Software Design

- Design Principles and Concepts
- Effective Modular Design
- Cohesion and Coupling
- Architectural Design

Software Design

- Design means to draw or plan something to show the look, functions and working of it.

- Once the requirements document regarding the software to be developed is presented, the phase of software design gets started.
- The requirement specification activity is considered to be purely related with the problem domain whereas design is considered as the initial phase of transforming the problem into a solution.
- In the design phase, all the relevant entities such as the customer, business requirements and technical considerations collaborate to formulate a product or a system.
- In the design process, there are several elements such a set of principles, concepts and practices, which help a software engineer to model the system or product which is to be built.
- The design model is assessed for quality and reviewed before generation of code and execution of tests.
- The design model gives detailed information regarding software data structures, architecture, interfaces, and components which are necessary to employ the system.

Software Design …. Contd.

- Software design sits at the technical core of software engineering and is applied regardless of the software process model that is used.
- The design task produces a data design, an architectural design, an interface design, and a component design.

A. 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 are defined in the entity relationship diagram.
- Part of data design may occur in combination with the design of software architecture.

Software Design …. Contd.

B. 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.

C. Interface Design:

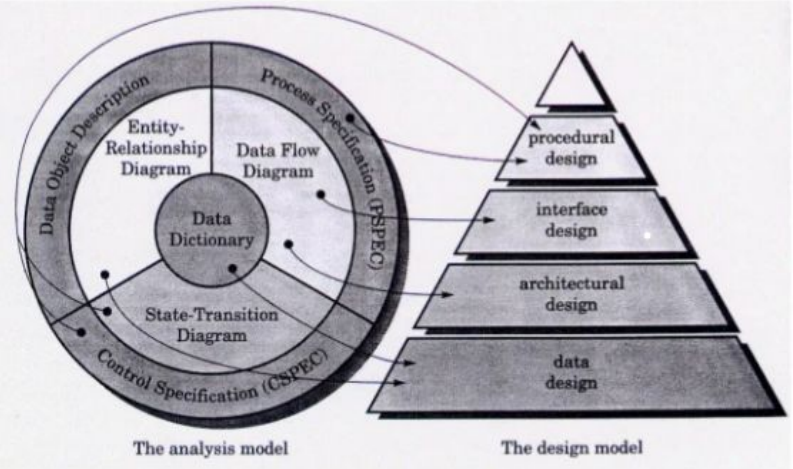
- The interface design describes how the software communicates within itself, with systems that interoperate with it, and with humans who use it.
- An interface implies a flow of information (e.g. data and/ or control) and a specific type of behavior.

Software Design …. Contd.

- Hence, data and flow control diagrams provide much of the information required for interface design.

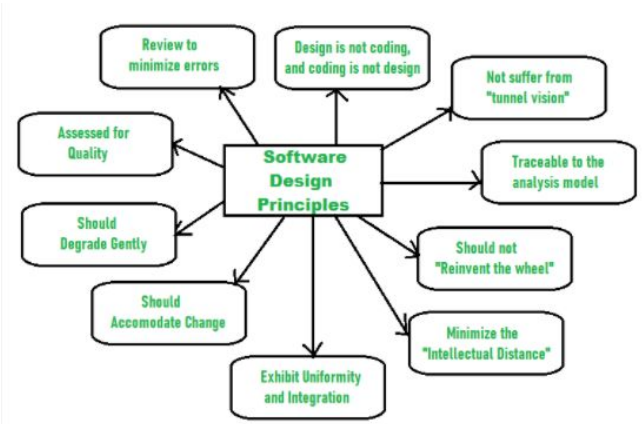
E. Component-level Design:

- The component-level design transforms structural elements of the software architecture into a procedural description of software components.



Design Principles

- Software Design is also a process to plan or convert the software requirements into a step that are needed to be carried out to develop a software system.
- There are several principles that are used to organize and arrange the structural components of Software design.
- Software Designs in which these principles are applied affect the content and the working process of the software from the beginning.



Design Principles …. Contd.

- **Principles Of Software Design :**

1.Should not suffer from "Tunnel Vision" –

- While designing the process, it should not suffer from "tunnel vision" which means that is should not only focus on completing or achieving the aim but on other effects also.

1.Traceable to analysis model –

- The design process should be traceable to the analysis model which means it should satisfy all the requirements that software requires to develop a high-quality product.

3.Should not "Reinvent The Wheel" –

- The design process should not reinvent the wheel that means it should not waste time or effort in creating things that already exist. Due to this, the overall development will get increased.

Design Principles …. Contd.

4. **Minimize Intellectual distance –**
   - The design process should reduce the gap between real-world problems and software solutions for that problem meaning it should simply minimize intellectual distance.

5. **Exhibit uniformity and integration** –
   - The design should display uniformity which means it should be uniform throughout the process without any change. Integration means it should mix or combine all parts of software i.e. subsystems into one system.

6. **Accommodate change** –
   - The software should be designed in such a way that it accommodates the change implying that the software should adjust to the change that is required to be done as per the user's need.

Design Principles …. Contd.

7. **Degrade gently –**
   - The software should be designed in such a way that it degrades gracefully which means it should work properly even if an error occurs during the execution.

8. **Assessed or quality** –
   - The design should be assessed or evaluated for the quality meaning that during the evaluation, the quality of the design needs to be checked and focused on.

9. **Review to discover errors** –
   - The design should be reviewed which means that the overall evaluation should be done to check if there is any error present or if it can be minimized.

Design Principles …. Contd.

10. **Design is not coding and coding is not design** –
    - Design means describing the logic of the program to solve any problem and coding is a type of language that is used for the implementation of a design.

Design Concepts

- Software Design is the process to transform the user requirements into some 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 the aim of this phase is to transform the SRS document into the design document.

- The following items are designed and documented during the design phase:
  - Different modules required.
  - Control relationships among modules.
  - Interface among different modules.
  - Data structure among the different modules.
  - Algorithms required to implement among the individual modules.

Objectives of Software Design

1. **Correctness:**
   A good design should be correct i.e. it should correctly implement all the functionalities of the system.

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

3. **Understandability:**
   A good design should be easily understandable, for which it should be modular and all the modules are arranged in layers.

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

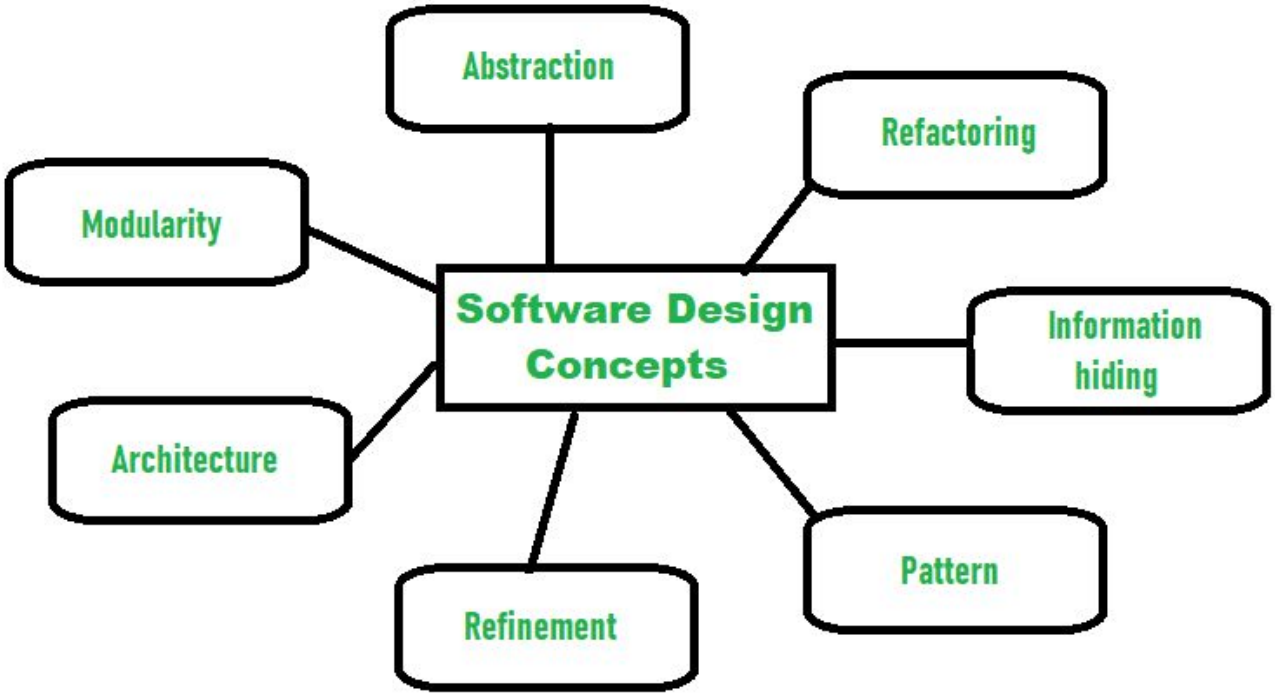
5. **Maintainability:**
   A good software design should be easily amenable to change whenever a change request is made from the customer side.

