Unit 4 Software Design

Prepared by: Dr. Pooja M Bhatt
Department of Computer Engineering
MBIT,CVM University

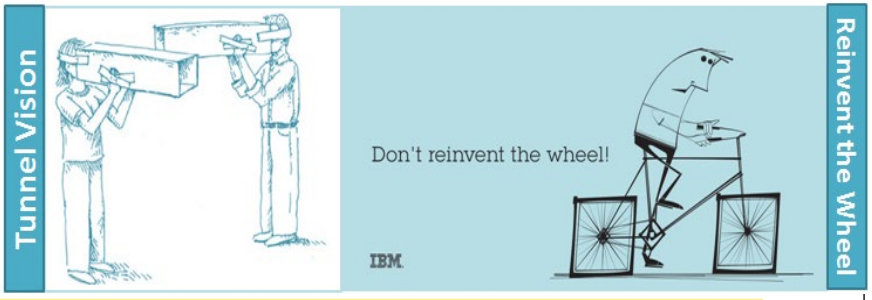
Index

- Design Concepts and Design Principal
- Architectural Design
- Component Level Design
- Function Oriented Design
- Object Oriented Design
- User Interface Design
- Web Application Design

Design

- A **meaningful representation** of something to be built.
- Software design is more creative than analysis.
- It is a problem solving activity.
- It's a **process** by which **requirements** are **translated** into **blueprint** for constructing a software.
- **Blueprint** gives us the **holistic view** (entire view) of a **software.**

Design principles



1. Design process should **not suffer** from **"tunnel vision".**
2. Design should be **traceable** to the **analysis model.**
3. Design should **not reinvent** the **wheel.**
4. Design should **"minimize the intellectual distance"** between the **software** and **the real world** problem.
5. Design should exhibit (**present**) **uniformity** and **integration.**

Design principles

6. Design should be **structured to accommodate change.**
7. Design should be **structured** to **degrade gently**, even when abnormal data, events, or operating conditions are encountered.
8. **Design is not coding, coding is not design**.
9. **Design** should be **assessed for quality** as it is **being created**, not after the fact.
10. Design should be **reviewed to minimize conceptual** (semantic) errors.

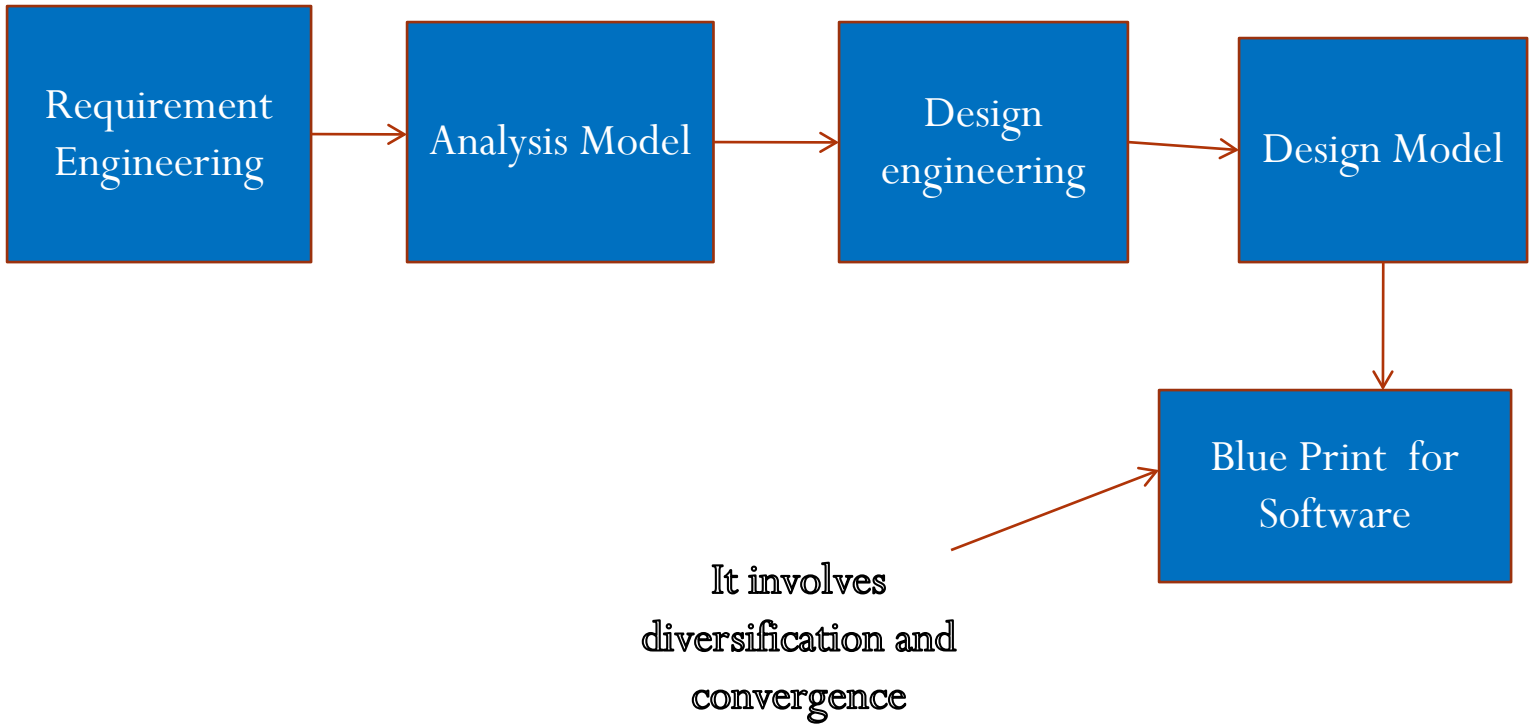
Design Concepts

The beginning of **wisdom** for a **software engineer** is to **recognize** the **difference** between **getting program to work** and **getting it right.**

