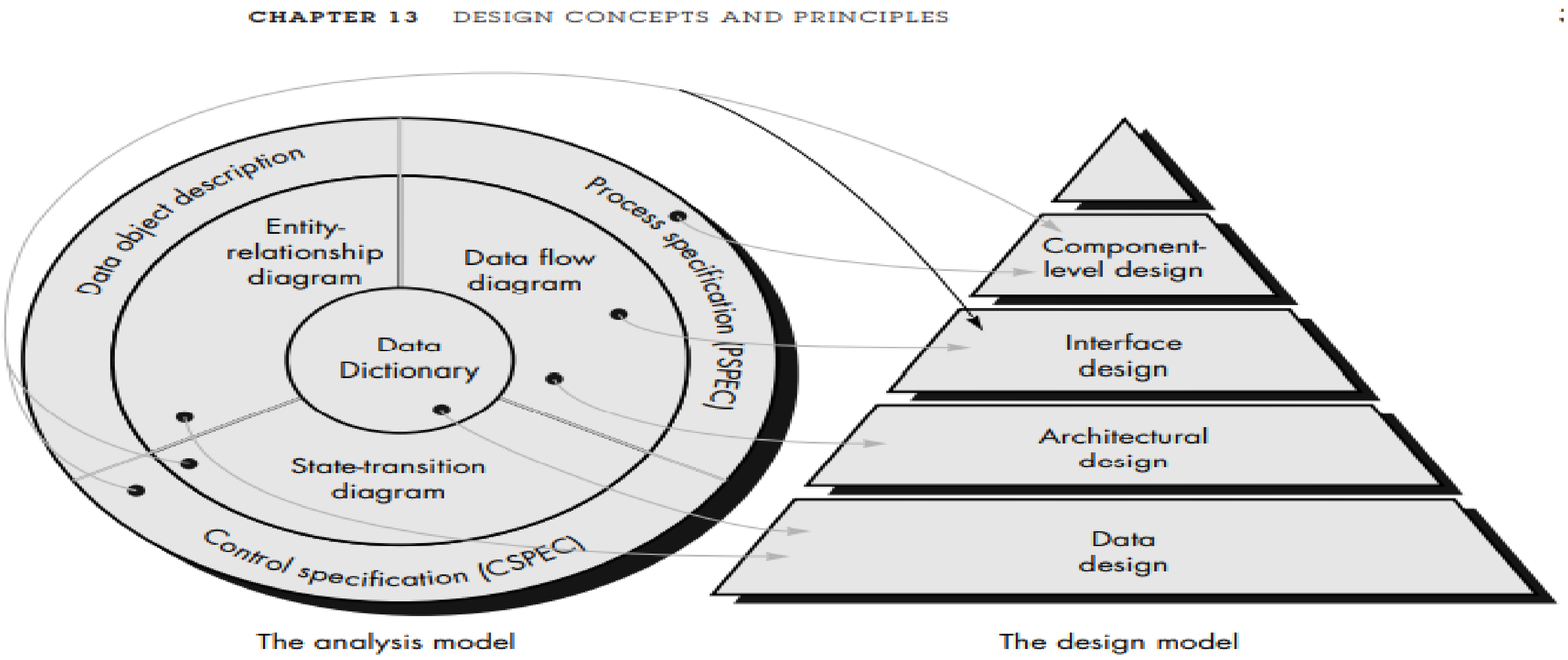
Chapter 4

Design Concepts and Principles

- Design is a **meaningful** engineering representation of something that is to be built.
- Design focuses on four major areas:
    - **Data**
    - **Architecture**
    - **Interfaces**
    - **Components**

Translating the analysis model into a software design



CHAPTER 13 DESIGN CONCEPTS AND PRINCIPLES

Translating the analysis model into a software design

- The **data design** transform the information domain model created during analysis into the **data structures.**
- Data objects and relationships defined in the ERD and the detailed data content depicted in the data dictionary provide the **basis**
- The Architectural design defines the relationship between major **structural elements** of the software.
- The **interface design** describes how the software communicates within systems .

Translating the analysis model into a software design

- The component level design transform **structural elements** of the software architecture into a **procedure description** of software components.
- Information obtained from PSPEC, CSPEC serves as the basis for the component design

