By Shubh Thakare +919595939264

Unit 2. Requirements engineering process.

Requirements engineering process :

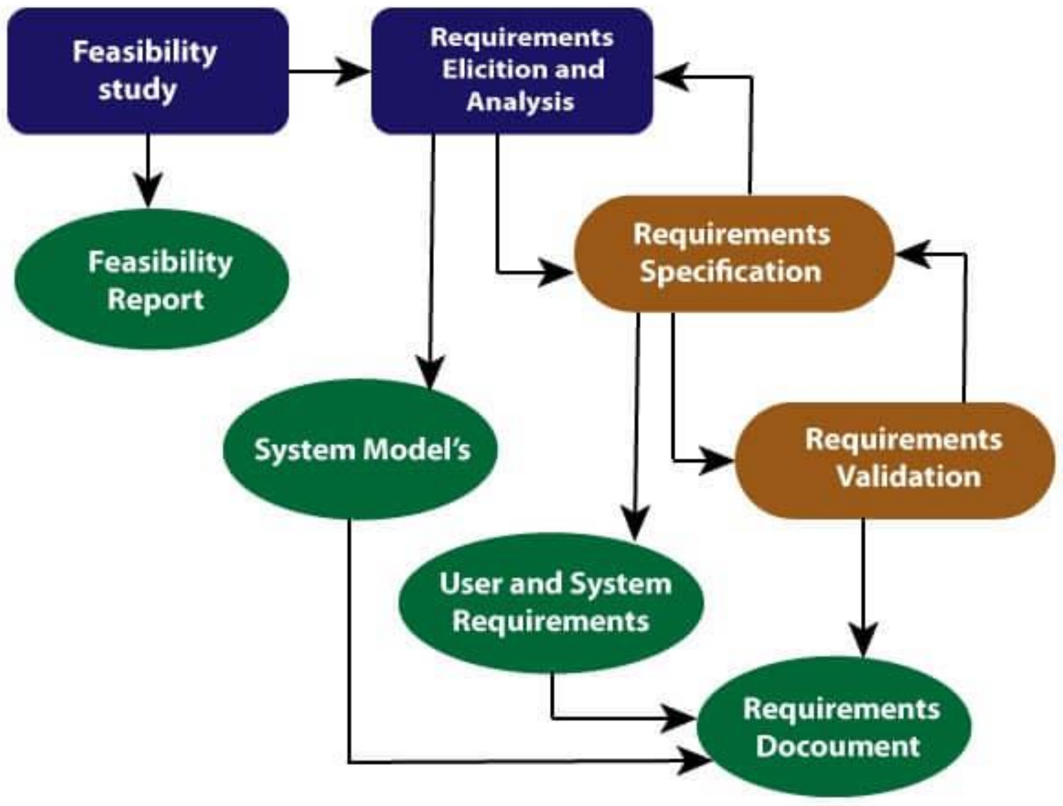
Requirement Engineering

Requirements engineering (RE) refers to the process of defining, documenting, and maintaining requirements in the engineering design process. Requirement engineering provides the appropriate mechanism to understand what the customer desires, analyzing the need, and assessing feasibility, negotiating a reasonable solution, specifying the solution clearly, validating the specifications and managing the requirements as they are transformed into a working system. Thus, requirement engineering is the disciplined application of proven principles, methods, tools, and notation to describe a proposed system's intended behavior and its associated constraints.

Requirement Engineering Process

It is a four-step process, which includes -

1. Feasibility Study
2. Requirement Elicitation and Analysis
3. Software Requirement Validation
4. Software Requirement Management



Requirement Engineering Process

1.Feasibility studies

The objective behind the feasibility study is to create the reasons for developing the software that is acceptable to users, flexible to change and conformable to established standards.

Types of Feasibility:

1. **Technical Feasibility** - Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.
2. **Operational Feasibility** - Operational feasibility assesses the range in which the required software performs a series of levels to solve business problems and customer requirements.
3. **Economic Feasibility** - Economic feasibility decides whether the necessary software can generate financial profits for an organization.

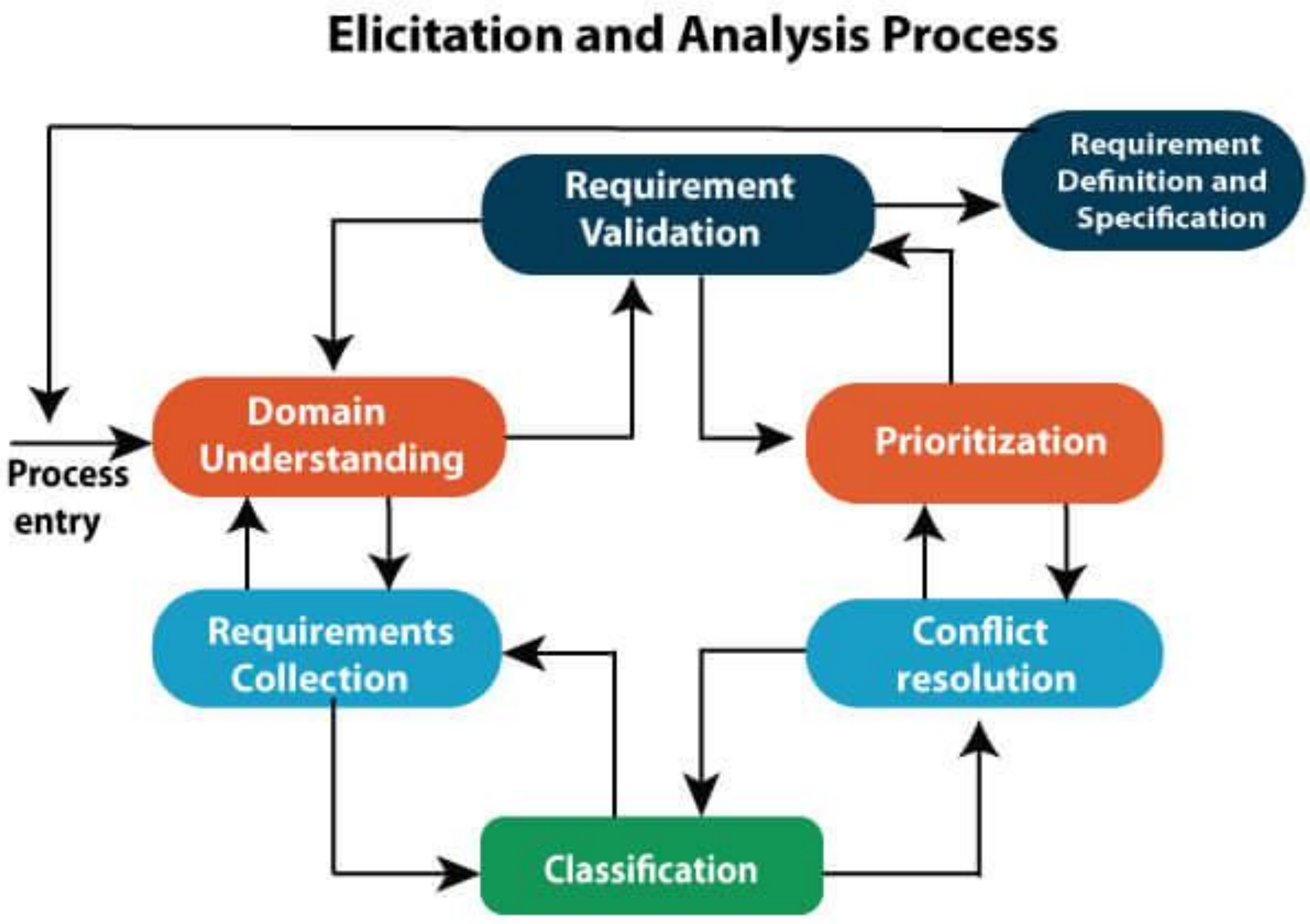
2.Requirements elicitation and analysis

This is also known as the **gathering of requirements**. Here, requirements are identified with the help of customers and existing systems processes, if available.

Analysis of requirements starts with requirement elicitation. The requirements are analyzed to identify inconsistencies, defects, omission, etc. We describe requirements in terms of relationships and also resolve conflicts if any.

Problems of Elicitation and Analysis

- Getting all, and only, the right people involved.
- Stakeholders often don't know what they want
- Stakeholders express requirements in their terms.
- Stakeholders may have conflicting requirements.
- Requirement change during the analysis process.
- Organizational and political factors may influence system requirements.



3.Requirements validation

After requirement specifications developed, the requirements discussed in this document are validated. The user might demand illegal, impossible solution or experts may misinterpret the needs. Requirements can be the check against the following conditions -

- If they can practically implement
- If they are correct and as per the functionality and specially of software
- If there are any ambiguities
- If they are full
- If they can describe

Requirements Validation Techniques

- **Requirements reviews/inspections:** systematic manual analysis of the requirements.
- **Prototyping:** Using an executable model of the system to check requirements.
- **Test-case generation:** Developing tests for requirements to check testability.
- **Automated consistency analysis:** checking for the consistency of structured requirements descriptions.

4. Requirements management

Requirement management is the process of managing changing requirements during the requirements engineering process and system development.

New requirements emerge during the process as business needs a change, and a better understanding of the system is developed.

The priority of requirements from different viewpoints changes during development process.

The business and technical environment of the system changes during the development.

Prerequisite of Software requirements

Collection of software requirements is the basis of the entire software development project. Hence they should be clear, correct, and well-defined.

A complete Software Requirement Specifications should be:

- Clear
- Correct

- Consistent
- Coherent (Logical Consistency)
- Comprehensible (Understandable)
- Modifiable
- Verifiable
- Prioritized
- Unambiguous (not more than one interpretation)
- Traceable
- Credible source

Software Requirements: Largely software requirements must be categorized into two categories:

1. **Functional Requirements:** Functional requirements define a function that a system or system element must be qualified to perform and must be documented in different forms. The functional requirements are describing the behavior of the system as it correlates to the system's functionality.
2. **Non-functional Requirements:** This can be the necessities that specify the criteria that can be used to decide the operation instead of specific behaviors of the system.
   Non-functional requirements are divided into two main categories:
   - **Execution qualities** like security and usability, which are observable at run time.
   - **Evolution qualities** like testability, maintainability, extensibility, and scalability that embodied in the static structure of the software system.

System models :

Objectives

- To explain why the context of a system should be modelled as part of the RE process
- To describe behavioural modelling, data modelling and object modelling
- To introduce some of the notations used in the Unified Modeling Language (UML)
- To show how CASE workbenches support system modelling

System modelling

- System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers.
- Different models present the system from different perspectives
  - External perspective showing the system's context or environment;
  - Behavioural perspective showing the behaviour of the system;
  - Structural perspective showing the system or data architecture.

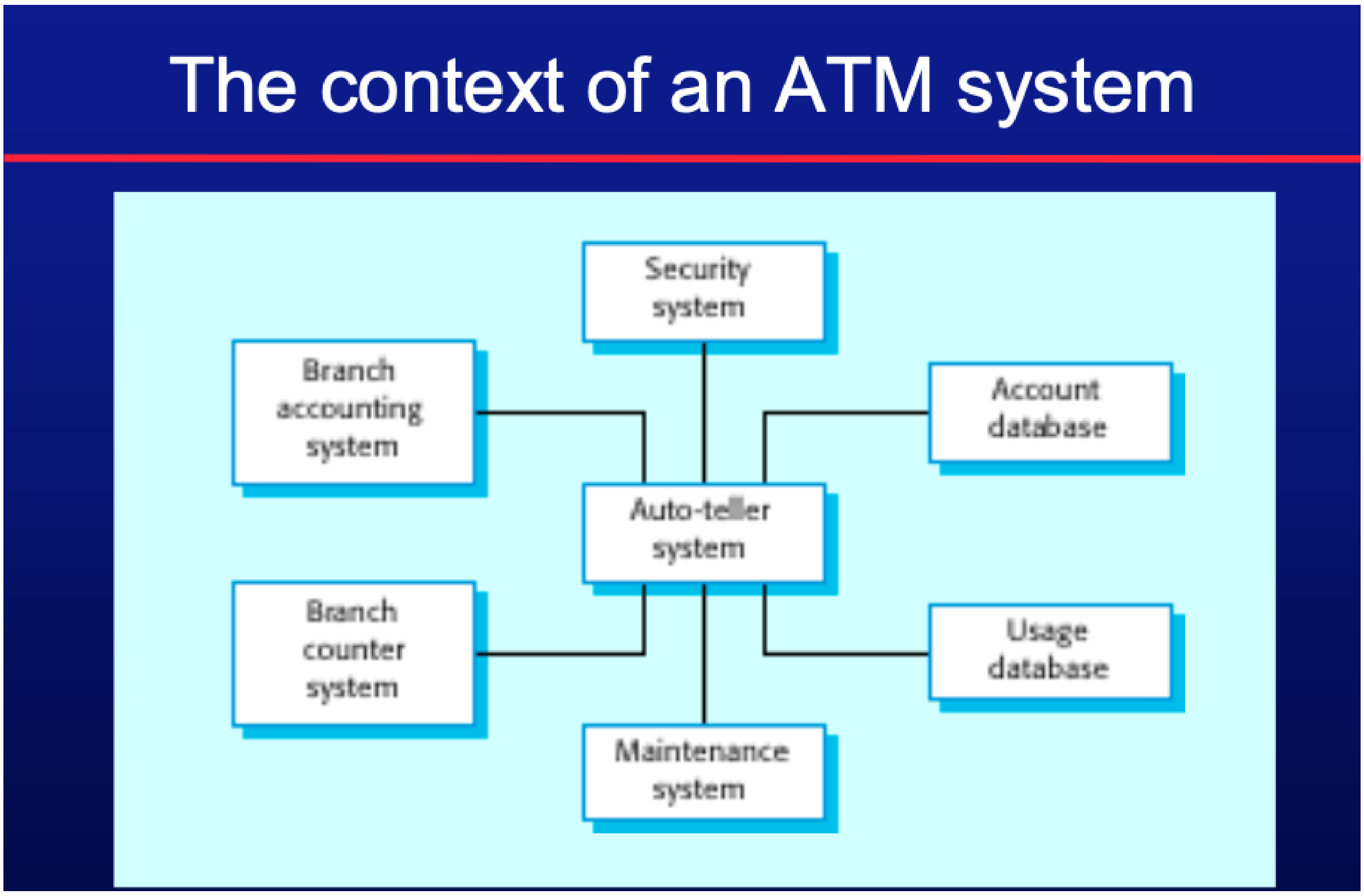
Model types

● Data processing model showing how the data is processed at different stages.
● Composition model showing how entities are composed of other entities.
● Architectural model showing principal sub-systems.
● Classification model showing how entities have common characteristics.
● Stimulus/response model showing the system's reaction to events.