Software Design Concepts

- ✔ Concepts are defined as a principal idea or invention that comes into our mind or in thought to understand something.
- ✔ The software design concept simply means the idea or principle behind the design.
    - It describes how you plan to solve the problem of designing software, the logic, or thinking behind how you will design software.
    - It allows the software engineer to create the model of the system or software or product that is to be developed or built.
    - The software design concept provides a supporting and essential structure or model for developing the right software.

Software Design Concepts ….

- ✔ Some of the concepts of software design are:



Software Design Concepts ….

The following points should be considered while designing Software:

a) Abstraction - hide Irrelevant data

- Abstraction simply means to hide the details to reduce complexity and increase efficiency or quality.
- Different levels of Abstraction are necessary and must be applied at each stage of the design process so that any error that is present can be removed to increase the efficiency of the software solution and to refine the software solution.
- The solution should be described in broad ways that cover a wide range of different things at a higher level of abstraction and a more detailed description of a solution of software should be given at the lower level of abstraction.

Software Design Concepts ….

b) 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.
- It is necessary to divide the software into components known as modules because nowadays there are different software available like Monolithic software that is hard to grasp for software engineers.
- So, modularity in design has now become a trend and is also important.
- If the system contains fewer components then it would mean the system is complex which requires a lot of effort (cost) but if we are able to divide the system into components then the cost would be small.

Software Design Concepts ….

c) 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.

d) Refinement - removes impurities

- Refinement simply means to refine something to remove any impurities, if present and increase the quality.
- The refinement concept of software design is actually a process of developing or presenting the software or system in a detailed manner that means to elaborate a system or software.
- Refinement is very necessary to find out any error if present and then to reduce it.

Software Design Concepts ….

e) Pattern - a repeated form

- The 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.

f) 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 not be accessed by any other modules.

Software Design Concepts ….

g) **Refactoring - reconstruct something**

- Refactoring simply means reconstructing something in such a way that it does not affect the behavior of any other features.
- Refactoring in software design means reconstructing the design to reduce complexity and simplify it without affecting the behavior or its functions.

Effective Modular Design

- Any software comprises of many systems which contains several sub-systems and those sub-systems further contains their sub-systems.
- So, designing a complete system in one go comprising of each and every required functionality is a hectic work and the process can have many errors because of its vast size.

- Thus, in order to solve this problem the developing team breaks down the complete software into various modules.
  - ✔ A module is defined as the unique and addressable components of the software which can be solved and modified independently without disturbing (or affecting in very small amount) other modules of the software.
  - ✔ Thus every software design should follow modularity.

Effective Modular Design ….

- The process of breaking down a software into multiple independent modules, where each module is developed separately is called Modularization.

- Effective modular design can be achieved if the partitioned modules are separately solvable, modifiable as well as compliable.
  - Here, separate compliable modules means that after making changes in a module there is no need of recompiling the whole software system.

- In order to build a software with effective modular design there is a factor "Functional Independence" which comes into play.
  - The meaning of Functional Independence is that a function is atomic in nature so that it performs only a single task of the software without or with least interaction with other modules.
  - It is considered as a sign of growth in modularity i.e., presence of larger functional independence results in a software system of good design and design further affects the quality of the software.

Effective Modular Design ….

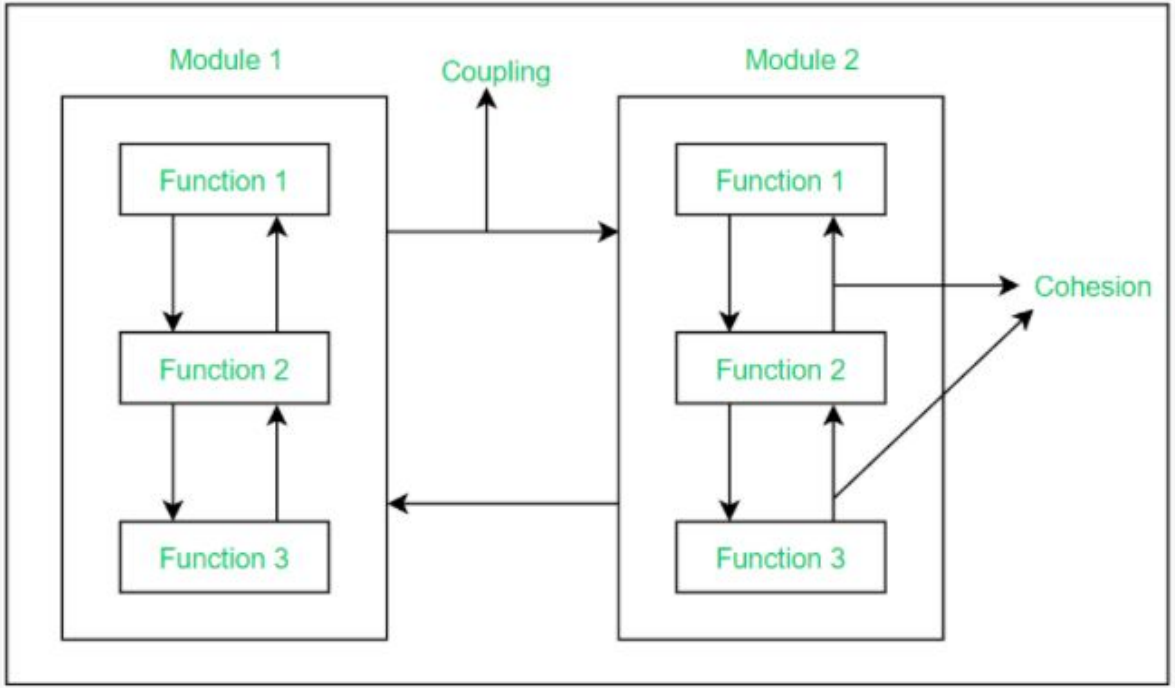
Benefits of Independent modules/functions in a software design:

- ❑As the functionalities of the software are broken down into atomic levels, the developers get a clear requirement of each and every function.
    - Hence designing of the software becomes easy and error free.

- ❑As the modules are independent, they have limited or almost no dependency on other modules.
    - So, making changes in a module without affecting the whole system is possible in this approach.
    - Error propagation from one module to another and further in whole system can be neglected and it saves time during testing and debugging.

Effective Modular Design ….

Independence of modules of a software system can be measured using two criteria : Cohesion, and Coupling.

