

Design Process :

■ Software design is an iterative process using which the user requirements can be translated into the software product.

■ At the initial stage of the software is represented as an abstract view. During the sub-sequent iterations data, functional and behavioral requirements are traced in detail. That means at each iteration the refinement is made to obtain lower level details of the software product.

Characteristics of Good Design

1. The good design should implement all the requirements specified by the customer. Even if there are some implicit requirements of the software product then those requirements should also be implemented by the software design.
2. The design should be simple enough so that the software can be understood easily by the developers, testers and customers.
3. The design should be comprehensive. That means it should provide complete picture of the software.

## Design Concepts

The software design concept provides a framework for implementing the right software.

Following are certain issues that are considered while designing the software -

- ☐ ■ Abstraction
- ☐ ■ Modularity
- ☐ ■ Architecture
- ☐ ■ Refinement
- ☐ ■ Pattern
- ☐ ■ Information hiding
- ☐ ■ Functional independence
- ☐ ■ Refactoring
- ☐ ■ Design classes
- ☐

## Abstraction

The abstraction means an ability to cope up with the complexity. Software design occurs at different levels of abstraction. At each stage of software design process levels of abstractions should be applied to refine the software solution. At the **higher level** of abstraction, the solution should be stated in **broad terms** and in the **lower level** more **detailed description** of the solution is given.

While moving through different levels of abstraction the procedural abstraction and data abstraction are created.

The **procedural abstraction** gives the named sequence of instructions in the specific function. That means the functionality of procedure is mentioned by its implementation details are hidden. For example : Search the record is a procedural abstraction in which implementation details are hidden(i.e. Enter the name, compare each name of the record against the entered one, if a match is found then declare success!! Other wise declare 'name not found')

In **data abstraction** the collection of data objects is represented. For example for the procedure search the data abstraction will be Record. The record consists of various attributes such as Record ID, name, address and designation.

## MODULARIZATION

Modularization is a technique to divide a software system into multiple discrete and independent modules, which are expected to be capable of carrying out task(s) independently. These modules may work as basic constructs for the entire software. Designers tend to design modules such that they can be executed and/or compiled separately and independently.

Modular design unintentionally follows the rules of 'divide and conquer' problem-solving strategy this is because there are many other benefits attached with the modular design of a software.

## Refinement

- Refinement is actually a process of elaboration.
- Stepwise refinement is a top-down design strategy proposed by Niklaus WIRTH.
- The architecture of a program is developed by successively refining levels of procedural detail.
- The process of program refinement is analogous to the process of refinement and partitioning that is used during requirements analysis.
- Abstraction and refinement are complementary concepts. The major difference is that - in the abstraction low-level details are suppressed. Refinement helps the designer to elaborate low-level details.

## Pattern

According to **Brad Appleton** the design pattern can be defined as - It is a named nugget(something valuable) of insight which conveys the essence of proven solution to a recurring problem within a certain context.

In other words, design pattern acts as a design solution for a particular problem occurring in specific domain. Using design pattern designer can determine whether-

- ■ Pattern can be **reusable**.
- ■ Pattern can be used for **current work**.
- ■ Pattern can be used to **solve similar kind of problem** with different functionality.

## Information Hiding

Information hiding is one of the important property of effective modular design. The term information hiding means the modules are designed in such a way that information contained in one module cannot be accessible to the other module (the module which does not require this information). Due to information hiding only limited amount of information can be passed to other module or to any local data structure used by other module.

The **advantage** of information hiding is basically in testing and maintenance. Due to information hiding some data and procedures of one module can be hidden from another module. This ultimately **avoids** introduction of **errors** module from one module to another. Similarly one can make **changes** in the desired module without affecting the other module.

## Functional Independence

- The functional independence can be achieved by developing the functional modules with single-minded approach.
- By using functional independence functions may be compartmentalized and interfaces are simplified.
- Independent modules are easier to maintain with reduced error propagation.
- Functional independence is a key to good design and design is the key to software

quality.

- The major benefit of functional independence is in achieving effective modularity.
- The functional independence is assessed using two qualitative criteria - **Cohesion and coupling**.

### **Cohesion**

- With the help of cohesion the information hiding can be done.
- A cohesive module performs only “one task” in software procedure with little interaction with other modules. In other words cohesive module performs only one thing.
- Different types of cohesion are :
  1. **Coincidentally cohesive** - The modules in which the set of tasks are related with each other loosely then such modules are called coincidentally cohesive.
  2. **Logically cohesive** - A module that performs the tasks that are logically related with each other is called logically cohesive.
  3. **Temporal cohesion** - The module in which the tasks need to be executed in some specific time span is called temporal cohesive.
  4. **Procedural cohesion** - When processing elements of a module are related with one another and must be executed in some specific order then such module is called procedural cohesive.
  5. **Communicational cohesion** - When the processing elements of a module share the data then such module is communicational cohesive.
- The goal is to achieve high cohesion for modules in the system.

### **Coupling :**

Coupling effectively represents how the modules can be “connected” with other module or with the outside world.

- Coupling is a measure of interconnection among modules in a program structure.
- Coupling depends on the interface complexity between modules.
- The goal is to strive for lowest possible coupling among modules in software design.
- The property of good coupling is that it should reduce or avoid change impact and ripple effects. It should also reduce the cost in program changes, testing and maintenance.
- Various types of coupling are :
  - i) **Data coupling** - The data coupling is possible by parameter passing or data interaction.
  - ii) **Control coupling** - The modules share related control data in control coupling.
  - iii) **Common coupling** - In common coupling common data or a global data is shared among the modules.
  - iv) **Content coupling** - Content coupling occurs when one module makes use of data or control information maintained in another module.

### **Refactoring**

Refactoring is necessary for simplifying the design without changing the function or behaviour. **Fowler** has defined refactoring as "the process of changing a software system in such a way that the external behaviour of the design do not get changed, however the internal structure gets improved".