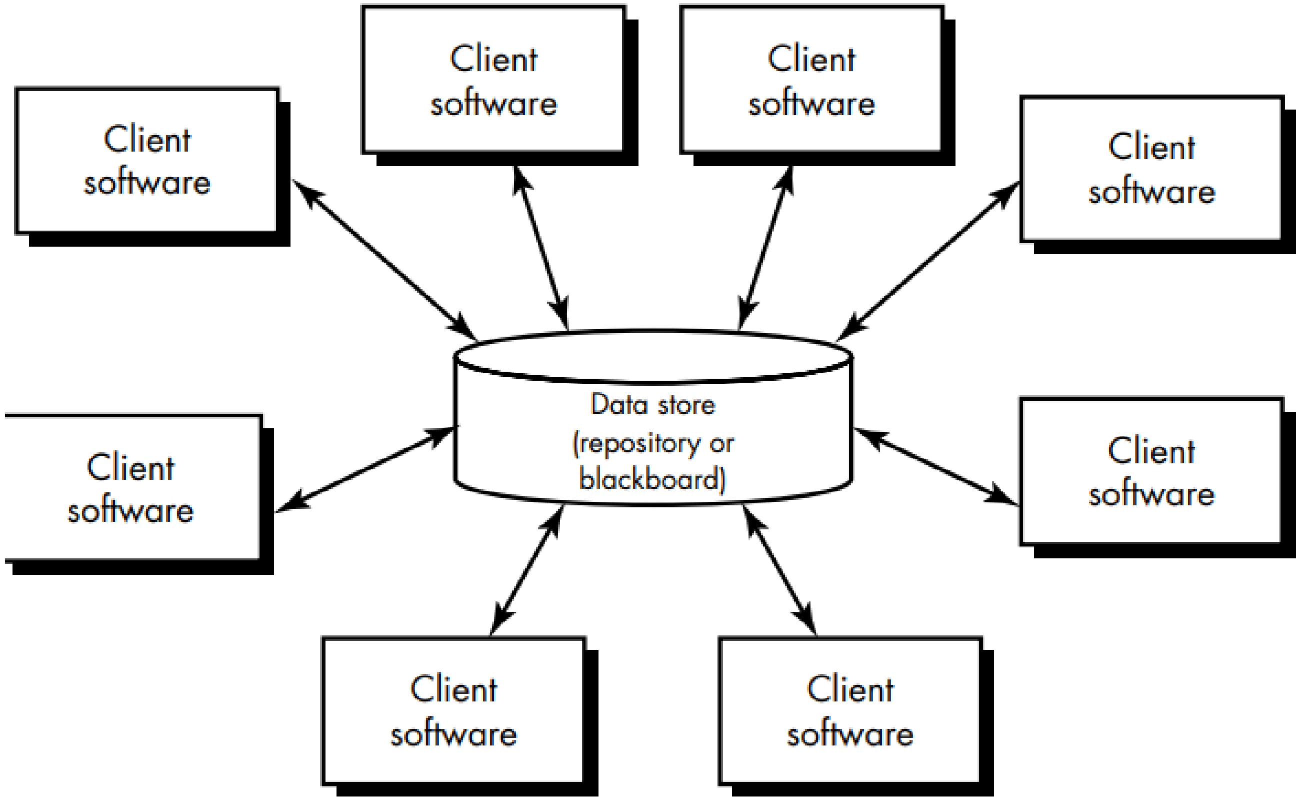
Data Design

- **Data design** sometimes referred to as **data architecting** creates a model of **data and/or information** that is represented at a high level of abstraction the customer/user’s view of data.
- **This data model** is then refined into **progressively** more **implementation-specific representations** that can be processed by the computer-based system.

Architectural Design

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

Architectural Design



Architectural Design

Who does it?

Although a software engineer can design both data and architecture, the job is often allocated to specialists when large, complex systems are to be built.

A database or data warehouse designer creates the data architecture for a system. The “system arc

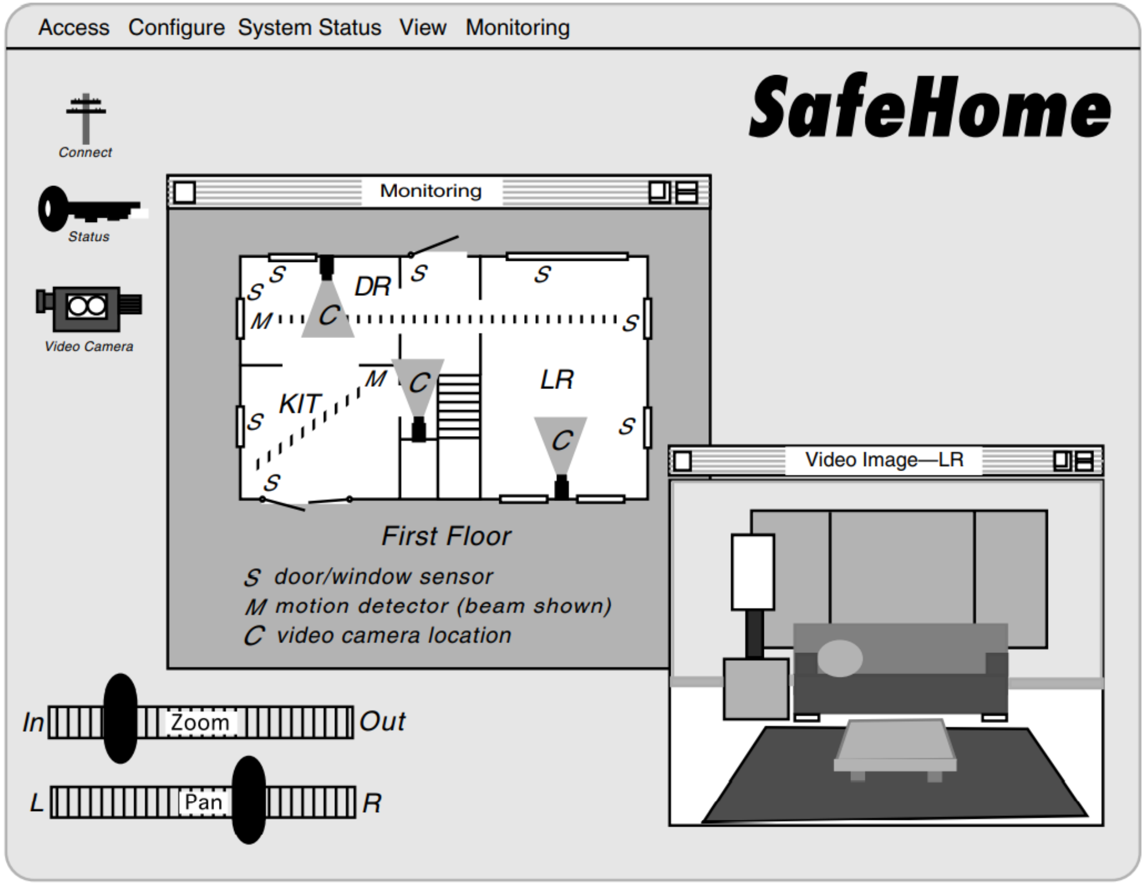
Interface Design

- User interface design creates an effective communication medium between a human and a computer.
- Following a set of interface design principles, design identifies interface objects and actions and then creates a screen layout that forms the basis for a user interface prototype.

Who does it?

- A software engineer designs the user interface by applying an iterative process that draws on predefined design principles

