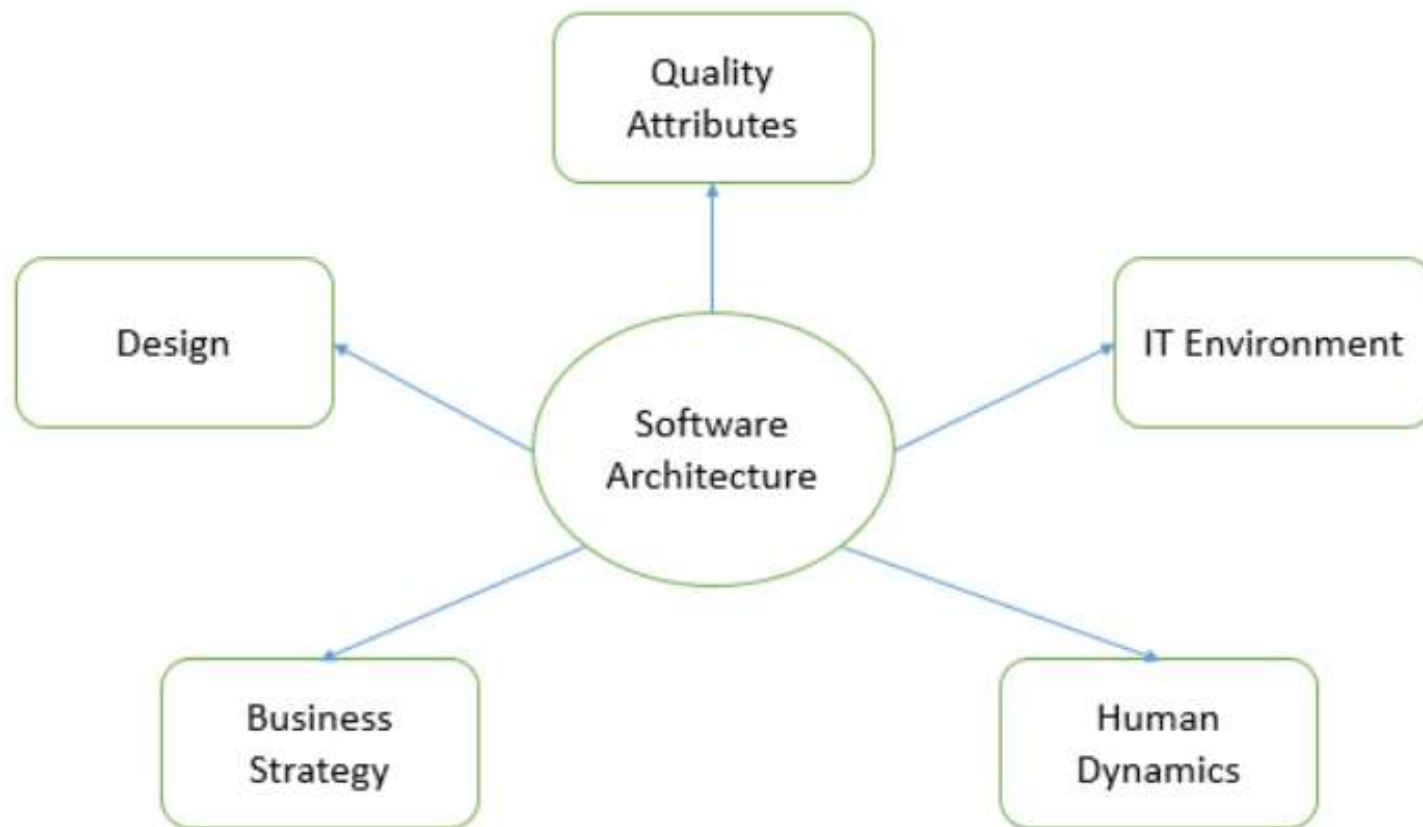
The architecture of a system **describes its major components, their relationships** (structures), and **how they interact with each other**.

It refers to the **high-level structure of a software system**.

It defines how software components interact, how they are organized, and how the system fulfills its functional and non-functional requirements.

A well-designed ***architecture acts as a blueprint***, guiding the development and evolution of the software.

Software Architecture





Software Architecture

Importance of Software Architecture

1. **Foundation for Development:** Provides a clear structure for developers to follow.
2. **Scalability:** Ensures the system can grow with user needs and technological changes.
3. **Maintainability:** Simplifies troubleshooting and future updates.
4. **Risk Management:** Identifies and addresses risks early in the development process.
5. **Communication:** Acts as a common language between stakeholders (developers, clients, testers).

Software Architecture

Principles of Software Architecture

Modularity

- Divide the system into smaller, manageable pieces.
- Example: Divide an e-commerce app into modules like User, Product, and Payment.

Abstraction

- Hide unnecessary details and focus on essential features.
- Example: Provide only essential methods for accessing database records.

Software Architecture

Encapsulation:

- Protect internal details of components and expose only necessary functionality.

Scalability:

- Design the system to handle growth in users or data.
- Example: Use a micro-services architecture to add services independently.



Software Architecture

Characteristics of Software Architecture

- **Scalability:** Supports growth in functionality or data without significant redesign.
- **Performance:** Optimized to meet the desired response times and throughput.
- **Flexibility:** Allows changes or extensions with minimal impact on the existing system.
- **Security:** Ensures data integrity and protection from unauthorized access.
- **Maintainability:** Facilitates debugging

Data Design

Data design in software engineering **involves creating a structured representation of data and its relationships within a system.**

This process is crucial for developing reliable and efficient system designs, enabling developers to understand, manage and manipulate data effectively.