**Benefits** of refactoring are -

- The **redundancy** can be achieved.
- **Inefficient algorithms** can be eliminated or can be replaced by efficient one.
- Poorly constructed or **inaccurate data structures** can be removed or replaced.
- Other **design failures** can be rectified.

The decision of refactoring particular component is taken by the designer of the software system.

### Design Classes

Design classes are defined as the classes that describe some elements of problem domain, focus on various aspects of problem from user's point of view.

The goal of design classes is :

1. To refine the analysis classes by providing the detail design, so that further implementation can be done easily.
2. To create new set of classes for implementing the infrastructure of the software.

There are five different types of design classes

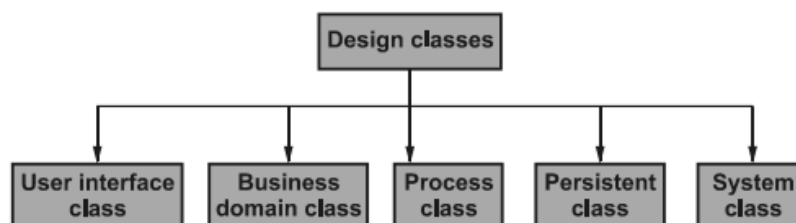


Fig. 3.3.2 Types of Design classes

#### 1. User Interface class

The user interface class defines all the abstractions that are necessary for Human Computer interface(HCI). The user interface classes is basically a **visual representation** of the HCI.

#### 2. Business Domain Class

Business domain classes are the refinement of analysis classes. These classes identify the **attributes and services** that are needed to implement some elements of **business domain**.

#### 3. Process Class

Process class implement lower level business abstractions used by the business domain.

#### 4. Persistent Class

These classes represent the data store such as databases which will be retained as it is even after the execution of the software.

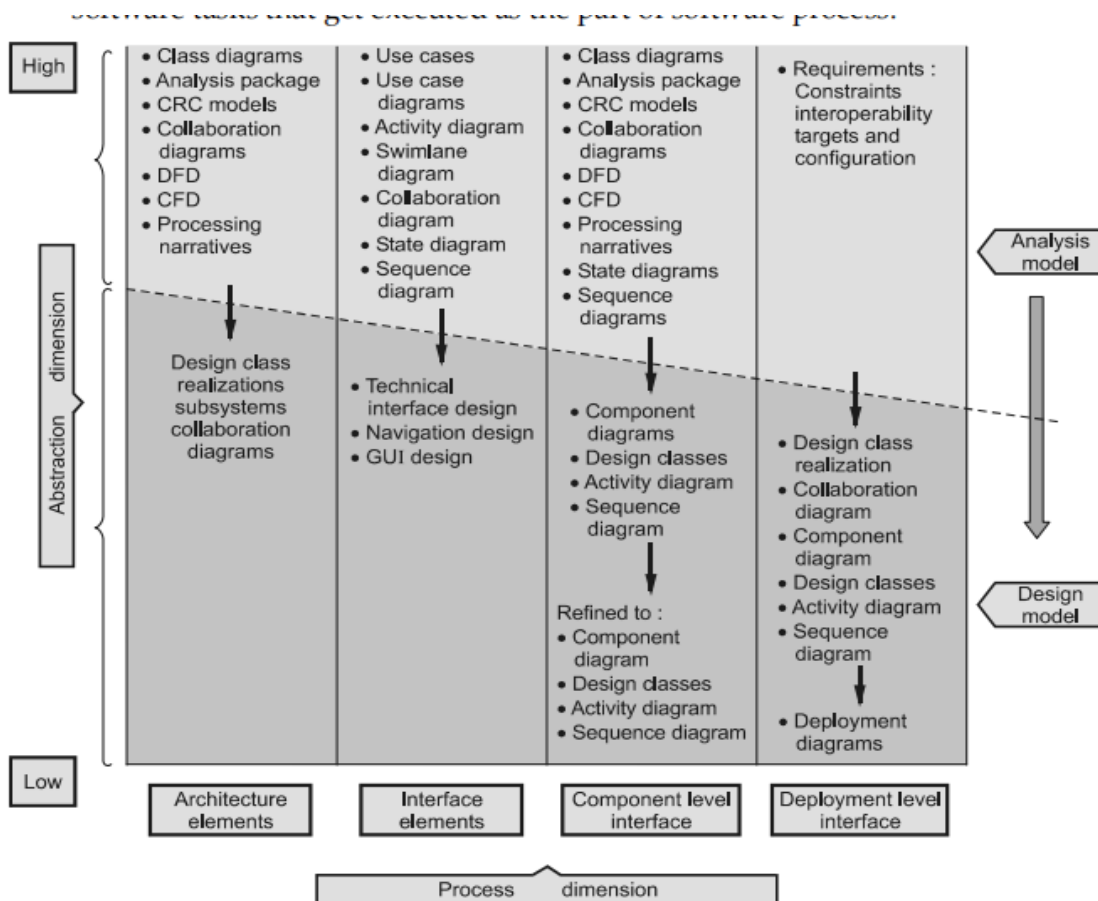
#### 5. System Class

These classes are responsible for software management and control functions that are used for system operation.

Each class must be **well formed design class**. Following are some characteristics of well formed design classes -



DESIGN MODEL



**Fig. 3.4.1 Dimension of Design Model**

The **abstract dimension** represents level of details as each element of analysis model is transformed into design equivalent.

■ In Fig. 3.4.1 the **dashed line** shows the boundary between analysis and design model.

■ In both the analysis and design models the same UML diagrams are used but in analysis model the UML diagrams are abstract and in design model these diagrams are refined and elaborated. Moreover in design model the implementation specific details are provided.