Interface Design



Component Design

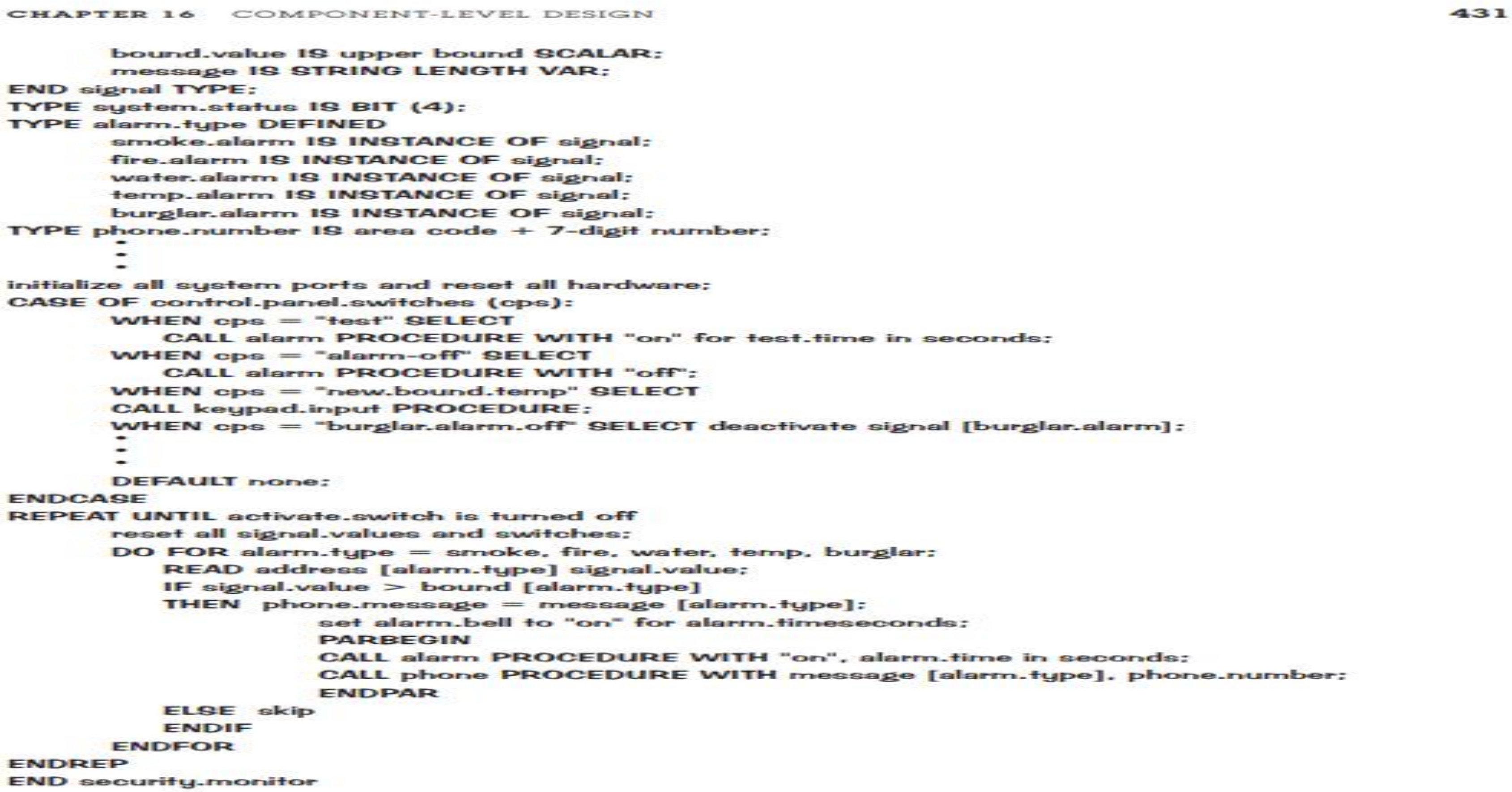
- Component-level design, also called procedural design, occurs after data, architectural, and interface designs have been established.
- The intent is to translate the design model into operational software.
- But the level of abstraction of the existing design model is relatively high, and the abstraction level of the operational program is low
- The translation can be challenging, opening the door to the introduction of subtle errors that are difficult to find and correct in later stages of the software process

Component Design

Who does it?

- A software engineer performs component-level design.

Component Design



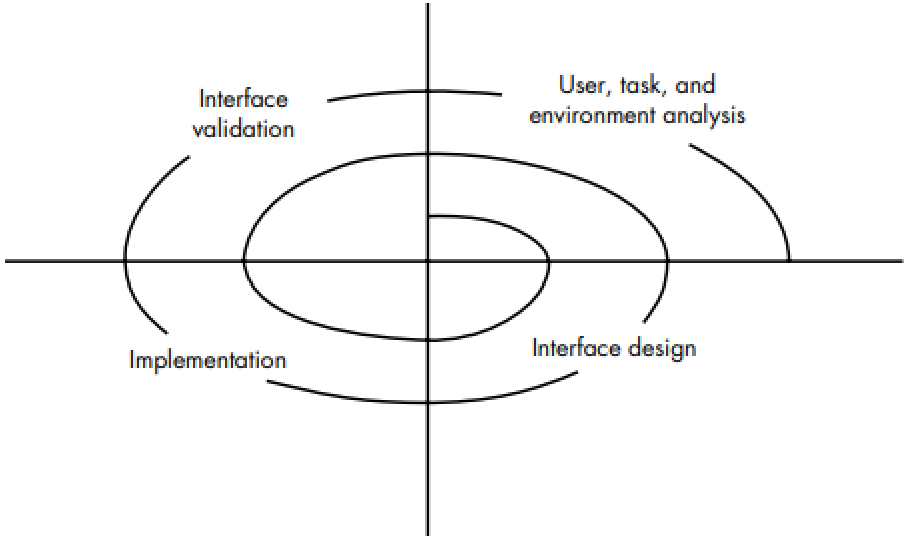
CHAPTER 16 COMPONENT-LEVEL DESIGN 431

```
        bound.value IS upper bound SCALAR;
        message IS STRING LENGTH VAR;
END signal TYPE;
TYPE system.status IS BIT (4);
TYPE alarm.type DEFINED
        smoke.alarm IS INSTANCE OF signal;
        fire.alarm IS INSTANCE OF signal;
        water.alarm IS INSTANCE OF signal;
        temp.alarm IS INSTANCE OF signal;
        burglar.alarm IS INSTANCE OF signal;
TYPE phone.number IS area code + 7-digit number;
        •
        •
        •
initialize all system ports and reset all hardware;
CASE OF control.panel.switches (cps):
        WHEN cps = "test" SELECT
            CALL alarm PROCEDURE WITH "on" for test.time in seconds;
        WHEN cps = "alarm-off" SELECT
            CALL alarm PROCEDURE WITH "off";
        WHEN cps = "new.bound.temp" SELECT
        CALL keypad.input PROCEDURE;
        WHEN cps = "burglar.alarm.off" SELECT deactivate signal [burglar.alarm];
        •
        •
        •
        DEFAULT none;
ENDCASE
REPEAT UNTIL activate.switch is turned off
        reset all signal.values and switches;
        DO FOR alarm.type = smoke, fire, water, temp, burglar;
            READ address [alarm.type] signal.value;
            IF signal.value > bound [alarm.type]
            THEN  phone.message = message [alarm.type];
                        set alarm.bell to "on" for alarm.timeseconds;
                        PARBEGIN
                        CALL alarm PROCEDURE WITH "on", alarm.time in seconds;
                        CALL phone PROCEDURE WITH message [alarm.type], phone.number;
                        ENDPAR
            ELSE  skip
            ENDIF
        ENDFOR
ENDREP
END security.monitor
```