Data Design

Best Practices for Data Design

- 1. Understand Business Requirements:** Analyze business requirements, user needs, and data dependencies to make informed data modeling choices.
- 2. Use Descriptive Names:** Select simple but meaningful names for entities, attributes, and relationships to avoid confusion and maintain readability.
- 3. Maintain Consistency:** Ensure uniformity by following guidelines like naming conventions, data types, and relationships.

Data Design

Best Practices for Data Design

- 3. Document the Data Model:** Provide thorough documentation, including entity definitions, attribute descriptions, relationship cardinalities, and business logic.
- 4. Iterate and Refine:** Continuously refine the data model based on feedback, changes in requirements, or evolving business needs

Data Design

Benefits of Data Design

- 1. Clarity and Communication:** Facilitates clear communication and understanding of data requirements among stakeholders.
- 2. Efficiency:** Promotes efficient database operations, reducing retrieval and update times.
- 3. Scalability:** Supports growth and changes in data requirements over time.
- 4. Data Quality:** Enforces consistency, quality standards, and data governance practices.

Architectural Styles and Patterns

Architectural Styles

Architectural styles define the **overall system structure** and its **organization**.

- Architectural styles in software engineering define the high-level structure and organization of a software system.
- Defines how components are arranged and interact in the system.
- Establishes a structural blueprint for system organization.
- They provide a set of rules and guidelines for arranging components and their interactions.
- Determines system behavior and scalability.

Architectural Styles and Patterns

Common Architectural Styles

1. Monolithic Architecture
2. Layered Architecture
3. Client-Server Architecture
4. Microservices Architecture
5. Event-Driven Architecture
6. Service-Oriented Architecture (SOA)
7. Component-Based Architecture

Architectural Styles and Patterns

Common Architectural Styles

- 8. Peer-to-Peer Architecture
- 9. N-Tier Architecture
- 10. Cloud-Based Architecture

Architectural Styles and Patterns

Architectural Patterns

Architectural patterns provide **specific solutions to common design problems within that structure.**

- Provides best practices for solving specific design problems within an architectural style.
- More specific, focusing on the design of components and interactions.
- Provides guidance for implementing parts of the system effectively.
- Helps optimize and refine the design within an existing style.

Architectural Design

“The process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system”

The software needs an architectural design to represent the design of the software.

Architectural Design

Components of Architectural Design

- System Organization
- Abstraction and Decomposition
- Design Patterns & Architectural Styles
- Data Management
- Interaction and Communication
- Scalability & Security
- Communication and Documentation
- Validation and Testing & Maintainability



Component level design

Component level design refers to the process of breaking down a system into its constituent parts to better understand how they interact and connect.

It is the phase that focuses on defining and developing the software components that will be used to build the overall system architecture.

Component level design

Characteristics of Component-Based Design

- ❖ Modularity
- ❖ Reusability
- ❖ Interoperability
- ❖ Encapsulation
- ❖ Scalability

Parul University

Component level design

Types of Components

1. UI Components
2. Service Components
3. Data Components
4. Infrastructure Components
5. Integration Components
6. Reusable Components

Procedural Design

Procedural design emphasizes the procedures or functions that operate on data. Procedures, also known as routines, subroutines or functions, simply consist of a series of computational steps to be carried out.

Procedural Design

Key Principles

- 1. Modularity:** Code is organized into procedures that can be called to perform specific tasks.
- 2. Control Structures:** Defines the flow of control through procedures using structures like loops and conditional statements.
- 3. Data Abstraction:** Procedures may manipulate data without exposing the underlying representation.

Object Oriented Design

Object-oriented design (OOD) is a design philosophy based on concepts of "objects", which are instances of classes, encapsulating both data and behavior

PU

Object Oriented Design

Key Principles

- 1. Classes and Objects:** A class defines a blueprint for objects, which are instances of classes.
- 2. Inheritance:** Mechanism by which one class can inherit properties and methods from another.
- 3. Encapsulation:** Bundling data and methods that operate on the data into a single unit (class).
- 4. Polymorphism:** Ability to process objects differently based on their data type or class..

Difference between Data and Information

Data

Raw, fact, figures, symbols, unorganized or unstructured fact is called data. **Or** we can say - Data are individual units of information. it can be generated from different sources like sensors, surveys, transactions, social media etc.

- Data contains numbers, text, observation, statements and characters in a raw form.
- Data is meaningless.
- Data does not directly help in decision making.



Difference between Data and Information

Types of Data

Quantitative data	Numerical form, like 50 Kg, 165 cm, 15887 etc.
Discrete Data	Whole numbers. example the number of employees in a department.
Continuous Data	Data that can take any value within a range. For example, wind speed, and temperature.
Qualitative data	It's available in a descriptive form for example name, gender, address, and features of a person.
Nominal Data	Data that represents categories with no inherent order. For example, colours, and gender.

Difference between Data and Information

Types of Data

Ordinal Data	Data that represents categories with a specific order or ranking. For example, ranking satisfaction levels as "poor," "average," or "excellent."
Categorical Data	The data which represents categories or labels and is often qualitative is called categorical data. It can include nominal and ordinal data.
Numerical Data	This type of data includes numbers.

Difference between Data and Information

Types of Data

Time Series Data	Data collected over time intervals like stock prices, weather data, and sales figures.
Spatial Data	Data associated with geographic locations like Google maps, GPS data, and satellite images.

Difference between Data and Information

Information

Processed data is called information. Information is presented in the form of words, language, thoughts, and ideas.