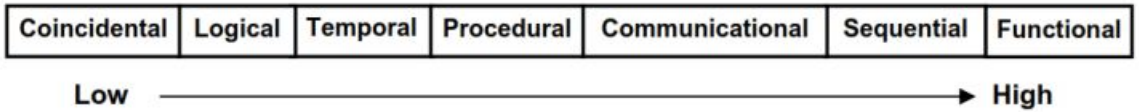


Cohesion

Cohesion:

- Cohesion defines the degree to which components of a system are related to each other.
  - It measures the strength of relationship between two components of system (various functions within a module).
  - It is an indication of relative functional strength of a module.
  - A cohesive module performs a single task, requiring little interaction with the other components in other parts of the program.
  - It is concept of intra-module.
  - It is the indication of relationship within the module.

- It is of 7 types which are listed below in the order of low to high cohesion:



Types of Cohesion

i. Co-incidental cohesion:

- A module is said to have coincidental cohesion, if it performs a set of tasks that relate to each other very loosely, if at all.
- In this case, the module contains 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.
  - ✔ The elements are not related(unrelated).
  - ✔ The elements have no conceptual relationship other than location in source code.
- It is accidental and the worst form of cohesion.
- Ex-
  - print next line and reverse the characters of a string in a single component.
  - In a transaction processing system (TPS), the get-input, print error, and summarize members functions are grouped together in one module.

Types of Cohesion …. Contd.

ii. Logical cohesion:

- A module is said to be logically cohesive, if all the elements of the module perform similar operations, e.g. error handling, data input, data output, etc.
- The elements are logically related and not functionally.
- Here, all elements of module contribute to same system operation.
- Ex-
  - A component reads inputs from tape, disk, and network. All the code for these functions is in the same component. Operations are related, but the functions are significantly different.
  - A set of print functions generating different output reports are arranged into a single module.

Types of Cohesion …. Contd.

iii. Temporal cohesion:

- When a module contains functions that are related by the fact all the functions must be executed in the same time span, the module is said to exhibit temporal cohesion.
- Ex-
  - The set of functions responsible for initialization, start-up, shut down of some process, etc. exhibit temporal cohesion.
    - This avoids the problem of passing the flag.

iv. Procedural cohesion:

- A module is said to possess 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.

Types of Cohesion …. Contd.

- Elements of procedural cohesion ensure the order of execution.
- Actions are still weakly connected and unlikely to be reusable.
- Ex-
  - calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.
  - algorithm for decoding the message

vi. Communicational cohesion:

- A module is said to have communicational cohesion, if the set of functions of the module refer to or update the same data structure.
  - ✔ Two elements operate on the same input data or contribute towards the same output data (when elements of module perform different functions but each function accepts same input and generates same output).

Types of Cohesion …. Contd.

- Ex-
  - update record in the database and send it to the printer.
  - the set of functions defined on an array or a stack.

vii. Sequential cohesion:

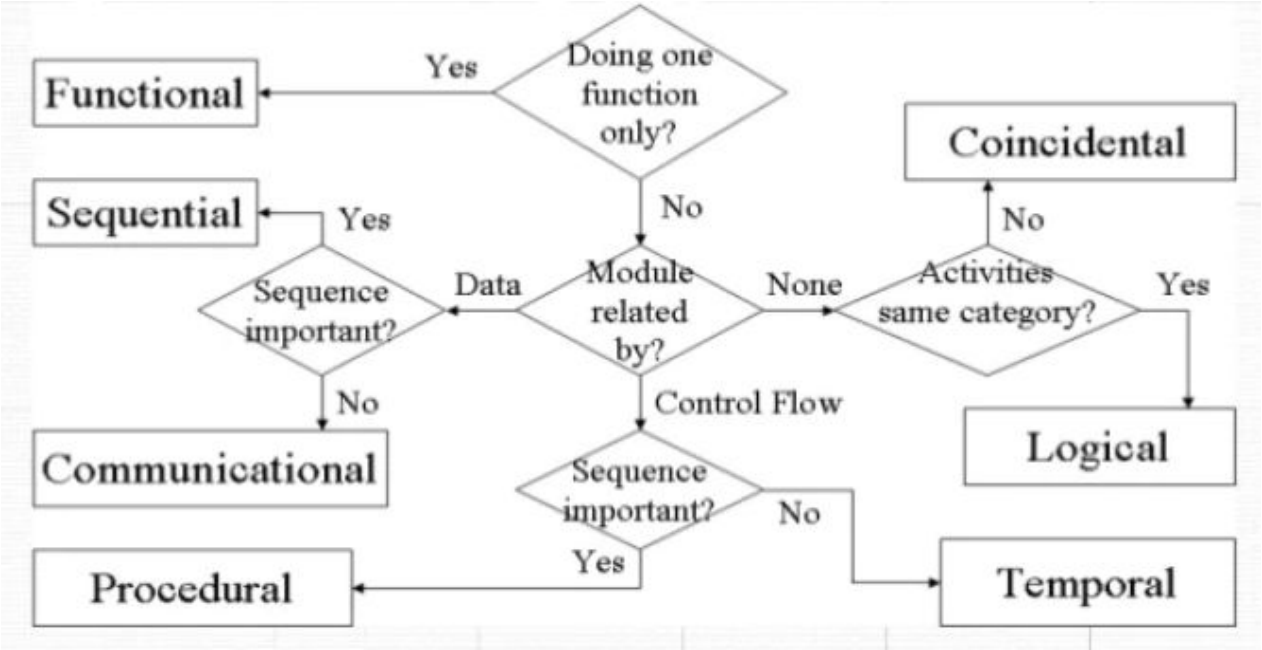
- A module is said to possess sequential cohesion, if the elements of a module form the parts of sequence, where the output from one element of the sequence is input to the next.
- An element outputs some data that becomes the input for other element, i.e., data flow between the parts.
- It occurs naturally in functional programming languages.
- Ex-
  - In a Transaction Processing System (TPS), the get-input, validate-input, sort-input functions are grouped into one module.

Types of Cohesion …. Contd.

vii. Functional cohesion:

- Functional cohesion is said to exist, if different elements of a module cooperate to achieve a single function.
- Every essential element for a single computation is contained in the component.
- A functional cohesion performs the task and functions.
- It is an ideal situation.
- Ex-
  - A module containing all the functions required to manage employee's pay-roll.

Types of Cohesion …. Contd.



Types of Cohesion …. Contd.

- **Advantages:**
  - High cohesion among system components results in better program design.
  - High cohesion components can be easily reused.
  - High cohesion components are more reliable.

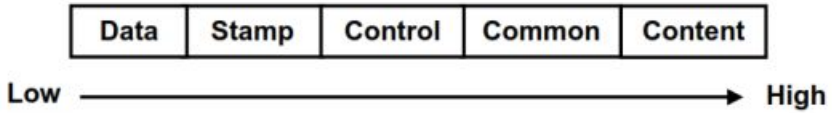
- **Disadvantages:**
  - Low cohesion components are difficult to maintain.
  - Low cohesion components cannot be reused.
  - Low cohesion components are difficult to understand.
  - Low cohesion components are less reliable.

Coupling

Coupling:

- Coupling is a measure of strength in relationship between various modules within a software.
  - It is an indication of the relative interdependence among modules.
  - Coupling depends on the interface complexity between the modules, the point at which entry or reference is made to a module, and what data passes across the interface.
  - It is concept of inter-module.