Design Process

- Software design is an **iterative process** through which requirements are translated into a “**blueprint**” for constructing the software.
- Initially, the **blueprint depicts** a holistic view of software.
- That is, the design is represented at a **high level of abstraction**— a level that can be directly traced to the specific system objective and more detailed data, functional, and behavioral requirements.
- As design iterations occur, subsequent refinement leads to design representations at much **lower levels of abstraction.**
- These can still be traced to requirements, but the connection is more subtle

Design Process



Design Guidelines

- Design should exhibit an **architectural structure** .
- A design should be **modular** → logically , function& Sub functions
- A design should contain **distinct representation** of data, architecture, interfaces and components
- A design should lead to component that exhibit **independent functional** characteristics

DESIGN PRINCIPLES

- **The design process should not suffer from “tunnel vision” –** alternative approach
- The design should be **traceable** to the analysis model.
- **The design should not reinvent the wheel**.-- > already exist (new ideas)
- The design should “minimize the **intellectual distance**” between the software and the problem
- The design should exhibit **uniformity** and **integration.–** rules and format should be defined for a design team
- The design should be structured to **accommodate** change.
- The design should be structured to degrade gently, even when aberrant data, events, or operating conditions are encountered.

Design Principles

- Design is not coding, coding is not design
- The design should be reviewed to minimize conceptual errors

DESIGN CONCEPTS

- A set of fundamental software design concepts has evolved over the past four decades
- Each helps the software engineer to answer the following questions:
    - *What **criteria** can be used to **partition software** into individual components?*
    - *How is **function** or **data structure** detail separated from a **conceptual representation** of the software?*
    - *What **uniform criteria** define the **technical quality** of a software design?*

Design Concept

Abstraction

Data abstraction → Door

Procedural abstraction → Open ()

Control abstraction : → synchronization semaphore

Design Concept

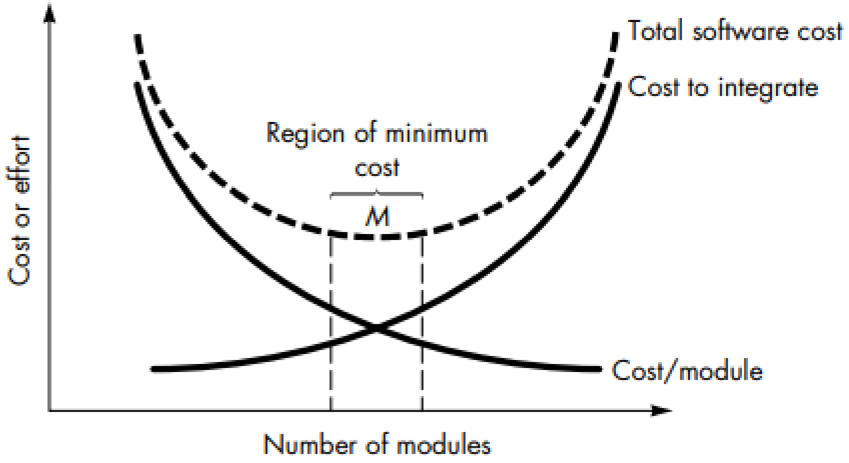
Refinement

- There is a tendency to move immediately to full detail, skipping the refinement steps.
- This leads to errors and omissions and makes the design much more difficult to review.
- Perform stepwise refinement for the design

Design Concept

Modularity

- **Don't over modularize.**
- The simplicity of each module will be overshadowed by the complexity of integration.



Design Concept

Modularity

Criteria that enable effective modular system:

- **Modular decomposability**:
- **Modular composability:**
- **Modular understandability**: module can be understood as a standard alone unit (without reference to other modules)
- **Modular continuity**: small change in the system requirement result in changes to individual modules, rather than systemwide change
- **Modular protection**: able to handle aberrant condition

Design Concept

Software Architecture

- Software architecture implies to “**the overall structure of the software**
- Design concept provides the conceptual integrity for a system
- Software architecture provides the hierarchical structure of program components

Design Concept

Software Architecture

- architectural design can be represented using one or more of a number of different models .
- **Structural models**: represents organized collection of program components
- **Framework models :**High level abstraction by identifying reusability
- **Dynamic models:** address the behavioral aspects of the program
- **Process models**: focus on the design of the business or technical process
- **functional models:** represents functional hierarchy
- A number of different architectural description languages (ADLs) have been developed to represent these models .