- Information is organized.
- It is meaningful.
- It is always useful and used in decision-making.
- Information depends on Data.
- Example of information: Merit List, receipts, reports, report cards etc.

E-R Diagram

The Entity Relationship Model is a model for identifying entities (like student, car or company) to be represented in the database and representation of how those entities are related.

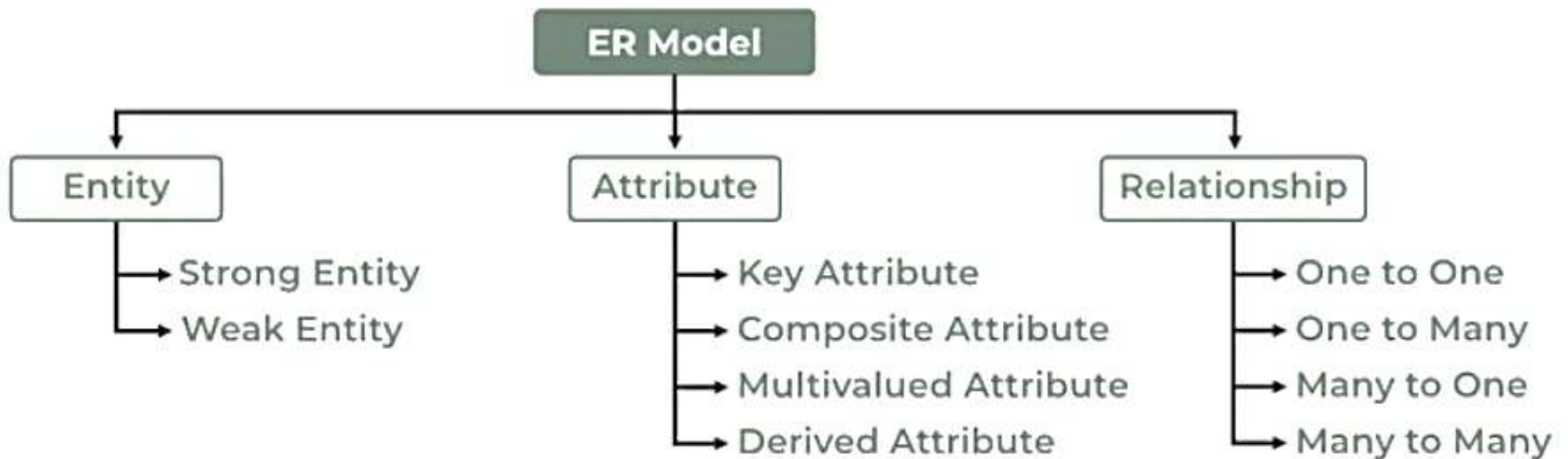
The ER data model specifies enterprise schema that represents the overall logical structure of a database graphically.

ER Model consists of Entities, Attributes, and Relationships among Entities in a Database System.

E-R Diagram

Components of ER Diagram






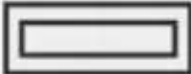
ER Model consists of Entities, Attributes, and Relationships among Entities in a Database System.





E-R Diagram

Symbols used in ER Diagram

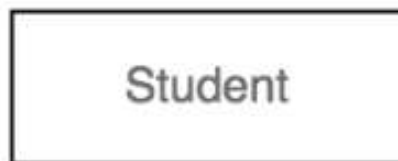
Figures	Symbols	Represents
Rectangle		Entities in ER Model
Ellipse		Attributes in ER Model
Diamond		Relationships among Entities
Line		Attributes to Entities and Entity Sets with Other Relationship Types
Double Ellipse		Multi-Valued Attributes
Double Rectangle		Weak Entity

E-R Diagram

Entity

An Entity may be an object with a physical existence – a particular person, car, house, or employee.

An Entity is an object of Entity Type and a set of all entities is called an entity set.



Entity Type



Entity Set

E-R Diagram

Types of Entity

1. Strong Entity

It is a type of entity that **has a key Attribute**. it is represented by **a rectangle**.



2. Weak Entity

An entity **does not have a primary key attribute**. It is represented by a **double rectangle**.



E-R Diagram

Attribute

Attributes are the **properties that define the entity type**. In ER diagram, the attribute is **represented by an oval**.



For example- Roll_No, Name, DOB, Age, Address, and Mobile_No are the attributes that define entity type Student.

E-R Diagram

Types of Attribute

1. Key Attribute

The attribute which **uniquely identifies each entity in the entity set** is called the key attribute. The key attribute is represented by an **oval with underlying lines**.

For example- Roll_No will be unique for each student.



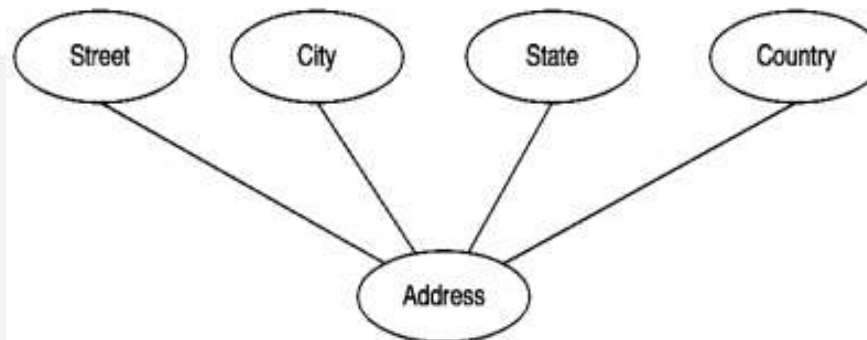
2. Composite Attribute

An attribute **composed of many other attributes** is called a composite attribute. The composite attribute is represented by **an oval comprising of ovals**.