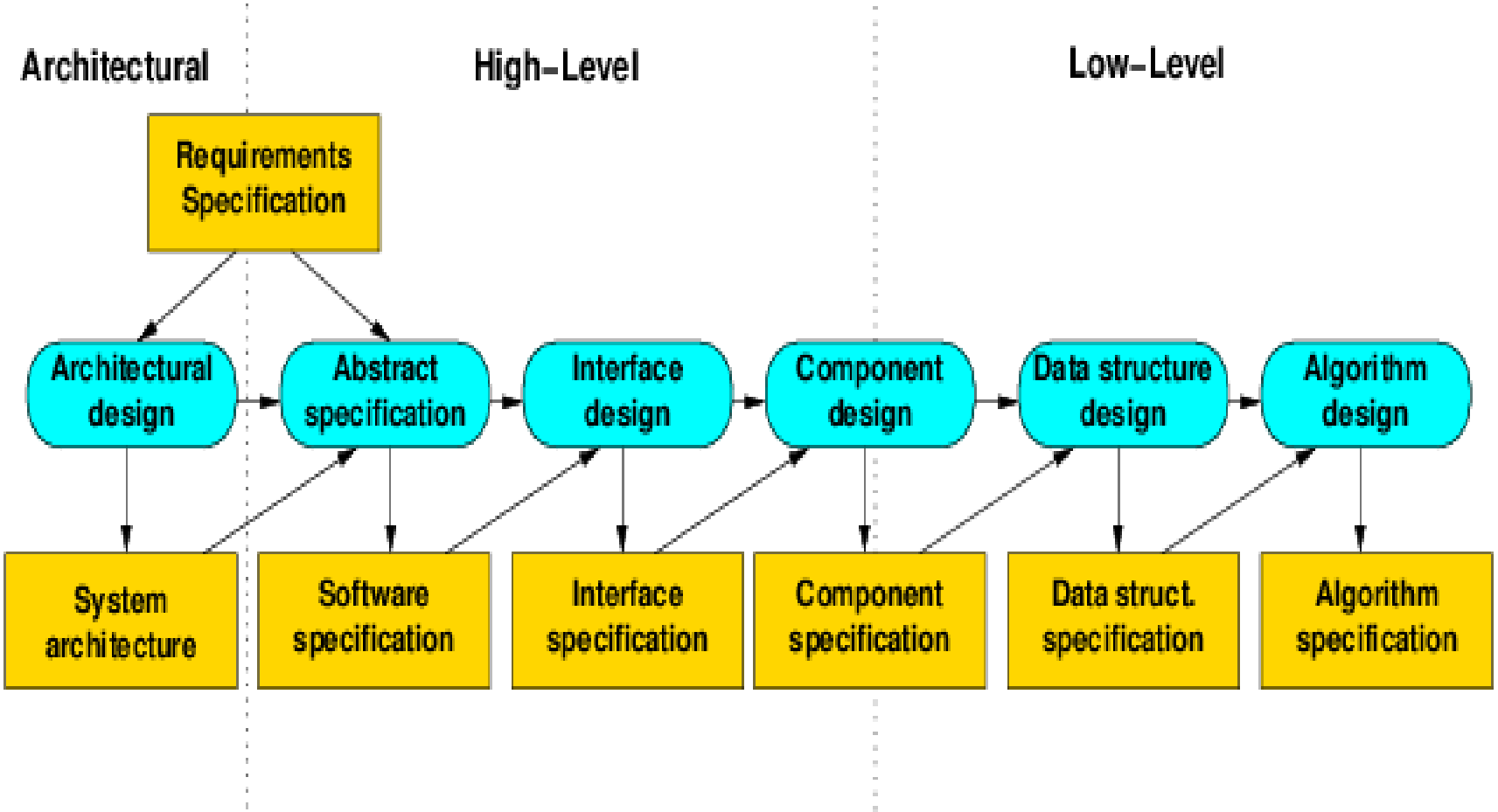
- **Abstraction** (data, procedure, control).
- **Architecture** (the overall structure of the software).
- **Pattern**("conveys the essence" of a proven design solution).
- **Separation of Concern**(any complex problem can be more easily handled by subdivisions in small pieces).

- **Modularity**(compartmentalization of data and function)
- **Information Hiding**(controlled interface)
- **Functional Independence**(single minded function and low coupling)
- **Refinement**(elaboration of details for all abstractions)
- **Aspect**(a mechanism to understand how global req. affect design)
- **Refactoring**(a reorganization technique the simplifies the design.

Software Design Process

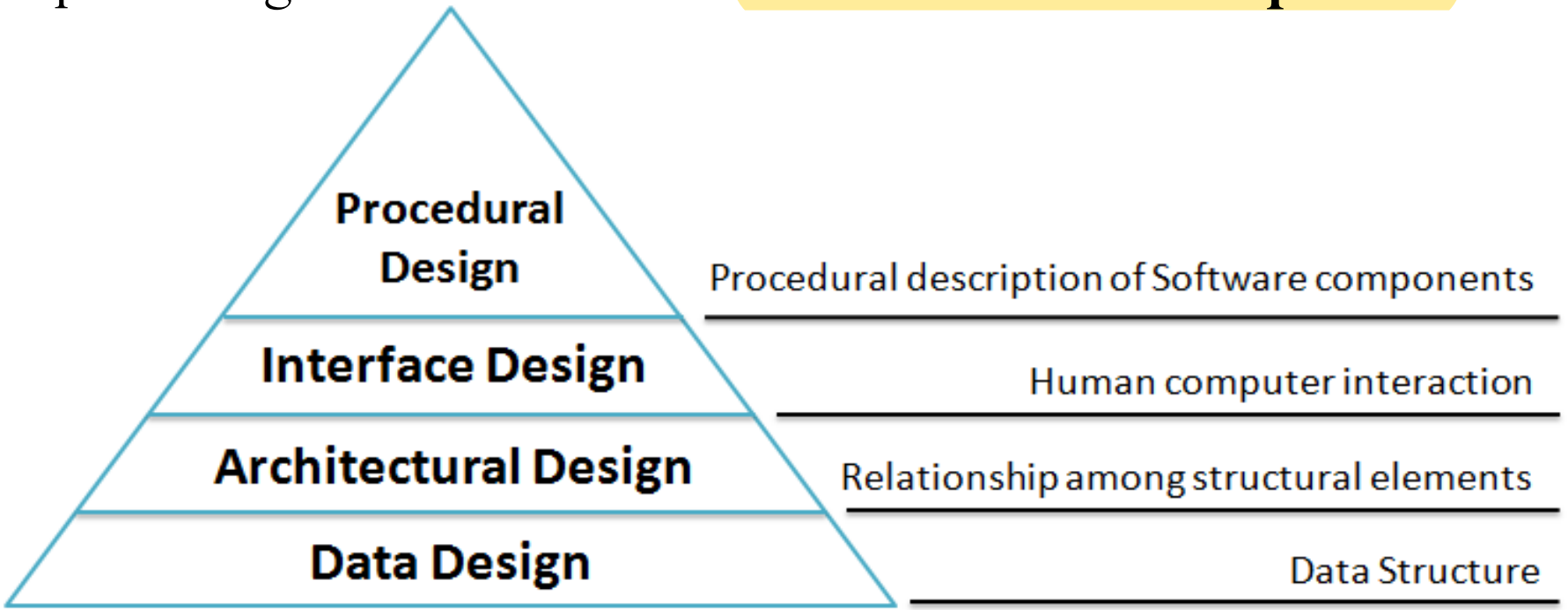


DESIGN PROCESS



Design Models

- It is **creative** activity.
- Its a most **critical activity** during system development.
- It has great **impact** on **testing** and **maintenances**.
- Prepare design document forms **reference for later phases.**



- ***Data Design***
- The data design transforms the information domain model created during analysis into the data structures that will be required to implement the software.
- The data objects and relationships define in the entity relationship diagram.
- Part of data design may occur in combination with the design of software architecture.
- ***Architectural Design***
- The architectural design defines the relationship between major structural elements of the software.
- The architectural design representation—the framework of a computer-based system—can be derived from the system specification, the analysis model, and the interaction of subsystems defined within the analysis model.