■ Along the horizontal axis various elements such as architecture element, interface element, component level elements and deployment level elements are given. It is not necessary that these elements have to be developed in sequential manner. First of all the preliminary architecture design occurs then interface design and component level design occur in parallel. The deployment level design ends up after the completions of complete design model.

### 1) **Data Design Element**

The data design represents the **high level of abstraction**. This data represented at data design level is **refined gradually** for implementing the computer based system. The data has great impact on the architecture of software systems. Hence structure of data is very important factor in software design. Data appears in the form of **data structures and algorithms** at the program **component level**. At the **application level** it appears as the **database** and at the **business level** it appears as **data warehouse and data mining**. Thus data plays an important role in software design

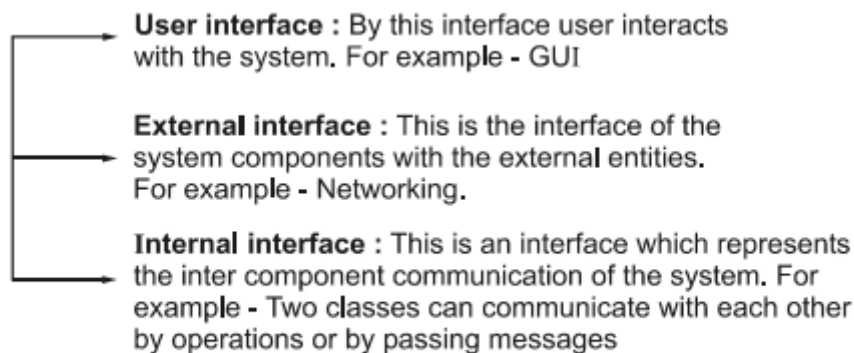
## 2 ) Architectural Design Element

The architectural design gives the layout for overall view of the software. Architectural model can be built using following sources -

- Data flow models or class diagrams
- Information obtained from application domain
- Architectural patterns and styles.

## 3 ) Interface Design Elements

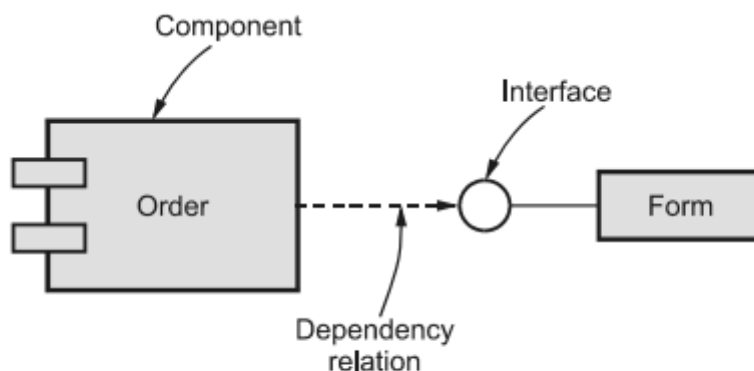
Interface Design represents the **detailed design** of the software system. In interface design how **information flows** from one component to other component of the system is depicted. Typically there are three types of interfaces –



## 4) Component Level Design Elements

The component level design is more detailed design of the software system along with the specifications. The component level design elements describe the internal details of the component. In component level design all the local data objects, required data structures and algorithmic details and procedural details are exposed.

Fig. 3.4.3 represents that component **order** makes use of another component **form**

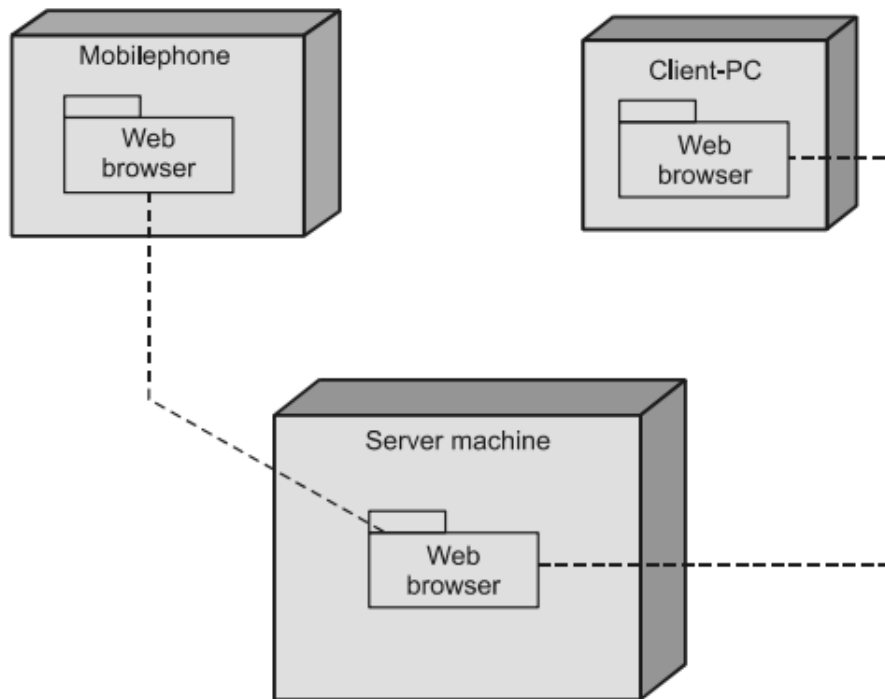


**Fig. 3.4.3 Components**

The **order** is dependent upon the component form. These two objects can be interfaced with each other.

### 5 ) Deployment Level Design Elements

The deployment level design elements indicate how software functions and software subsystems are assigned to the physical computing environment of the software product.



**Fig. 3.4.4 Deployment diagram**

For example web browsers may work in mobile phoned or they may run on client PC or can execute on server machines.

### Design Heuristic

The program structure can be manipulated according the design heuristics as shown below.

