Unit – 3
Software Design

Design process – Design Concepts-Design Model– Design Heuristic – Architectural Design – Architectural styles, Architectural Design, Architectural Mapping using Data Flow- User Interface Design: Interface analysis, Interface Design –Component level Design: Designing Class based components, traditional Components.

Design process

3.1 Introduction- Software Design

Software design is model of software which translates the requirements into finished software product in which the details about software data structures, architecture, interfaces and components that are necessary to implement the system are given.

3.1.1 Designing within the Context of Software Engineering

Software design is at the core of software engineering and it is applied irrespective of any process model.

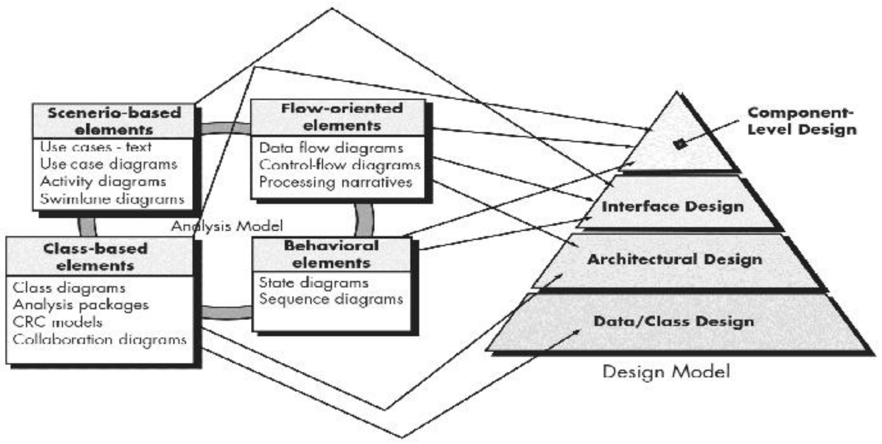
After analyzing and modelling the requirements, software design is done which serves as the basis for code generation and testing.

Software designing is a process of translating analysis model into the design model.

The analysis model is manifested by scenario based, class based, flow oriented and behavioral elements and feed the design task.

The classes and relationships defined by CRC index cards and other useful class based elements provide the basis for the data or class design.

The **architectural design** defines the relationship between major structural elements of the software. The architectural styles and design patterns can be used to achieve the requirements defined for the system.



The **interface design** describes how software communicates with systems. These systems are interacting with each other as well as with the humans who operate them. Thus interface design represents the flow of information and specific type of behavior. The usage scenarios and behavioral models of analysis modelling provide the information needed by the interface design.

The component-level design transforms structural elements of software architecture into procedural description of software module. The information used by the component design is obtained from class based model, flow based model and behavioral model.

Software design is **important** to assess the quality of software. Because design is the only way that we can accurately translate the user requirements into the finished software product.

Without design unstable system may get developed. Even if some small changes are made then those changes will go fail .It will become difficult to test the product. The quality of the software product cannot be assessed until late in the software process.

3.2 Design Process

Software design is an iterative process in which the requirements are translated into the blueprint of the software. Initially the software is represented at high level of abstraction, but during the subsequent iterations data, functional and behavioral requirements are traced in more detail. Thus refinement made during each iteration leads to design representations at much lower level of abstraction. Throughout the software design process the quality of the software is assessed by considering certain characteristics of the software design.

Characteristics of good design-

1. The good design should implement all the requirements that are explicitly mentioned in the analysis model. It should accommodate all the implicit requirements demanded by the customer.
2. The design should be simple enough so that the code developer, code tester as well as those who are supporting the software will find it readable and understandable.
3. The design should be comprehensive. That means it should provide a complete picture of software, addressing the data, functional and behavioral domains from an implementation perspective.

Quality Guideline

The goal of any software design is to produce high quality software. In order to evaluate quality of software there should be some predefined rules or criteria that need to be used to assess the software product. Such criteria serve as characteristics for good design. The quality guidelines are as follows

1. The design architecture should be created using following issues **-**
   - The design should be created using architectural styles and patterns.
   - Each component of design should possess good design characteristics
   - The implementation of design should be evolutionary, so that testing can be performed at each phase of implementation.
2. In the design the data, architecture, interfaces and components should be clearly represented.
3. The design should be modular. That means the subsystems in the design should be logically partitioned.
4. The data structure should be appropriately chosen for the design of specific problem.
5. The components should be used in the design so that functional independency can be achieved in the design.

6. Using the information obtained in software requirement analysis the design should be created.
7. The interfaces in the design should be such that the complexity between the connected components of the system gets reduced. Similarly interface of the system; with external interface should be simplified one.
8. Every design of the software system should convey its meaning appropriately and effectively.

Quality Attributes

The design quality attributes popularly known as FURPS (Functionality, Usability, Reliability, Performance and Supportability) is a set of criteria developed by Hewlett and Packard. Following table represents meaning of each quality attribute

Quality Attribute	Meaning
Functionality	Functionality can be checked by assessing the set of features and capabilities of the functions. The functions should be general and should not work only for particular set of inputs. Similarly the security aspect should be considered while designing the function.
Usability	The usability can be assessed by knowing the usefulness of the system.
Reliability	Reliability is a measure of frequency and severity of failure Repeatability refers to the consistency and repeatability of the measures The mean time to failure (MTTF) is a metric that is widely used to measure the product's performance and reliability.
Performance	It is a measure that represents the response of the system. Measuring the performance means measuring the processing speed, memory usage, response time and efficiency.
Supportability	It is also called, maintainability. It is the ability to adopt the enhancement or changes made in the software. It also means, the ability to withstand in a given environment

Design Concepts

3.3 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

1. **Abstraction**
2. **Modularity**
3. **Architecture**
4. **Refinement**
5. **Pattern**
6. **Information hiding**
7. **Functional independence**
8. **Refactoring**
9. **Design classes**

3.3.1 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, and if a match is found then declare success!! Otherwise 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.

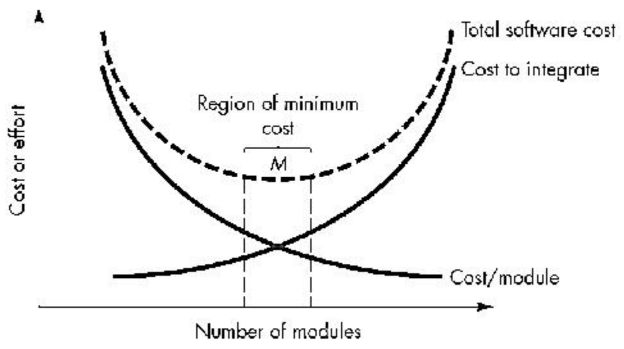
3.3.2 Modularity

The software is divided into separately named and addressable components that called as modules.

Monolithic software is hard to grasp for the software engineer, hence it has now become a trend to divide the software into number of products. But there is a co-relation between the number of modules and overall cost of the software product. Following argument supports this idea

"Suppose there are two problems A and B with varying complexity. If the complexity of problem A is greater than the complexity of the problem B then obviously the efforts required for solving the problem A is greater than that of problem B. That also means the time required by the problem A to get solved is more than that of problem B."

The overall complexity of two problems when they are combined is greater than the sum of complexity of the problems when considered individually. This leads to divide and conquer strategy (according to divide and conquer strategy the problem is divided into smaller sub problems and then the solution to these sub problems is obtained. Thus dividing the software problem into manageable number of pieces leads to the concept of modularity. It is possible to conclude that if we subdivide the software indefinitely then effort required to develop each software component will become very small. But this conclusion is invalid because the total number of modules get increased the efforts required for developing each module also gets increased. That means the cost associated with each effort gets increased. The effort (cost) required for integration these modules will also get increased. The total cost required to develop such software product is shown in the figure



The above figure provides useful guideline for the modularity and that is – over modularity or the under modularity while developing the software product must be avoided. We should modularize the software but the modularity must remain near the region denoted by M

- Modularization should be such that the development can be planned easily, software increments can be defined and delivered, changes can be more easily accommodated and long term maintenance can be carried out effectively.
- **Meyer** defines five criteria that enable us to evaluate a design method with respect to its ability to define an effective modular system :
- **Modular decomposability**

  A design method provides a systematic mechanism for decomposing the problem into sub-problems. This reduces the complexity of the problem and the modularity can be achieved.
- **Modular composability**

  A design method enables existing design components to be assembled into a new system.
- **Modular understandability**

  A module can be understood as a standalone unit. It will be easier to build and easier to change.
- **Modular continuity**

  Small changes to the system requirements result in changes to individual modules, rather than system-wide changes.
- **Modular protection**

  An aberrant condition occurs within a module and its effects are constrained within the module.

3.3.3 Architecture

- Architecture means representation of overall structure of an integrated system.
- In architecture various components interact and the data of the structure is used by various components.
- These components are called system elements.
- Architecture provides the basic framework for the software system so that important framework activities can be conducted in systematic manner.
- In architectural design various system models can be used and these are

Model	Functioning
Structural Model	Overall architecture of the system can be represented using this model.
Framework Model	This model shows the architectural framework framework and corresponding applicability
Dynamic Model	This model shows the reflection of changes on the system due to external events.
Process model	The sequence of processes and their functioning is represented in this model.
Functional Model	The functional hierarchy occurring in the system is represented by this method.