For example- the Address attribute of the student Entity type consists of Street, City, State, and Country.

E-R Diagram

Types of Attributes



3. Multivalued Attribute

An attribute consisting of **more than one value for a given entity**. A multivalued attribute is represented by **a double oval**.

For example, Phone_No (can be more than one for a given student).



E-R Diagram

Types of Attributes

4. Derived Attribute

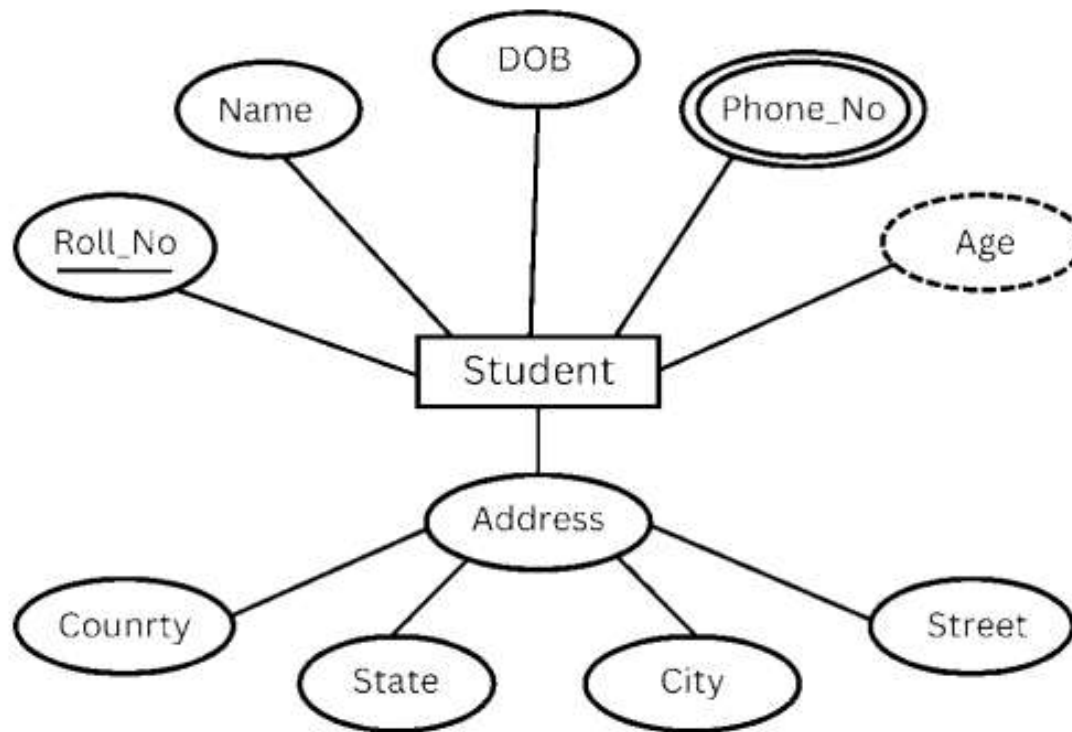
An attribute that can be **derived from other attributes** of the entity type is known as a derived attribute. The derived attribute is represented by **a dashed oval**.

For example - Age (can be derived from DOB).



E-R Diagram

The Complete Entity Type Student with its Attributes can be represented as:



E-R Diagram

Relationship

The association among entities is known as relationship. Relationships are represented by the **diamond-shaped box**.

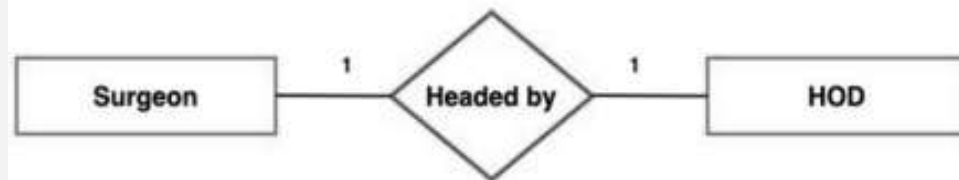
For example - an employee works_at a department, a student enrolls in a course. Here Works_at and Enrolls are called relationships.



E-R Diagram

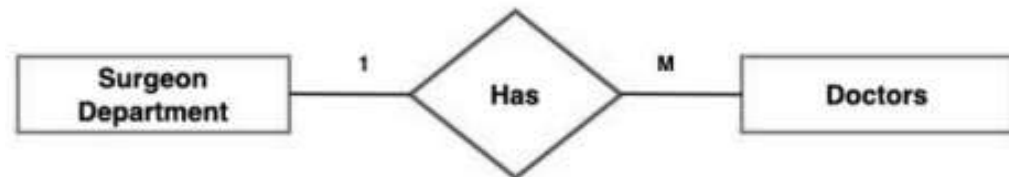
Relationship

1. **One-to-One:** When each entity in each entity set can take part only once in the relationship, the cardinality is one-to-one.



2. **One-to-Many:** When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationships.