- ***Interface Design***
- The interface design describes how the software communicates within itself, with systems that interoperate with it, and with humans who use it.
- An interface implies a flow of information (e.g., data and/or control) and a specific type of behavior. Therefore, data and control flow diagrams provide much of the information required for interface design.
- ***Procedural-level Design***
- The component-level design transforms structural elements of the software architecture into a procedural description of software components.

Quality attributes of software design

- **Functionality:** assessed by **feature set** and **capabilities of** the **program**, **generality** of the **functions** & **security** of overall **system.**
- **Usability:** assessed by considering **human factors**, overall **aesthetics**, **consistency** & **documentations.**
- **Reliability:** assessed by measuring **frequency** & **severity** of **failures**, **accuracy** of **outputs**, **mean-time-of-failure (MTTF)**, **ability** to **recover** from **errors.**
- **Performance:** measured by **processing speed**, **response time**, **resource consumption**, **throughput** and **efficiency.**
- **Supportability: Ability to extend program**, adaptability, serviceability, testability, compatibility

Software Architecture & Design

- **Large systems** are **decomposed** into **subsystems**
- **Sub-systems** provide **related services.**
- Initial design process includes Identifying sub-systems , Establishing a framework for sub-system control and communication.
- Why to document the Architecture?
- **Stakeholder Communication:** High-level presentation of system
- **System Analysis:** Big effect on performance, reliability, maintainability and other –ilities **(Usability, Maintainability, Scalability, Reliability, Extensibility, Security, Portability)**
- **Large-scale Reuse:** Similar requirements similar architecture

Software Architecture & Design

- Architectural design represents the **structure of data and program components**
- It considers, **Architectural style** that the system will take **Structure** and **properties** of the **components** that constitute the system, and **Interrelationships** that occur among all architectural components of a system.
- Representations of software architecture are an **enable for communication between all parties** (stakeholders).
- Architecture "constitutes a relatively small, intellectually graspable model of how the system is structured and how its components work together".

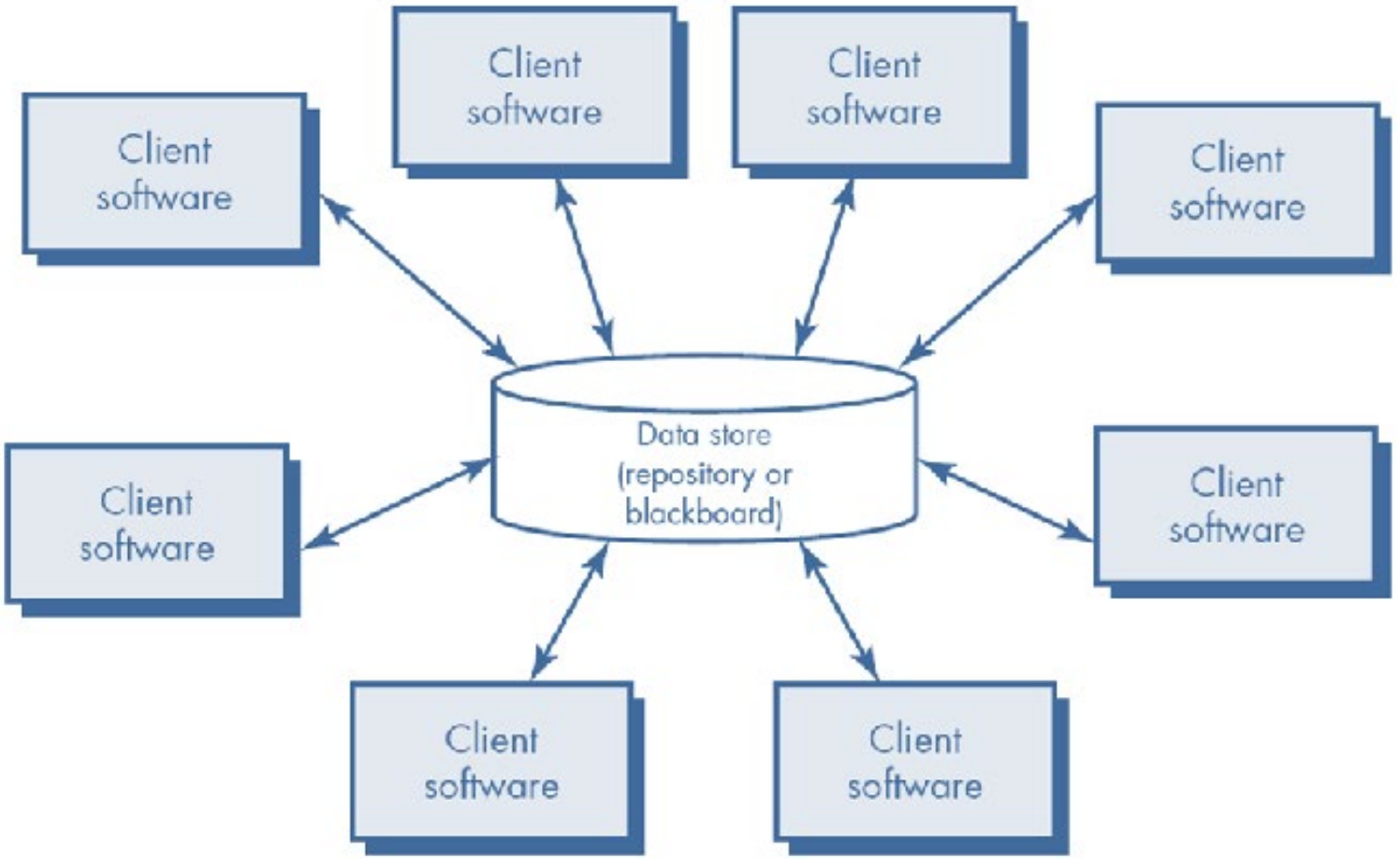
Architectural Styles

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

Architectural Styles

- Each style describes a system category that encompasses,
- **A set of components** (ex. a database, computational modules) that perform a function required by a system.
- **A set of connectors** that enable "communication, coordination and cooperation" among components.
- **Constraints** that define how components can be integrated to form the system.
- **Semantic models** that enable a designer to understand the overall properties of a system.