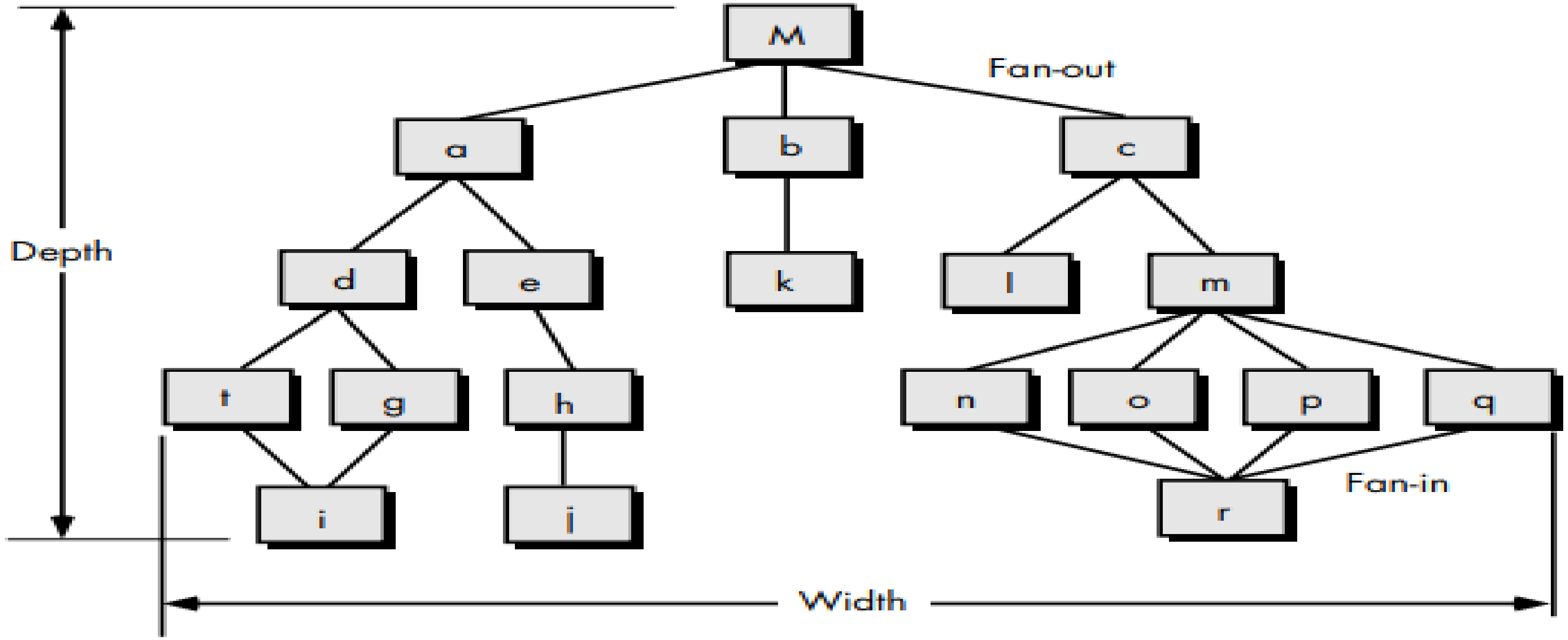
Control Hierarchy

- Control hierarchy, also called **program structure**, represents the organization of program components and implies a hierarchy of control.
- It does not represent procedural aspects of software such as sequence of processes, occurrence or order decisions
- The most common is the treelike diagram that represents hierarchical control for **call and return architectures**.

Control Hierarchy

- **Depth and width** provide an indication of the number of levels of control and overall span of control
- **Fan-out** is a measure of the number of modules that are directly controlled by another module.
- **Fan-in indicates** how many modules directly control a given module.

Control Hierarchy



Control Hierarchy

- The control relationship among modules is expressed in the following way:
- A module that controls another module is said to be **superordinate** to it,
- conversely, a module controlled by another is said to be **subordinate** to the controller .
- module M is superordinate to modules a, b, and c.
- Module h is subordinate to module e and is ultimately subordinate to module M.

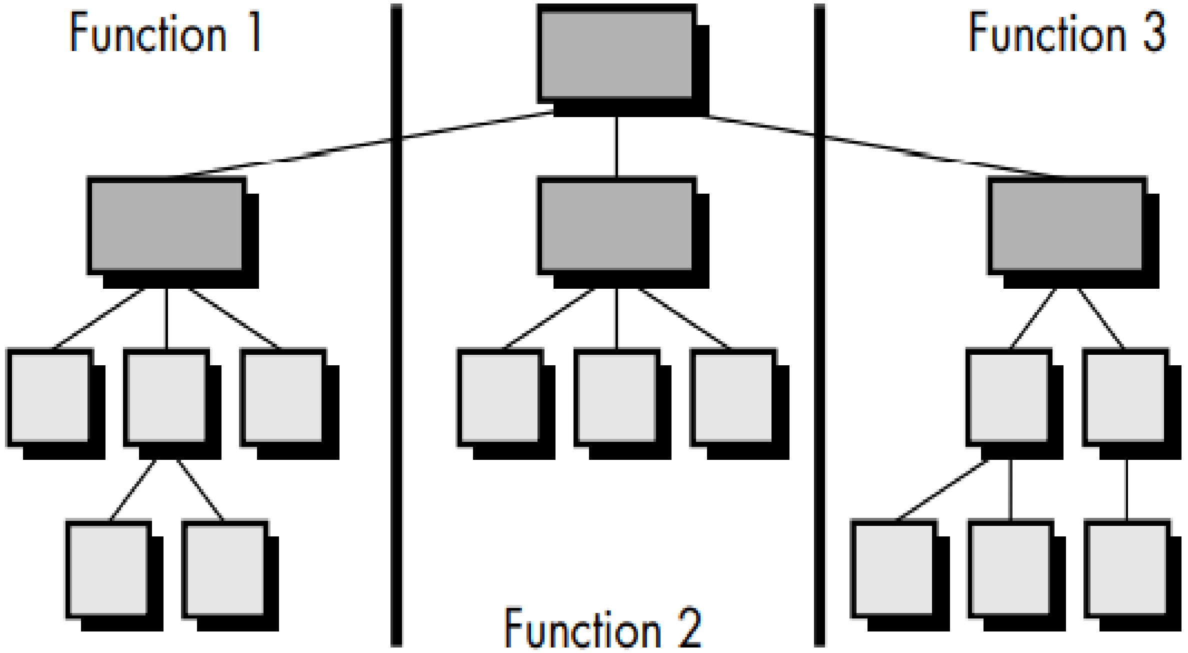
Structural Partitioning

- If the architectural style of a system is hierarchical, the program structure can be partitioned both **horizontally and vertically.**
- **horizontal partitioning** defines separate branches of the modular hierarchy for each major program function.
- **Control modules**, represented in a **darker shade** are used to coordinate communication between and execution of the functions.
- The simplest approach to **horizontal partitioning** defines three partitions—input, data transformation and output.
- Partitioning the architecture horizontally provides a number of distinct benefits