1.Context Models

● Context models are used to illustrate the operational context of a system - they show what lies outside the system boundaries.
● Social and organisational concerns may affect the decision on where to position system boundaries.
● Architectural models show the system and its relationship with other systems.

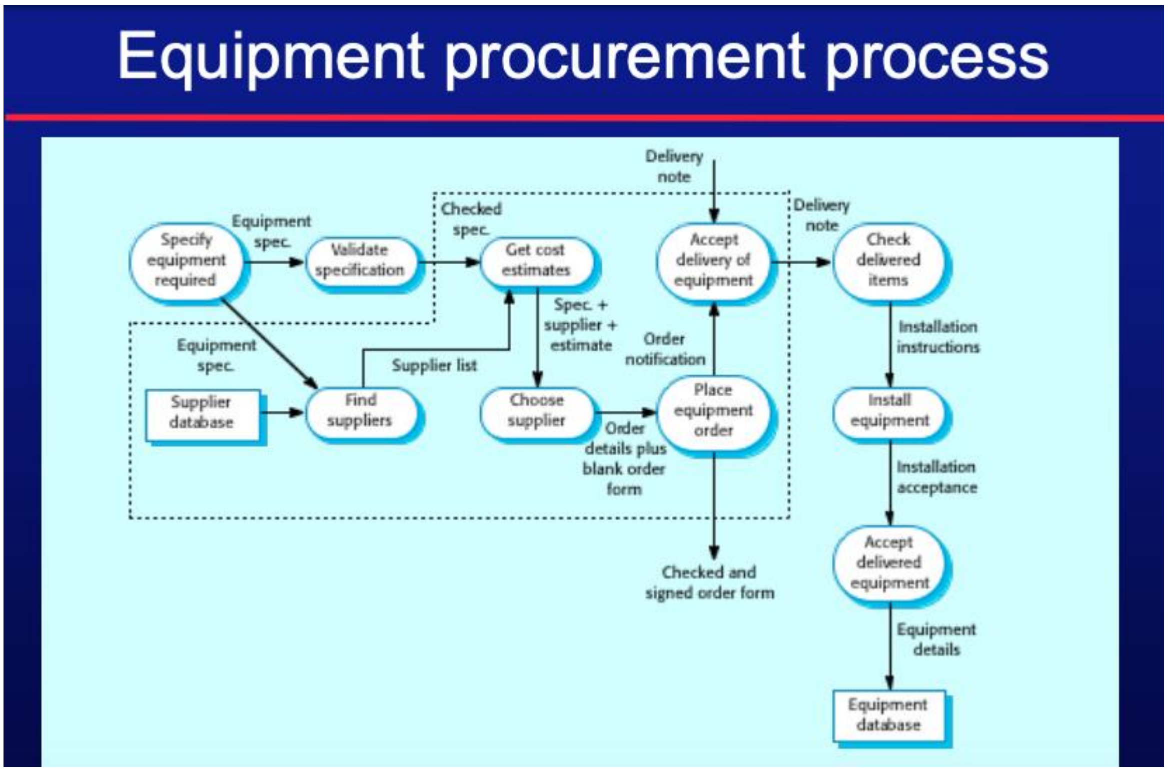
The context of an ATM system



Process models

● Process models show the overall process and the processes that are supported by the system.

● Data flow models may be used to show the processes and the flow of information from one process to another.



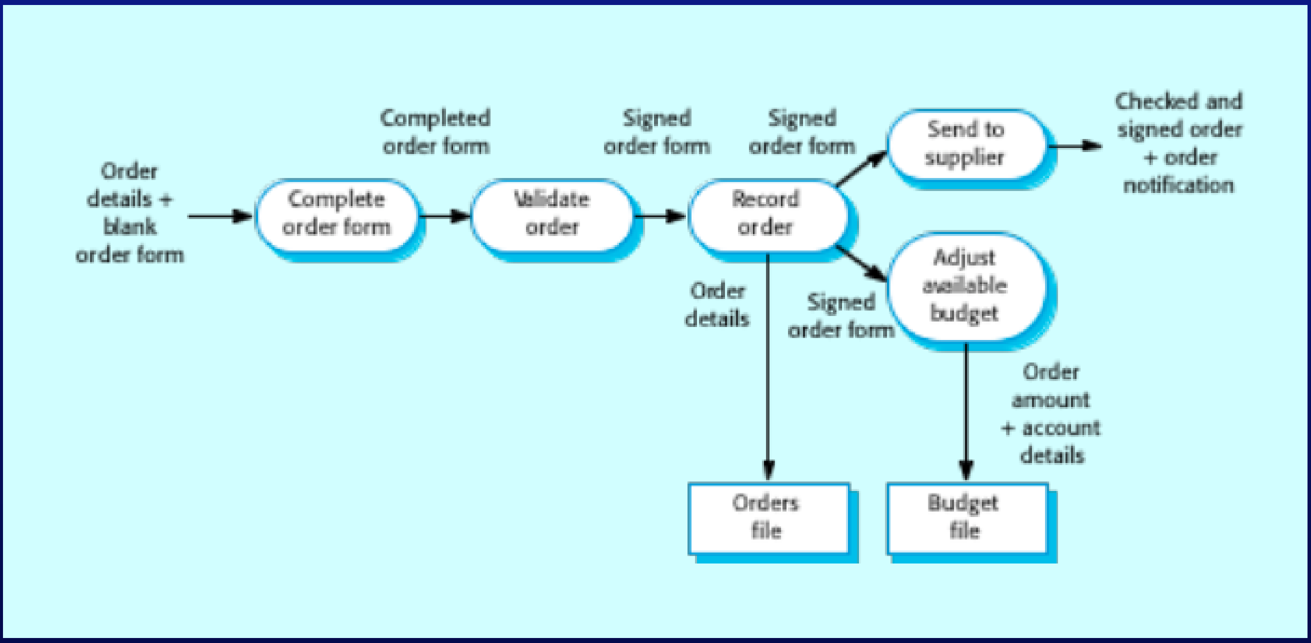
2.Behavioral models

● Behavioural models are used to describe the overall behaviour of a system.
● Two types of behavioural model are:
  • Data processing models that show how data is processed as it moves through the system;
  • State machine models that show the systems response to events.
● These models show different perspectives so both of them are required to describe the system's behaviour.

3.Data models

● Data flow diagrams (DFDs) may be used to model the system's data processing.
● These show the processing steps as data flows through a system.
● DFDs are an intrinsic part of many analysis methods.
● Simple and intuitive notation that customers can understand.
● Show end-to-end processing of data.

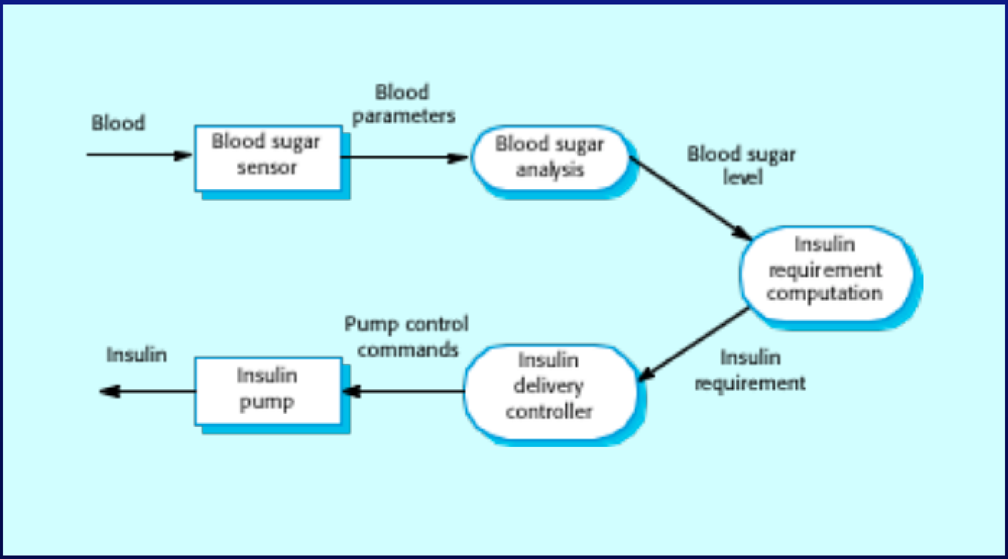
Order processing DFD



Data Flow Diagrams

● DFDs model the system from a functional perspective.
● Tracking and documenting how the data associated with a process is helpful to develop an overall understanding of the system.
● Data flow diagrams may also be used in showing the data exchange between a system and other systems in its environment.

Insulin pump DFD

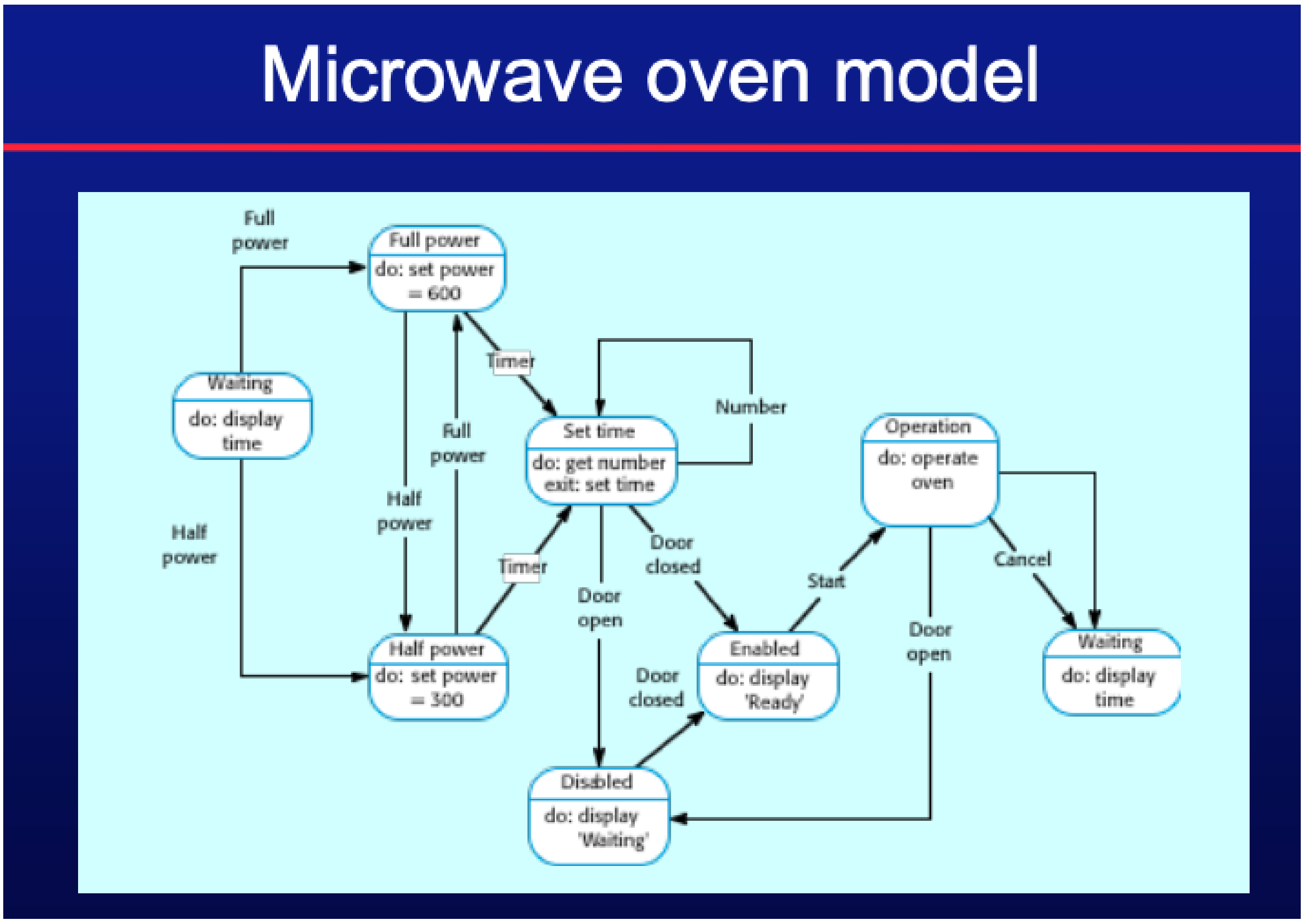


State machine models-Status Of Sys.

● These model the behaviour of the system in response to external and internal events.
● They show the system's responses to stimuli so are often used for modelling real-time systems.
● State machine models show system states as nodes and events as arcs between these nodes. When an event occurs, the system moves from one state to another.
● State charts are an integral part of the UML and are used to represent state machine models.

Statecharts

● Allow the decomposition of a model into submodels (see following slide).
● A brief description of the actions is included following the 'do' in each state.
● Can be complemented by tables describing the states and the stimuli.