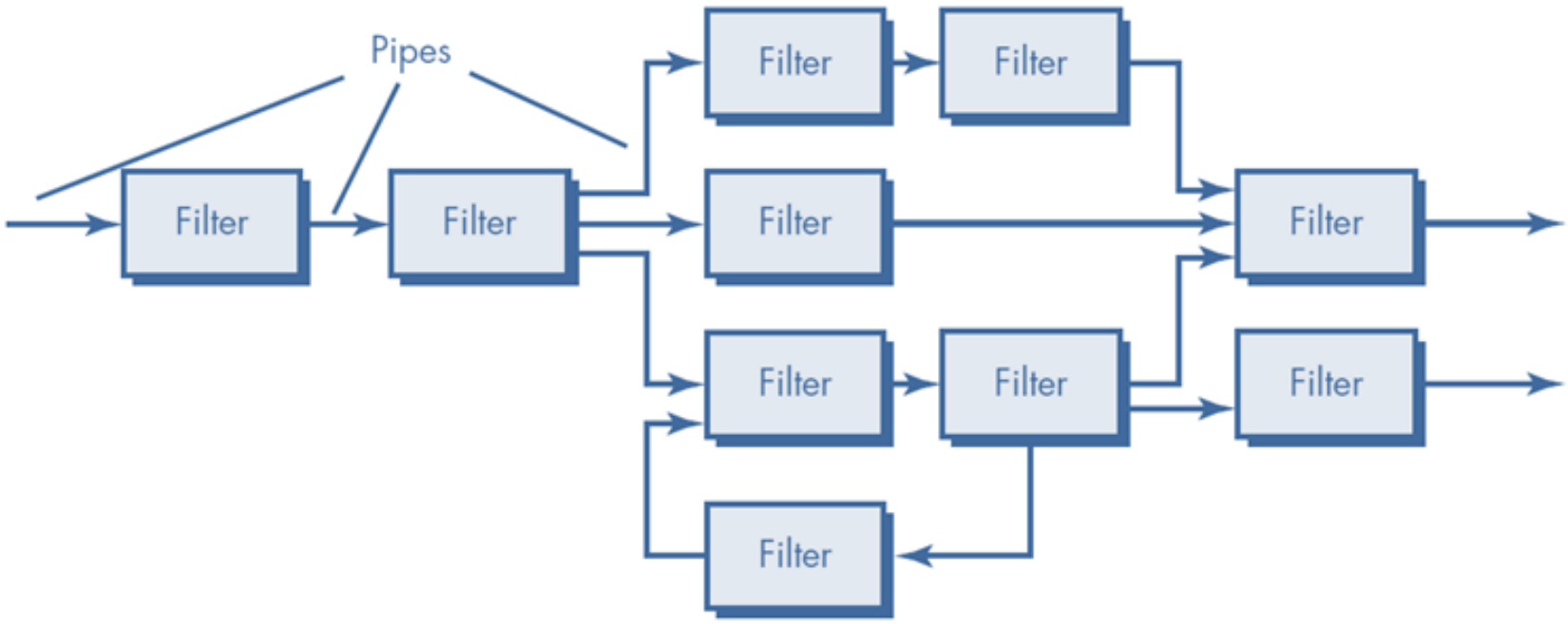
Data-centered architecture style



Data-centered architecture style

- **A data store** (Ex., a file or database) **resides at the center** of this architecture and is **accessed frequently** by other components.
- Client **software accesses a central repository**.
- In **some cases** the data **repository is passive**.
- That is, client software accesses the data independent of any changes to the data or the actions of other client software.