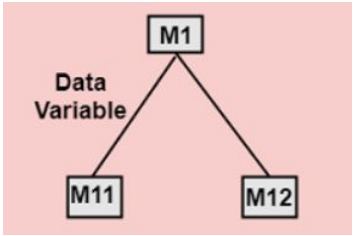
- It is of 5 types which are listed below in the order of low to high coupling:



Types of Coupling

i. **Data Coupling:**

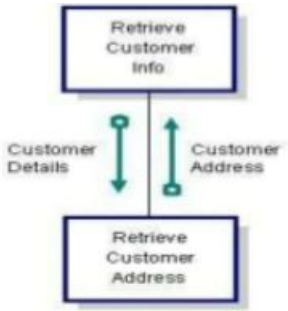
- Two modules are data coupled, if they communicate through a parameter.
- It occurs between two components of a system when they pass data to each other by parameter using argument list and each element in argument list is used.
  - ✔ This data item should be problem related and not used for the control purpose.
- A component becomes difficult to maintain if too many parameters are passed.
- Ex-
  - An elementary data item passed as a parameter between two modules, e.g. an integer, a float, a character, etc.
  - Customer-billing system



Types of Coupling …. Contd.

ii. Stamp Coupling:

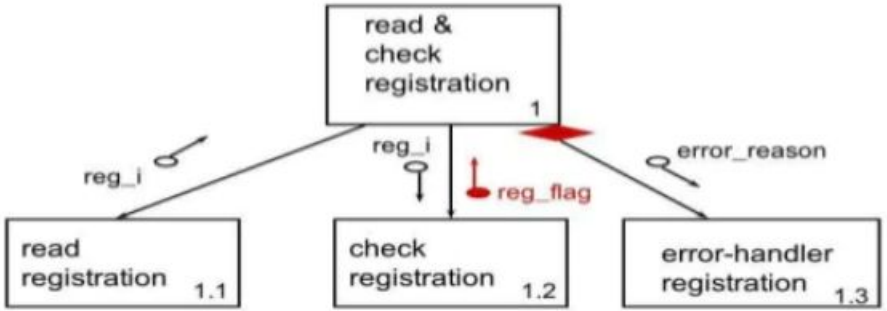
- Two modules are stamp coupled, if they communicate using a composite data item such as a record in PASCAL, or a structure in C.
- The complete data structure is passed from one module to another module.
- Ex-
  - passing structure variable in C or object in C++ language to a module.



Types of Coupling …. Contd.

iii. Control Coupling:

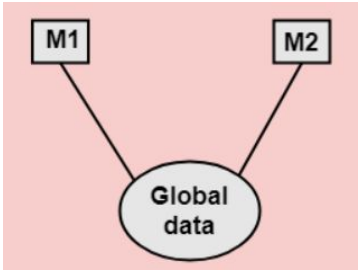
- Control coupling exists between two modules, if data from one module is used to direct the order of instructions execution in another.
  - ✔ One module decides the function of the other module or changes its flow of execution.
- Ex-
  - A flag set in one module and tested in another module.



Types of Coupling …. Contd.

iv. Common Coupling:

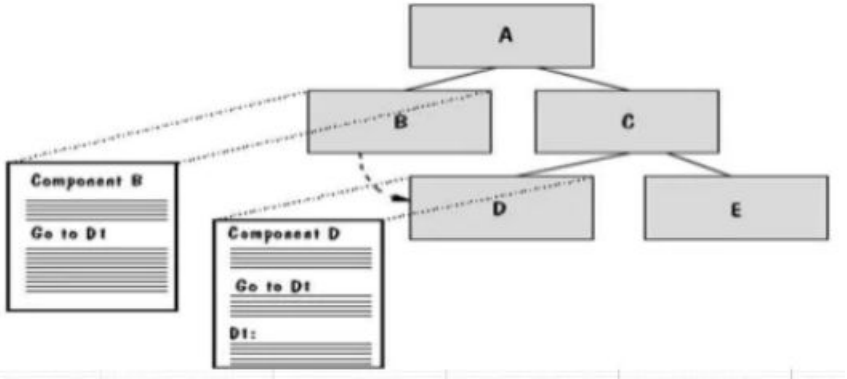
- Two modules are common coupled, if they share data through some global data items.
- When multiple modules have read and write access to some global data, it is called common or global coupling.
  - ✔ The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change.
  - ✔ So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses and reduced maintainability.
- Ex-
  - Modules sharing data such as global data structures



Types of Coupling …. Contd.

v. Content Coupling:

- Content coupling exists between two modules, if they share code.
  - ✔ One module can modify the data of another module or control flow is passed from one module to the other module.
- Ex-
  - a branch from one module into another module.



Types of Coupling …. Contd.

vi. **External Coupling:**

- External Coupling arises when two modules share an externally imposed data format, communication protocols, or device interface.
- This is related to communication to external tools and devices.
- The modules depend on other modules, external to the software being developed or to a particular type of hardware.
- Ex-
  - protocol, external file, device format, etc.

vii. **Message Coupling:**

- Message coupling occurs between two components of system when those components communicate with each other via message passing.
- In this coupling, the components are independent of each other.

Types of Coupling …. Contd.

- **Advantages:**
  - Low coupling components do not force ripple effect to other components.
  - Low coupled components can be easily reused.
  - With low coupling among components, system can be built faster.

- **Disadvantages:**
  - High coupling components are difficult to understand.
  - High coupling components force changes in other components, if one component changes.
  - High coupling components slow down the development process of a system.

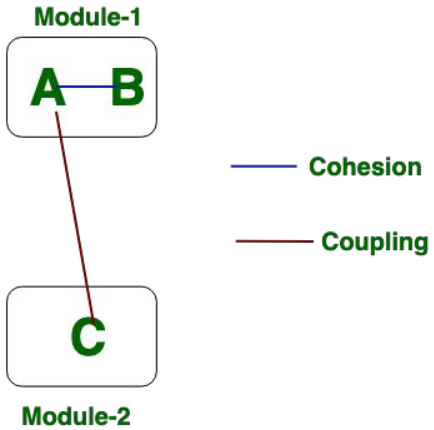
- A good software design requires high cohesion and low coupling.

Cohesion Vs Coupling

Cohesion	Coupling
Cohesion is the concept of intra module.	Coupling is the concept of inter module.
Cohesion is the indication of the relationship within the module.	Coupling is the indication of the relationships between modules.
Cohesion represents the functional strength of modules.	Coupling represents the independence among modules.
High cohesion is good for software.	High coupling is avoided for software.
In cohesion, module focuses on the single thing.	In coupling, modules are connected to the other modules.
While designing you should strive for high cohesion i.e. a cohesive component/ module focus on a single task (i.e., single-mindedness) with little interaction with other modules of the system.	While designing you should strive for low coupling i.e. dependency between modules should be less.

Cohesion Vs Coupling ….

Cohesion	Coupling
Cohesion is the kind of natural extension of data hiding, for example, a class having all members visible with a package having default visibility.	Making private fields, private methods and non-public classes provide loose coupling.
Cohesion shows the module's relative functional strength.	Coupling shows the relative independence among the modules.



Architectural Design

- Architectural design represents the structure of data and program components that are required to build a computer-based system.
- It considers the architectural style that the system will take, the structure and properties of the components that constitute the system, and the interrelationships that occur among all architectural components of a system.
- Representations of a software architecture are an enabler for communication between all parties (stakeholders) interested in the development of a computer-based system.
- The architecture highlights early design decisions that will have a reflective impact on all software engineering work that follows and, as important, on the ultimate success of the system as an operational entity.
- Architecture ‘constitutes a relatively small, intellectually graspable model of how the system is structured and how its components work together’.

Software Architecture

- Software architecture of a program or computing system is the structure or structures of the system which comprises of -
  - The software components
  - The externally visible properties of those components
  - The relationships among the components

- Software architectural design represents the structure of the data and program components that are required to build a computer-based system.

- An architectural design model is transferable
  - It can be applied to the design of other systems
  - It represents a set of abstractions that enable software engineers to describe architecture in predictable ways.

Why Architecture?

- The architecture is not the operational software. Rather, it is a representation that enables a software engineer to:
  i. Analyze the effectiveness of the design in meeting its stated requirements.
  ii. Consider architectural alternatives at a stage when making design changes is still relatively easy, and
  iii. Reduce the risks associated with the construction of the software.