Microwave oven model

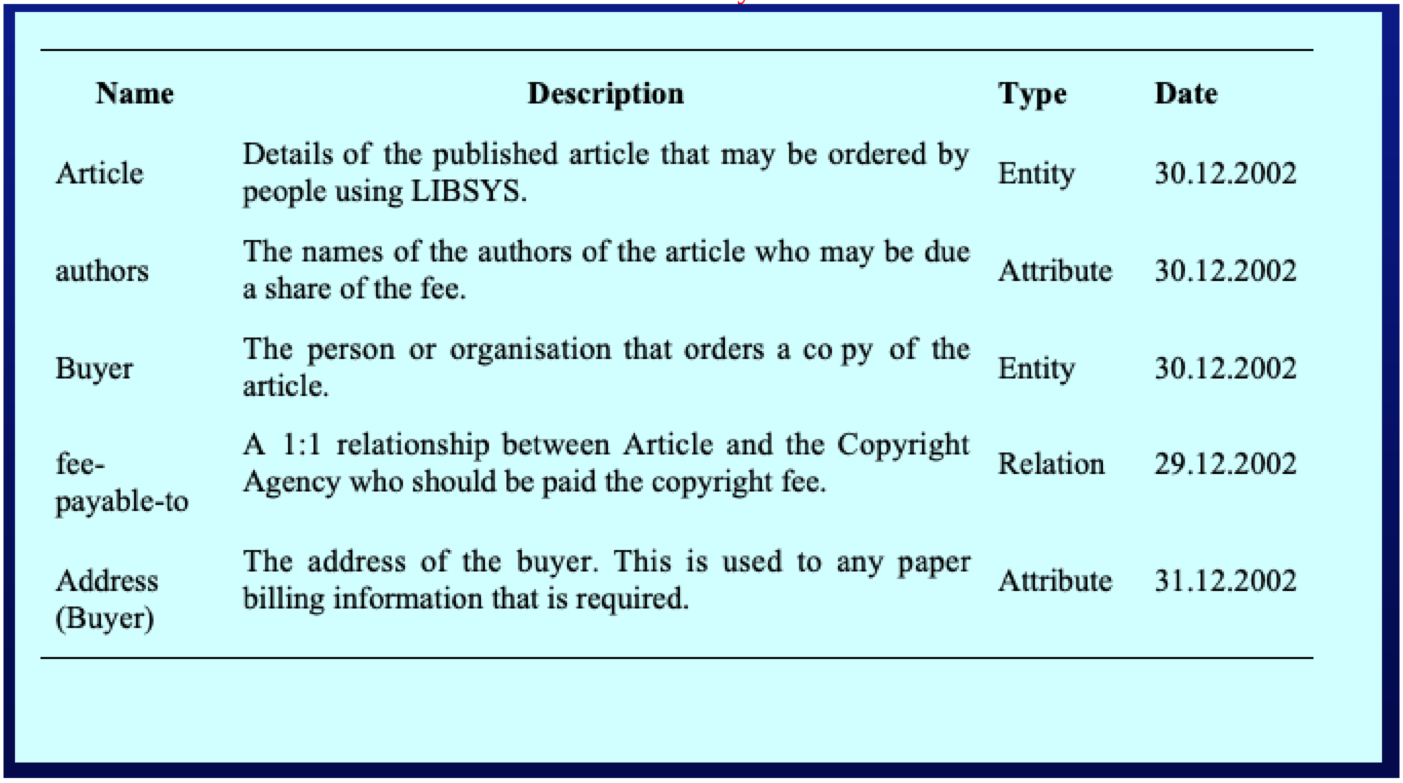
Semantic data models

● Used to describe the logical structure of data processed by the system.
● An entity-relation-attribute model sets out the entities in the system, the relationships between these entities and the entity attributes
● Widely used in database design. Can readily be implemented using relational databases.
● No specific notation provided in the UML but objects and associations can be used.

Data dictionaries

● Data dictionaries are lists of all of the names used in the system models. Descriptions of the entities, relationships and attributes are also included.
● Advantages
  • Support name management and avoid duplication;
  • Store of organisational knowledge linking analysis, design and implementation;
● Many CASE workbenches support data dictionaries.

Data dictionary entries



Name	Description	Type	Date
Article	Details of the published article that may be ordered by people using LIBSYS.	Entity	30.12.2002
authors	The names of the authors of the article who may be due a share of the fee.	Attribute	30.12.2002
Buyer	The person or organisation that orders a co py of the article.	Entity	30.12.2002
fee-payable-to	A 1:1 relationship between Article and the Copyright Agency who should be paid the copyright fee.	Relation	29.12.2002
Address (Buyer)	The address of the buyer. This is used to any paper billing information that is required.	Attribute	31.12.2002

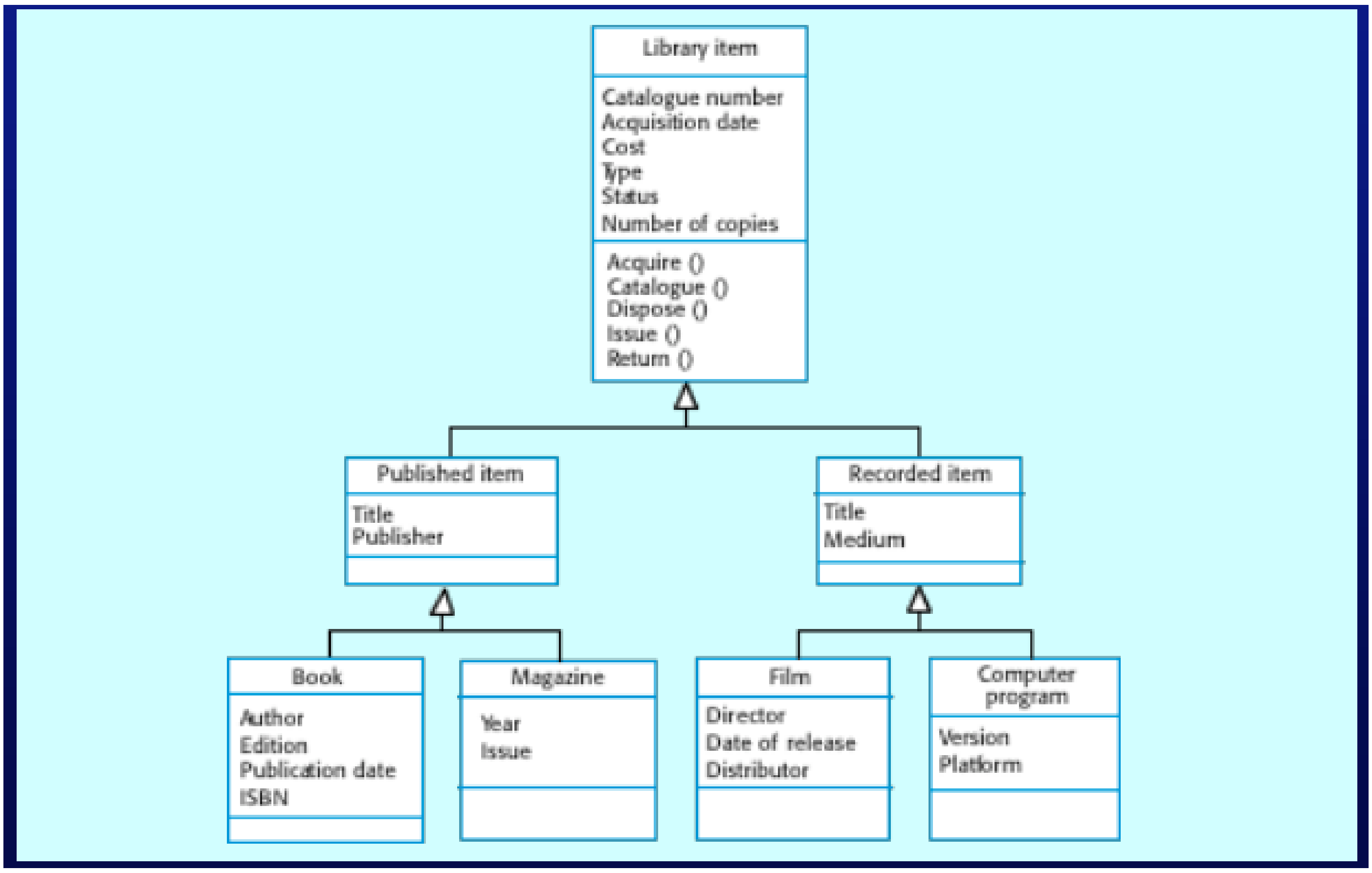
4.Object models

● Object models describe the system in terms of object classes and their associations.
● An object class is an abstraction over a set of objects with common attributes and the services (operations) provided by each object.
● Various object models may be produced
  • Inheritance models;
  • Aggregation models;
  • Interaction models.
● Natural ways of reflecting the real-world entities manipulated by the system
● More abstract entities are more difficult to model using this approach
● Object class identification is recognised as a difficult process requiring a deep understanding of the application domain
● Object classes reflecting domain entities are reusable across systems

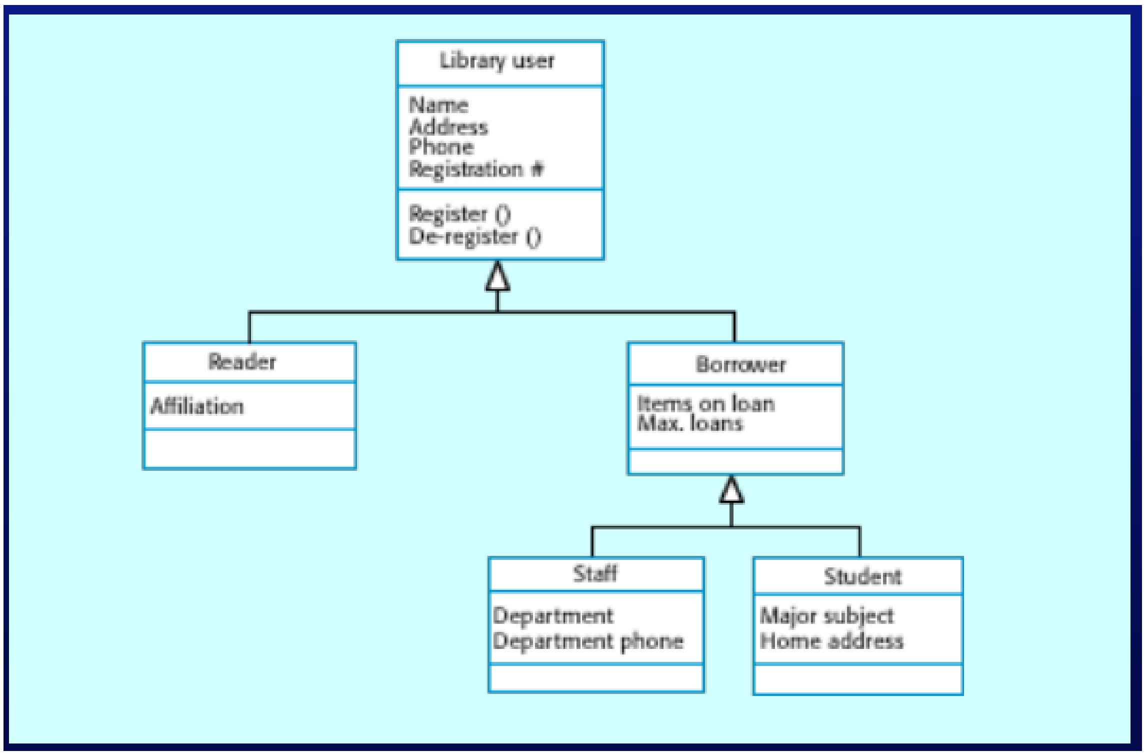
Inheritance models

● Organise the domain object classes into a hierarchy.
● Classes at the top of the hierarchy reflect the common features of all classes.
● Object classes inherit their attributes and services from one or more super-classes. these may then be specialised as necessary.
● Class hierarchy design can be a difficult process if duplication in different branches is to be avoided.

Library class hierarchy



User class hierarchy

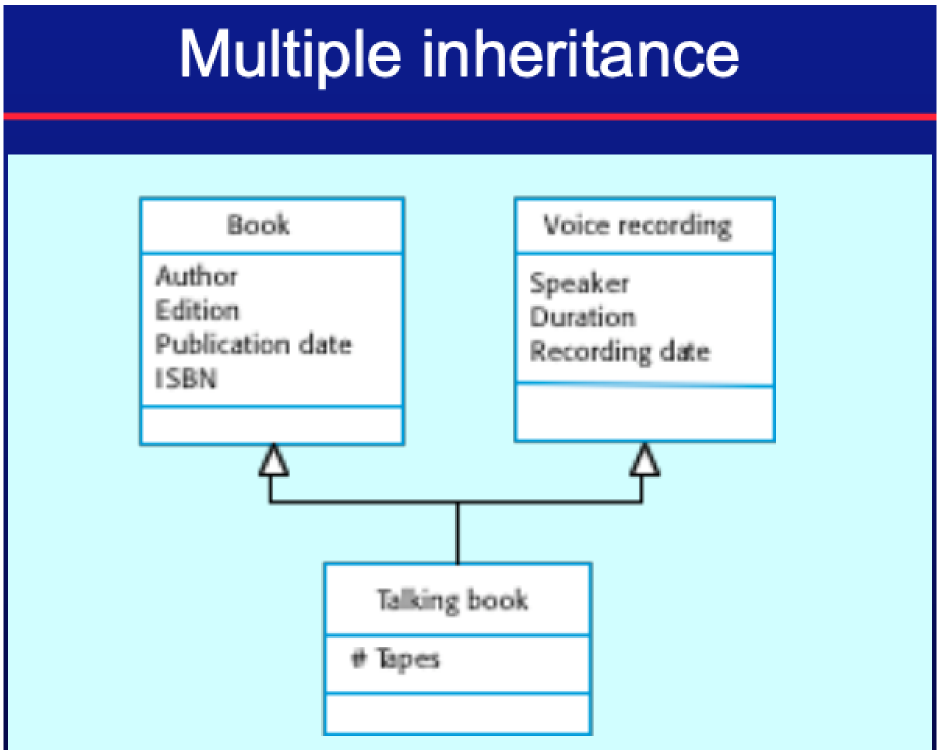


Multiple inheritance

● Rather than inheriting the attributes and services from a single parent class, a system which supports multiple inheritance allows object classes to inherit from several super-classes.