Data-flow architectures



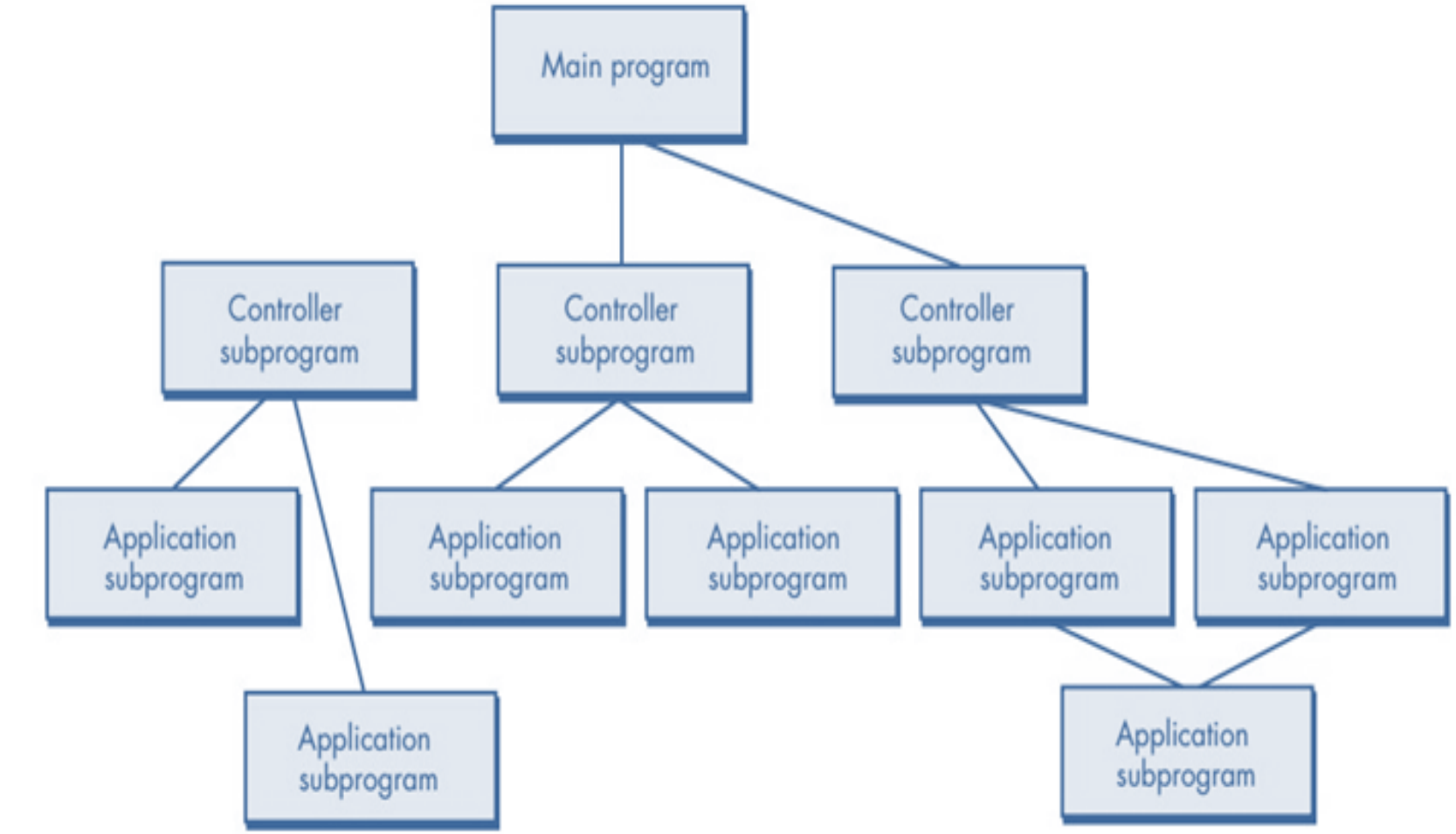
Pipes and filters

Data-flow architectures

- This architecture is applied **when input data are to be transformed**.
- A set of components (called **filters**) **connected by pipes** that **transmit data** from one component to the next.
- Each filter works independently of those components upstream and downstream, is designed to expect data input of a certain form, and produces data output (to the next filter) of a specified form.

Call and return architecture

- This architectural style **enables a software designer (system architect) to achieve a program structure** that is relatively easy to modify and scale.
- A number of sub styles exist within this category as below.
- **1. Main program/subprogram architectures**
- This classic program structure **decomposes function into a control hierarchy** where a **"main" program invokes a number of program components**, which in turn may invoke still other components.
- **2. Remote procedure call architectures**
- The components of a main **program/subprogram** architecture are **distributed across multiple computers** on a network.

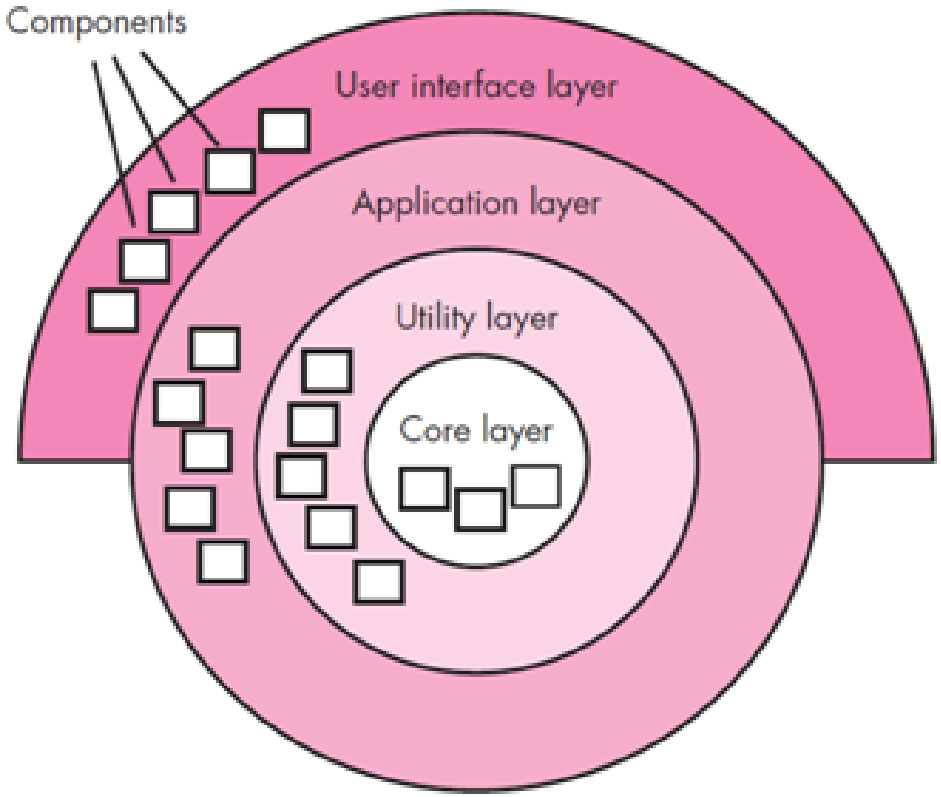

Main program
Controller subprogram
Controller subprogram
Controller subprogram
Application subprogram
Application subprogram
Application subprogram
Application subprogram
Application subprogram
Application subprogram
Application subprogram

Object-oriented architecture

- The **components** of a system **encapsulate data and the operations** that must be applied to manipulate the data.
- **Communication** and **coordination** between components is accomplished **via 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 layer**, components service **user interface** operations.
- At the **inner layer**, components perform **operating system interfacing**.
- **Intermediate layers** provide **utility services** and application **software functions**.


Components
User interface layer
Application layer
Utility layer
Core layer

Component(procedural) Level design

- **Component** is a modular, deployable and replaceable part of a system that **encapsulates** implementation and exposes a set of system that interfaces.
- Component-level design **occurs after data, architectural and interface designs** have been established.
- It **defines** the **data structures, algorithms, interface characteristics,** and **communication mechanisms** allocated to each component.
- The intent is to **translate** the **design model** into **operational software**.
- But the abstraction level of the existing design model is relatively high and the abstraction level of the operational program is low.

Function Oriented Approach

- The following are the features of a typical function-oriented design approach:
- **1. A system is viewed as something that performs a set of functions.**
- Starting at this high level view of the system, **each function** is successively **refined into more detailed functions**.
- For example, consider a **function create-new-library member -**which essentially creates the record for a new member, assigns a unique membership number to him, and prints a bill towards his membership charge.

Continue..