3.3.4 Refinement

- Refinement is actually a process of elaboration.
- We begin with a statement of function (or description of information) that is defined at a high level of abstraction and reach at the lower level of abstraction. In each step (of the refinement), one or several instructions of the given program are decomposed into more detailed instructions.
- This successive decomposition or refinement of specifications terminates when all instructions are expressed in terms of any underlying computer or programming language.
- Abstraction and refinement are complementary concepts. Abstraction enables a designer to specify procedure and data and yet suppress low level details.
- Refinement helps the designer to reveal low-level details as design progresses.

3.3.5 Pattern

According to **Brad Appleton** the design pattern can be defined as-It is named nugget of insight which conveys the essence of proven solution to a recurring problem with 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.**

3.3.6 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 another 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 nodule.

3.3.7 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.

3.3.7.1 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.

3.3.7.2 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:

[1] Data coupling

The data coupling is possible by parameter passing or data interaction.

[2] Control coupling

The modules share related control data in control coupling.

[3] Common coupling

In common coupling common data or a global data is shared among the modules.

[4] Content coupling

Content coupling occurs when one module makes use of data or control information maintained in another module.

3.3.8 Refracting

Refreactoring ii necessary for simplifying the design without changing the function or behaviour.

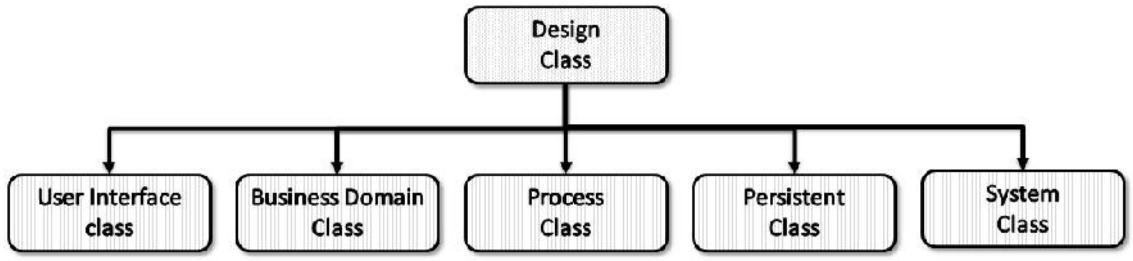
Fowler has designed refractoring as the process of changing a software system in such a way that the external behaviour of the design do not get changed,howeever the interanl structure gets improved".

3.3.9 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



[1] User Interface class
The user interface class defines all the abstractions that are necessary for Human Computer interface (HCI). The user interface class 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 -

❖ **Complete and efficient**
A design class must be properly encapsulated with corresponding attributes and methods. Design class must contain all those methods that are sufficient to achieve the intent of the class.

❖ **Primitiveness**
Methods associated with one particular class must perform unique service and the class should not provide another way to implement the same service.

❖ **High Cohesion**
A cohesive designed class must possess small, focused set of responsibilities and these responsibilities must be associated with all the attributes of that class

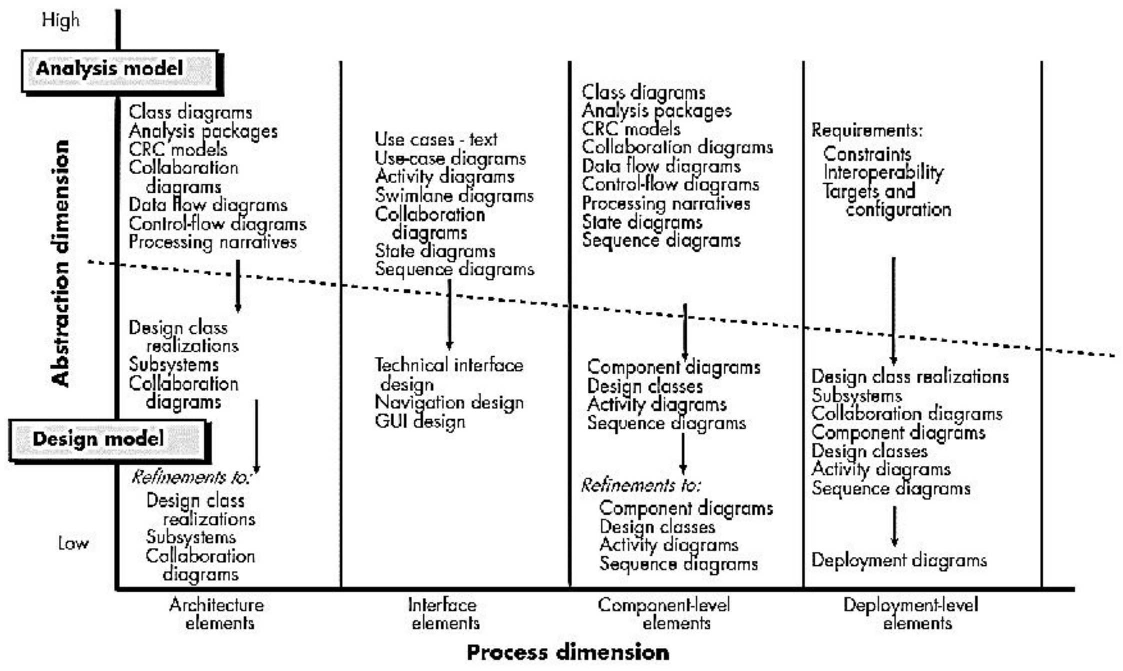
❖ **Low Coupling**

Design classes must be collaborated with manageable number of classes. If one design class is collaborated with all other design classes then the implementation, testing and maintenance of the system becomes complicated. The law of Demeter suggests that the one particular design class should send messages to the methods of neighboring design classes only.

Design Model

3.4 Design Model

- The **process dimension** denotes that the design model evolution due to various software tasks that get executed as the part of software process.
- The **abstract dimension** represents level of details as each element of analysis model is transformed into design equivalent.
- In following figure 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.

3.4.1 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.

3.4.2 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.

3.4.3 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 –

[1] User interface

By this interface user interacts with the system.

For example - GUI

[2] External interface

This is the interface of the system components with the external entities.

For example - Networking.

[3] Internal interface

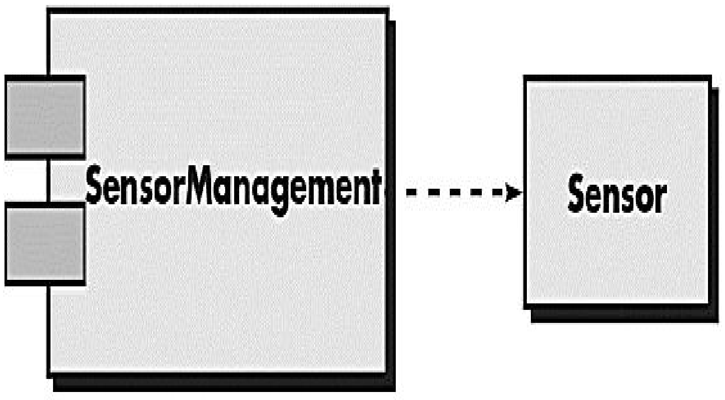
This is an interface which represents the inter component communication of the system.

For example - Two classes can communicate with each other by operations or by passing messages

3.4.4 Component Level Design Elements

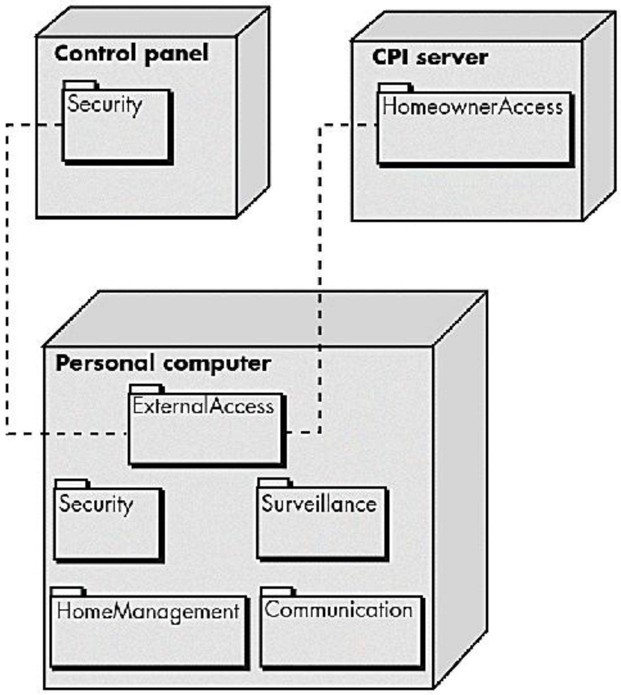
The component level design is more detailed design of the software system along5 with the specifications. The component level design elements describe the internal details1 of the component. In component level design all the local data objects, required data structures and algorithmic details and procedural details are exposed.

The below figure represents that component order makes use of another component form.



3.4.5 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.



3.5 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.

3.5.1 Heuristic evaluation

A heuristic evaluation is a usability inspection method for computer software that helps to identify usability problems in the user interface (UI) design. It specifically involves evaluators examining the interface and judging its compliance with recognized usability principles (the "heuristics").