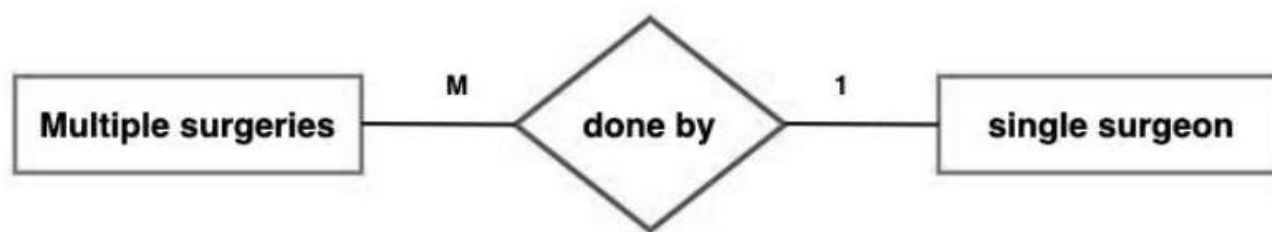
For example, a client can place many orders; a order cannot be placed by many customers.



E-R Diagram

Relationship

3. Many-to-One: More than one entity from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.



Dataflow Model

Data Flow Diagram (DFD) is a **graphical representation of data flow** in any system. It is capable of illustrating **incoming data flow**, **outgoing data flow**, and **stored data**. Data flow diagram describes anything about how data flows through the system.


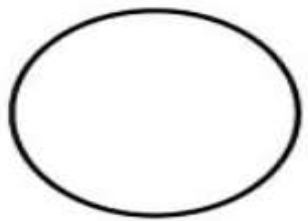


Dataflow Model

Components of Data Flow Diagrams

DFDs consist of **four main components**:

- 1. Process:** Represents the transformation of **input data to output data**. It is usually depicted as a **circle or a rounded rectangle**.
- 2. Data Flow:** Shows the **direction of data movement between processes**, data stores, and external entities. It is **represented by arrows**.
- 3. Data Store:** Indicates where data is stored within the system. It is depicted as **two parallel lines**.
- 4. External Entity:** Represents **sources or destinations of data outside the system**. It is shown as a **rectangle**.

Dataflow Model

Symbol	Name	Function
	Data flow	Used to Connect Processes to each , other , to sources or Sinks; te arrow head indicates direction of data flow.
	Process	Performs Some transformation of Input data to yield output data.
	Source of Sink (External Entity)	A Source of System inputs or Sink of System outputs.
	Data Store	A repository of data; the arrow heads indicate net inputs and net outputs to store.

Dataflow Model

Levels of Data Flow Diagrams

DFDs can be divided into different levels to represent varying degrees of detail:

1. **0-Level DFD:** Also known as a **context diagram**, it provides an overview of the entire system as a single process with its interactions with external entities.
2. **1-Level DFD:** Breaks down the main process of the 0-level DFD into sub-processes, providing more detail about the system's functionality.
3. **2-Level DFD:** Further decomposes the sub-processes of the 1-level DFD into more detailed processes.