Structural Partitioning

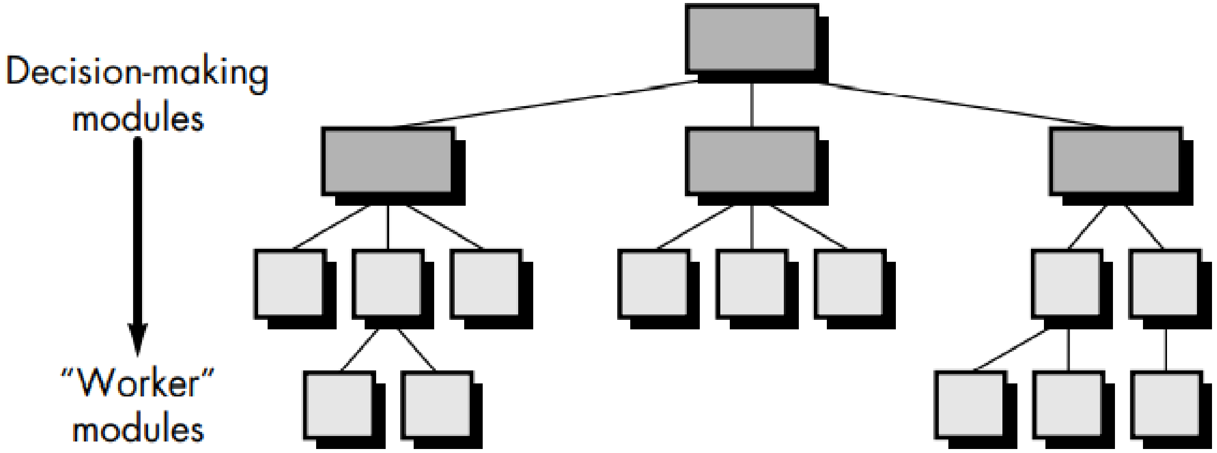
- software that is easier to test
- software that is easier to maintain
- propagation of fewer side effects
- software that is easier to extend

Horizontal partitioning



(a) Horizontal partitioning

Vertical Partitioning



(b) Vertical partitioning

Vertical Partitioning

- often called **factoring**
- control (decision making) and work should be distributed top-down in the program structure.
- Top level modules should perform **control functions** and do little actual processing work.
- Modules that reside low in the structure should be the workers, performing all input, computation, and output tasks.

Vertical Partitioning

- “Worker” modules tend to change more frequently than control modules.
- By placing the workers low in the structure, side effects (due to change) are reduced.

Vertical Partitioning

- change in a control module will have a higher probability of propagating side effects to modules that are subordinate to it.
- A change to a worker module, given its low level in the structure, is less likely to cause the propagation of side effects.
- changes to computer programs revolve around changes to input, computation or transformation, and output.
- The overall control structure of the program (i.e., its basic behavior is far less likely to change).
- For this reason vertically partitioned structures are less likely to be susceptible to side effects
- when changes are made and will there fore be more maintainable—a key quality fact

Data Structure

- Data structure is a representation of the logical relationship among individual elements of data.
- Because the structure of information will invariably affect the final procedural design,
- data structure is as important as program structure to the representation of software architecture.
- the organization, methods of access, degree of associativity, and processing alternatives for information.

Data Structure

- A **scalar item** is the **simplest** of all data structures.
- a scalar item represents a **single element** of information
- The size and format of a scalar item may vary within bounds that are dictated by a programming language.
- For example, a scalar item may be a logical entity one bit long, an integer or floating point number that is 8 to 64 bits long, or a character string that is hundreds or thousands of bytes long.
- When scalar items are organized as a **list or contiguous group**, a **sequential vector** is formed.

Data Structure

- When the sequential vector is extended to two, three, and ultimately, an arbitrary number of dimensions, an n-dimensional space is created.
- The most common n-dimensional space is the two-dimensional matrix. In many programming languages, an n-dimensional space is **called an array**.

Software Procedure

- **Program structure** defines control hierarchy without regard to the sequence of processing and decisions.
- Software procedure focuses on the **processing details** of each module individually.
- Procedure must provide a precise specification of processing, including sequence of events, exact decision points, repetitive operations, and even data organization and structure.

Information Hiding

- The concept of modularity leads every software designer to a fundamental question: "**How do we decompose a software solution to obtain the best set of modules?“**
- The principle of information hiding suggests that modules be characterized by design decisions that hides from all others.
- modules should be specified and designed so that information (procedure and data) contained within a module is inaccessible to other modules that have no need for such information.

Effective Modular Design

Functional Independence:

- Functional independence is achieved by developing modules with "single-minded" function
- Functional Independence removes the excessive interaction with other modules

Coupling and Cohesion

- **Cohesion:**
  Cohesion is the indication of the relationship **within module**.
- It is concept of intra-module.
- Cohesion has many types but usually highly cohesion is good for software.