1. **Evaluate the first iteration of the program structure to reduce the coupling and improve cohesion** - The module independency can be achieved in the program structure by exploding or imploding the modules. Exploding the modules means obtaining two or more modules in the final stage and imploding the modules means combining the result of different modules.
2. **Attempt to minimize the structures with high fan-out and strive for fan-in as depth increases** - At the higher level of program structure the distribution of control should be made. Fan-out means number of immediate subordinates to the module. Fan-in means number of immediate ancestors the module have.
3. **Keep scope of the effect of a module within the scope of control of that module** - The decisions made in particular module 'a' should not affect the module 'b' which lies outside the scope of module 'a'.

**4. Evaluate the module interfaces to reduce complexity and redundancy and improve consistency** - Mostly the cause of software errors is module interfaces. The module interfaces should simply pass the information and should be consistent with the module.

**5. Define module whose function is predictable but avoid modules that are too restrictive** - The module should be designed with simplified internal processing so that expected data can be produced as a result. The modules should not restrict the size of local data structure, options with control flow or modes of external interfaces. Being module too much restrictive causes large maintenance.

**6. Strive for controlled entry modules by avoiding pathological connections** - Software interfaces should be constrained and controlled so that it will become manageable. Pathological connection means many references or branches into the middle of a module.

## Introduction to Architectural Design

**Definition : Architectural design** is a design created to represent the data and program components that are required to build the computer based systems.

■ Architectural design is a specialized activity in which using specific architectural style and by considering the system's structure and properties of system components - a large and complex system is built.

■ The person who is responsible to design such system is called **software architect**.

■ The architectural design gives a **layout** of the system that is to be built. In short, the **program structure** is created during architectural design along with the description of component properties and their inter-relationship.

## Software Architecture

The architectural design is the design process for identifying the subsystems making up the system and framework for subsystem control and communication.

The **goal** of architectural design is to establish the overall structure of software system.

Architectural design represents the link between design specification and actual design process.

*Software Architecture is a structure of systems which consists of various components, externally visible properties of these components and the inter-relationship among these components.*

### 1) Structural Partitioning

The program structure can be partitioned horizontally or vertically.

#### A) Horizontal partitioning

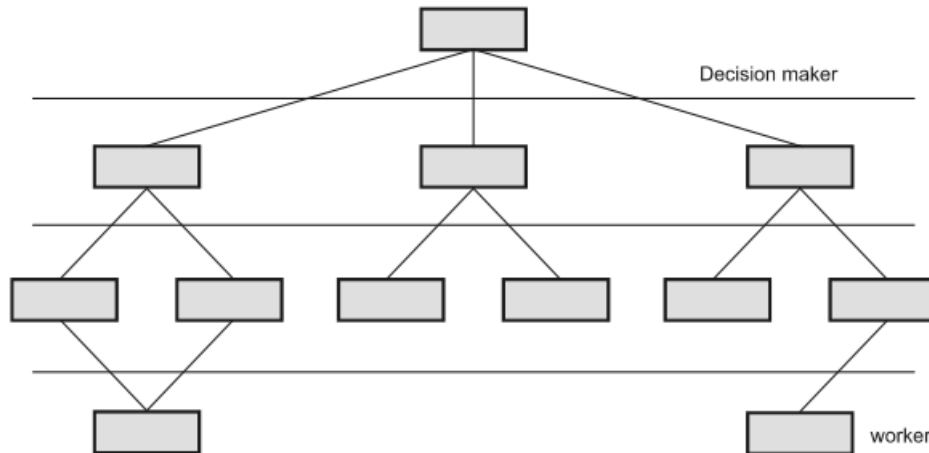
Horizontal partitioning defines separate branches of the modular hierarchy for each major program function.

Horizontal partitioning can be done by partitioning system into : input, data transformation (processing) and output.



In horizontal partitioning the design making modules are at the top of the architecture.

In horizontal partitioning the design making modules are at the top of the architecture



**Fig. 3.7.1 Horizontal partitioning**

#### **Advantages of horizontal partition**

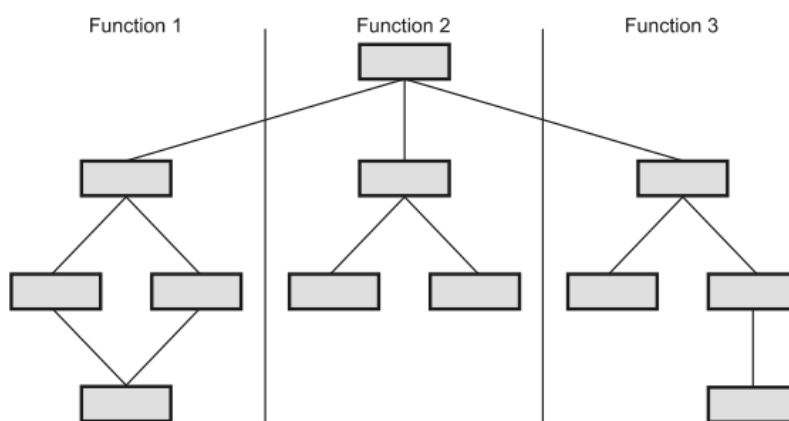
1. These are easy to test, maintain and extend.
2. They have fewer side effects in change propagation or error propagation.

#### **Disadvantage of horizontal partition**

More data has to be passed across module interfaces which complicate the overall control of program flow.

#### **B)Vertical partitioning**

Vertical partitioning suggests the control and work should be distributed top-down in program structure.



**Fig. 3.7.2 Vertical partitioning**

In vertical partitioning

- Define separate branches of the module hierarchy for each major function.
- Use control modules to co-ordinate communication between functions.

#### **Advantages of vertical partition**

1. These are easy to maintain the changes.

2. They reduce the change impact and error propagation.

### Architectural Styles

■ The **architectural model or style** is a pattern for creating the system architecture for given problem. However, most of the large systems are heterogeneous and do not follow single architectural style.