● This can lead to semantic conflicts where attributes/services with the same name in different super-classes have different semantics.

● Multiple inheritance makes class hierarchy reorganisation more complex.



5.structured methods

Structured methods

● Structured methods incorporate system modelling as an inherent part of the method.

● Methods define a set of models, a process for deriving these models and rules and guidelines that should apply to the models.

● CASE tools support system modelling as part of a structured method.

Structured methods

Structured methods were invented in the 1970s to support function-oriented design (Constantine and Yourdon, 1979) and evolved in the 1980s and 1990s to support object-oriented development (Coad and Yourdon, 1990, Robinson, 1992, Jacobson et al., 1993, Graham, 1994, Booch, 1994).

A structured method includes a design process model, notations to represent the design, report formats, rules and design guidelines. Structured methods may support some or all of the following models of a system:

- An object model that shows the object classes used in the system and their dependencies.
- A sequence model that shows how objects in the system interact when the system is executing.

- A state transition model that shows system states and the triggers for the transitions from one state to another.
- A structural model where the system components and their aggregations are documented.
- A data flow model where the system is modelled using the data transformations that take place as it is processed. This is not normally used in object-oriented methods but is still frequently used in real-time and business system design.
- A use-case model that shows the interactions between users and the system.

What is Structured Analysis?

Structured Analysis is a development method that allows the analyst to understand the system and its activities in a logical way.

It is a systematic approach, which uses graphical tools that analyze and refine the objectives of an existing system and develop a new system specification which can be easily understandable by user.

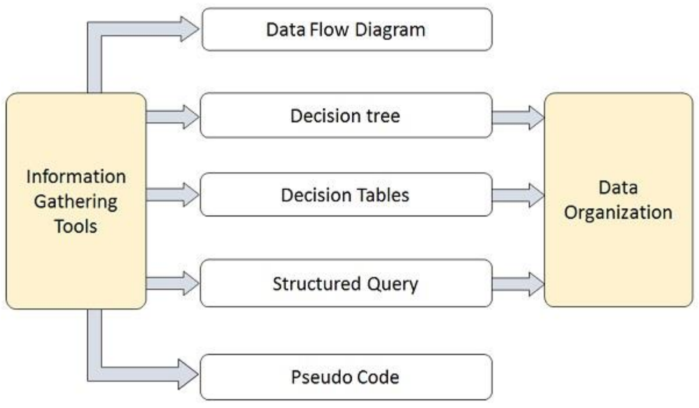
It has following attributes −

- It is graphic which specifies the presentation of application.
- It divides the processes so that it gives a clear picture of system flow.
- It is logical rather than physical i.e., the elements of system do not depend on vendor or hardware.
- It is an approach that works from high-level overviews to lower-level details.

Structured Analysis Tools

During Structured Analysis, various tools and techniques are used for system development. They are −

- Data Flow Diagrams
- Data Dictionary
- Decision Trees
- Decision Tables
- Structured English
- Pseudocode



6.Modeling with UML

UML - Modeling Types

It is very important to distinguish between the UML model. Different diagrams are used for different types of UML modeling. There are three important types of UML modeling.

Structural Modeling

Structural modeling captures the static features of a system. They consist of the following −

- Classes diagrams
- Objects diagrams
- Deployment diagrams
- Package diagrams
- Composite structure diagram
- Component diagram

Structural model represents the framework for the system and this framework is the place where all other components exist. Hence, the class diagram, component diagram and deployment diagrams are part of structural modeling. They all represent the elements and the mechanism to assemble them.

The structural model never describes the dynamic behavior of the system. Class diagram is the most widely used structural diagram.

Behavioral Modeling

Behavioral model describes the interaction in the system. It represents the interaction among the structural diagrams. Behavioral modeling shows the dynamic nature of the system. They consist of the following −

- Activity diagrams
- Interaction diagrams
- Use case diagrams

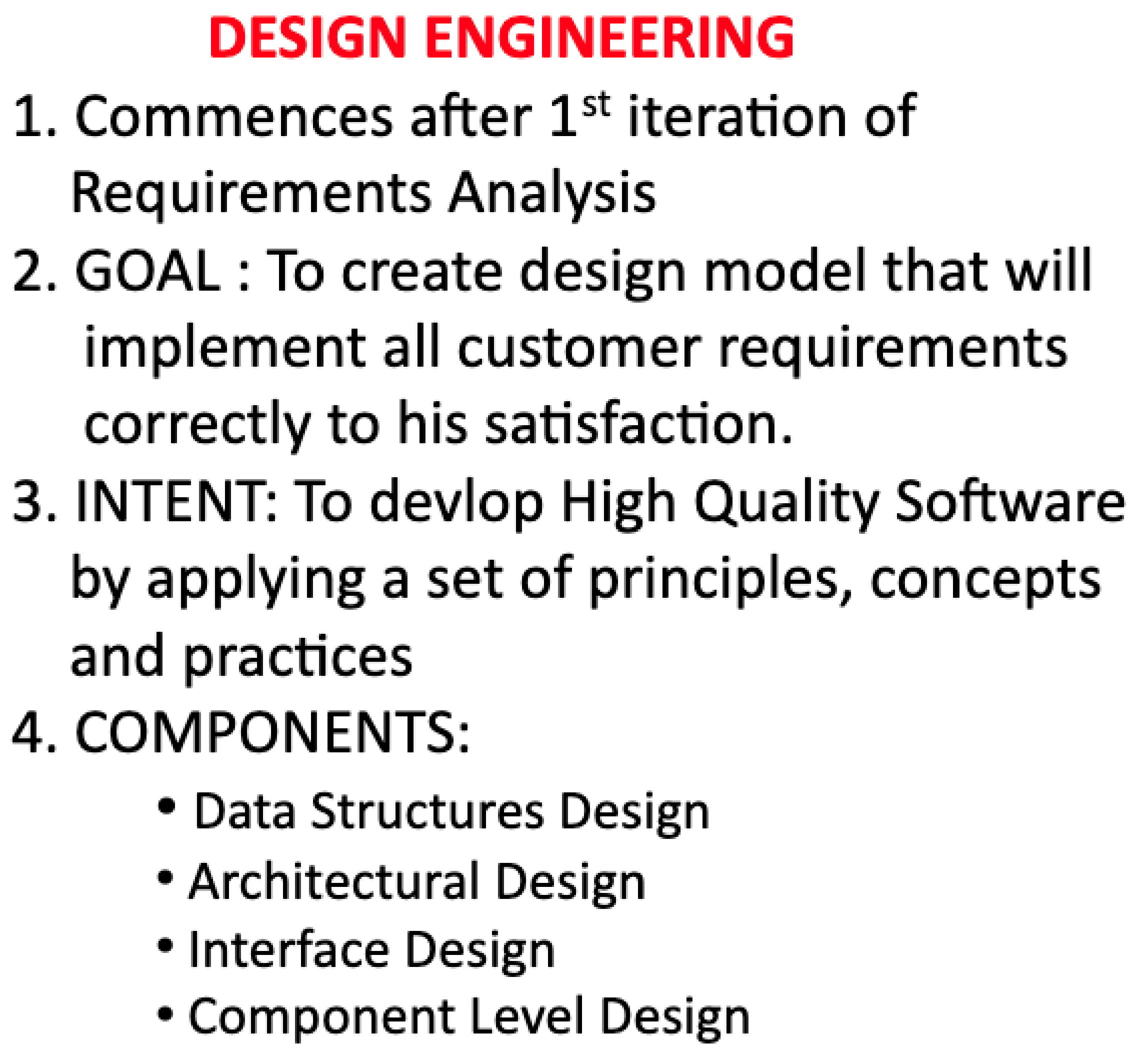
All the above show the dynamic sequence of flow in a system.

Architectural Modeling

Architectural model represents the overall framework of the system. It contains both structural and behavioral elements of the system. Architectural model can be defined as the blueprint of the entire system. Package diagram comes under architectural modeling.

By Shubh Thakare +919595939264

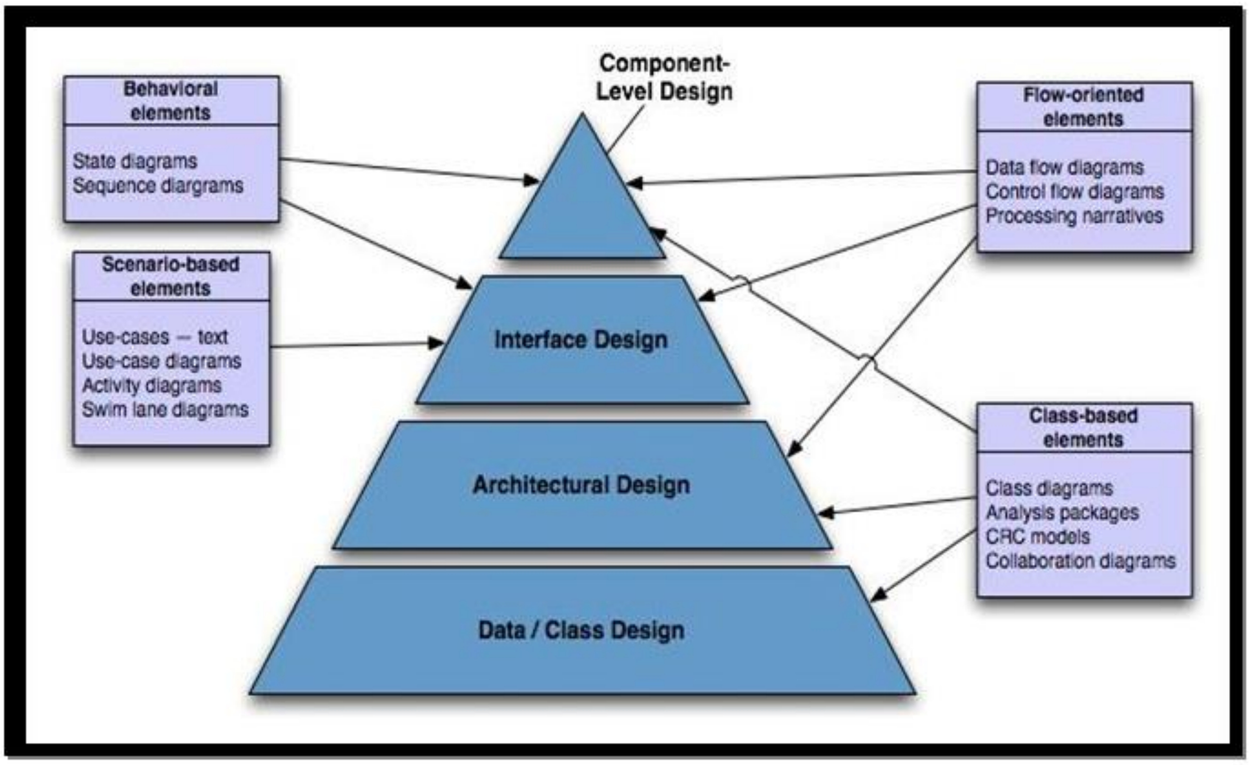
Design Engineering :



DESIGN ENGINEERING

1. Commences after 1st iteration of Requirements Analysis
2. GOAL : To create design model that will implement all customer requirements correctly to his satisfaction.
3. INTENT: To devlop High Quality Software by applying a set of principles, concepts and practices
4. COMPONENTS:
   - Data Structures Design
   - Architectural Design
   - Interface Design
   - Component Level Design