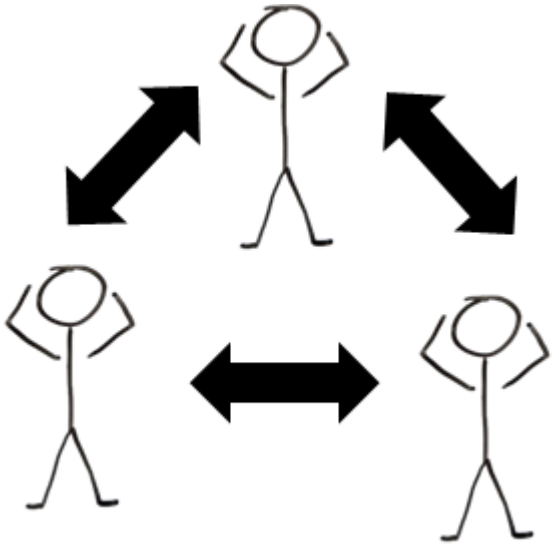
- This function may consist of the following **sub-functions**: » **assign-membership-number, create-member-record and print-bill**
- Each of these sub-functions may be split into more detailed sub-functions and so on.
- **2. The system state is centralized and shared among different functions.**
- For Ex., data such as **member-records** is available for reference and updating to several functions such as:
- – create-new-member – delete-member
- – update-member-record

Object Oriented Approach

- In the object-oriented design approach, the **system is viewed** as **collection of objects** (i.e., entities).
- The **state is decentralized** among the objects and **each object manages its own state** information.
- For example, in a Library Automation Software,
- each library member may be a separate object with its own data and functions to operate on these data.
- In fact, the functions defined for one object cannot refer or change data of other objects.
- **Objects** have their **own internal data** which define their state.

Cohesion & Coupling

- A **good software design** implies **clean decomposition** of the **problem** **into** **modules**, and the **neat arrangement** of these **modules** in a **hierarchy**.
- The primary **characteristics** of **neat** **module decomposition** are **high cohesion** and **low coupling**.



- A **cohesive module** performs a single task, requiring little interaction with other components.
- A **Coupling** is an indication of the relative interdependence among modules.

Cohesion

- Cohesion is an **indication** of the **relative functional strength** of a module.
- A **cohesive module** performs a **single task**, **requiring little interaction** with other components.
- Stated simply, a **cohesive module** should (ideally) **do just one thing**.
- A module having **high cohesion** and **low coupling** is said to be **functionally independent** of other modules.
- By the term functional independence, we mean that a **cohesive module performs a single task or function**.

Classification of Cohesion

Classification of Cohesion		
	Coincidental	**Low**
	Logical	
	Temporal	
	Procedural	
	Communicational	
	Sequential	
	Functional	**High**

- **Coincidental cohesion :**
- A module is said to have coincidental cohesion, if it performs a set of tasks that relate to each other very loosely.
- In this case, the module contains a random collection of functions.
- It is likely that the functions have been put in the module out of pure coincidence without any thought or design.
- For Ex., in a transaction processing system (TPS), the get-input, print-error, and summarize-members functions are grouped into one module.

- **Logical cohesion:**
- A module is said to be logically cohesive, if all elements of the module perform similar operations.
- For Ex., error handling, data input, data output, etc.
- An example of logical cohesion is the case where a set of print functions generating different output reports are arranged into a single module.
- **Temporal cohesion:**
- When a module contains functions that are related by the fact that all the functions must be executed in the same time span.
- For Ex., the set of functions responsible for initialization, start-up, shutdown of some process, etc.

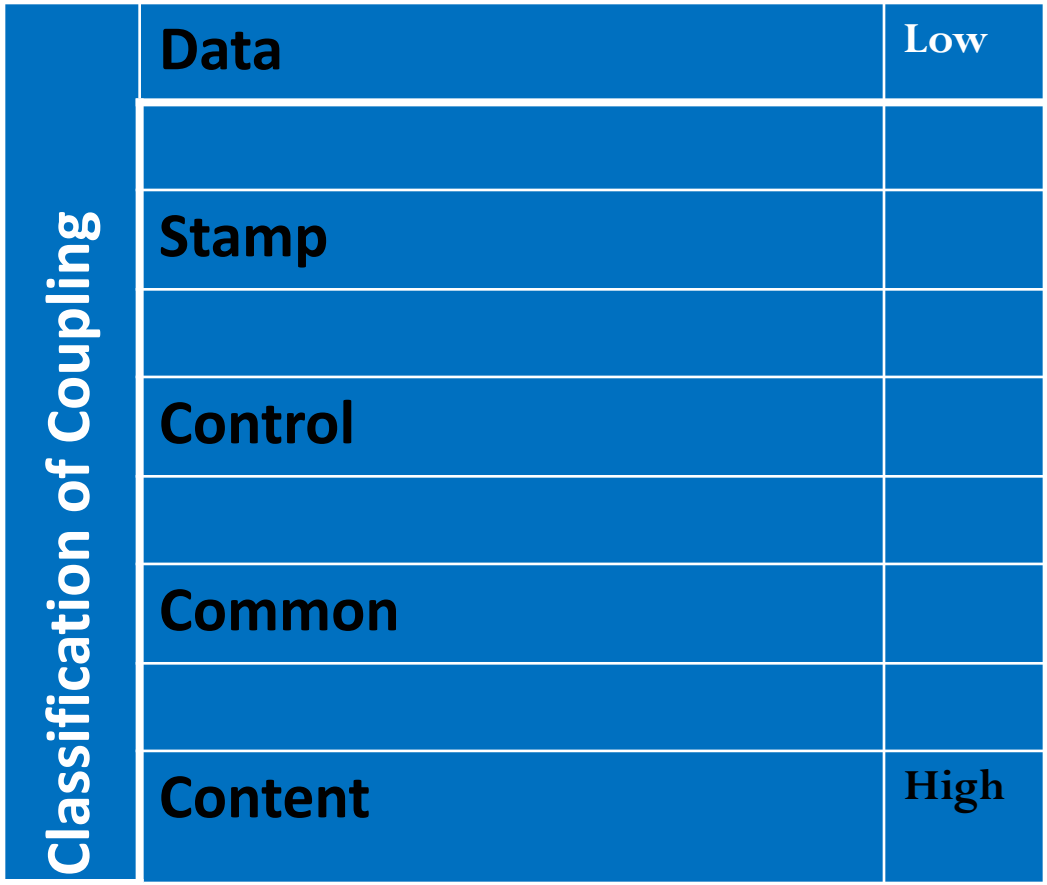
- **Procedural cohesion:**
- If the set of functions of the module are all part of a procedure (algorithm) in which certain sequence of steps have to be carried out for achieving an objective
- For Ex., the algorithm for decoding a message.
- **Communicational cohesion:**
- If **all functions** of the module **refer** to the **same data structure**
- For Ex., the set of functions defined on an array or a stack.