- Following are the reasons for the importance of software architecture:-
  1. The representation of software architecture allows the communication between all stakeholder and the developer.
  2. The architecture focuses on the early design decisions that impact on all software engineering work and it is the ultimate success of the system.
  3. The software architecture composes a small and intellectually graspable model.
  4. This model helps the system for integrating the components using which the components are work together.

Architectural Styles

- The software that is built for computer-based systems also exhibits one of many architectural styles.
- Each style describes a system category that encompasses:
  a) A set of components (e.g. a database, computational modules) that perform a function required by a system.
  b) A set of connectors that enable “communication, coordination and cooperation” among components.
  c) Constraints that define how components can be integrated to form the system.
  d) Semantic models that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

- The following are the architectural styles …..

Data Centered Architectural Style

- A data store (e.g. a file or database) will reside at the center of this architecture and is accessed frequently by the other components that update, add, delete or modify the data present within the store.
- The client software access a central repository. Variation of this approach are used to transform the repository into a blackboard when data related to client or data of interest for the client change the notifications to client software.
- In some cases, the repository is passive.
    - i.e. the client assesses the data independent of any changes to the data or the actions of other client software.
- This data-centered architecture will promote integrability. This means that the existing components can be changed and new client components can be added to the architecture without the permission or concern of other clients.

Data Centered Architectural Style….

- Data can be passed among clients using blackboard mechanism.
- There are two types of components −
  - ✔ A central data structure or data store or data repository, which is responsible for providing permanent data storage. It represents the current state.
  - ✔ A data accessor or a collection of independent components that operate on the central data store, perform computations, and might put back the results.

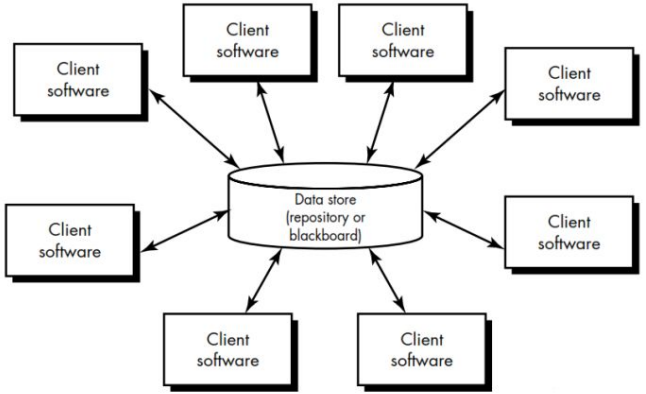


Figure: Data-centered architecture

Data Centered Architectural Style….

- **Advantages:**
  - Data repository is not dependent of the clients.
  - Clients work independent of each other.
  - Provides data integrity, backup and restore features.
  - Provides scalability and reusability of agents as they do not have direct communication with each other (adding and modification of clients is easy).
  - Reduces overhead of transient data between software components.

- **Disadvantages:**
  - It is more vulnerable to failure and data replication or duplication is possible.
  - High dependency between data structure of data store and its agents.
  - Changes in data structure highly affect the clients.
  - Evolution of data is difficult and expensive.
  - Cost of moving data on network for distributed data.

Data Flow Architectural Style

- This architecture is applied when input data are to be transformed through a series of computational or manipulative components into output data.
- A pipe and filter pattern has 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.
  - ✔ They are designed to expect data input of a certain form, and produce data output (to the next filter) of a specified form.
- However, the filter does not require knowledge of the working of its neighboring filters.
- If the data flow degenerates into a single line of transforms, then it is termed as batch sequential. This structure accepts the batch of data and then applies a series of sequential components to transform it.

Data Flow Architectural Style

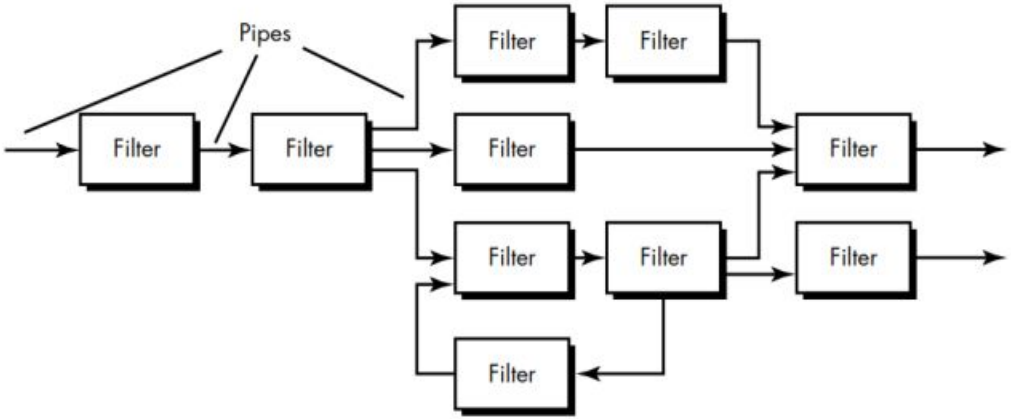
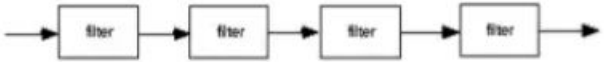
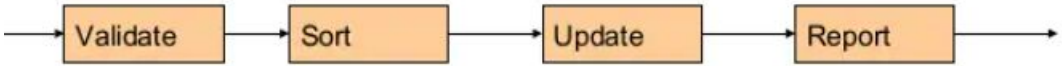


Figure: Data-flow architectures





Data Flow Architectural Style….

- **Advantages:**
  - Provides concurrency and high throughput for excessive data processing.
  - Provides reusability and simplifies system maintenance.
  - Provides modifiability and low coupling between filters.
  - Provides simplicity by offering clear divisions between any two filters connected by pipe.
  - Provides flexibility by supporting both sequential and parallel execution.

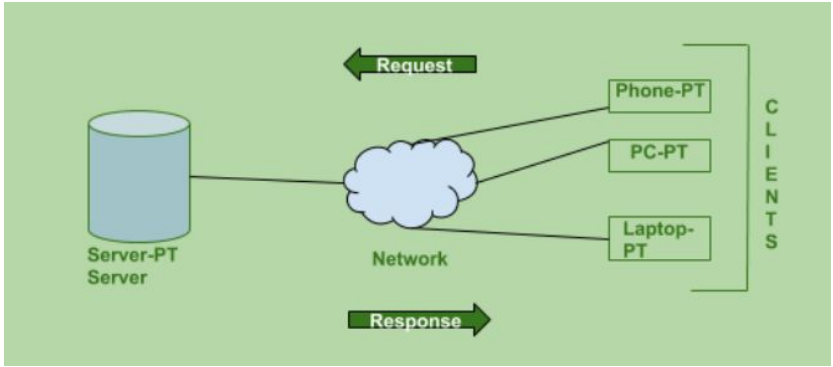
- **Disadvantages:**
  - Not suitable for dynamic interactions.
  - A low common denominator is needed for transmission of data in ASCII formats.
  - Overhead of data transformation between filters.
  - Does not provide a way for filters to cooperatively interact to solve a problem.
  - Difficult to configure this architecture dynamically.

Distributed Architectural Style

- The Client-server model is a distributed application structure that partitions task or workload between the providers of a resource or service, called servers, and service requesters called clients.
- In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and deliver the data packets requested back to the client.
- Clients do not share any of their resources.
- Examples of Client-Server Model are Email, World Wide Web, etc.
- **Client:**
  - When we talk the word Client, it mean to talk of a person or an organization using a particular service.
  - Similarly in the digital world a Client is a computer (Host) i.e. capable of receiving information or using a particular service from the service providers (Servers).

Distributed Architectural Style ….