■ System categories define the architectural style

**1. Components :** They perform a function.

For example : Database, simple computational modules, clients, servers and filters.

**2. Connectors :** Enable communications. They define how the components communicate, co-ordinate and co-operate.

For example : Call, event broadcasting, pipes

**3. Constraints :** Define how the system can be integrated.

**4. Semantic models :** Specify how to determine a system's overall properties from the properties of its parts.

■ The commonly used architectural styles are

**1. Data centered architectures.**

**2. Data flow architectures.**

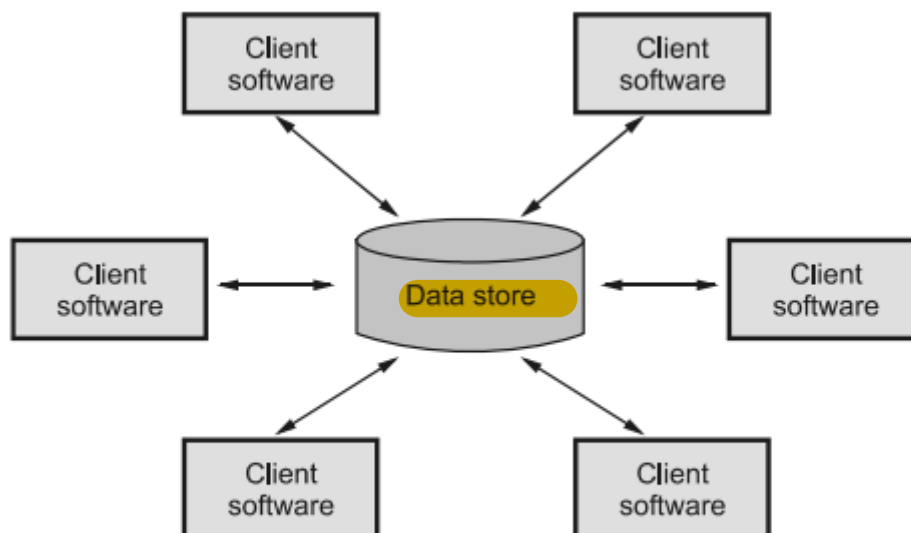
**3. Call and return architectures.**

**4. Object oriented architectures.**

**5. Layered architectures.**

### Data Centered Architectures

In this architecture the data store lies at the centre of the architecture and other components frequently access it by performing add, delete and modify operations. The client software requests for the data to central repository. Sometime the client software accesses the data from the central repository without any change in data or without any change in actions of software actions.



**Fig. 3.9.1 Data centered architecture**

Data centered architecture possesses the property of interchangeability.

Interchangeability means any component from the architecture can be replaced by a new

component without affecting the working of other components.  
In data centered architecture the data can be passed among the components.

In data centered architecture,

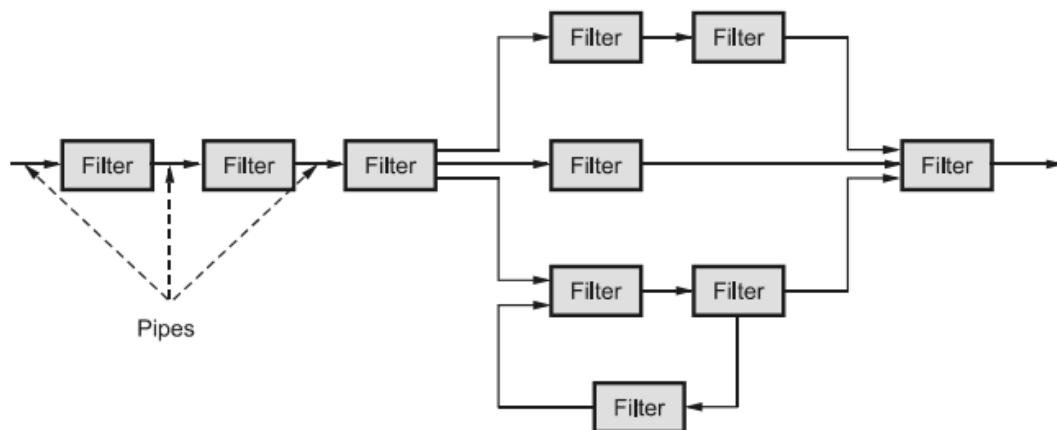
**Components are :** Database elements such as tables, queries.

**Communication are :** By relationships

**Constraints are :** Client software has to request central data store for information.

### Data Flow Architectures

In this architecture series of transformations are applied to produce the output data. The set of components called filters are connected by pipes to transform the data from one component to another. These filters work independently without a bothering about the working of neighbouring filter.



**Fig. 3.9.2 Pipes and filters**

If the data flow degenerates into a single line of transforms, it is termed as batch sequential.