Analysis → Design



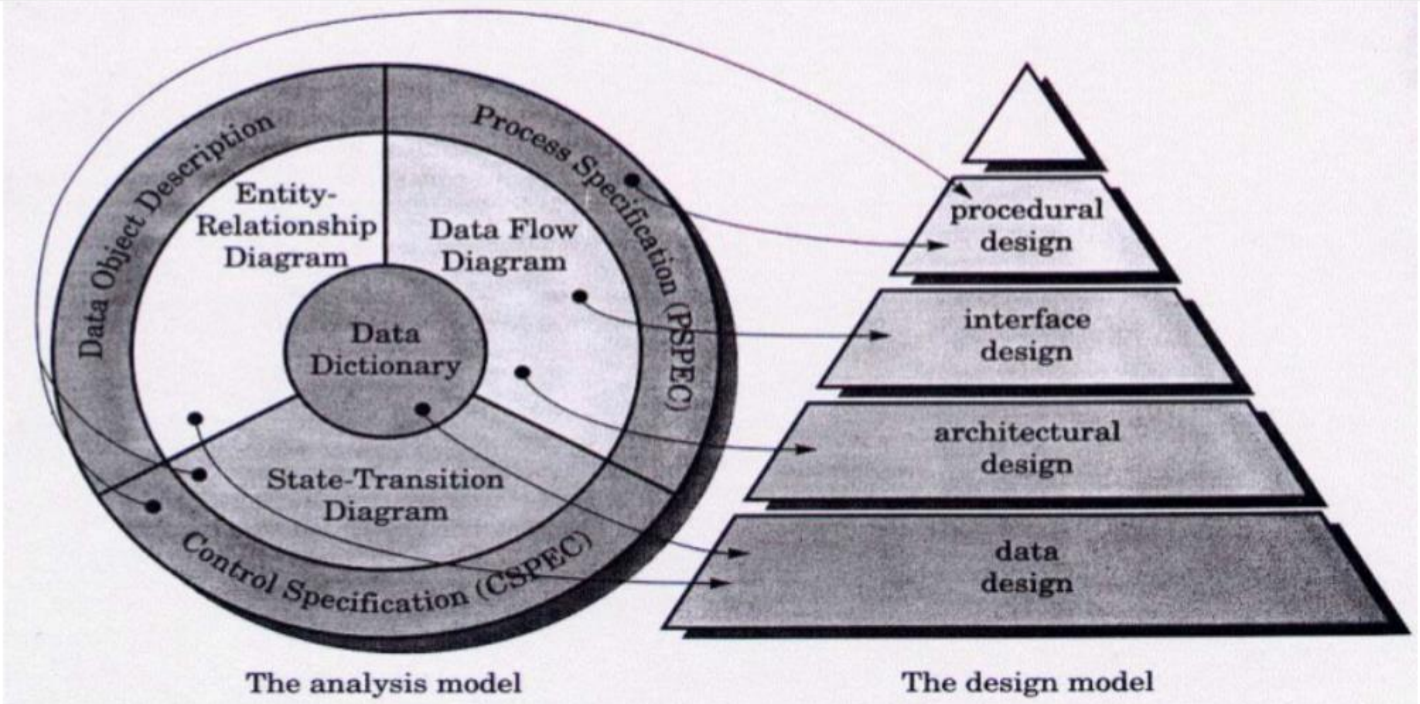


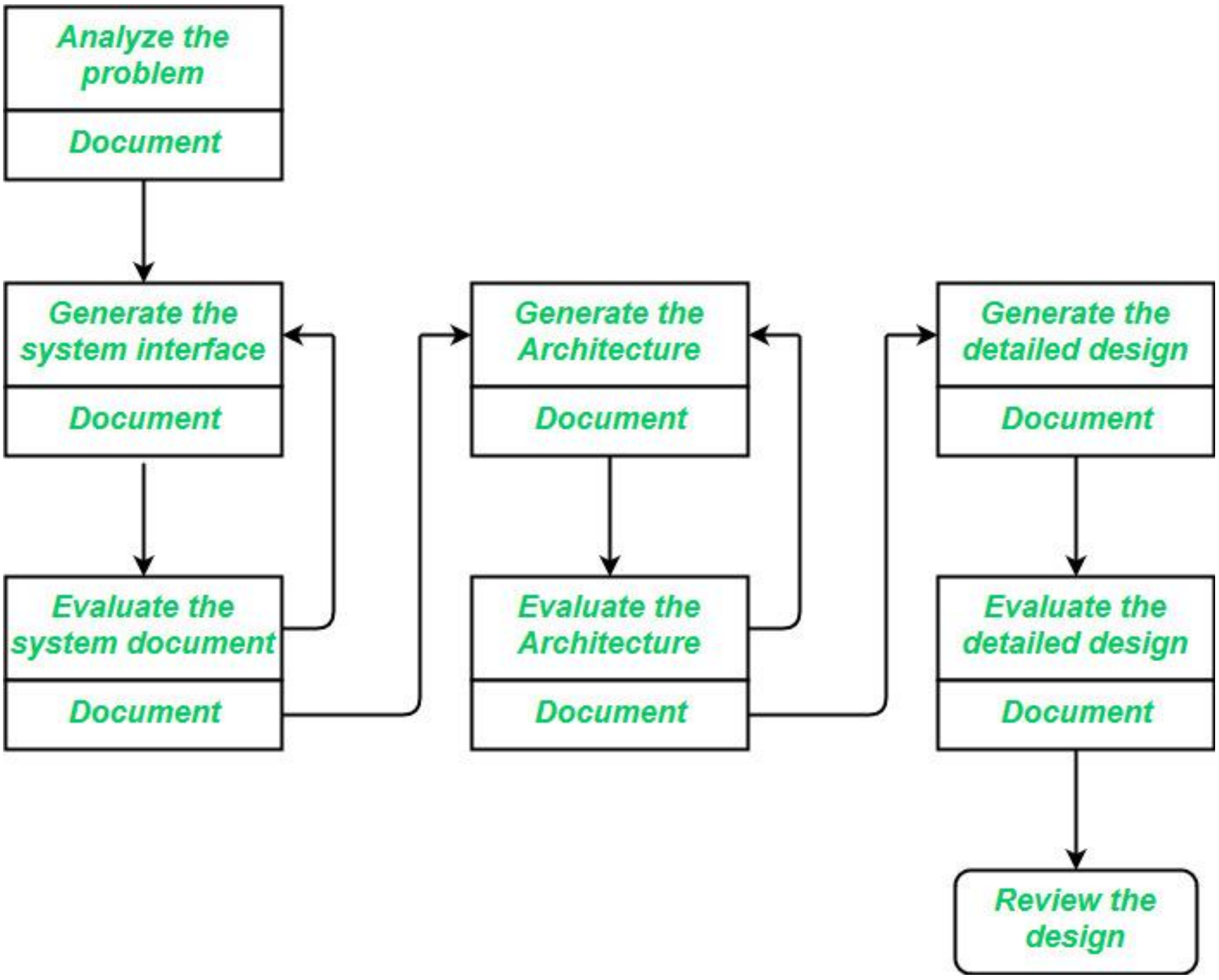
FIGURE 13.1. Translating the analysis model into a software design

1.Design process and Design quality

Software Design Process

The design phase of software development deals with transforming the customer requirements as described in the SRS documents into a form implementable using a programming language.
The software design process can be divided into the following three levels of phases of design:

1. Interface Design
2. Architectural Design
3. Detailed Design


Analyze the problem
Document
Generate the system interface
Document
Evaluate the system document
Document
Generate the Architecture
Document
Evaluate the Architecture
Document
Generate the detailed design
Document
Evaluate the detailed design
Document
Review the design

Interface Design:
Interface design is the specification of the interaction between a system and its environment. this phase proceeds at a high level of abstraction with respect to the inner workings of the system
i.e,1 during interface design, the internal of the systems are completely ignored and the system is treated as a black box. Attention is focussed on the dialogue between the target system and the users, devices, and other systems with which it interacts. The

design problem statement produced during the problem analysis step should identify the people, other systems, and devices which are collectively called *agents*.'
i.e2 is Your Operating System

Interface design should include the following details:

- Precise description of events in the environment, or messages from agents to which the system must respond.
- Precise description of the events or messages that the system must produce.
- Specification on the data, and the formats of the data coming into and going out of the system.
- Specification of the ordering and timing relationships between incoming events or messages, and outgoing events or outputs.

Architectural Design:
Architectural design is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them. In architectural design, the overall structure of the system is chosen, but the internal details of major components are ignored.

Issues in architectural design includes:

- Gross decomposition of the systems into major components.
- Allocation of functional responsibilities to components.
- Component Interfaces
- Component scaling and performance properties, resource consumption properties, reliability properties, and so forth.
- Communication and interaction between components.

The architectural design adds important details ignored during the interface design. Design of the internals of the major components is ignored until the last phase of the design.

Detailed Design:
Design is the specification of the internal elements of all major system components, their properties, relationships, processing, and often their algorithms and the data structures.
The detailed design may include:

- Decomposition of major system components into program units.
- Allocation of functional responsibilities to units.
- User interfaces
- Unit states and state changes
- Data and control interaction between units
- Data packaging and implementation, including issues of scope and visibility of program elements
- Algorithms and data structures

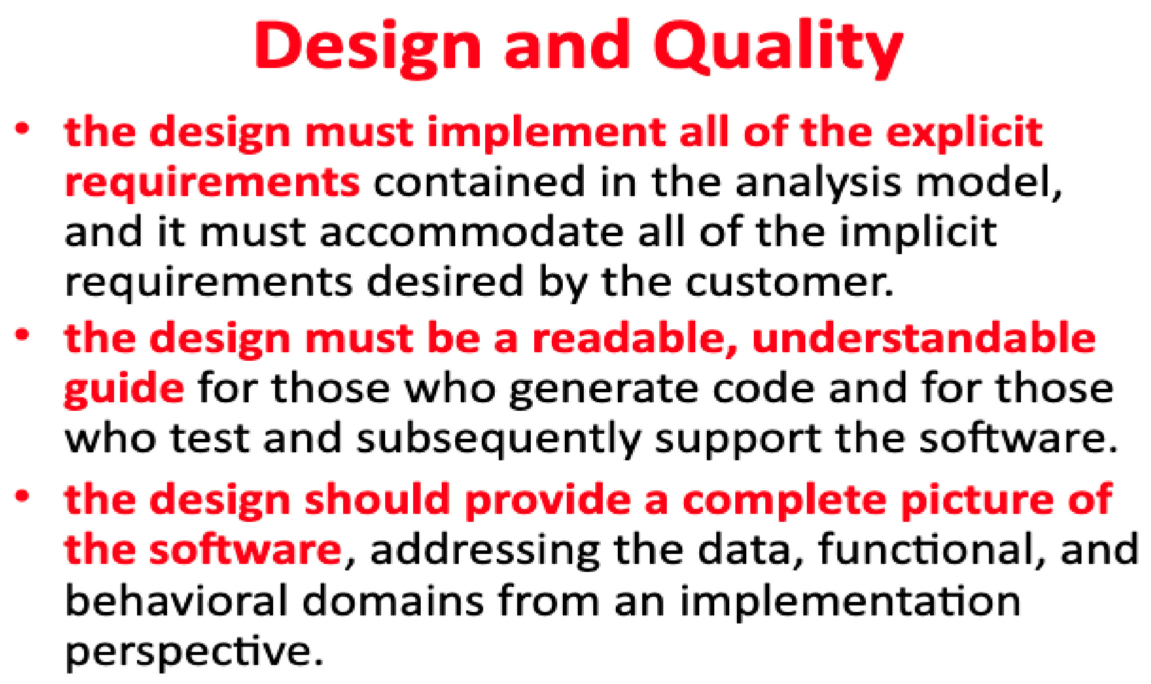
Design Process:

- Software design is an iterative process through which requirements are translated into a "blueprint" for constructing the software
- During the design process the software requirements model is transformed into design models that describe the details of the data structures, system architecture, interface, and components.
- Design is considered to be high level of abstraction.

Fundamental Design concepts

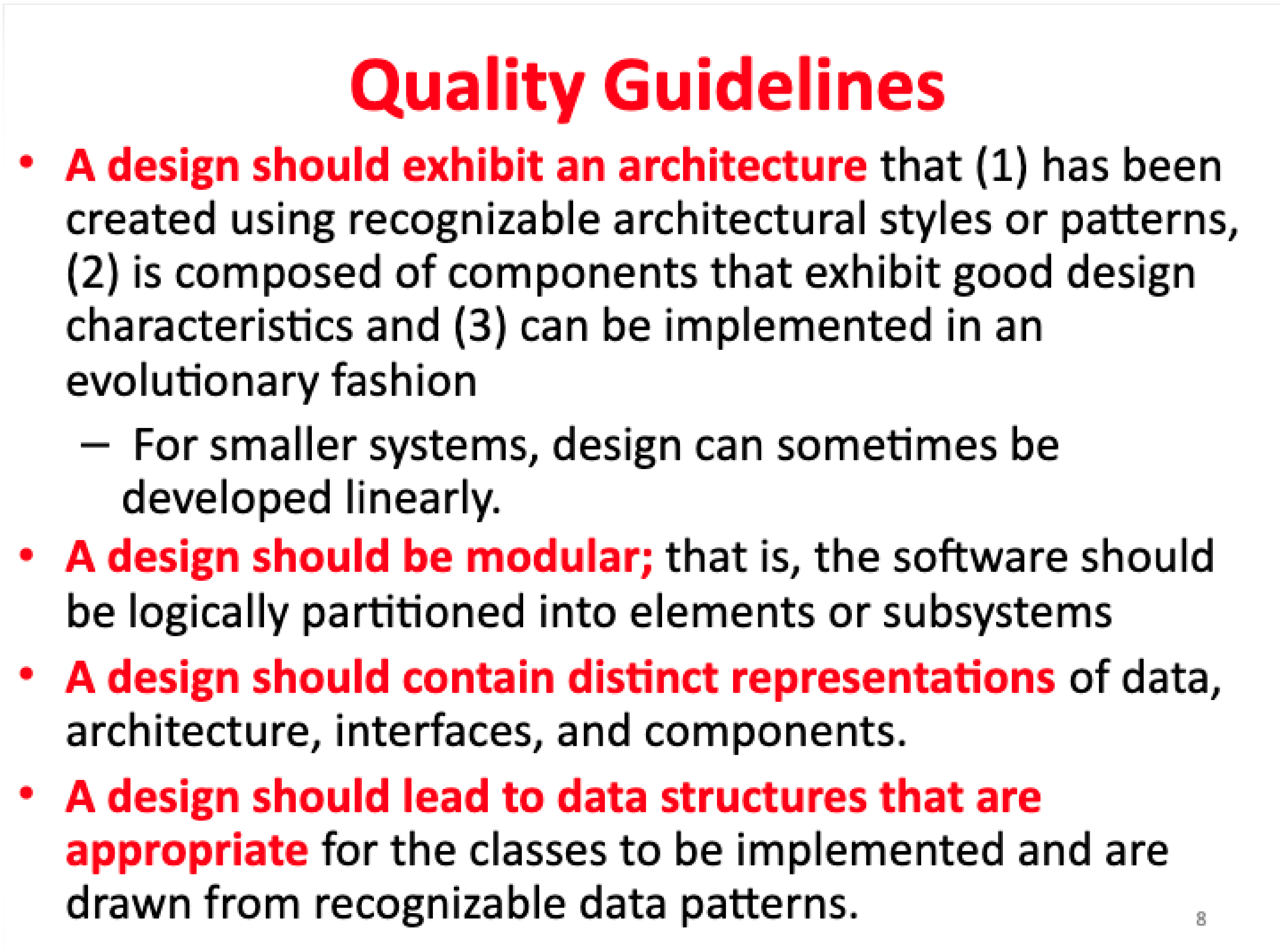
- Abstraction
- Architecture
- Patterns
- Modularity
- Information hiding
- Functional independence
- Refinement
- Refactoring
- Design classes