[1] Visibility of system status:

The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

[2] Match between system and the real world:

The system should speak the user's language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

[3] User control and freedom:

Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

[4] Consistency and standards:

Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

[5] Error prevention:

Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

[6] Recognition rather than recall:

Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

[7] Flexibility and efficiency of use:

Accelerators—unseen by the novice user—may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

[8] Aesthetic and minimalist design:

Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

[9] Help users recognize, diagnose, and recover from errors:

Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

[10] Help and documentation:

Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large

Architectural Design

3.6 Architectural Design

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.

3.7 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.

Importance of software architecture

There are three reasons why the software architecture is so important?

[1] Software architecture gives the representation of the computer based system that is to be built. Using this system model even the stakeholders can take active part in the software development process. This helps in clear specification/understanding of requirements.
[2] Some early design decisions can be taken using software architecture and hence system performance and operations remain under control.
[3] The software architecture gives a clear cut idea about the computer based system which is to be built.

3.7.1 Structural Partitioning

The program structure can be partitioned horizontally or vertically.

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.

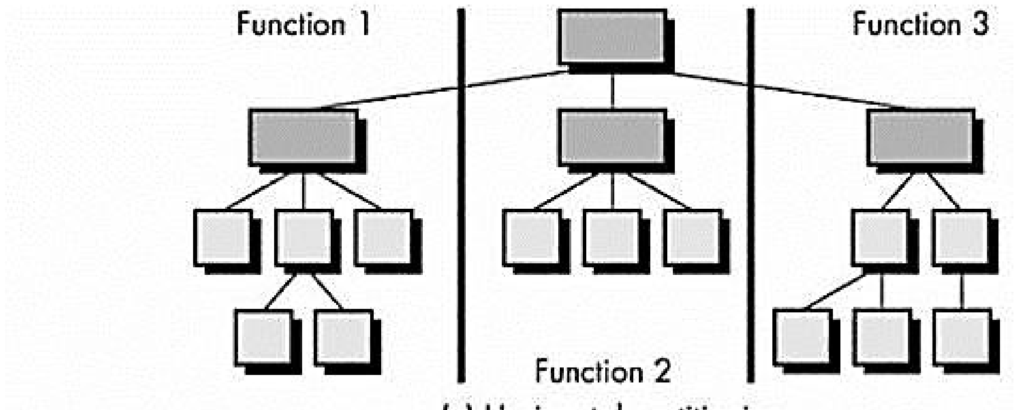
In horizontal partitioning the design making modules are at the top of the architecture.

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

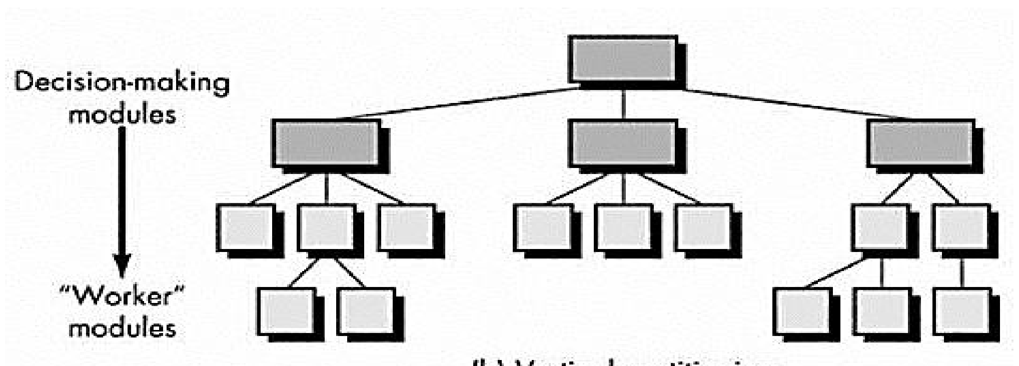
More data has to be passed across module interfaces which complicate the overall control of program flow.



(a) Horizontal partitioning

Vertical partitioning

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



(b) 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.

3.8 Data Design

Data design is basically the model of data that is represented at the high level of abstraction.

The data design is then progressively refined to create implementation specific representations.

Various elements of data design are

[1] Data object
The data objects are identified and relationship among various data objects can be represented using entity relationship diagrams or data dictionaries.

[2] Databases
Using software design model, the data models are translated into; data structures and databases at the application level.

[3] Data warehouses
At the business level useful information is identified from various databases and the data warehouses are created. For extracting or navigating the useful business information stored in the huge data warehouse then data mining techniques are applied.

Guideline for data design

[1] Apply systematic analysis on data
Represent data objects, relationships among them and data flow along with the contents.

[2] Identify data structures and related operations
For the design of efficient data structures all the operations that will be performed on it should be considered.

[3] Establish data dictionary
The data dictionary explicitly represents various data objects, relationships among them and the constraints on the elements of data structures.

[4] Defer the low-level design decisions until late in the design process
Major structural attributes are designed first to establish an architecture of data. And then low-level design attributes are established.

[5] Use information hiding in the design of data structures
The use of information hiding helps in improving quality of software design. It also helps in separating the logical and physical views.

[6] Apply a library of useful data structures and operations
The data structures can be designed for reusability. A use of library of data structure templates (called as abstract data types) reduces the specification and design efforts for data.

[7] Use a software design and programming language to support data specification and abstraction

The implementation of data structures can be done by effective software design and by choosing suitable programming language.

Architectural Style

3.9.1 Architectural Style

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.

3.9.1.1 Data Centered Architectures

In this architecture the data store lies at the center 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.

Data centered architecture possess 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.

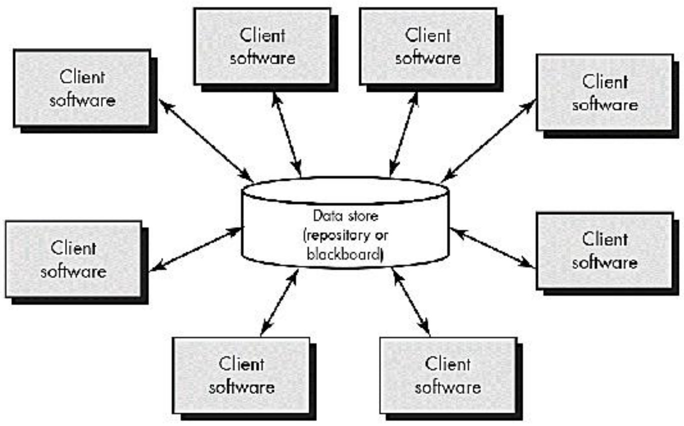
In data centered architecture the data can be passed among the components.

In data centered architecture,

Components are: Database elements such as tables, queries.

Communications are: By relationships

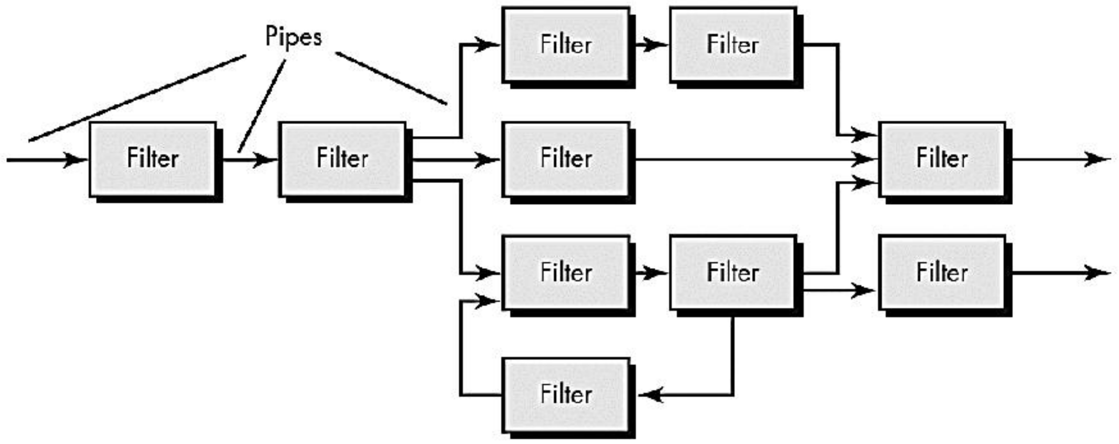
Constraints are: Client software has to request central data store for information.



3.9.1.2 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 neighboring filter.

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

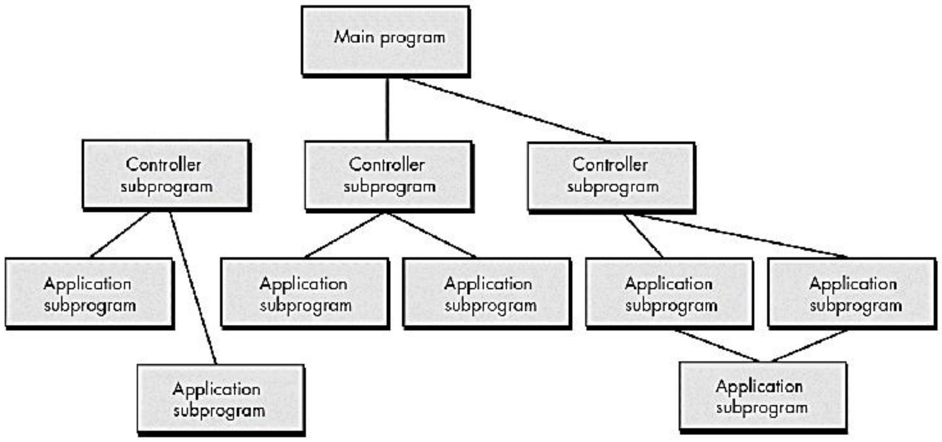


Pipes and filters

3.9.1.3 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.