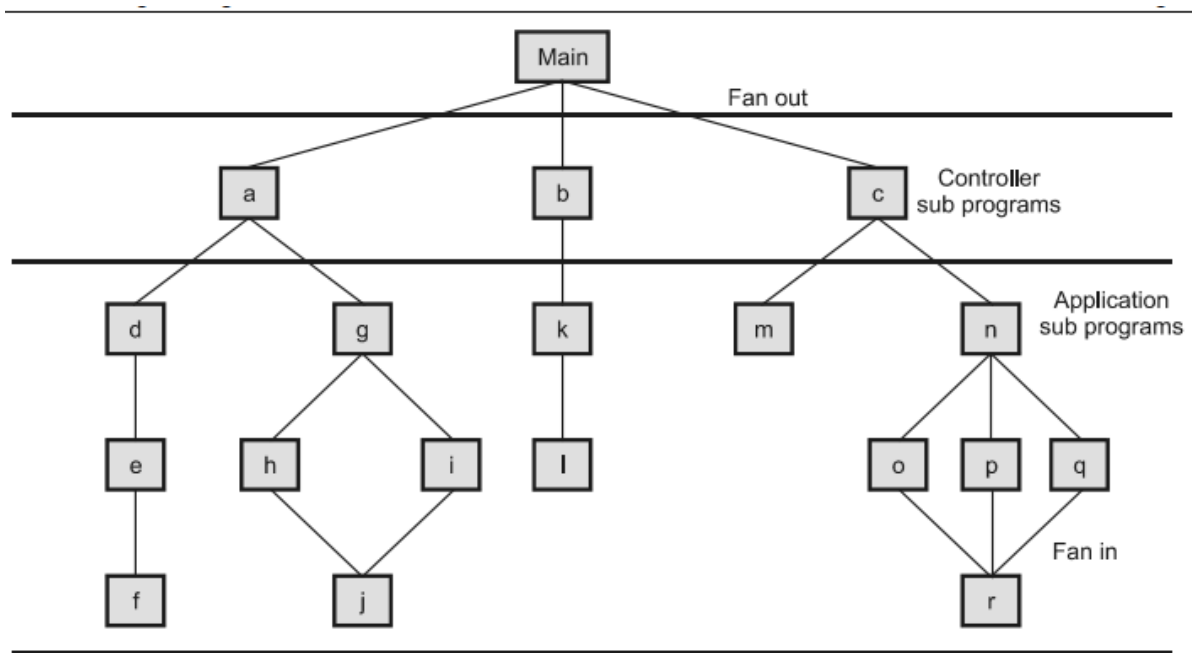


**Fig. 3.9.3 Batch sequential**

### Call and Return Architecture

The program structure can be easily modified or scaled. The program structure is organized into modules within the program. In this architecture how modules call each other. The program structure decomposes the function into control hierarchy where a main program invokes number of program components.

In this architecture the hierarchical control for call and return is represented.

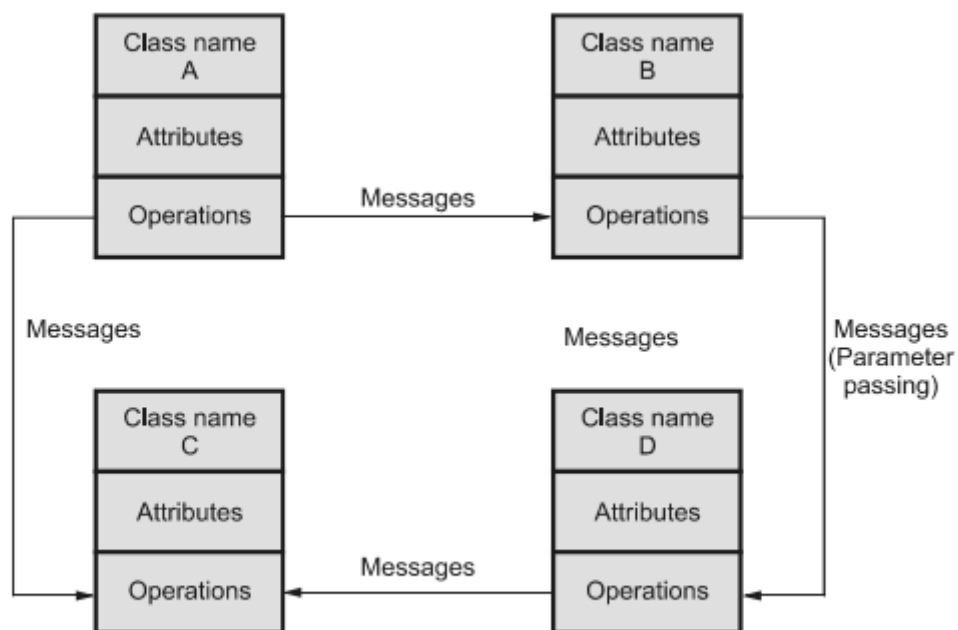


**Fig. 3.9.4 Call and return Architecture**

### Object Oriented Architecture

In this architecture the system is decomposed into number of interacting objects. These objects encapsulate data and the corresponding operations that must be applied to manipulate the data.

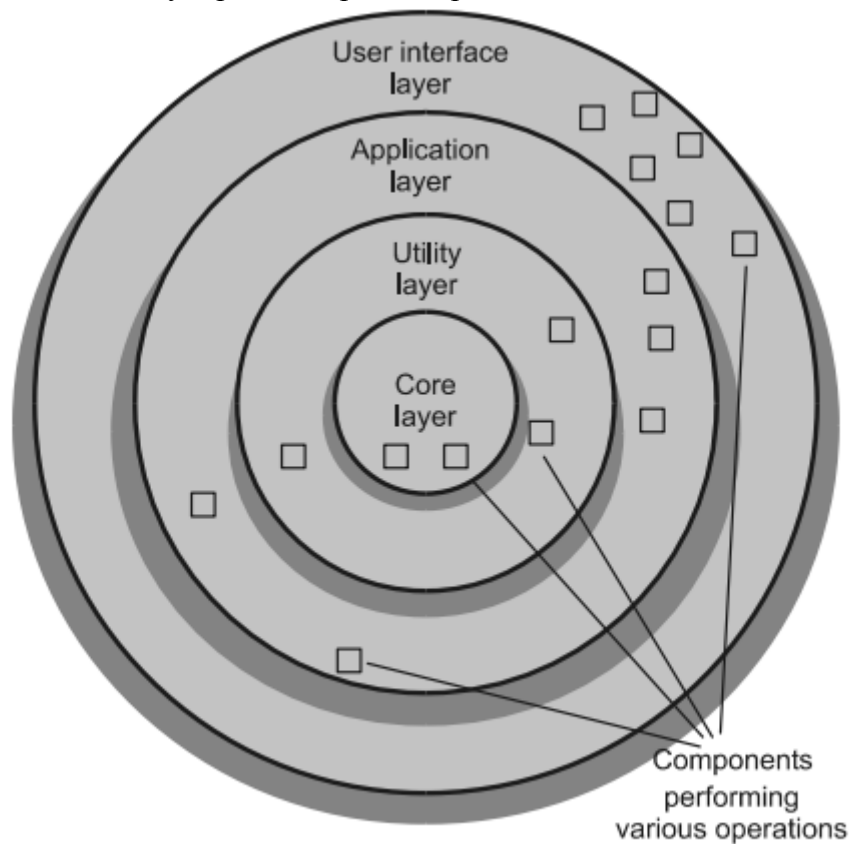
The object oriented decomposition is concerned with identifying objects classes, their attributes and the corresponding operations. There is some control models used to co ordinate the object operations.