Software Design Quality



Design and Quality

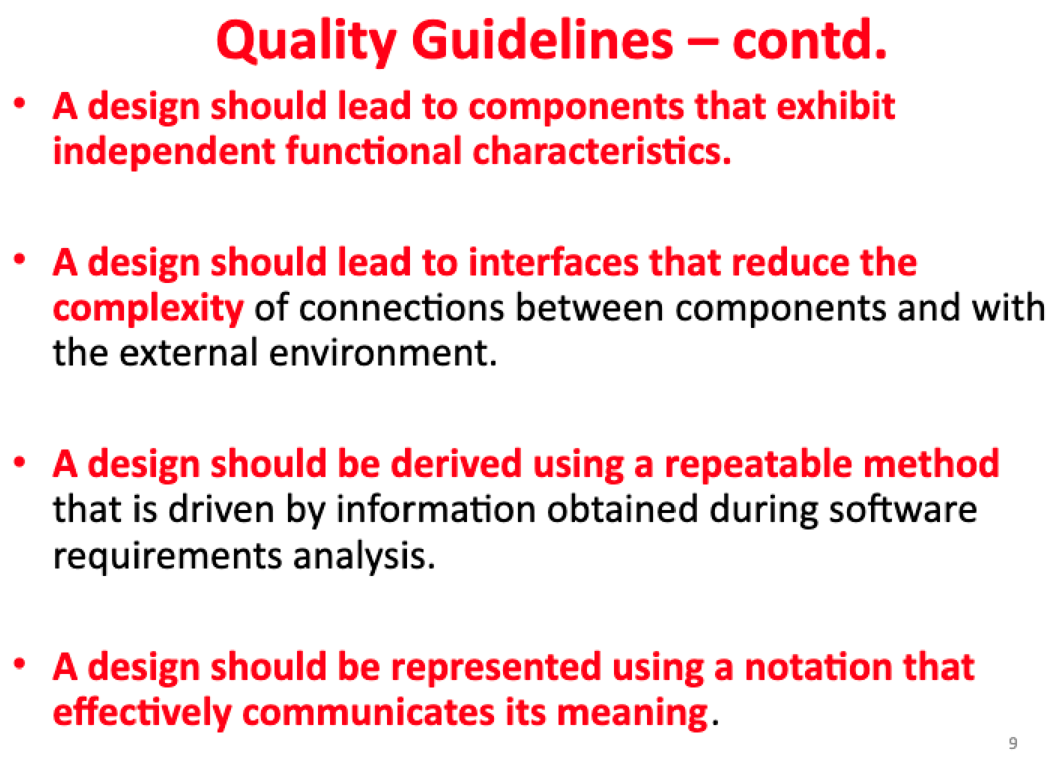
- **the design must implement all of the explicit requirements** contained in the analysis model, and it must accommodate all of the implicit requirements desired by the customer.
- **the design must be a readable, understandable guide** for those who generate code and for those who test and subsequently support the software.
- **the design should provide a complete picture of the software**, addressing the data, functional, and behavioral domains from an implementation perspective.



Quality Guidelines

- **A design should exhibit an architecture** that (1) has been created using recognizable architectural styles or patterns, (2) is composed of components that exhibit good design characteristics and (3) can be implemented in an evolutionary fashion
  - For smaller systems, design can sometimes be developed linearly.
- **A design should be modular;** that is, the software should be logically partitioned into elements or subsystems
- **A design should contain distinct representations** of data, architecture, interfaces, and components.
- **A design should lead to data structures that are appropriate** for the classes to be implemented and are drawn from recognizable data patterns.

8



Quality Guidelines – contd.

- **A design should lead to components that exhibit independent functional characteristics.**

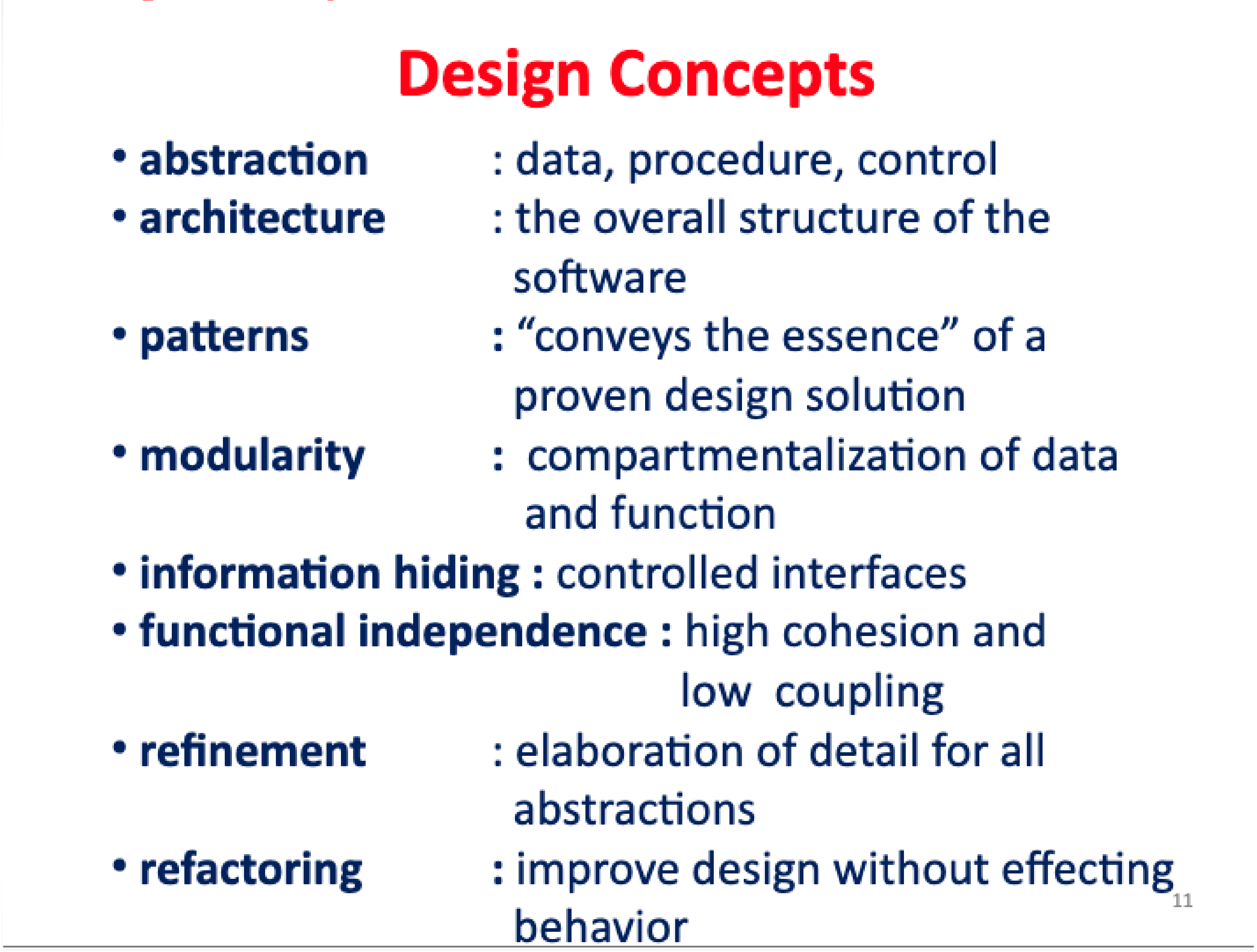
- **A design should lead to interfaces that reduce the complexity** of connections between components and with the external environment.

- **A design should be derived using a repeatable method** that is driven by information obtained during software requirements analysis.

- **A design should be represented using a notation that effectively communicates its meaning**.

9

2.Design concepts



Design Concepts

- **abstraction** : data, procedure, control
- **architecture** : the overall structure of the software
- **patterns** : "conveys the essence" of a proven design solution
- **modularity** : compartmentalization of data and function
- **information hiding :** controlled interfaces
- **functional independence :** high cohesion and low coupling
- **refinement** : elaboration of detail for all abstractions
- **refactoring** : improve design without effecting behavior

11

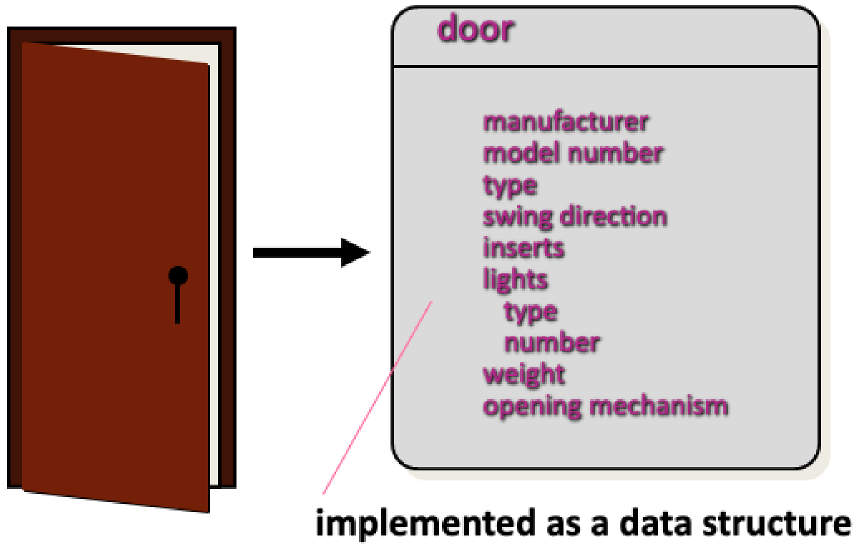
Design Concepts : Abstraction

- When we consider a modular solution to any problem, many levels of abstraction can be posted. At the highest level of abstraction, a solution is stated in broad terms using the language of problem environment. At lower levels of abstraction, a more detailed description of the solution is provided.
- There are two types of abstractions, procedural and data
- For example, procedural : compute result
  sequence of instructions
  Data : subject wise marks obtained

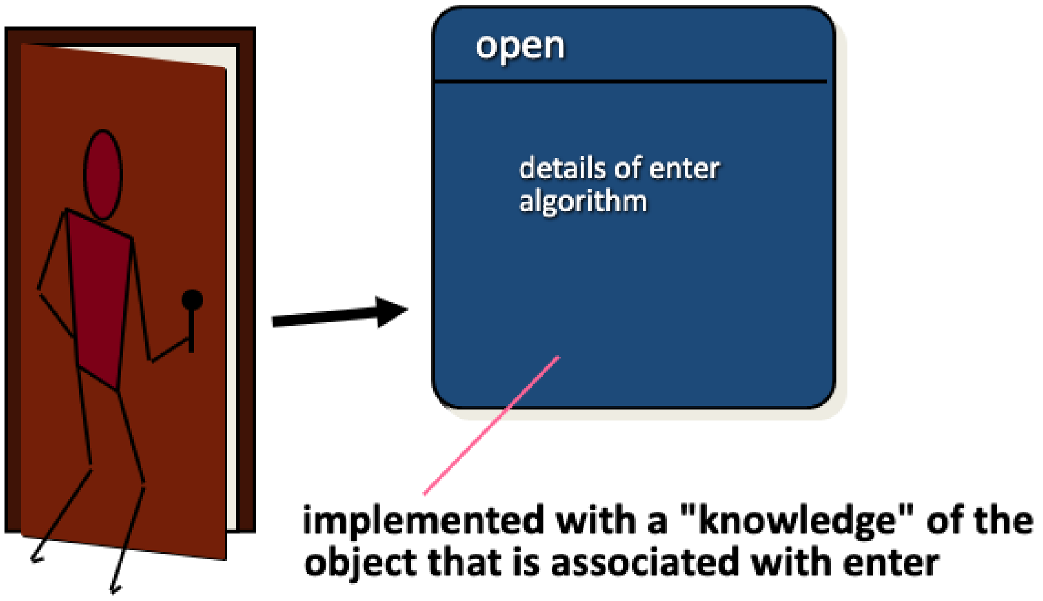
Design Concepts : Abstraction

- **Abstraction**
  - process – extracting essential details
  - entity – a model or focused representation
- **Information hiding**
  - the suppression of inessential information
- **Encapsulation**
  - process – enclosing items in a container
  - entity – enclosure that holds the items

Data Abstraction

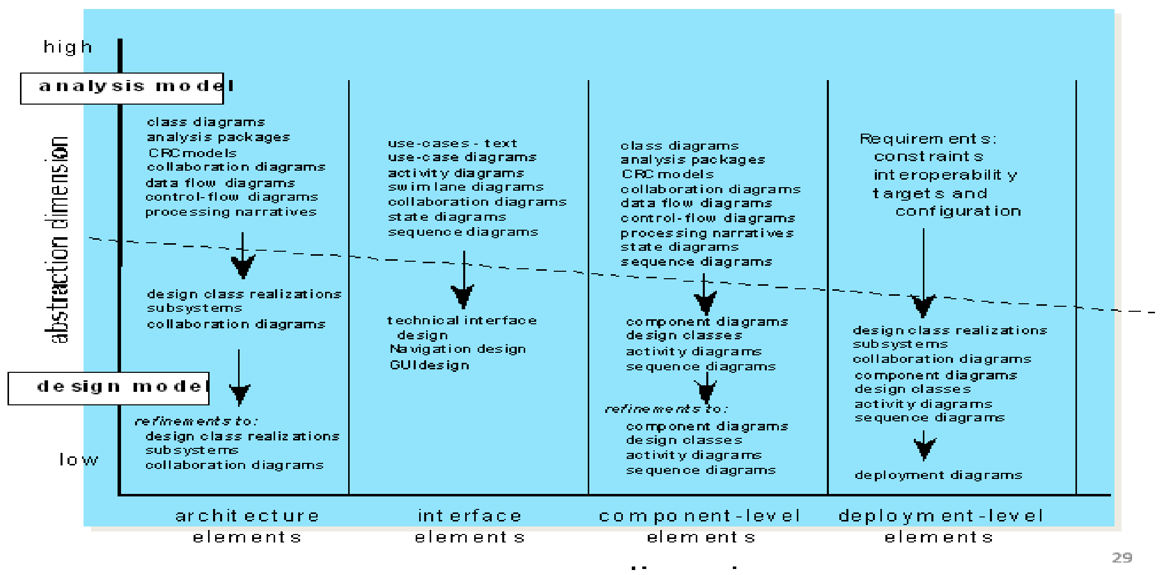


Procedural Abstraction


open
details of enter algorithm
implemented with a "knowledge" of the object that is associated with enter