3.9.1.4 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.

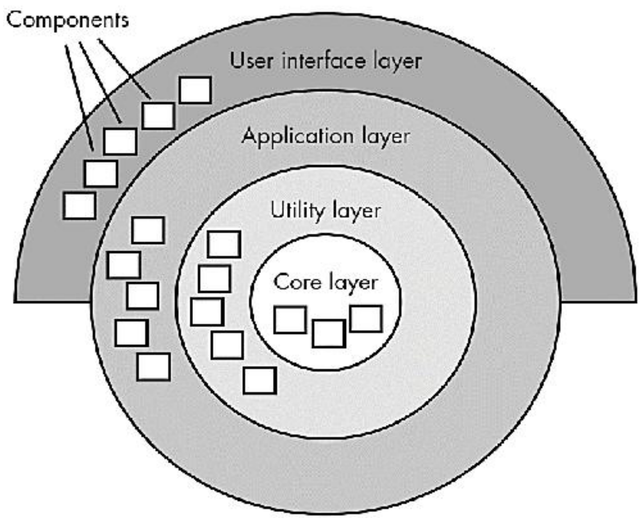
The object oriented decomposition is concerned with identifying objects classes, their attributes and the corresponding operations. There is some control models used to co-ordinate the object operations.

3.9.1.5 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.

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.



3.9.2 Architectural Patterns

In this section we will understand what are architectural patterns? The architectural pattern is basically an approach for handling behavioral characteristics of software systems. Following are the architectural pattern domains

1) Concurrency

Concurrency means handling multiple tasks in parallel. For example in operating system, multiple tasks are executed in parallel. Hence concurrency is a pattern which represents that the system components can interact with each other in parallel. The benefit of this pattern is that system efficiency can be achieved.

2) Persistence

Continuity in the data can be maintained by the persistence pattern. In other words the data used in earlier execution can be made available further by storing it in files or in databases. These files/databases can be modified in the software system as per the need. In object oriented system the values of all attributes various operations that are to be executed are persistent for further use. Thus broadly there are two patterns,

i. Database management pattern
ii. Application level pattern.

3) Distribution

Distribution pattern refers to the way in which the system components communicate with each other in distributed systems. There are two major problems that occur in distribution pattern

- ❖ The nature of interconnection of the components
- ❖ The nature of communication

These problems can be solved by other pattern called broker pattern. The broker pattern lies between server and client components so that the client server communication can be established properly. When client want some service from server, it first sends message to broker. The broker then conveys this message to server and completes the connection. Typical example is CORBA. The CORBA is a distributed architecture in which broker pattern is used.

3.10 Architectural Design

The architectural design process begins by representing the system in context.

As architectural design begins, the software to be developed must be put into context—

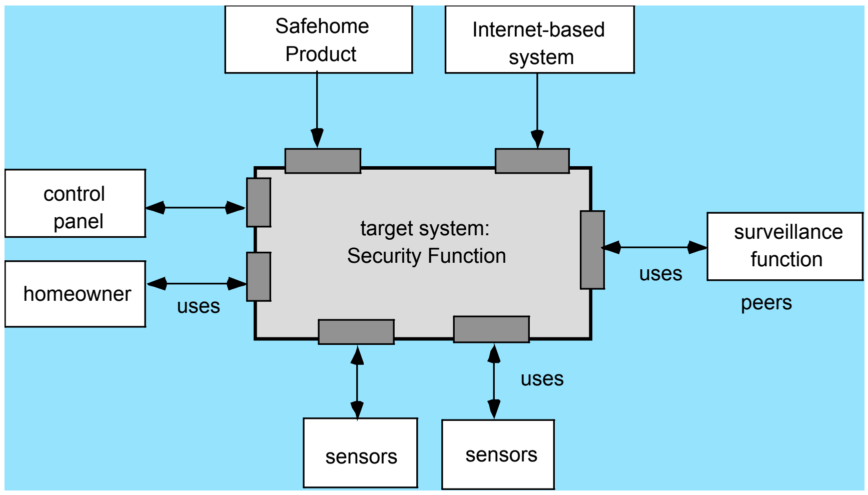
That is, the design should define the external entities (other systems, devices, people) that the software interacts with and the nature of the interaction. This information can generally be acquired from the requirements model and all other information gathered during requirements engineering. Once context is modeled and all external software interfaces have been described, you can identify a set of architectural archetypes. **An archetype is an abstraction (similar to a class) that represents one element of system behavior**. The set of archetypes provides a collection of abstractions that must be modeled architecturally if the system is to be constructed, but the archetypes themselves do not provide enough implementation detail. Therefore, the designer specifies the structure of the system by defining and refining software components that implement each archetype. This process continues iteratively until a complete architectural structure has been derived.

3.10.1 Representing the System in Context

Architectural context represents how the S/W interacts with entities external to its boundaries.

The design should define the external entities (other systems, devices, people) that the software interacts with and the nature of the interaction

At the architectural design level, a software architect uses an**architectural context diagram (ACD)** to model the manner in which software interacts with entities external to its boundaries.
The generic structure of the architectural context diagram is illustrated in Figure.



In figure, systems that interoperate with the target system (the system for which an architectural design is to be developed) are represented as

Super ordinate systems: those systems that use the target system as part of some higher-level processing scheme.

Subordinate systems—those systems that are used by the target system and provide data or processing that are necessary to complete target system functionality.

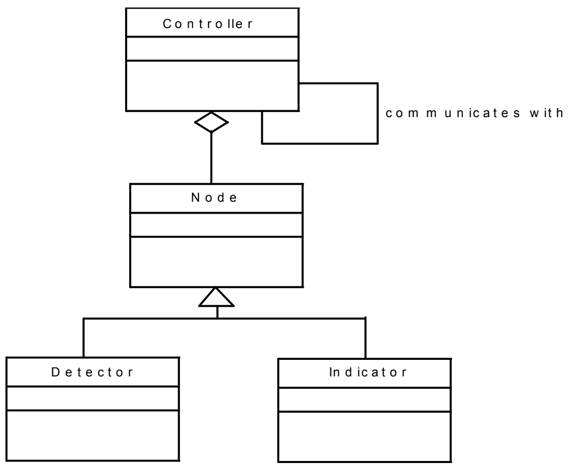
Peer-level systems—those systems that interact on a peer-to- peer basis (i.e., information is either produced or consumed by the peers and the target system.

Actors—entities *(people, devices)* that interact with the target system by producing or consuming information.

Each of these external entities communicates with the target system through an interface (the small shaded rectangles).

3.10.2 Defining Archetypes

- An *archetype* is an abstraction (similar to a class) that represents one element of system behavior
- The designer specifies the structure of the system by defining and refining software components that implement each archetype



An archetype is a class or pattern that represents a core abstraction that is critical to the design of an architecture for the target system.

In general, a relatively small set of archetypes is required to design even relatively complex systems.

Archetypes are the abstract building blocks of an architectural design.

In many cases, archetypes can be derived by examining the analysis classes defined as part of the requirements model. An archetype is a generic, idealized model of a person, object, or concept from which similar instances are derived, copied, patterned, or emulated. :

Node : Represents a cohesive collection of input and output elements of the home security function.

For example a node might be included of (1) various sensors and (2) a variety of alarm (output) indicators.

Detector : An abstraction that covers all sensing equipment that feeds information into the target system.

Indicator. An abstraction that represents all mechanisms (e.g., alarm siren, flashing lights, bell) for indicating that an alarm condition is occurring.

Controller. An abstraction that describes the mechanism that allows the arming (Supporting) or disarming of a node. If controllers reside on a network, they have the ability to communicate with one another.

3.10.3 Refining the Architecture into Components

As the software architecture is refined into components.

Analysis classes represent entities within the application (business) domain that must be addressed within the software architecture.

In some cases (e.g., a graphical user interface), a complete subsystem architecture with many components must be designed.

For Example : The SafeHome home security function example, you might define the set of top-level components that address the following functionality:

External communication management — coordinates communication of the security function with external entities such as other Internet-based systems and external alarm notification.

Control panel processing— manages all control panel functionality.

Detector management — coordinates access to all detectors attached to the system.

Alarm processing — verifies and acts on all alarm conditions

3.10.4 Describing Instantiations of the System

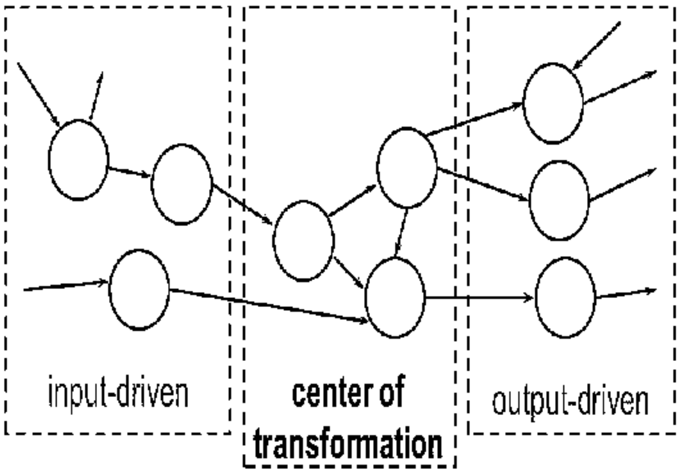
The architectural design that has been modelled to this point is still relatively high level. The context of the system has been represented Archetypes that indicate the important abstractions within the problem domain have been defined, The overall structure of the system is apparent, and the major software components have been identified. However, further refinement is still necessary. To accomplish this, an actual instantiation of the architecture is developed. It means, again it simplify by more details.