- **Sequential cohesion:**
- If the elements of a module form the parts of sequence, where the output from one element of the sequence is input to the next.
- For Ex., In a Transaction Processing System, the get-input, validate-input, sort-input functions are grouped into one module.
- **Functional cohesion:**
- If different elements of a module cooperate to achieve a single function.
- For Ex., A module containing all the functions required to manage employees' pay-roll exhibits functional cohesion.

Coupling

- **Coupling** between two modules is a **measure of the degree** of **interdependence** or interaction **between** the **two modules**.
- A module having high cohesion and low coupling is said to be functionally independent of other modules.
- If **two modules interchange large amounts of data**, then they are **highly interdependent**.
- The degree of coupling between two modules depends on their interface complexity.
- The interface complexity is basically determined by the number of types of parameters that are interchanged while invoking the functions of the module.

Classification of Coupling



Classification of Coupling Cont.

- **Data coupling:**
- Two modules are data coupled, if **they communicate through a parameter**.
- An example is an elementary (primal) data item passed as a parameter between two modules, e.g. an integer, a float, a character, etc.
- **Stamp coupling:**
- This is a special case (or extension) of data coupling
- Two modules (``A'' and ``B'') exhibit stamp coupling if **one passes** directly to the other a **composite data item** - such as a record (or structure), array, or (pointer to) a list or tree.
- This occurs when **ClassB** is **declared as a type** for an argument of an **operation of ClassA** **.**

- **Control coupling:**
- If **data** from **one** module **is used to direct the order of instructions** execution **in another**.
- An example of control coupling is a flag set in one module and tested in another module.
- **Common coupling:**
- Two modules are common coupled, if they **share data** through some **global data items**.
- Common coupling can **leads to uncontrolled error propagation** and **unforeseen side effects** when changes are made.

- **Content coupling:**
- Content coupling occurs when **one component secretly modifies data** that is **internal to another component.**
- This violets information hiding – a basic design concept
- Content coupling exists between two modules, if they share code.

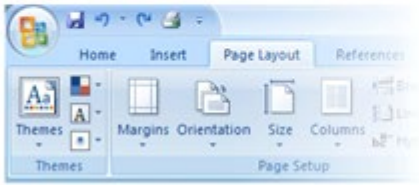
Golden Rules of design

Place the User in Control

Reduce the User's Memory Load

Make the Interface Consistent

Microsoft Word



Microsoft Excel

Microsoft Powerpoint

Place the User in Control

- During a requirements-gathering session for a major new information system, a key user was asked about.
- Following are the **design principles** that allow the **user to maintain control**:
- **Define interaction modes** in a way that **does not force a user** into unnecessary or **undesired actions**.
- Provide for **flexible interaction**.
- Allow user **interaction** to be interruptible and **undoable**.
- Streamline interaction as skill levels advance and allow the interaction to be customized.
- **Hide technical internals** from the casual user.
- Design for **direct interaction with objects** that appear on the screen.

Reduce the User's Memory Load

- The **more a user has to remember, the more error-prone** the interaction with the system will be.
- Following are the design principles that enable an interface to reduce the user's memory load:
- **Reduce demand** on **short-term memory**.
- Establish **meaningful defaults**.
- **Define shortcuts** that are intuitive.
- The **visual layout** of the interface should be **based on** a **real-world metaphor**.
- Disclose **information** in a **progressive fashion**.

Make the Interface Consistent

- The interface should **present** and acquire **information** in a **consistent fashion**.
- Following are the design principles that help make the interface consistent:
- Maintain **consistency across a family of applications**.
- If past interactive models have created user expectations, **do not make changes unless there is a compelling** (convincing) **reason** to do so.

User Interface Analysis and Design Models

- **User profile model** – Established by a software engineer • Establishes the profile of the end-users of the system
- based on age, gender, physical abilities, education, cultural background, motivation, goals, and personality.
- **Design model** – Created by a software engineer
- Derived from the analysis model of the requirements.
- Incorporates data, architectural, interface, and procedural representations of the software.
- **Implementation model** – Created by the software implementers
- Consists of the look and feel of the interface combined with all supporting information (books, videos, help files) that describe system syntax and semantics.

- **User's mental model** – Developed by the user when interacting with the application
- Often called the user's system perception.
- Consists of the image of the system; that users carry in their heads.
- The **role of the interface designer** is to merge these differences and derive a consistent representation of the interface.

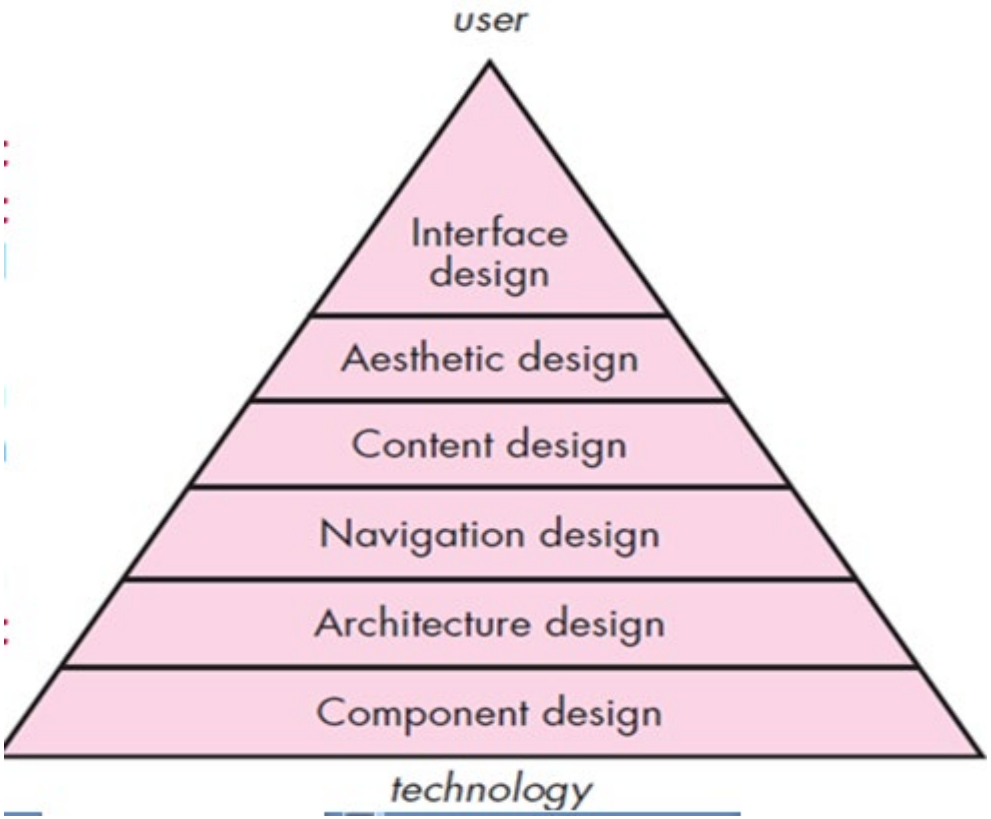
Web Application Design

- Design for WebApp encompasses technical and nontechnical activities that include:
- **Establishing the look and feel** of the WebApp
- Defining the **overall architectural structure**
- **Developing** the **content** and **functionality** that reside within the architecture
- **Planning** the **navigation** that occurs within the WebApp