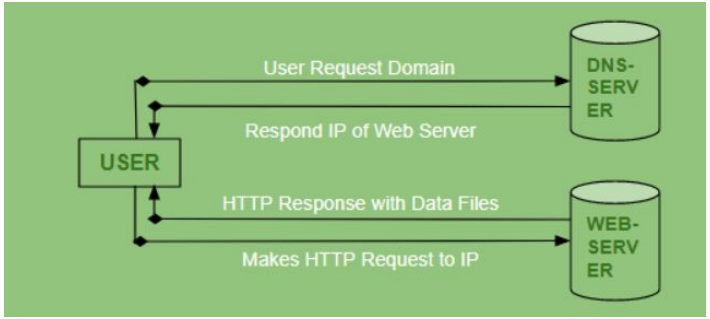
- **Servers:**
  - Similarly, when we talk the word Servers, it means a person or medium that serves something.
  - Similarly in this digital world a Server is a remote computer which provides information (data) or access to particular services.
- So, its basically the Client requesting something and the Server serving it as long as its present in the database.



Distributed Architectural Style ….

The browser follows few steps to interact with the server:

i. User enters the URL(Uniform Resource Locator) of the website or file. The Browser then requests the DNS(DOMAIN NAME SYSTEM) Server.
ii. DNS Server lookup for the address of the WEB Server.
iii. DNS Server responds with the IP address of the WEB Server.
iv. Browser sends over an HTTP/HTTPS request to WEB Server's IP (provided by DNS server).
v. Server sends over the necessary files of the website.
vi. Browser then renders the files and the website is displayed. This rendering is done with the help of DOM (Document Object Model) interpreter, CSS interpreter and JS Engine collectively known as the JIT or (Just in Time) Compilers.



Distributed Architectural Style ….

- **Advantages:**
  - Centralized system with all data in a single place.
  - Cost efficient, requires less maintenance cost and data recovery is possible.
  - The capacity of the Client and Servers can be changed separately.

- **Disadvantages:**
  - Clients are prone to viruses, Trojans and worms if present in the Server or uploaded into the Server.
  - Server are prone to Denial of Service (DOS) attacks.
  - Data packets may be spoofed or modified during transmission.
  - Phishing or capturing login credentials or other useful information of the user are common and MITM(Man in the Middle) attacks are common.

Peer-to-Peer Architecture

- The peer to peer computing architecture contains nodes that are equal participants in data sharing.
- All the tasks are equally divided between all the nodes.
- The nodes interact with each other as required as share resources.
- The different characteristics of peer to peer networks are as follows −
  - Peer to peer networks are usually formed by groups of a dozen or less computers.
    - These computers all store their data using individual security but also share data with all the other nodes.
  - The nodes in peer to peer networks both use resources and provide resources.
    - So, if the nodes increase, then the resource sharing capacity of the peer to peer network increases.
    - This is different than client server networks where the server gets overwhelmed if the nodes increase.

Peer-to-Peer Architecture

- Since nodes in peer to peer networks act as both clients and servers, it is difficult to provide adequate security for the nodes.
  - This can lead to denial of service attacks.
- Most modern operating systems such as Windows and Mac OS contain software to implement peer to peer networks.

Advantages:

- Each computer in the peer to peer network manages itself.
  - So, the network is quite easy to set up and maintain.
- In the client server network, the server handles all the requests of the clients.
  - This provision is not required in peer to peer computing and the cost of the server is saved.

Peer-to-Peer Architecture

- It is easy to scale the peer to peer network and add more nodes.
  - This only increases the data sharing capacity of the system.
- None of the nodes in the peer to peer network are dependent on the others for their functioning.

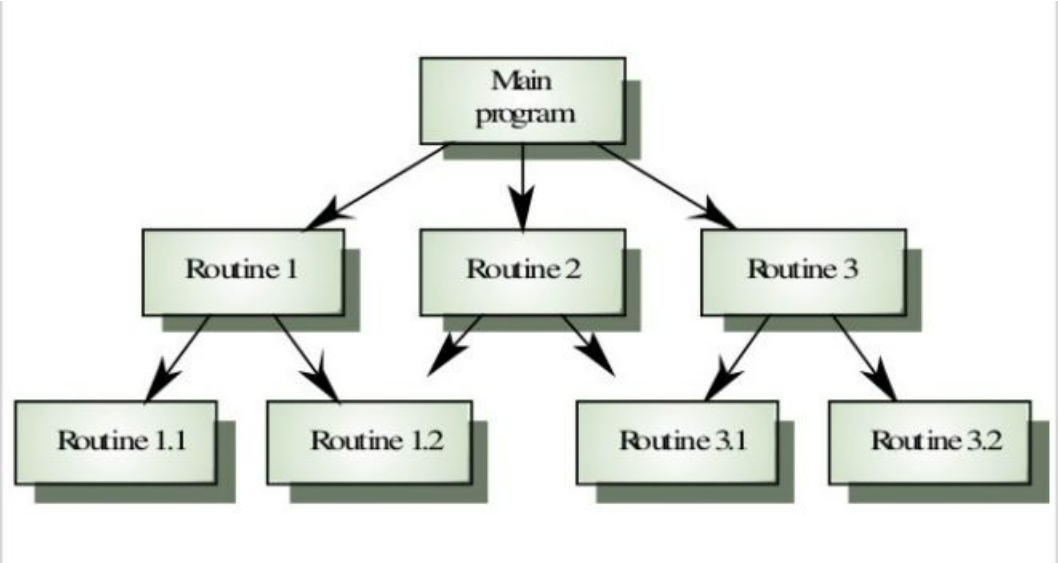
Disadvantages:

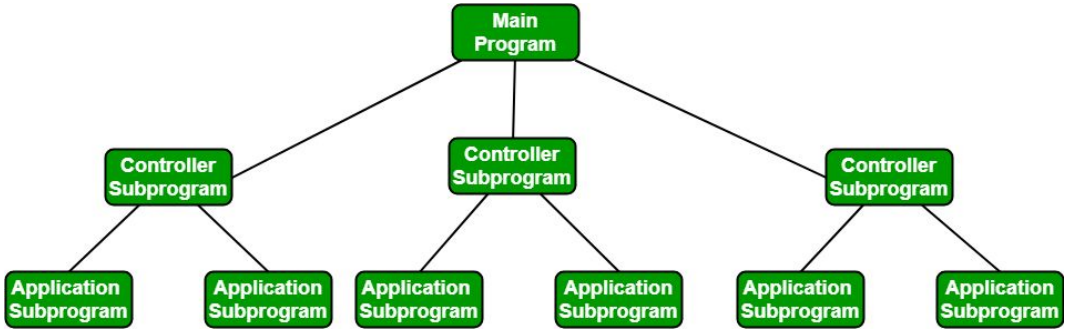
- It is difficult to backup the data as it is stored in different computer systems and there is no central server.
- It is difficult to provide overall security in the peer to peer network as each system is independent and contains its own data.

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:
  - A. Main program/ sub program architectures
    - This classic program structure decomposes function into a control hierarchy where a “main” program invokes a number of program components.
    - The program components in turn may invoke still other components.
  - B. Remote procedure call architecture:
    - The components of main program/ sub program architecture are distributed among multiple computers on a network.
    - The aim is to increase the performance.

Call and Return Architecture ….





Call and Return Architecture ….

- **Advantages:**
  - Modularity:
    - subroutines can be replaced as long as interface semantics are unaffected

- **Disadvantages:**
  - Usually fails to scale
  - Inadequate attention to data structures
  - Effort to accommodate new requirement - unpredictable
  - Relation to programming languages/environments
    - Traditional programming languages: BASIC, Pascal, C…

Object Oriented Architecture

- This architecture is the latest version of call-and-return architecture.
- It consist of the bundling of data and methods.
- The components of a system encapsulate data and the operations that must be applied to manipulate the data.
- The coordination and communication between the components are established via the message passing.
- Object-Oriented architecture views a system as a series of cooperating objects, instead of a set of routines or procedural instructions.
- It is a significant methodology for the development of any software.