Architectural Mapping using Data Flow

3.11 Architectural Mapping using Data Flow

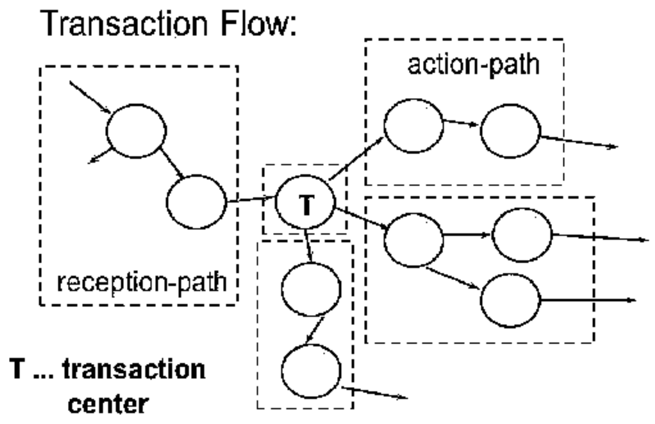
Transform flow

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



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.



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.

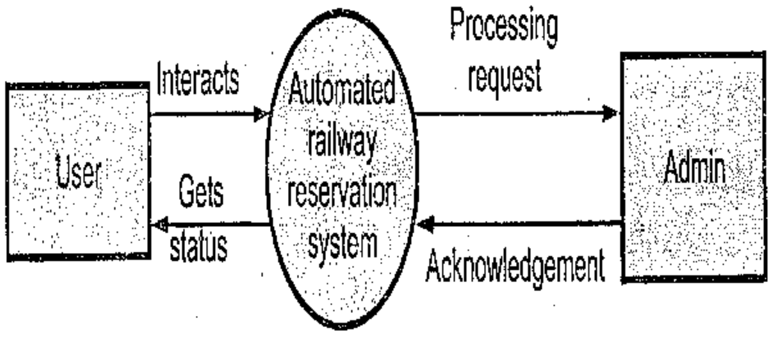
After registering himself/herself to the system, user must have to log in. He /she can make the enquiry about the trains either by entering the train number or by entering the source and destinations.

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.

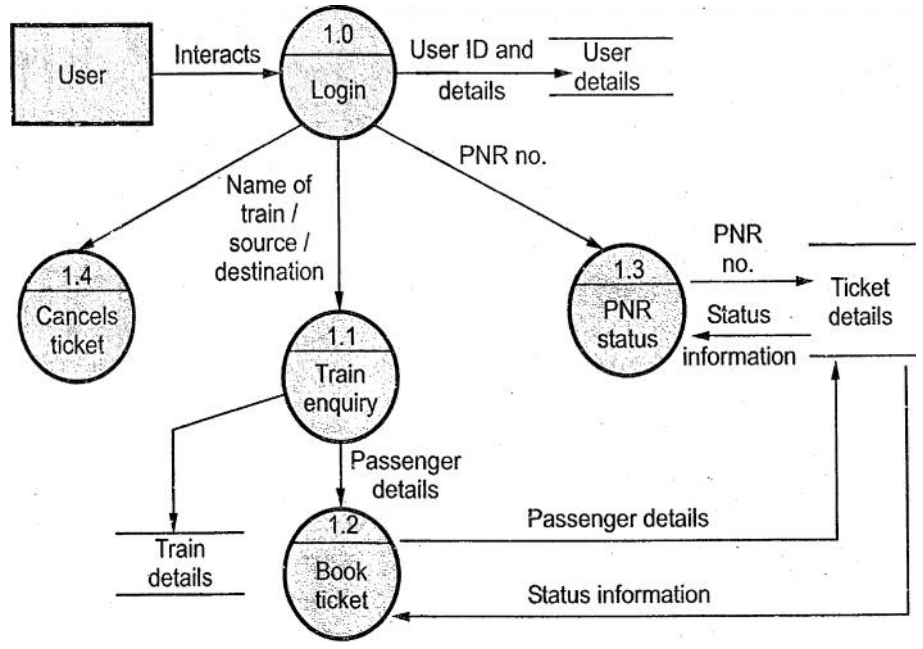
User can view the latest status of his reservations by simply entering the PNR number.

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.

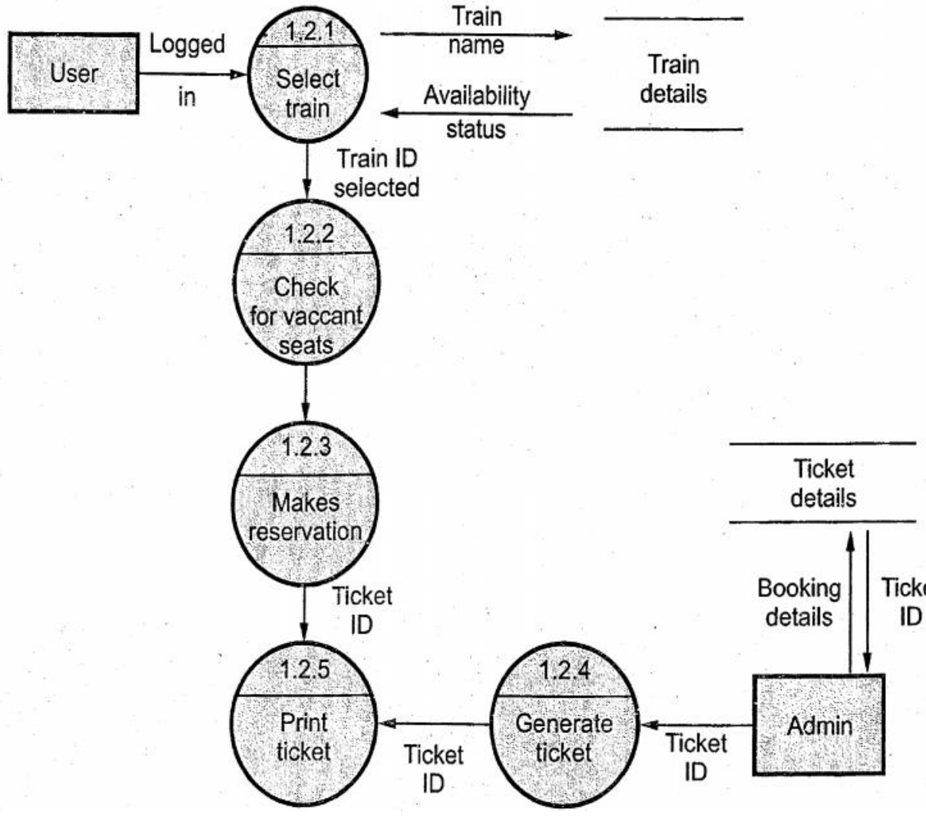
The Level 0, Level 1 and Level 2 DFD, Level 3 DFD are as shown below –