Web App Interface Design

- The objectives of a WebApp interface are to:
- **Design pyramid for WebApps**

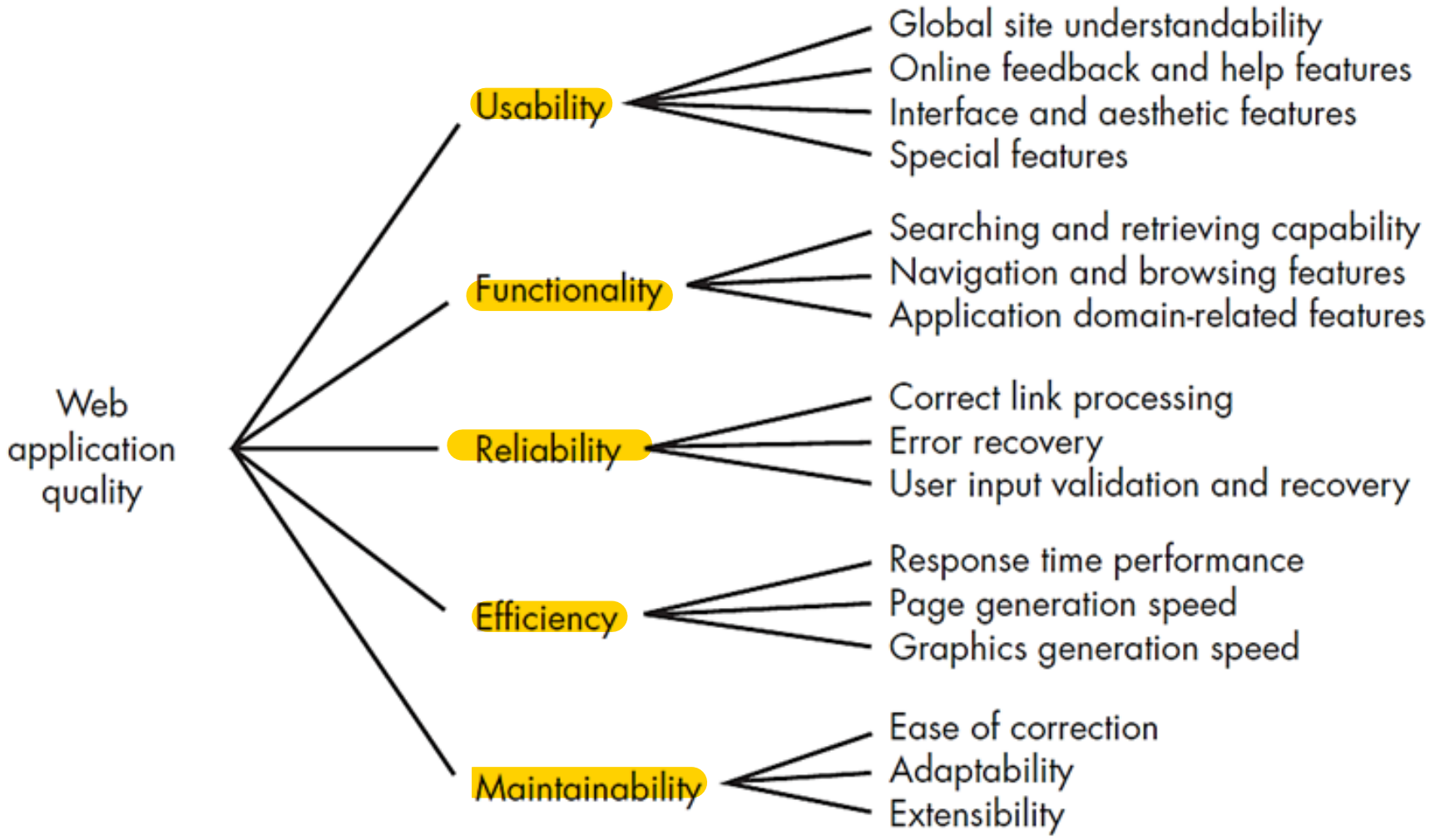


- Establish a **consistent window** into the **content** and **functionality** provided by the interface.
- **Guide the user** through a series of interactions with the WebApp.
- **Organize** the **navigation options** and **content** available to the user.

- **Interface Design**
- One of the challenges of interface design for WebApps is the **nature of the user's entry point**.
- **Aesthetic Design**
- Also called **graphic design**, is an **artistic endeavor** (offer) that **complements the technical aspects** of WebApp design.
- **Content Design**
- **Generate content and design** the representation for content to be used within a WebApp.
- **Architecture Design**
- It is tied to the goals established for a WebApp,
- the content to be presented, the users who will visit, and the navigation that has been established.

- **Navigation Design**
- **Define navigation pathways** that enable users to access WebApp content and functions.
- **Component-Level Design**
- Modern WebApps deliver increasingly sophisticated processing functions that,
- **Perform localized processing** to generate content and navigation capability in a dynamic fashion,
- Provide **computation** or **data processing capability** that are appropriate for the WebApp's business domain.
- Provide **sophisticated database query** and access.
- Establish **data interfaces with external corporate** systems.

WebApp Design Quality Requirement



References

Introduction

- https://www.youtube.com/watch?v=F440S-HibGQ
- https://www.youtube.com/watch?v=fN9IrxdxXUg
- https://www.youtube.com/watch?v=KN9Y70RxZ-o
- https://www.youtube.com/watch?v=X7EJck9XnBE
- https://www.youtube.com/watch?v=-biMMy9sNxw
- https://www.youtube.com/watch?v=xu2Sug6ztk8

Data architecture style:

- https://www.youtube.com/watch?v=TzYYG06x9e0

Assignment 5

1. What are different design concepts? Explain in details each.
2. Define coupling and cohesion. Explain different types of it.
3. Explain difference between coupling and cohesion.
4. Compare procedure oriented design and function oriented design.
5. What is User interface? Explain design issues wile designing user interface.
6. Explain design rules(Golden rules) for UI.
7. What is architectural style? Explain different architectural styles in detail.