Object Oriented Architecture …..

- **Advantages:**
  - It maps the application to real world objects for making it more understandable.
  - It is easy to maintain and improves the quality of the system due to program reuse.
  - This architecture provides reusability through polymorphism and abstraction.
  - It has ability to manage the errors during execution. (Robustness)
  - It has ability to extend new functionality and does not affected on the system.
  - It improves testability through encapsulation.
  - Object-Oriented architecture reduces the development time and cost.

Object Oriented Architecture …..

- **Disadvantages:**
  - It has difficulty to determine all the necessary classes and objects required for a system.
  - It is difficult to complete a solution within estimated time and budget because object-oriented architecture offers new kind of project management.
  - This methodology do not lead to successful reuse on a large scale without an explicit reuse procedure.

Layered Architecture

- A number of different layers are defined with each layer performing a well-defined set of operations.
  - ✔ Each layer will do some operations that becomes closer to machine instruction set progressively.
- At the outer layer, components will receive the user interface operations and
- At the inner layers, components will perform the operating system interfacing(communication and coordination with OS)
- Intermediate layers provide utility services and application software functions.

Layered Architecture …..

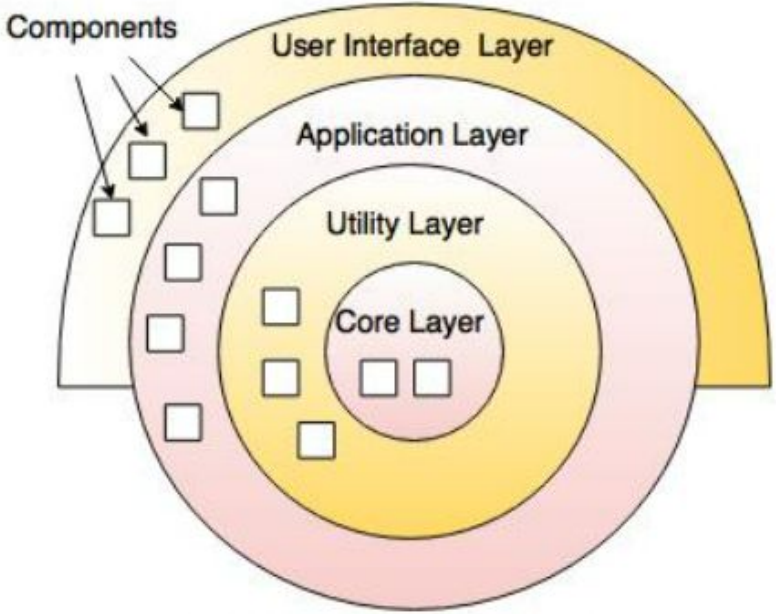


Fig.- Layered Architecture

Layered Architecture ….

- **Advantages:**
  - It provides modularity and clear interfaces.
  - Implementation simplicity, maintainability, flexibility and scalability are maintained
  - It supports portability
  - It provides robustness and preserves stability
  - It is a time tested approach

- **Disadvantages:**
  - Problem of data overhead and processing, due to the duplication of functionality
  - The more layers you have, the more risks you have for things to breakdown or data to get lost
  - Various issues occur regarding higher layer Vs lower layers
  - Results in complex exploitation of user-intensive applications
  - This leads to sluggish operation modes on various applications

Data Design

- Data design is considered both at architectural level as well as component level.

I. Data design at Architectural Level:

- The challenge is extract useful information from the data environment, particularly when the information desired is cross-functional.
- To solve this challenge, the business IT community has developed data mining techniques, also called knowledge discovery in database (KDD), that navigate through existing databases in an attempt to extract appropriate business-level information.
- However, the existence of multiple databases, their different structures, and the degree of detailed contained with the databases, and many other factors make data mining difficult within an existing database environment.

Data Design …. Contd.

- An alternative solution, called a data warehouse, adds on additional layer to the data architecture.
  - A data warehouse is a separate data environment that is not directly integrated with day-to-day applications that encompasses all data used by a business.

III. Data design at Component Level:

- At the component level, data design focuses on specific data structures required to realize the data objects to be manipulated by a component.
  - Refine data objects and develop a set of data abstractions.
  - Implement data object attributes as one or more data structures.
  - Review data structures to ensure that appropriate relationships have been established.

Data Design …. Contd.

- Set of principles for data specification:
  - The systematic analysis principles applied to function and behavior should also be applied to data.
  - All data structures and the operations to be performed on each should be identified.
  - A data dictionary should be established and used to define both data and program design.
  - Low level data design decisions should be deferred until late in the design process.
  - The representation of data structures should be known only to those modules that must make direct use of the data contained within the structure.
  - A library of useful data structures and the operations that may be applied to them should be developed.
  - A software design and programming language should support the specification and realization of abstract data types.

Component Level Design

- Component-level design, also called procedural design, occurs after data, architectural, and interface designs have been established.
- Component-level design defines the data structures, algorithms, interface characteristics, and communication mechanisms allocated to each software component.
- 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.
- Component is a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces.
- The approach of Component level design is classified into two parts:
  - Function Oriented approach
  - Object Oriented approach

Function Oriented Approach

- Following are the salient features of a typical function-oriented 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.
     - Ex.
       - » Consider a function create new library member which essentially creates the record for a new member, assigns a unique membership number to him/her, prints a bill towards his/her membership charge. This function may consist of the following sub-functions:
         - » assign-membership-number
         - » create-member-record
         - » print-bill

Function Oriented Approach ….

- » Each of these sub-functions may be split into more detailed sub-functions and so on.

3. The system state is centralized and shared among different functions, e.g. 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 a collection of objects (i.e. entities).
  - The state is decentralized among the objects and each object manages its own state information.
- Ex. In an 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.
  - Similar objects constitute a class.
- In other words, each object is a member of some class.
  - Classes may inherit features from super class.
  - Conceptually, objects communicate by message passing.

Function-Oriented Vs Object- Oriented Design

- Unlike function-oriented design methods, in ODD, the basic abstraction are not real world functions such as sort, display, track, etc. but real-world entities such as employee, picture, machine, radar system, etc.
- Ex.
  - In ODD, an employee pay-roll software is not developed by designing functions such as update-employee-record, get-employee-address, etc. but by designing objects such as employees, departments, etc.
- In object-oriented design, state information is not represented in a centralized shared memory but is distributed among the objects of the system.
- Ex.
  - While developing employee pay-roll system, the employee data such as the names of the employees, their code numbers, basic salaries, etc. are usually implemented as global data in a traditional programming system; whereas in an object-oriented system, these data are distributed among different employee objects of the system.

Function-Oriented Vs Object- Oriented Design

- Objects communicate by passing messages.
  - Therefore, one object may discover the state information of another object by interrogating it.
  - Thus, somewhere or the other, the real-world functions must be implemented.
- Function-oriented techniques such as SA/SD group functions together if, as a group, they constitute a higher-level function.
  - On the other hand, object-oriented techniques group functions together on the basis of the data they operate on.

User Interface Design

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

- Design rules for User Interface:

1) Place the User in control

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*
  - An interaction mode is the current state of the interface.
  - Ex.
    - ✔ If spell check is selected in a word-processor menu, the software moves to a spell-checking mode.
    - ✔ There is no reason to force the user to remain in spell-checking mode if the user desires to make a small text

User Interface Design ….

- *Provide for flexible interaction*
  - Because different users have different interaction preferences, choices should be provided.
  - Ex.
    - ✔ Software might allow a user to interact via keyboard commands, mouse movement, a digitizer pen, a multi touch screen, or voice recognition commands.
- *Allow user interaction to be interruptible and undoable*
  - Even when involved in a sequence of actions, the user should be able to interrupt the sequence to do something else.
- *Streamline interaction as skill levels advance and allow the interaction to be customized*
  - Users often find that they perform the same sequence of interactions repeatedly.