**Fig. 3.9.5 Object oriented architecture**

### Layered Architecture

■ The layered architecture is composed of different layers. Each layer is intended to perform specific operations so machine instruction set can be generated. Various components in each layer perform specific operations.



**Fig. 3.9.6 Layer architecture of components**

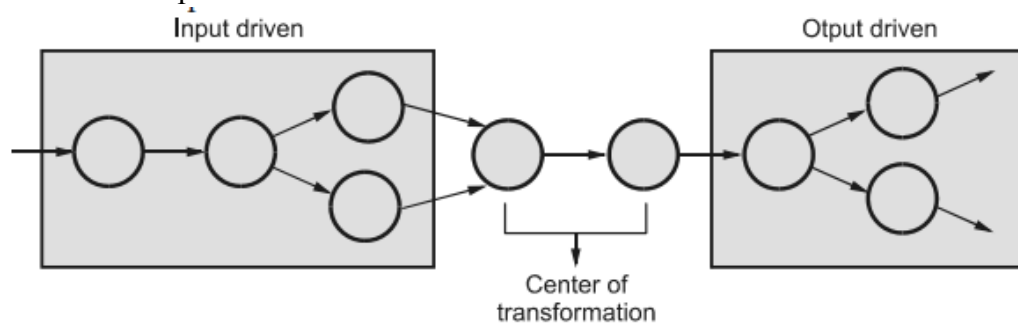
The outer layer is responsible for performing the user interface operations while the components in the inner layer perform operating system interfaces.

■ The components in intermediate layer perform utility services and application software functions.

### Architectural Mapping using Data Flow

#### Transform flow

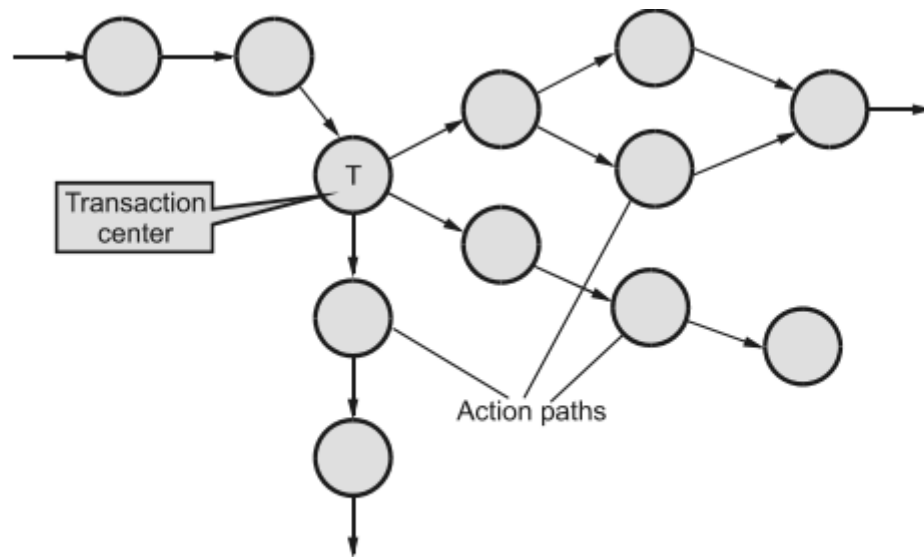
A transform flow is a sequence of paths which forms transition in which input data are transformed into output data.



**Fig. 3.11.1 Transform flow**

#### Transaction flow

A transaction flow represents the information flow in which single data item triggers the overall information flow along the multiple paths. This triggering data item is called transaction.



**Fig. 3.11.2 Transaction flow**

#### Example : Automated Railway Reservation System

**Problem description :** The automated railway reservation system is prepared for reserving the railway ticket online. Various activities that the user can perform are -

1. Registers for using the system
2. Make the train enquiry
3. Can view the status of reservation by entering PNR number
4. Can make the reservations
5. Can cancel the reservations.

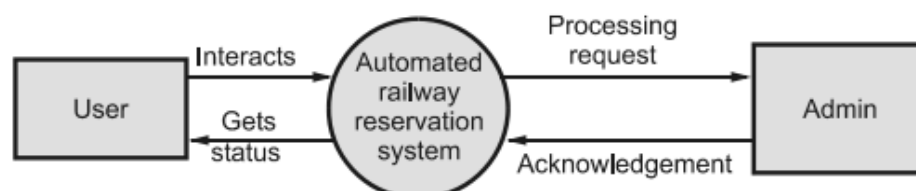
After registering himself/herself to the system, user must have to log in. He/she can make the enquiry about the trains either by entering the train number or by entering the source and destinations.

User can see the availability of seats by entering the passenger details and journey details. If the seats are available then he can make the reservations by making the payment. The user then prints the E-Ticket.