Dataflow Model

Level-0 DFD

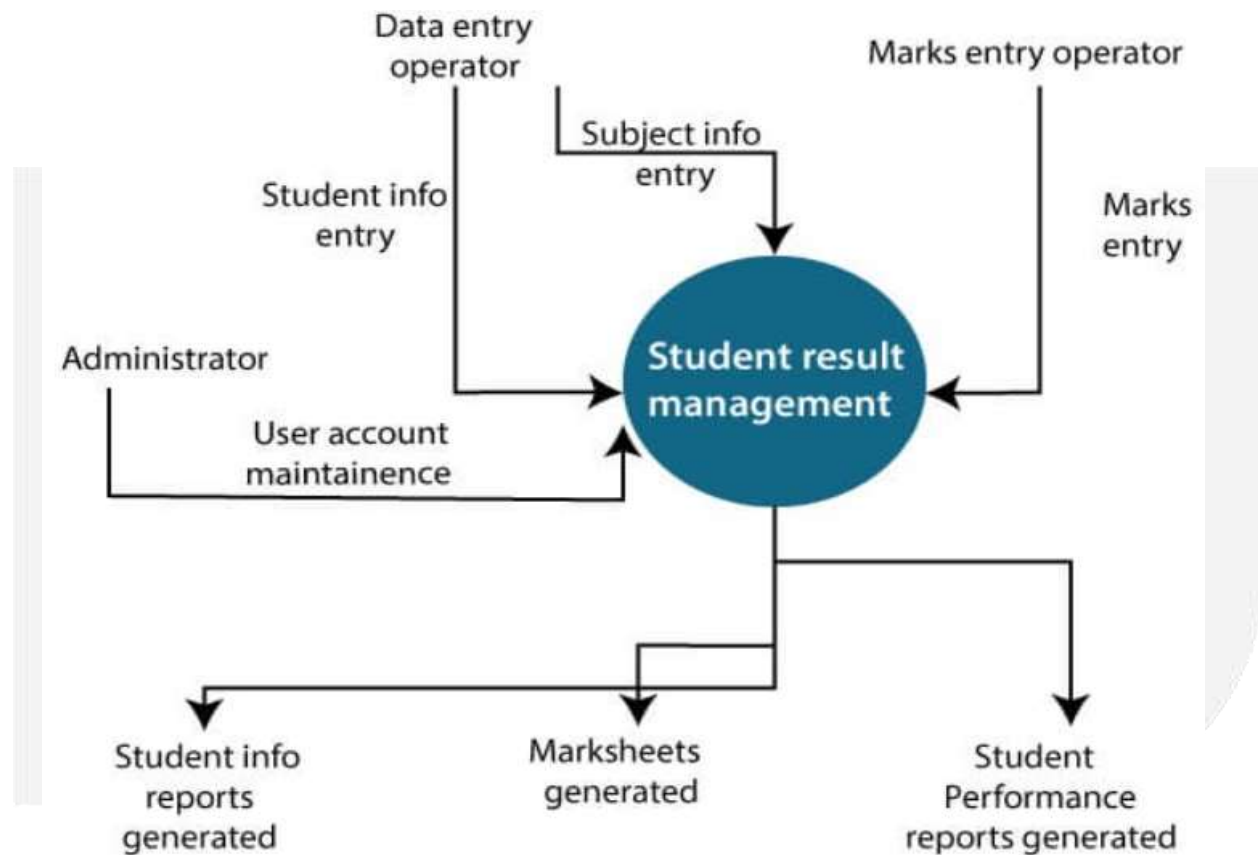


Fig: Level-0 DFD of result management system



Dataflow Model

Level-1 DFD

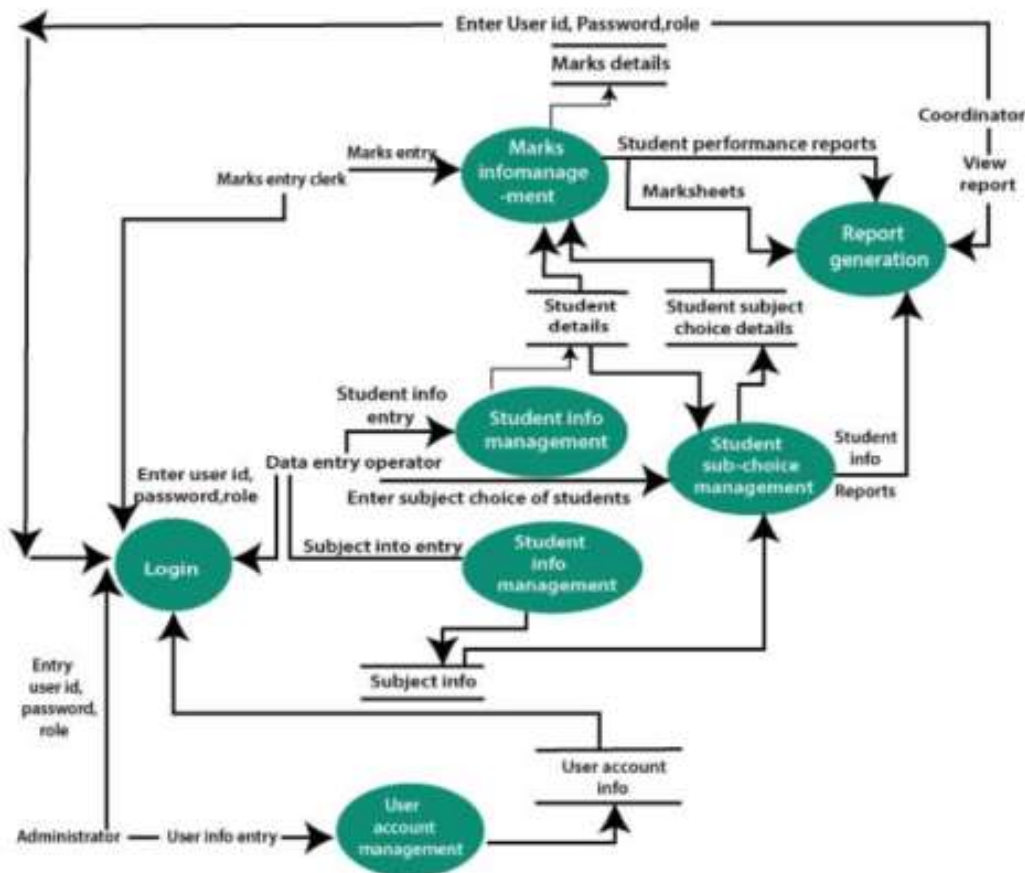
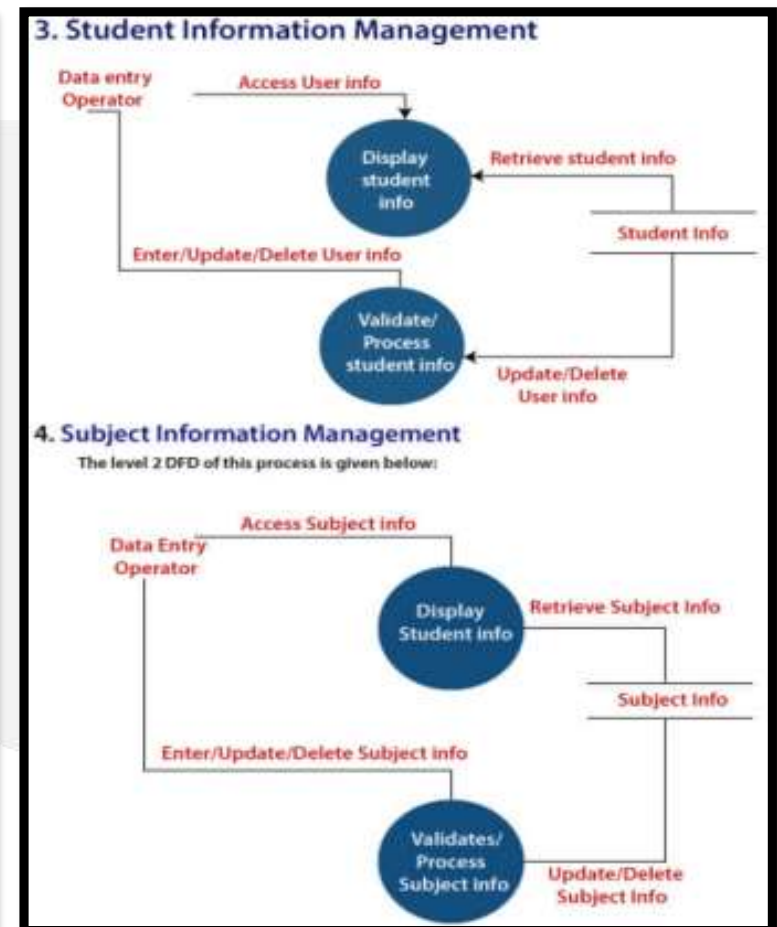
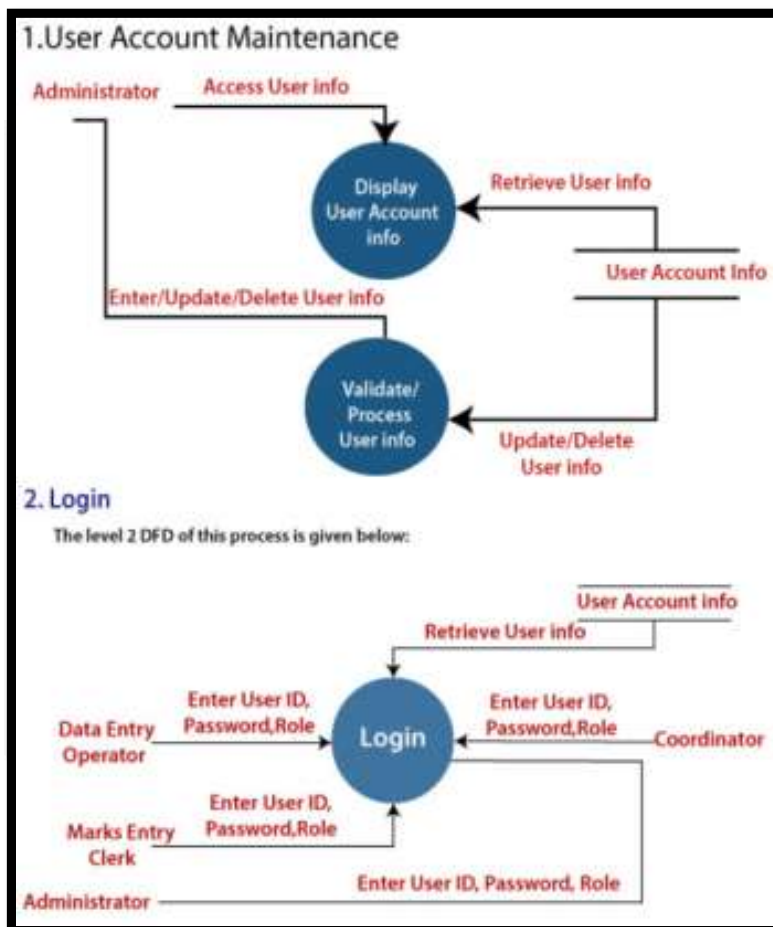