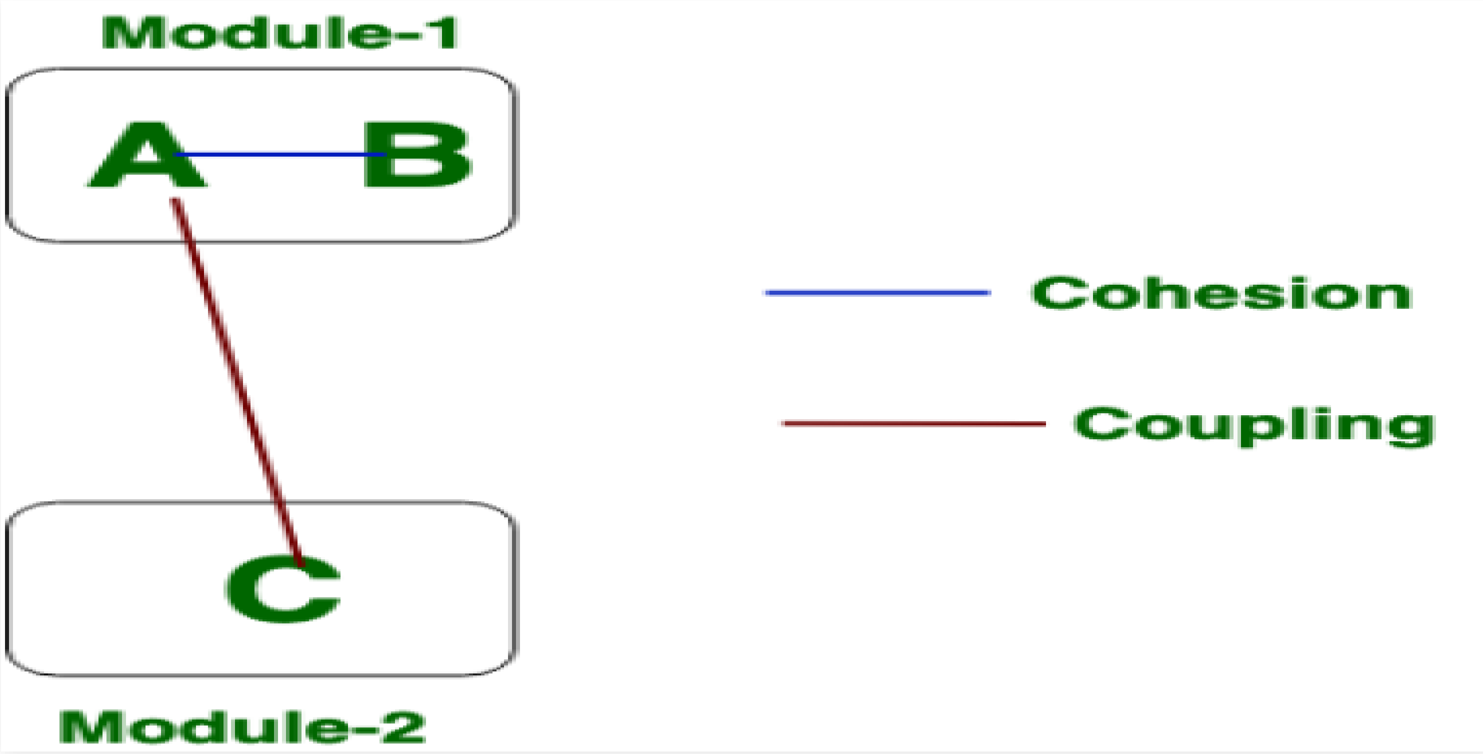
Coupling and Cohesion

- Coupling is also the indication of the relationships **between** modules.
- It is concept of Inter-module.
- Coupling has also many types but usually low coupling is good for software.

Coupling and Cohesion

- While you are designing the system always focus on low coupling and high cohesion.

Coupling and Cohesion



Architecture Design

ARCHITECTURAL STYLES

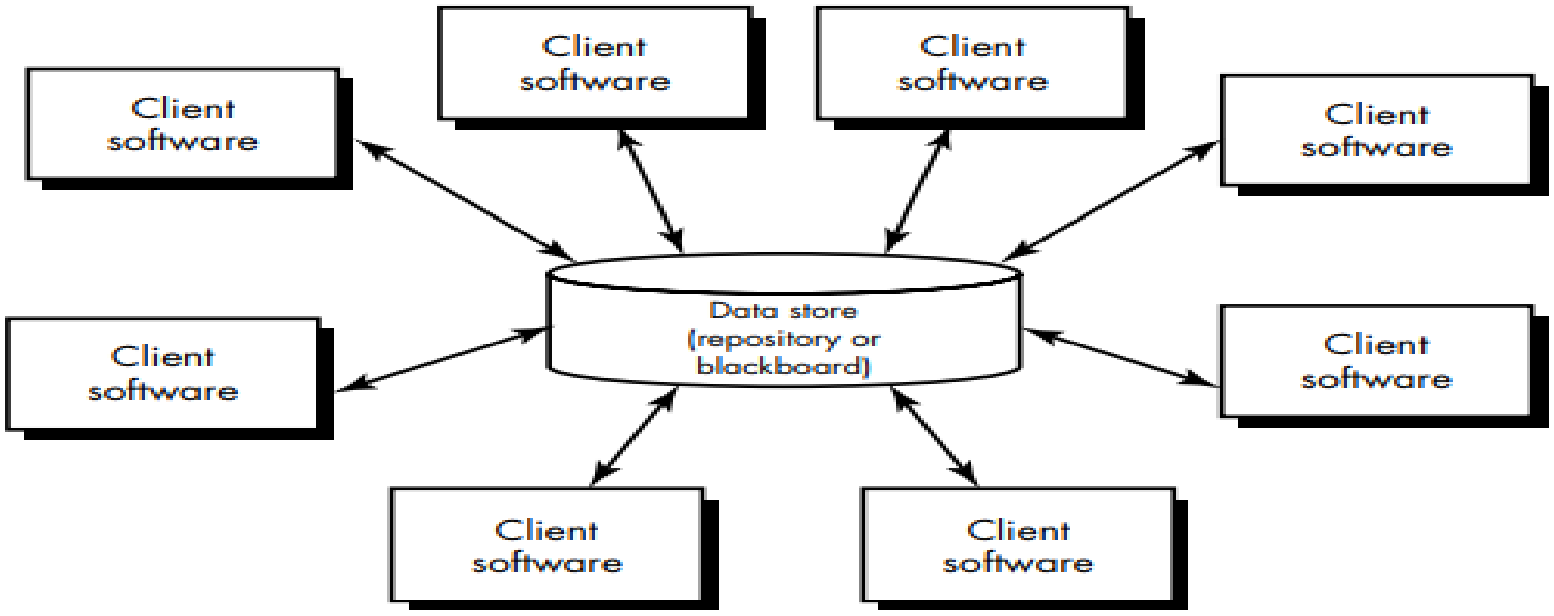
When a builder uses the phrase **“center hall colonial”** to describe a house, most people familiar with houses in the United States will be able to conjure a general image of what the house will look like and what the floor plan is likely to be

Architecture Patterns/ Styles

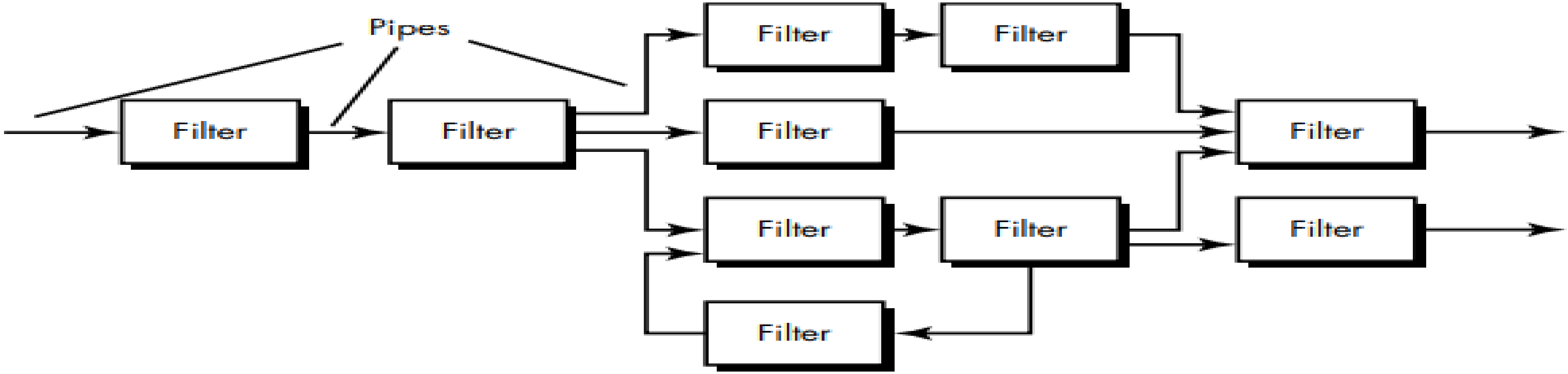
Data-centered architectures

- A data store resides at the center of this architecture
- Data is accessed frequently by other components that update, add, delete, or otherwise modify data within the store.
- Client software accesses a central repository.