User Interface Design ….

- *Hide technical internals from the casual user*
  - The user interface should move the user into the virtual world of the application.
  - The user should not be aware of the operating system, file management functions, or other arcane computing technology.
- *Design for direct interaction with objects that appear on the screen*
  - The user feels a sense of control when able to manipulate the objects that are necessary to perform a task in a manner similar to what would occur if the object were a physical thing.

3) 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.

User Interface Design ….

- *Reduce demand on short-term memory*
  - When users are involved in complex tasks, the demand on short-term memory can be significant.
  - The interface should be designed to reduce the requirement to remember past actions, inputs and results.
- *Establish meaningful defaults*
  - The initial set of defaults should make sense for the average user, but a user should be able to specify individual preferences.
  - However, a ‘reset’ option should be available, enabling the redefinition of original default values.
- *Define shortcuts that are intuitive*
  - When mnemonics are used to accomplish a system function, the mnemonic should be tied to the action in a way that is easy to remember.

User Interface Design ….

- *The visual layout of the interface should be based on a real-world metaphor*
  - » This enables the user to rely on well-understood visual cues, rather than memorizing an arcane interaction sequence.
- *Disclose information in a progressive fashion*
  - » The interface should be organized hierarchically.
  - » i.e. information about a task, an object, or some behavior should be presented first at a high level of abstraction.

4) 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.

User Interface Design ….

- *Allow the user to put the current task into meaningful context*
  - » Many interfaces implement complex layers of interactions with dozens of screen images.
  - » It is important to provide indicators that enable the user to know the context of the world at hand.
- *Maintain consistency across a family of applications*
  - » A set of applications should all implement the same design rules so that consistency is maintained for all interactions.
- *If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so.*
  - » Once a particular interactive sequence has become a de facto standard, the user expects this in every application he encounters.

User Interface Design Models

- Four different models come into play when a user interface is analyzed and designed.

A. **User profile model** – Established by a human engineer or software engineer

- Establishes profile of the end-users of the system based on age, gender, physical abilities, education, cultural or ethnic background, motivation, goals and personality.
- The underlying sense of the application; an understanding of the functions that are performed, the meaning of input and output, and the objectives of the system categories users as
  - **Novices**
    - No syntactic knowledge of the system, little semantic knowledge of the application, only general computer usage.

User Interface Design Models ….

- **Knowledgeable, intermittent users**
  - Reasonable semantic knowledge of the system, low recall of syntactic information to use the interface.
- **Knowledgeable, frequent users**
  - Good semantic and syntactic knowledge (i.e. power user), look for shortcuts and abbreviated modes of operation.

C. **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.
- Constrained by information in the requirements specification that helps define the user of the system.

User Interface Design Models ….

C. **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.
   - Strives to agree with the user's mental model; users then feel comfortable with the software and use it effectively.

D. **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.
   - Accuracy of the description depends upon the user's profile and overall familiarity with the software in the application domain.

- The role of the interface designer is to merge all these differences and derive a consistent representation of the interface.

Web Application Design

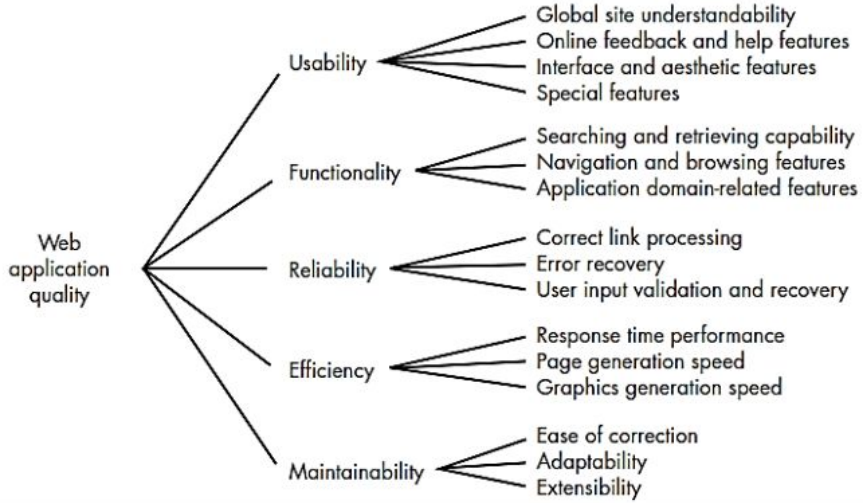
Design for WebApp encompasses technical and non-technical activities that include:

- establishing the look and feel of the WebApp
- creating the aesthetic layout of the user interface
- defining the overall architectural structure
- developing the content and functionality that reside within the architecture
- planning the navigation that occurs within the WebApp

WebApp Design Quality Requirement:

- Design is the engineering activity that leads to a high-quality product.
- This leads us to a recurring question that is encountered in all engineering disciplines.

Web Application Design …. Contd.



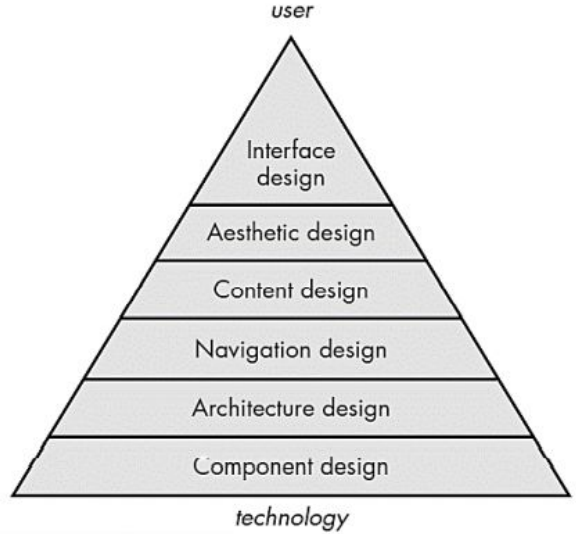
WebApp Interface Design:-

The objectives of a WebApp interface are to:

a. 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.

Web Application Design …. Contd.



- **Aesthetic Design:**
  - Aesthetic Design, also called graphic design, is an artistic endeavor that complements the technical aspects of WebApp design.
  - Without it, a WebApp may be functional, but unappealing.
    - With it, a WebApp draws its users into a world that embraces them on a primitive, as well as an intellectual level.

Web Application Design …. Contd.

- **Content Design:**
  - Content design focuses on two different design tasks, each addressed by individuals with different skill sets.
  - First, a design representation for content objects and the mechanisms required to establish their relationship to one another is developed.
  - In addition, the information within a specific content object is created.
  - The latter task may be conducted by copywriters, graphic designers, and others who generate the content to be used within a WebApp.

- **Architecture Design:**
  - Architecture design is tied to the goals established for a WebApp, the content to be presented, the users who will visit, and the navigation philosophy that has been established.

Web Application Design …. Contd.

- In most cases, the architecture design is conducted in parallel with interface design, aesthetic design, and content design.
- Because the WebApp architecture may have a strong influence on navigation, the decisions made during this design action will influence work conducted during navigation design.

❖ **Navigation Design:**

- Once the WebApp architecture has been established and the components (pages, scripts, applets, and other processing functions) of the architecture have been identified, you must define navigation pathways that enable users to access WebApp content and functions.

Web Application Design …. Contd.

- **Component-level Design:**
  - Modern WebApps deliver increasingly sophisticated processing functions that -
    i. Perform localized processing to generate content and navigation capability in a dynamic fashion.
    ii. Provide computation or data processing capability that are appropriate for the WebApp's business domain.
    iii. Provide sophisticated database query and access.
    iv. Establish data interfaces with external corporate systems.