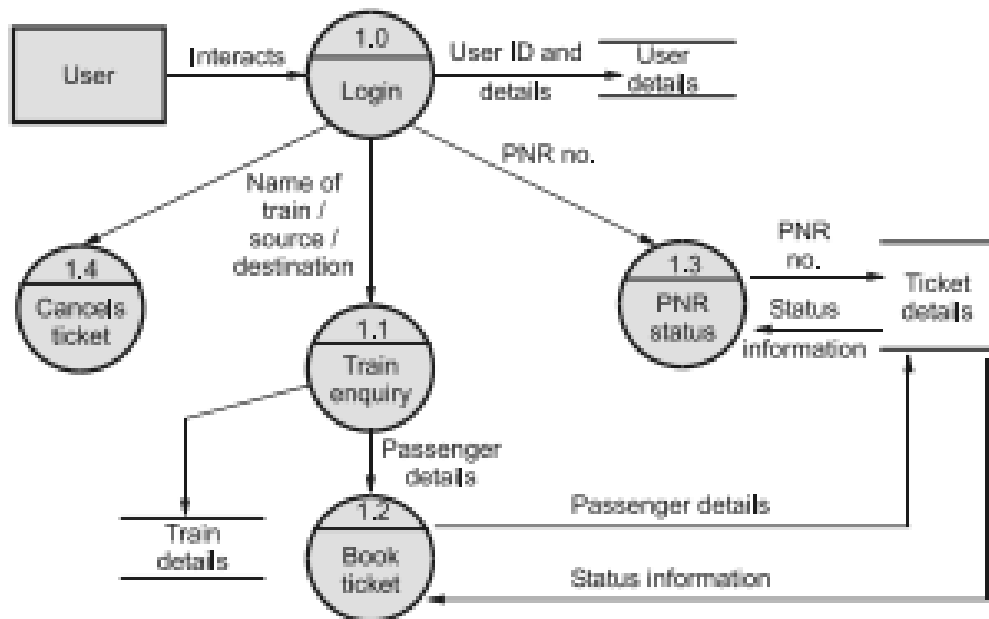
User can view the latest status of his reservations by simply entering the PNR number.

If the user wants to cancel some reservations then he/she has to enter the PNR number.

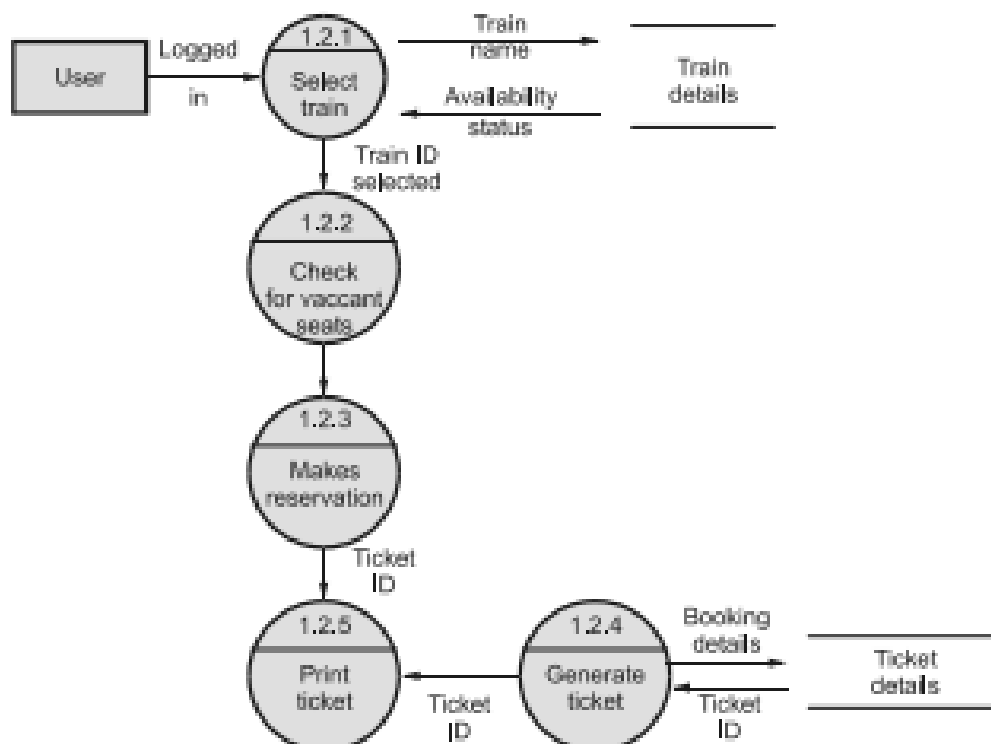
When user selects for the cancel reservations option then refund amount is paid to the customer and appropriate records are updated.



**Fig. 3.11.3 Context Level DFD (Level 0)**





**Fig. 3.11.4 Level 1 DFD**



**Fig. 3.11.5 Level 2 DFD for book ticket functionality**

## Component Level Design

-  Component is nothing but a model for computer software.
-  The OMG Unified Modeling Language Specification defines the concept of component more formally and it is -
- "Component is a modular, deployable and replaceable part of system that encapsulates the implementation and exposes the set of interfaces".

- ❑ 🎬 Components are the part of software architecture and hence they are important factors in achieving the objectives and requirements of system to be built.
- ❑ 🎬 Components can communicate and collaborate with other components.
- ❑ 🎬 Design models can be prepared using object oriented views and conventional views.

### Designing Class based Components

🎬 Component is represented as a part of architectural model. It collects the information about the system as a part of analysis model.

🎬 If **object oriented software engineering approach** is adopted then the component level design will emphasize on elaboration of analysis classes and refinement of **infrastructure classes**.

🎬 The detailed description of attributes, operations and interfaces of these infrastructure classes is required during the system building

### Traditional Components

🎬 **Component level design** is also called as **procedural design**. After data, architectural and interface design the component level design occurs.

🎬 The **goal** of component level design is to translate design model into operational software.

🎬 Graphical, tabular or text based notations are used to create a set of **structured programming constructs**. These structured programming constructs translate each component into **procedural design model**. Hence the work product of component level design is procedural design model.