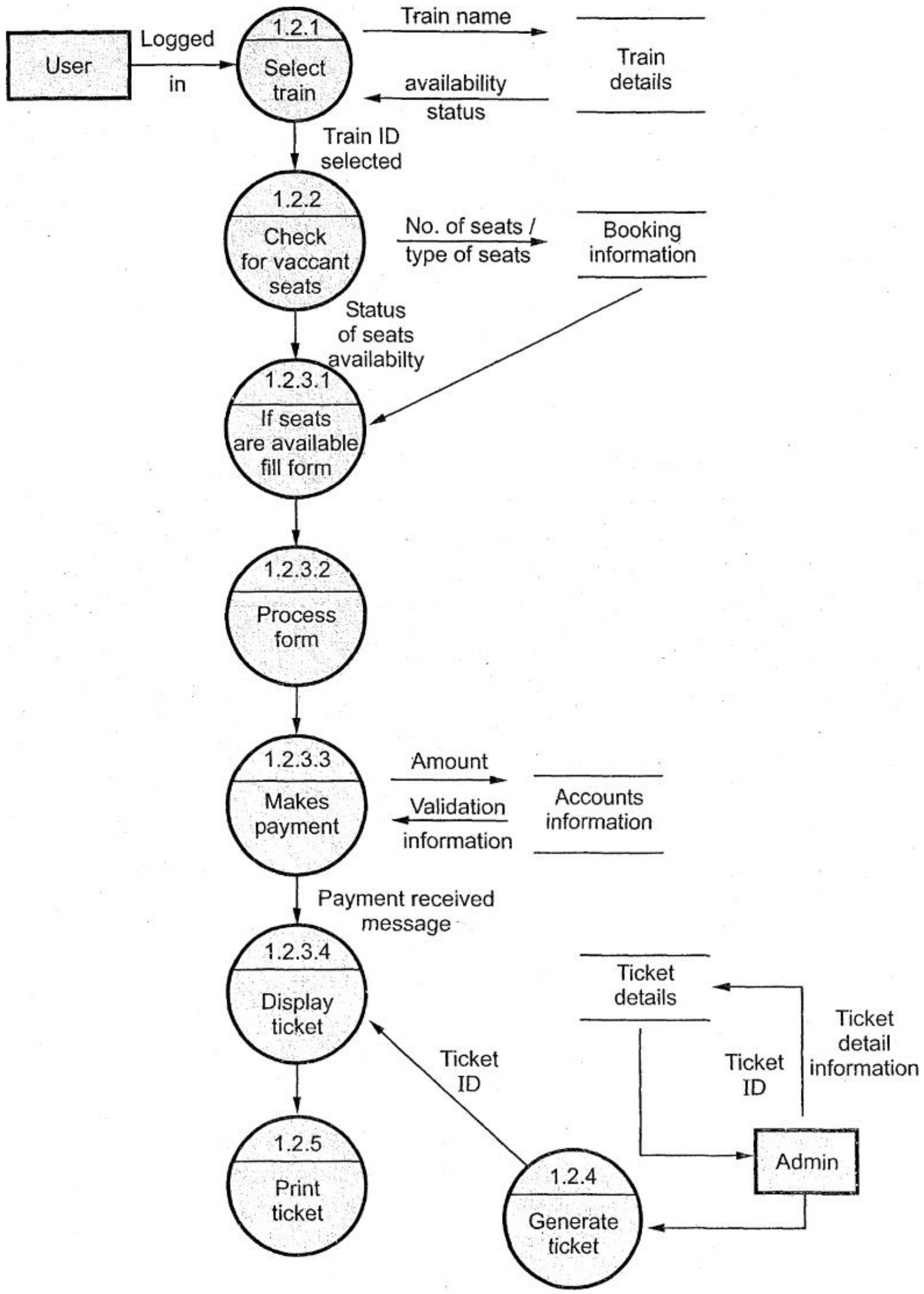
Context Level DFD (Level 0)



Level 1 DFD



Level 2 DFD for book ticket functionality



Level 3 DFD for "Make Reservation"

3.11.1 Transform Mapping

Transform mapping is a collection of design steps using which a DFD containing transform flow characteristics is mapped into specific architectural style.

Design steps for transform mapping

Step 1: Review fundamental system model to identify information flow

The fundamental system model can be represented by level 0 DFD and supporting information. This supporting information can be obtained from the two important documents called 'system specification' and 'software requirement specifications'. Both of them describe the information flow and structure at software interface.

Step 2: Review and refine the data flow diagram for software

The level 0 DFD is refined and the higher level DFD is drawn. Refer Level 1 DFD and Level 2 in which the Book Ticket functionality is reviewed and refined.

Step 3: Determine if the DFD has the transform or transaction flow characteristics

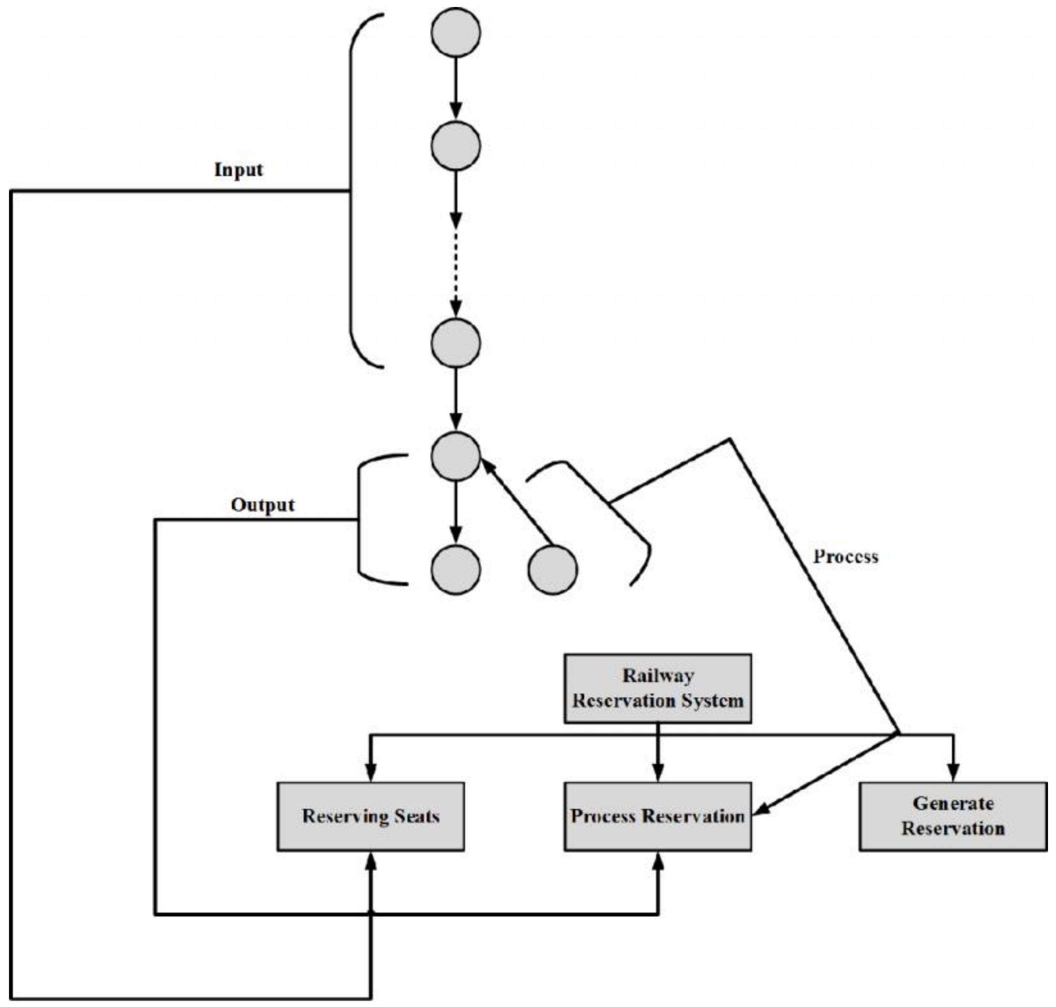
The information flow within the system is usually represented as transform flow. However, there can be dominance of transaction characteristics in the DFD. Based on characteristics of the DFD the transformation flow or transaction flow is decided.

Step 4: Isolate the transform center by specifying incoming and outgoing flow boundaries

After identifying that the DFD shows the transformation flow, we can easily separate | out the incoming and outgoing flow boundaries and the transform center can be identified.

Step 5: Perform first level factoring

After performing the first level factoring the program structure results in top down architecture where Top level components perform decision making



Low-level components perform most input, computation and output

Middle-level perform some control and some amount of work.

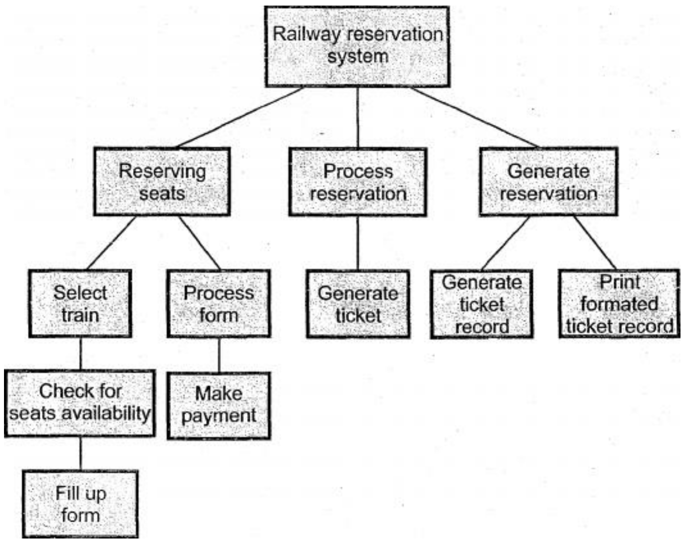
When transform flow is identified the DFD is mapped into call and return architecture.

Step 6: Perform second level factoring

In the second level factoring individual bubble of DFD is mapped into appropriate module within architecture.

There could be one-to-one mapping of bubble of DFD into the software module or two or three bubbles can be combined together to form a single software module.

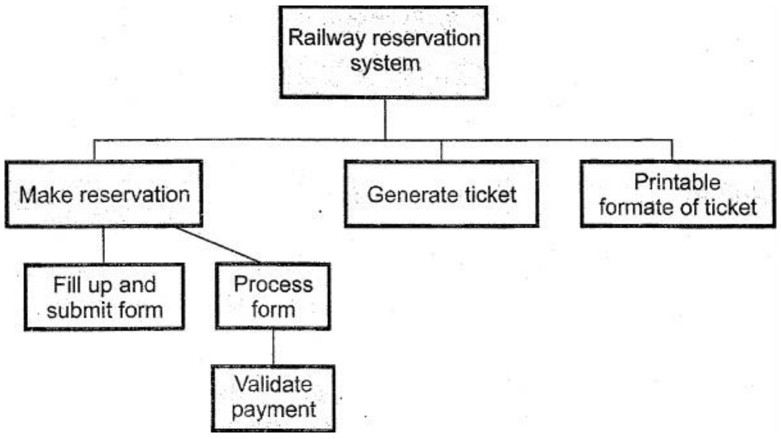
After performing the second level factoring the architecture serves as the first-iteration design.