Fig: Level-1 DFD of result management system



Dataflow Model

Level-2 DFD



Control Flow Graph

The control flow graph was originally developed by *Frances E. Allen*.

A Control Flow Graph (CFG) is the graphical representation of control flow or computation during the execution of programs or applications. Control flow graphs are mostly used in static analysis as well as compiler applications, as they can accurately represent the flow inside a program unit.

There exist 2 designated blocks in the Control Flow Graph:

- 1. Entry Block:** The entry block allows the control to enter into the control flow graph.
- 2. Exit Block:** Control flow leaves through the exit block.



Control Flow Graph

Advantage of CFG

- **Visualizes program flow:** Easy to see how a program runs.
- **Helps find errors:** Detects [unreachable code](#) or infinite loops.
- **Useful for optimization:** Improves program performance.
- **Aids testing:** Ensures all parts of the code are tested.

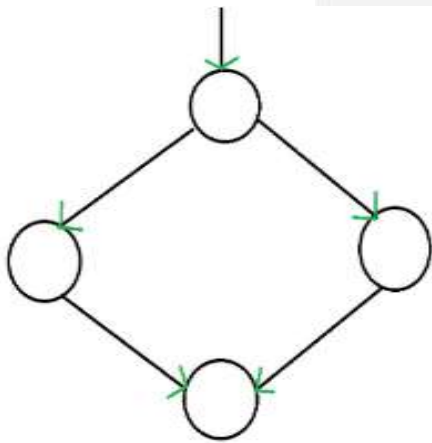
Disadvantages of CFG

- ❖ **Complex for big programs:** Hard to understand.
- ❖ **No unpredictable behavior:** Can't show unclear paths.
- ❖ **No data info:** Only shows program flow.
- ❖ **Not scalable:** Becomes messy for large projects.