Data Center Architecture



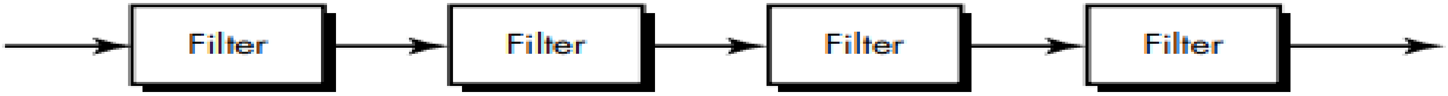
Data Flow Architecture



(a) Pipes and filters

Data Flow Architecture

(a) Pipes and filters



(b) Batch sequential

Data Flow Architecture

- This architecture is applied when input data are to be transformed through the series of computational components into output.
- Pipe and filter pattern is used for the DATA FLOW ARCHITECTURE.
- Each **processing** step is encapsulates in a filter component
- Data is **passed** through the pipe between adjacent filters.
- If the data flow degenerates into a single line of transformation it is termed as sequential
- This pattern accepts a batch of data and applied series of sequential components.

Call and return architecture

- This architecture helps designer to modify and scale program easily.

A number of substyles exist within this category.

- **Main Program and Subprogram architecture :-**
- **Remote procedure call architecture**:

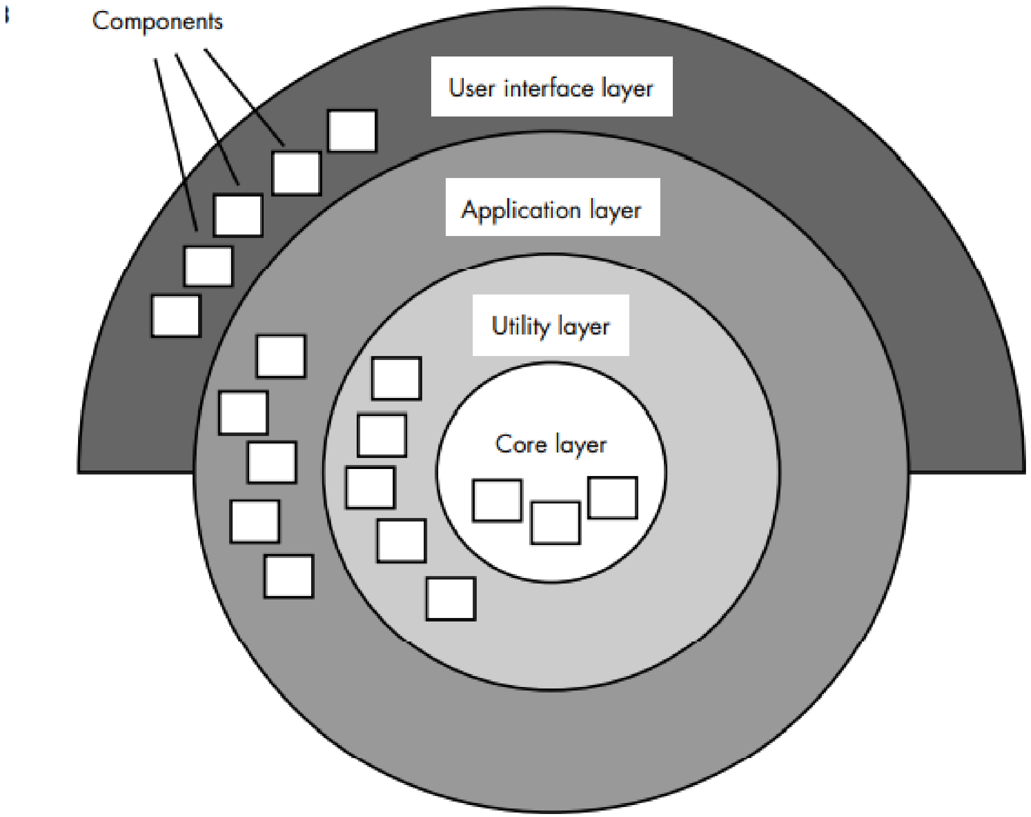
Object Oriented Architecture

- Communication and co-ordination between components is accomplished by message passing.

Layered Architecture

- A number of different layers are defined.
- Each accomplishing operations that progressively become closer to the machine instruction set.
- At the **outer layers,** components service the user interface operations.
- At the **inner layer**, components perform operating system interfacing.

Layered Architecture



DATA DESIGN

- Data Design creates a **model** of data that is represented at a high level of abstraction.
- This data model is then refined into progressively more implementation specific representation that can be processed by the computer-based system.
- Data Design plays an important role in following cases

At the program component level

At the application data design

At the business level

Data Modeling, Data Structure, Databases and Data Ware house

- The data objects **defined** during the software requirements analysis are modeled using **ERD and Data Dictionary.**
- Data Design activity translate these elements of the model into data structures.
- Data warehouse is a separate data environment that is not directly integrated with day-to day application but encompasses all data used by a business
- Data warehouse is a large, independent database that serve the set of applications required by a business.

Principles of Data Design

Set of Principles for the data Design

1. The systematic analysis principles should be applied to the data.
2. All data structures and the operations to be performed on each should be identified.
3. A data dictionary should be established

Principles of Data Design

4. The representation of data structures should be known only to those modules that must make direct use of the data

5.. A library of useful data structures and the operations should be developed.

6. A software design and programming language should support the specification and realization of abstract data types.