Step 7: Refine the first iteration architecture using design heuristics for improved software quality

The first-iteration architecture can be refined by applying the module independency.

The modules can be exploded or imploded with high cohesion and minimum coupling. The refinement should be such that the structure can be implemented without -difficulty, tested without confusion and can be easily maintained.



3.12 Concept of User Interface Design

Any computer based system requires two things: its computational ability and functionality. The computer based systems are typically used by casual users. Hence the purpose of user interface design is to have effective communication medium between the computerized system and the user. This kind of interface design is necessary because of many reasons such as: software is difficult to use, the use of software forces the user to make mistakes due to lack of understandings about the system.

While designing for user interface the very first step is to identify user, tasks and environmental techniques. Based on user tasks, user scenarios are prepared. Such user scenarios help to design user interface objects and corresponding actions. This set of objects and actions help in deciding the screen layout. Once such a layout has been prepared, appropriate icons, screen texts and menu items can be placed at respective positions.

3.13 Golden Rules

Thao Mandel has proposed three golden rules for user interface design **-**

1) Place the user in control
2) Reduce the user's memory load
3) Make the interface consistent.

3.13.1 Place the User in Control

While analyzing any requirement during requirement analysis the user often demands for the system which will satisfy user requirements and help him to get the things done. That means the user always wants to control the computerized system. Following are the design principles that allow user to control the system:

❖ **Define the interaction modes in such a way that user will be restricted from doing the unnecessary actions**
Interaction mode means the current state in which user is working and being in such a mode user is supposed to do the related tasks only if the user has to perform unnecessary actions at such time then the GUI becomes frustrating.

❖ **The interaction should be flexible**
The user interaction should be flexible. For example in the Microsoft power point slide show the slide transition is possible using mouse clicks as well as using keyboard. Such flexibility allows any user to operate the system as per his comfort.

❖ **Provide the facility of ’undo’ or ’interruption1 in user interaction**
This is the feature which allows the user to correct himself whenever necessary without loosing his previous work. For example: In the software like 'Paint' one can draw some objects and perform 'redo' or 'undo' actions for performing his desired drawing.

❖ **Allow user to customize the interaction**
It is observed that in while handling user interface certain actions need to be done repeatedly. It saves the time if these actions are collected in a Macro.

❖ **Hide technical details from the user**
This feature is essential for a casual user. User should not be aware of system commands, operating system functions or file management functions. For example while printing some document instead of giving the command for printing if user clicks on the icon indicating print then it becomes convenient for a casual user to take the print out.

❖ **The objects appearing on the screen should be interactive with the user**
This also means that user should be in position to adjust the object appearing on the screen. For example in ’Paint’ one should be able to stretch the oval or edit the text written in the object.

3.13.2 Reduce the User's Memory Load

If the user interface is good then user has to remember very less. In fact the design should be such that the system remembers more for the user and ultimately it assists the user to handle the computer based systems. Following are the principles suggested by Mandel to reduce the memory load of the user.

❖ **Do not force the user to have short term memory**
When user is involved in complex tasks then he has to remember more. The user interface should be such that the user need not have to remember past actions and results. And this can be achieved by providing proper visual interface.

❖ **Establish meaningful defaults**
Meaningful default options should be available to the user. For example in the Microsoft word the user can set the default font size as per his choice. Not only this, there should be some reset option available to the user in order to get back the original values.

❖ **Use intuitive shortcuts**
For quick handling of the system such short cuts are required in the user interface. For example control+s key is for saving the file or control+o is for opening the file. Hence use of meaningful mnemonics is necessary while designing an interface.

❖ **The visual layout of the interface should be realistic**
When certain aspect/feature of the system needs to be highlighted then use of proper visual layout helps a casual user to handle the system with ease. For example in an online purchase system if pictures of Master card/Visa card are given then it guides the user to understand the payment mode of online purchasing.

❖ **Disclose the information gradually**
There should not be bombarding of information on user. It should be presented to the user in a systematic manner. For instance one can organize the information in hierarchical manner and narrate it to the user. At the top level of such hierarchical structure the information should be in abstract form and at the bottom level more detailed information can be given.

3.13.3 Make the Interface Consistent

The user interface should be consistent. This consistency can be maintained at three levels such as

- ⇨ The visual information (all screen layouts) should be consistent throughout and it should be as per the design standards.
- ⇨ There should be limited set of input holding the non-conflicting information.
- ⇨ The information flow transiting from one task to another should be consistent. Mandel has suggested following principles for consistent interface design.

❖ **Allow user to direct the current task into meaningful manner**
This principle suggests that create a user interface with proper indicators on it so that user can understand his current task and how to proceed for new task.

❖ **Maintain consistency across family of product**
If an application comes in a packaged manner then every product of that application family should posses the consistent user interface. For example Microsoft Office family has several application products such as MS WORD, MS Power Point, MS Access and so on. The interface of each of these products is consistent (of course with necessary changes according to its working!). That means there is a title bar, menu bar having menus such as File, Edit, View, Insert,

Format, Tools, Table, Window, Help on every interface. Then tool bars and then design layouts are placed on the interface.

❖ **If certain standards are maintained in previous model of application do not change it until and unless it is necessary**

Certain sequence of operation becomes a standard for the user, then do not change these standards because user becomes habitual with such practices. For instance control + s is for saving the file then it has become a standard rule now, if you assign different short cut key for saving then it becomes annoying to the user

User Interface Analysis and Design

3.14 User Interface Analysis and Design

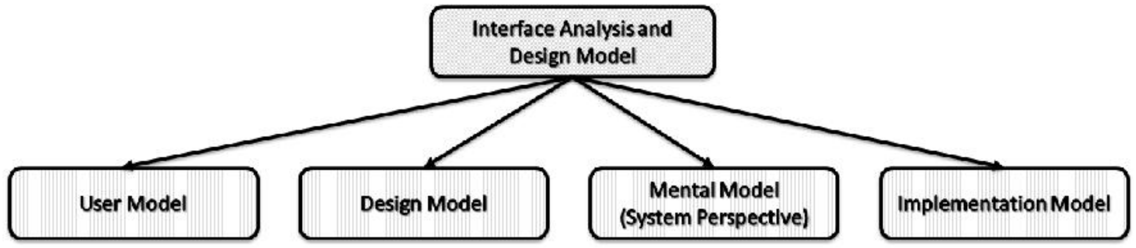
User interface analysis and design can be done with the help of following steps.

1) Create different models for system functions.
2) In order to perform these functions identify the human-computer interface tasks.
3) Prepare all interface designs by solving various design issues.
4) Apply modern tools and techniques to prototype the design.
5) Implement design model.
6) Evaluate the design from end user to bring quality in it.

These steps can be broadly categorized in two classes.

1) Interface analysis and design models
2) The process

3.14.1 Interface Analysis and Design Model


Interface Analysis and Design Model
User Model
Design Model
Mental Model (System Perspective)
Implementation Model

Typically there are four types of models that can be prepared in interface analysis and design. Let us understand each of them

❖ **User model**

To design any user interface it is a must to understand the user who is using the system. Hence in this model the profile of user is prepared by considering age, sex, educational, economic and cultural background. The software engineer prepares user model. Globally there are three kinds of users

User	Description
Novice	The user with very little knowledge of the computer who simply knows semantics (simply the wants) of the system and does not know the implementation of such system.
Knowledgeable and intermittent	The user having knowledge about, the semantics of the system as well as having little knowledge of syntactic of the system.

Knowledgeable and frequent	The user with good semantic as well as syntactic knowledge of the system.

❖ **Design Model**

It consists of data, architectural, interface and procedural representation of the software. While preparing this model the requirement specification must be used properly to set the system constraints. Software engineer prepares the design model of the interface.

❖ **Mental Model (system perception)**

The user model is the representation of what user thinks about the system? Basically any user interface design is heavily dependent upon the description obtained from the user about his wants and needs. If the user is knowledgeable then more complete description of the system can be obtained than that of novice user.

❖ **Implementation model**

The implementation model generates the look and feel of interface. This model describes the system's semantic and syntax. It is very necessary to match the implementation model with the user's mental model then only user can feel comfortable with the developed system.

Finally the interface designer has to resolve any differences within these models. The supreme principle that has to be followed in interface analysis and design method is that: know the user and know the task

3.14.2 The Process

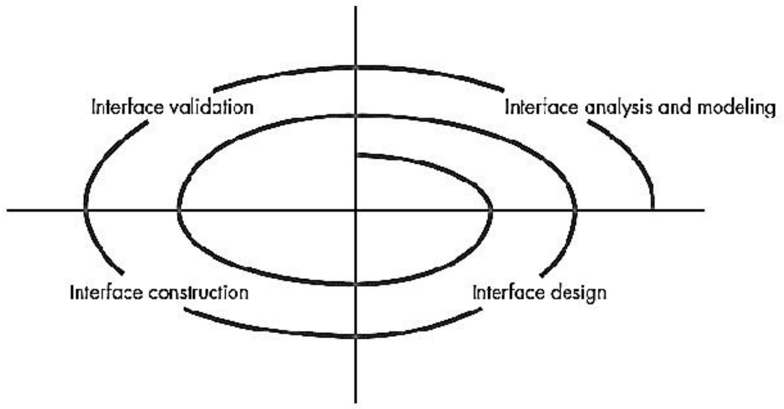
The user interface analysis and design process can be implemented using iterative Spiral model. It consists of four framework activities.