3.The design model

The Design Model

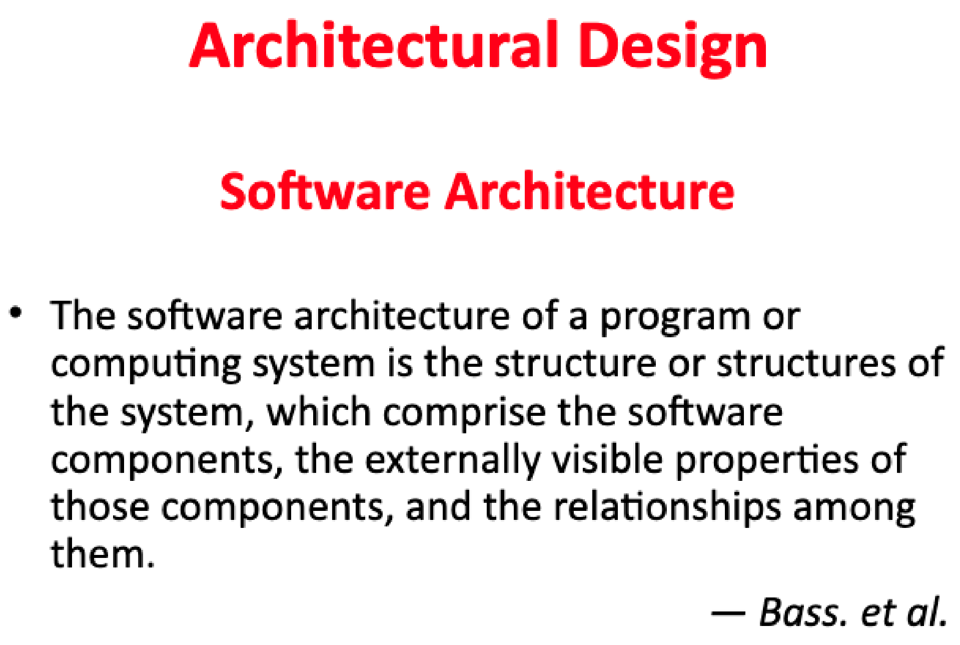


Design Model Elements

- **Data elements**
  - Architectural level → databases and files
  - Component level → data structures
- **Architectural elements**
  - An architectural model is derived from:
    - Application domain
    - Analysis model
    - Available styles and patterns
- **Interface elements**
  - There are three parts to the interface design element:
  - The user interface (UI)
  - Interfaces to external systems
  - Interfaces to components within the application
- **Component elements**
- **Deployment elements**

Creating an architectural design :

1.Software architecture

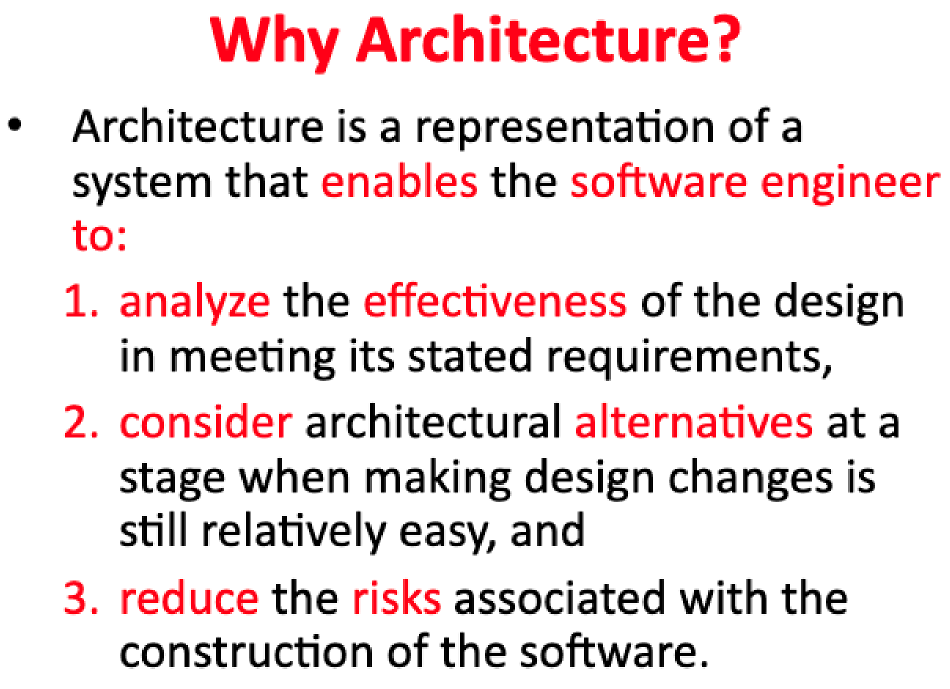


Architectural Design

Software Architecture

- The software architecture of a program or computing system is the structure or structures of the system, which comprise the software components, the externally visible properties of those components, and the relationships among them.

— *Bass. et al.*

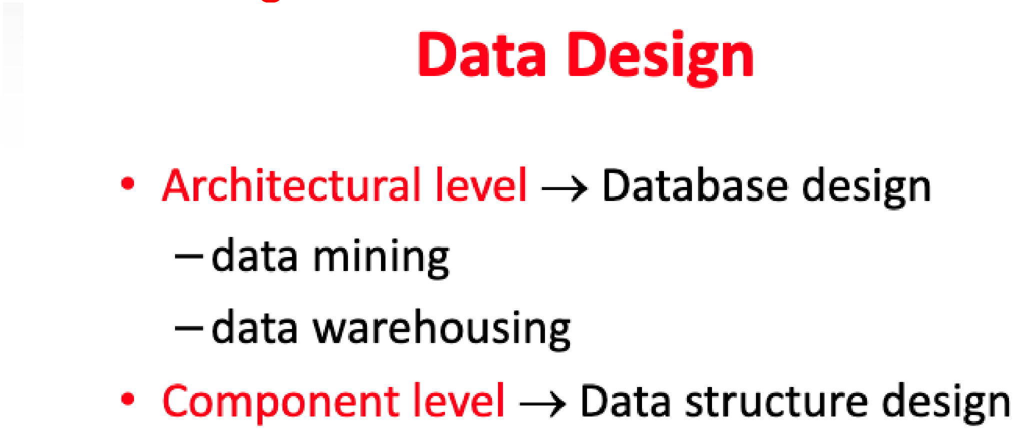


Why Architecture?

- Architecture is a representation of a system that enables the software engineer to:

1. analyze the effectiveness of the design in meeting its stated requirements,
2. consider architectural alternatives at a stage when making design changes is still relatively easy, and
3. reduce the risks associated with the construction of the software.

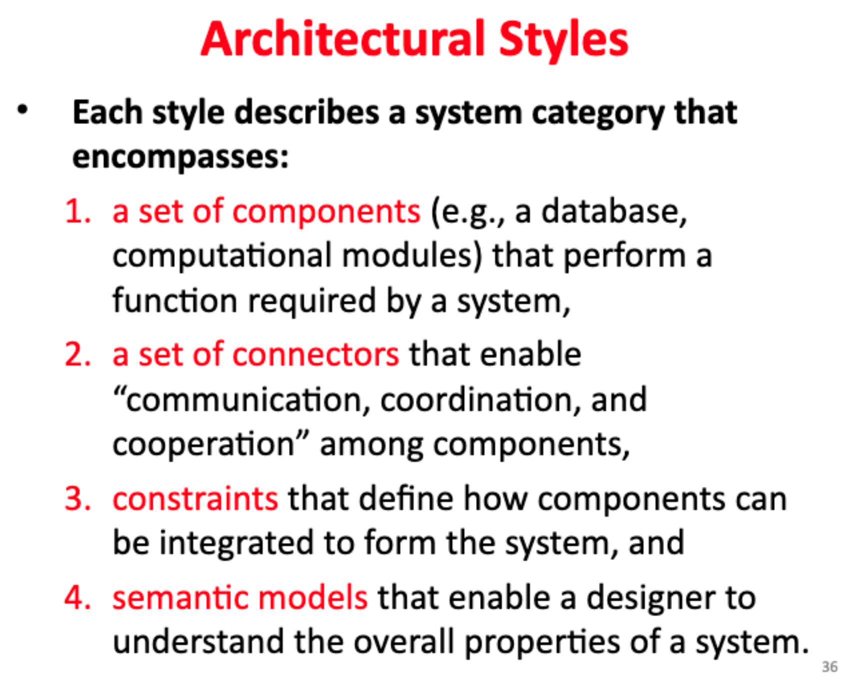
2.Data design



Data Design

- Architectural level → Database design
  - data mining
  - data warehousing
- Component level → Data structure design

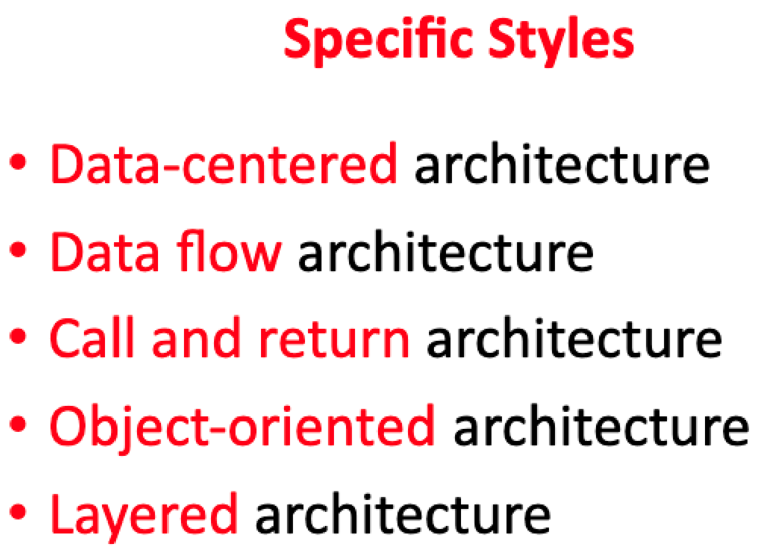
3.Architectural styles and patterns



Architectural Styles

- **Each style describes a system category that encompasses:**
  1. a set of components (e.g., a database, computational modules) that perform a function required by a system,
  2. a set of connectors that enable "communication, coordination, and cooperation" among components,
  3. constraints that define how components can be integrated to form the system, and
  4. semantic models that enable a designer to understand the overall properties of a system.