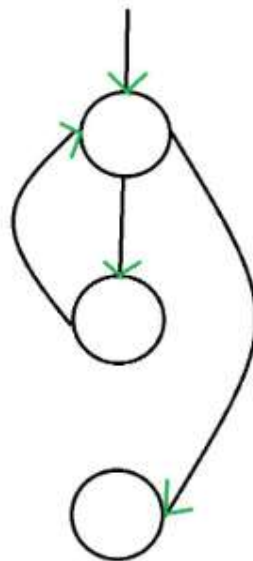
Control Flow Graph

General Control Flow Graphs

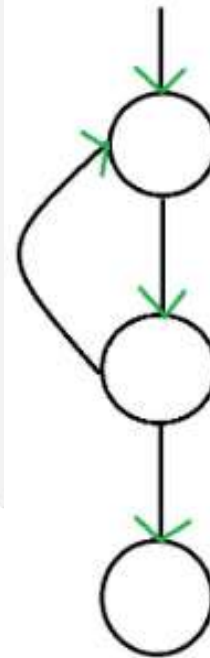
1. if else



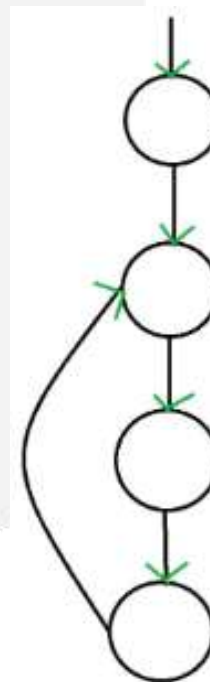
2. while



3. do while



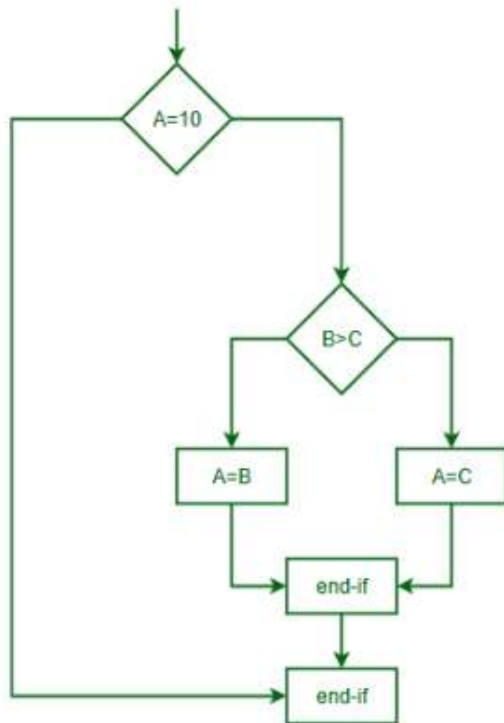
4. for



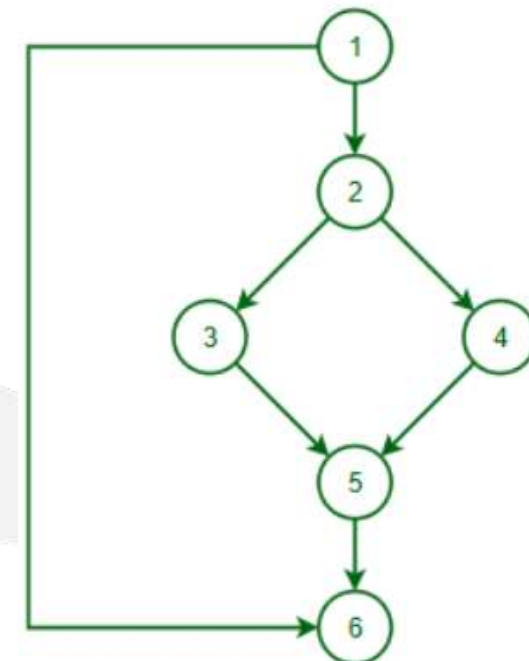
Control Flow Graph

General Control Flow Graphs

Flowchart



Control Flow Graph



Process and Control Specification

In process specification, **the work starts with the testers who test for various types of bugs that can be found in the system or software and then they report these bugs to the manager**, once the report has been submitted the manager selects the team and gives them the task to fix these bugs.

we can follow the below steps:

Identify the Bug -> Log the Bug -> Assign Priority for the Bug -> Fixing the Bug -> Verifying the Fix -> Updating the Status of Bug.

Process and Control Specification

It helps us **to create a clear process and set of rules that can define the control of the system**. By defining the control specification, we can have a better understanding of the system.

Data Dictionary

Data which simply means information being collected through some sources and **Dictionary** means where this information is available.

A Data Dictionary can be defined as **a collection of information on all data elements or contents of databases** such as **data types and text descriptions of the system.**

It makes it easier for users and analysts to use data as well as understand and have common knowledge about inputs, outputs, components of a database, and intermediate calculations.

Data Dictionary

Data Dictionary provides all information about names that are used in system models. Data Dictionary also provides information about **entities, relationships, and attributes** that are present in the system model.

Data Dictionary

How to Create a Data Dictionary?

Data Dictionary can be formed with the help of the following relational database like MySQL, SQL Server, etc. Several notations are present everywhere which enhances the quality of the Data Dictionary.

Database Administrators can have the following templates like SQL Server, that help in making Data Dictionary.

Data Dictionary

Notations in Data Dictionary

Data Construct	Notation	Meaning
Composition	=	Is composed of
Sequence	+	Represents AND
Selection	[]	Represents OR
Repetition	{ } ⁿ	Represents n repetitions or repetition for n times
Parentheses	()	Optional data that may or may not be present
Comment	* ... *	Delimits a comment or commented information



Data Dictionary

Example: Let us understand this by taking an example of the reservation system. In the reservation system, **“passenger”** is a data item whose information is available on the data dictionary as follows:

Keys	Values
Name	Passenger
Aliases	None
Where or how it's used?	Passenger's query (input) Ticket (output)
Description	<ul style="list-style-type: none"> • Passenger = Passenger_name + Passenger_address • Passenger_name = Passenger_lastname + Passenger_firstname + Passenger_middle_initial • Passenger_address = Local_address + Community_address + Zip_code • Local_address = House_number + street_name + Apartment_number • Community_Address = City_name + State_name

x DIGITAL LEARNING CONTENT

o



Parul[®] University



www.paruluniversity.ac.in