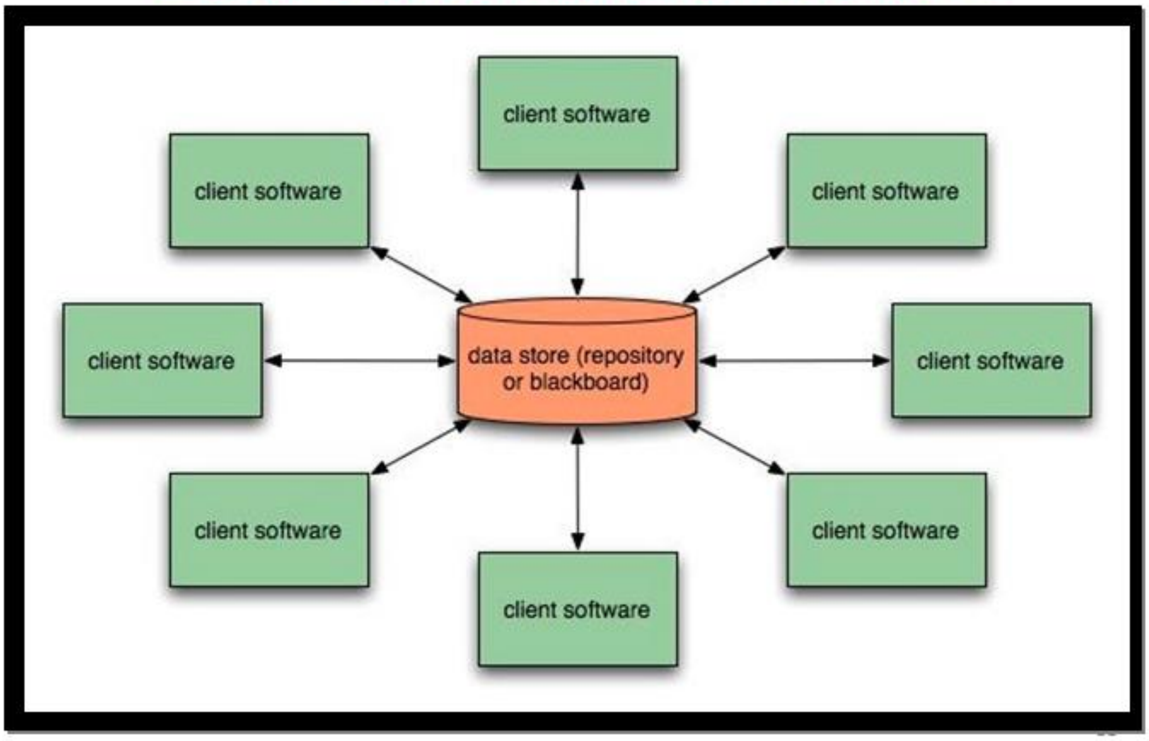
36



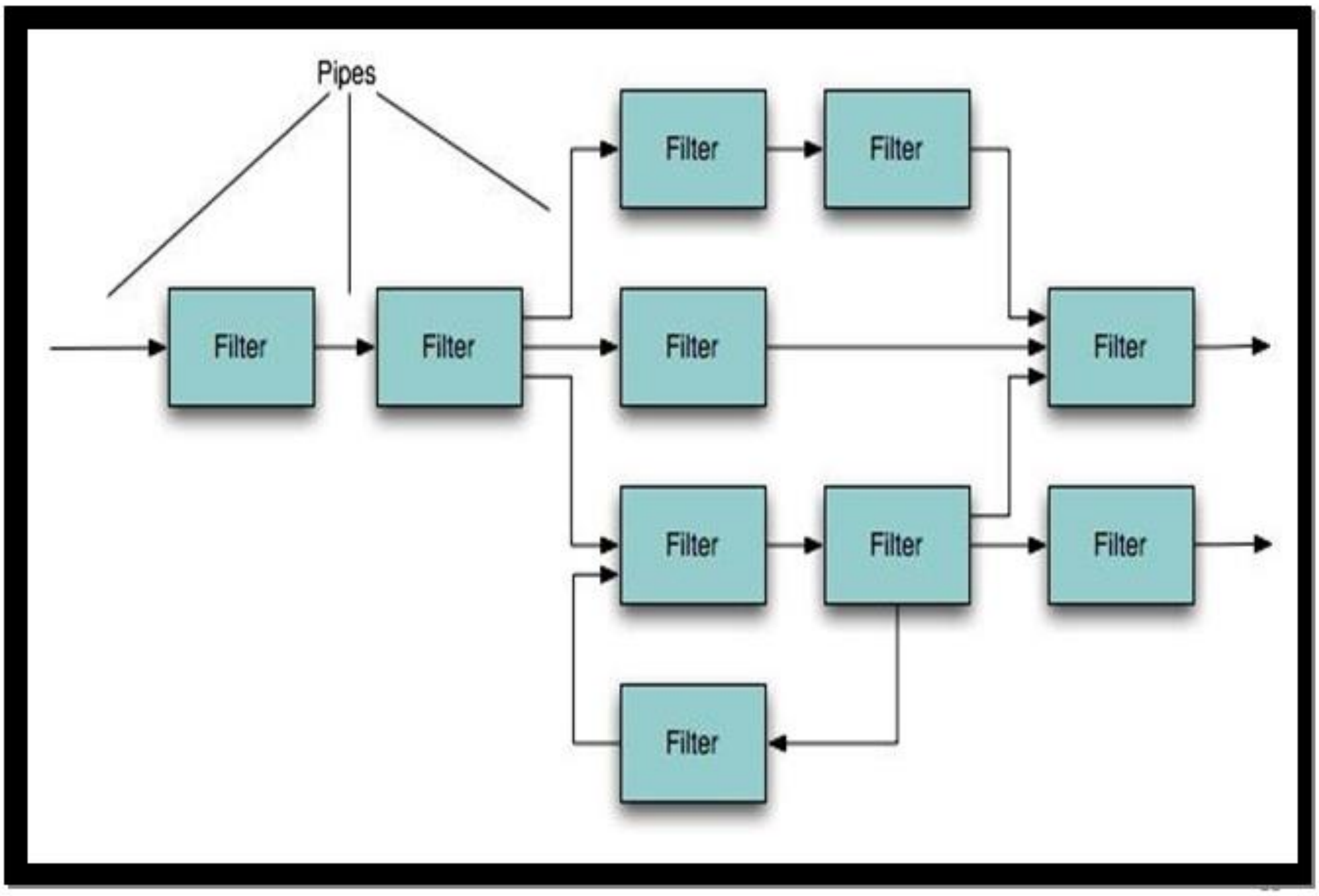
Specific Styles

- Data-centered architecture
- Data flow architecture
- Call and return architecture
- Object-oriented architecture
- Layered architecture

Data-Centered Architecture

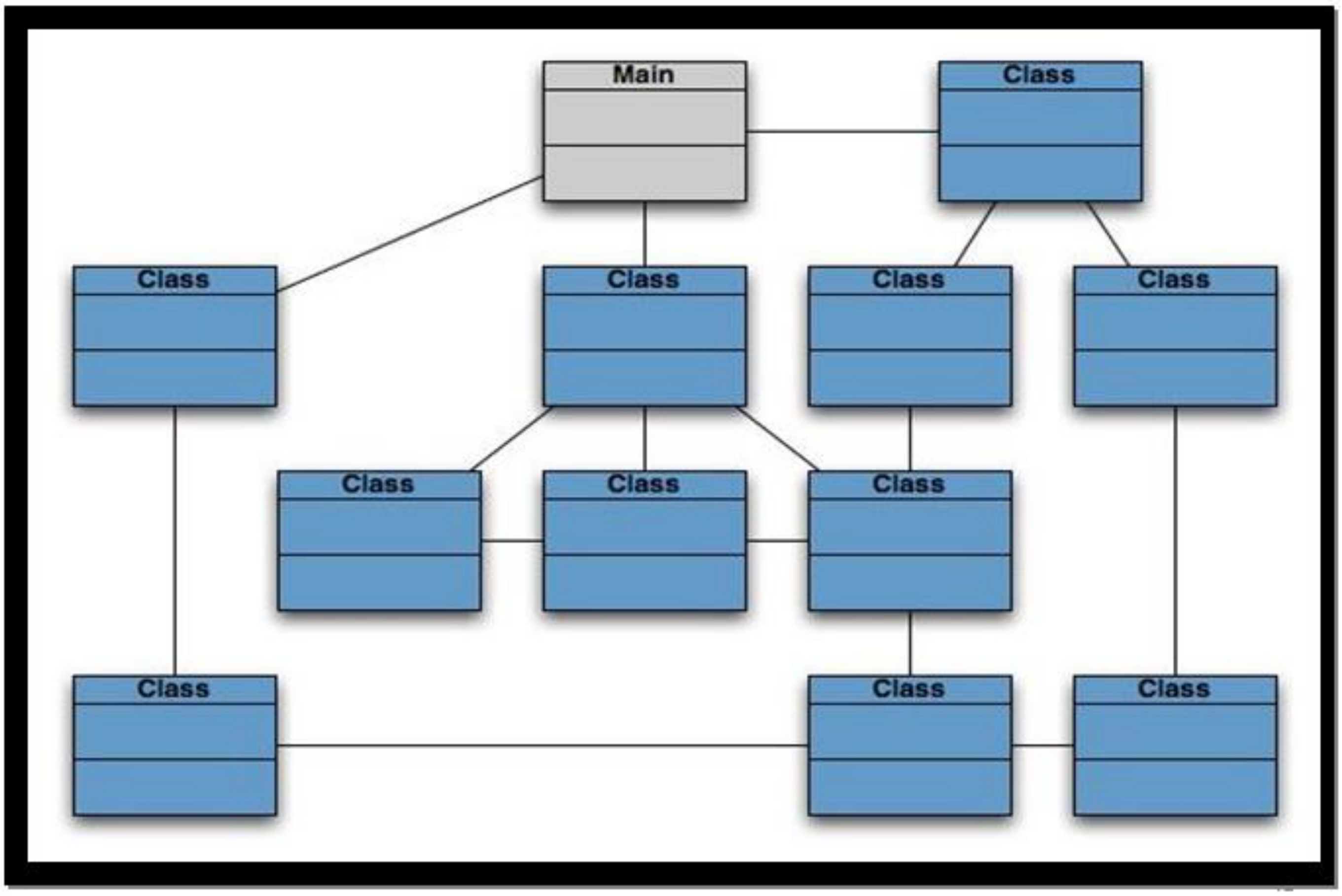


Data-Flow Architecture

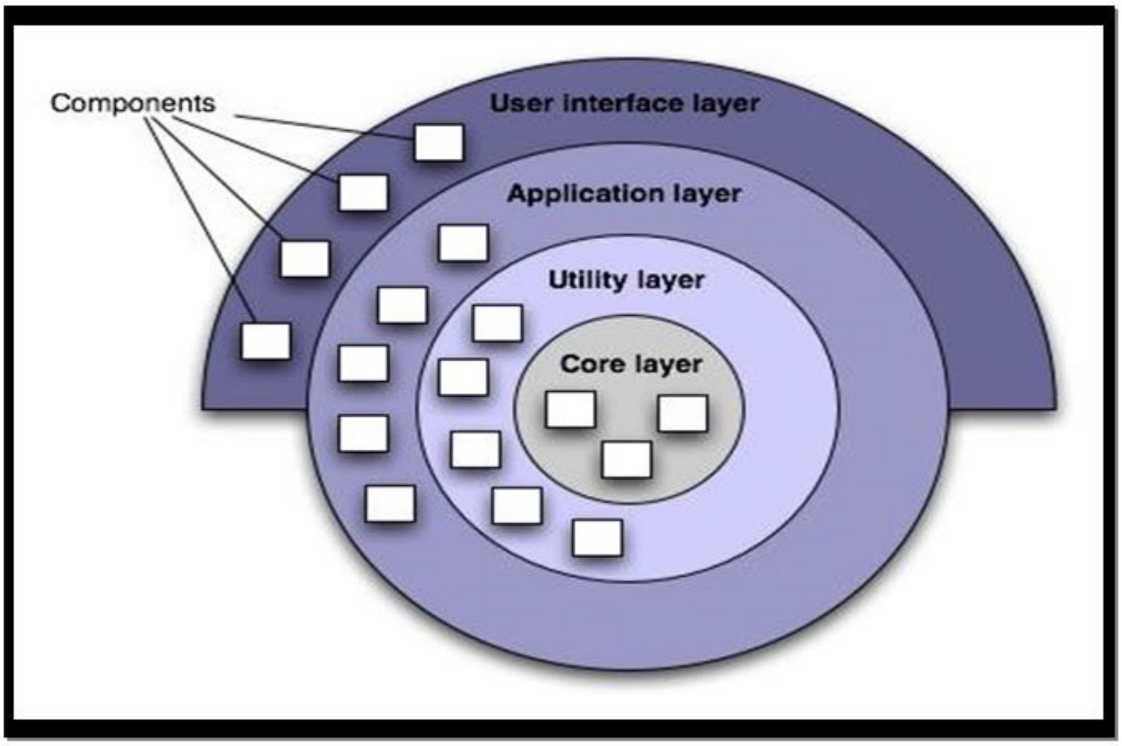


By Shubh Thakare +919595939264

Object-Oriented Architecture



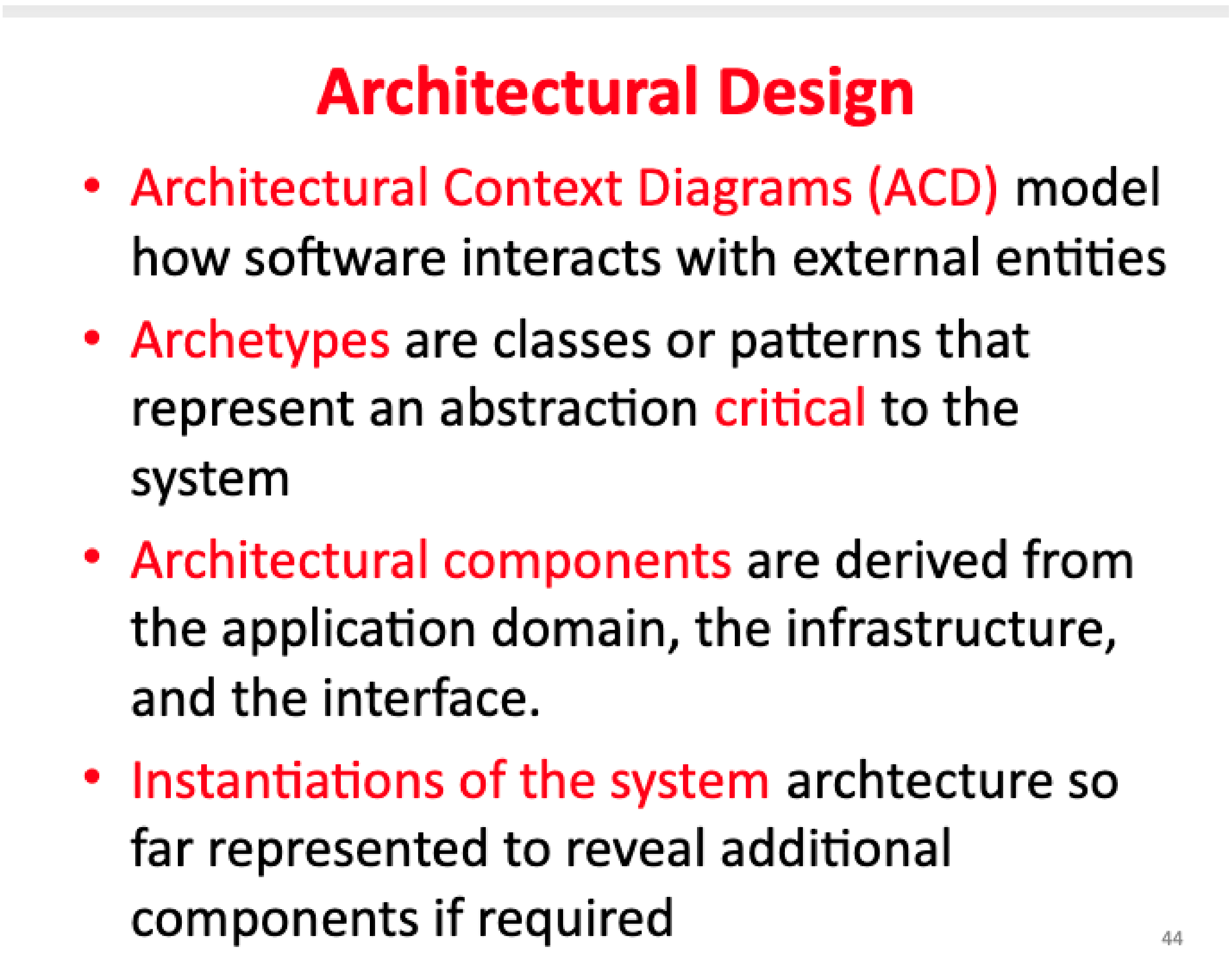
Layered Architecture



Architectural Patterns

- Concurrency
  - operating system process management
  - task scheduler
- Persistence
  - database management system
  - application level persistence
- Distribution
  - broker

4.Architectural Design



Architectural Design

- Architectural Context Diagrams (ACD) model how software interacts with external entities
- Archetypes are classes or patterns that represent an abstraction critical to the system
- Architectural components are derived from the application domain, the infrastructure, and the interface.
- Instantiations of the system archtecture so far represented to reveal additional components if required

44