1) Environment analysis and modelling
2) Interface design
3) Implementation
4) Interface validation

As shown in the above figure each of these tasks can be performed more than once. At each pass around the system more requirements can be elaborated and detailed design can be performed.

[1] Environment analysis and modelling

In this phase three major factors are focused ie. User, task and environment. First of all the user profile is analyzed to understand the user and to elicit the requirements, then the tasks that are required to carry out desired functionality are identified and analyzed. The analysis is made on user environment which involves the physical work environment. Finally analysis model is created for the interface. This model serves as a basis for the design of user interface.



Interface Analysis and Design Process

[2] Interface design

The interface design is a phase in which all the interface objects and corresponding actions of each task are defined.

[3] Implementation

The implementation phase involves creation of prototype using| which the interface scenarios can be evaluated. To accomplish the construction of user interface some automated tools can be used.

[4] Validation

The goal of validation is to validate the interface for its correct performance. During validation it is tested whether all the user requirements gel satisfied or not. The purpose of validation is also to check whether the interface is easy to learn and easy to use.

3.15 Interface Analysis

Before proceeding for interface design it is necessary to understand the problem.

Understanding the problem means understanding

1) The people or user who actually interacts with the system.
2) The task that are performed by the end user for interacting the system
3) The contents of the interface that will be displayed to the user.
4) The environment in which the task will be conducted.

3.15.1 User Analysis

Following are the ways by which one can learn what the user wants from the user interface -

User interviews:

This is the most effective technique in which some representatives from software team meet the end user to better understand the user needs, motivations and many other issues. Meetings are conducted for this purpose.

Sales input

Sales people interact with the users regularly and collect the information. Based on this information the users are categorized in particular groups and thereby their requirements are better understood.

Marketing input

Market analysis is made in order to understand the usage of software.

Support input

Support staff keeps a regular interaction with the user interaction for knowing the certain things like the likings and dis-likes of the users, which features are easy to use and so on.

Following is a set of questions that will help the interface designer better understand the users of a system -

1) Are user trained professionals, technicians, or worker?
2) Are the users capable of learning from written material? OR they require any classroom training?
3) What is the formal education of the user?
4) Are users aware of using keyboard?
5) What is the age group of user?
6) Will user be represented dominantly by one gender?
7) Does the user work in office hour or do they work until the job is done?
8) What kind of compensation will be given to the user for the work they perform?
9) Will the user make use of the software frequently or occasionally?
10) What is the primary spoken language among the users?
11) What will happen if the user makes a mistake in handling the system?
12) Are the expert users addressed by the system?
13) Do users want to know the technology used behind the interface?

The answers to these questions help in understanding the end-user.

3.15.2 Task Analysis and Modelling

In task analysis following questions are answered

[1] In specific situation what work the user should perform?
[2] When user performs the work what are the tasks and sub tasks that should be performed?
[3] When user performs the work, what are all those problem domain objects that user will manipulate?
[4] What is the hierarchy of the task?
[5] What is the workflow for accomplishing the task?

In order to answer these questions following techniques are used –

1) Use-cases

Use cases describe the manner in which the actor interacts with the system. Use cases always show how the end user performs specific work related task. Use cases are mostly written in informal way i.e. in paragraphs.

2) **Task elaboration**

For task elaboration the functional decomposition or the stepwise refinement of the processing task is done. The task analysis for interface design adopts elaborative approach. During the task elaboration, identification and classification of tasks is performed.

3) **Object elaboration**

The software engineers examine the use case and the other information obtained from user and extract the information about the physical objects. These objects are further classified into classes. The attributes of each class are defined and evaluation of actions will identify the list of operations.

4) **Workflow analysis**

When there are many users who are simultaneously using the system then for understanding the flow of the series of tasks conducted the work flow analysis technique is applied. Workflow processes can be easily modelled using swim lane diagram. In swim lane diagram, the modelling is done for the classes who are responsible for carrying out the activities.

The reservation system makes use of some important classes such as Booking service, Passenger service and Finance service. Following are the set of activities **-**

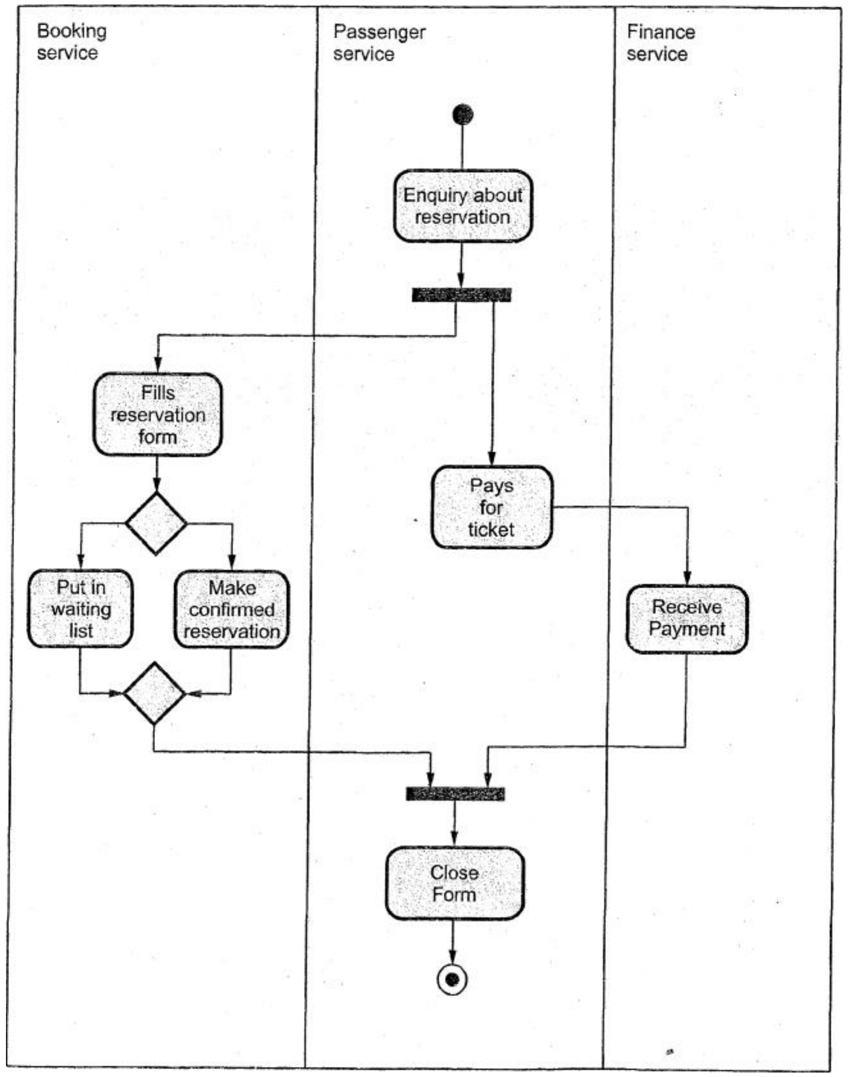
- ❖ Passenger makes enquiry about reservation.
- ❖ He has to fill up the form. The passenger will fill the up the information such as source and destination city, date and time of travel, number of passengers and type of reservation.
- ❖ If the desired seats are available then he gets the confirmation for the; reservation otherwise he will be put in the waiting list.
- ❖ If the confirmed reservation is made, the passenger pays for the ticket. 0 The finance service will receive the payment.
- ❖ The booking service will issue the tickets.

5) **Hierarchical representation**

In workflow analysis process elaboration occurs. After establishing the workflow analysis, the task hierarchy for each user type should be specified. In this hierarchical representation, there is a stepwise representation of each task identified for each user. For example

Fills reservation form

- ❖ Provide the personal information
    1. Specify name(s)
    2. Specify age
    3. Desired source city
    4. Desired destination city
    5. Date and time Reservation details
    6. Name(s)
    7. Age
    8. Source city
    9. Destination city
    10. Date and time of train
    11. Seat numbers
    12. Type of reservation
    13. Name of the train



3.16 Interface Design

After interface analysis all the tasks and corresponding actions are identified. Interface design is an iterative process in which each design process occurs more than once. Each time the design step gets elaborated in more detail. Following are the commonly used interface design steps **-**

1. During the interface analysis step define interface objects and corresponding actions or operations.
2. Define the major events in the interface. These events depict the user actions. Finally model these events.
3. Analyze how the interface will look like from user's point of view.
4. Identify how the user understands the interface with the information provided along with it.

While designing the interface software engineer communicates the user and according to his thinking about the interface, software engineer draws the sketches. Get [\ approved from the user and then work on defining objects and corresponding actions, While designing the interface the designer has to follow

- ➪ Golden rules
- ➪ Model the interface
- ➪ Analyze the working environment

Analysis of Display Content

The contents that get displayed as a user interface are called the display contents.

The display contents can be spreadsheets, 3D models, pictures or images, a graph or specialised information such as audio or video files.

During this analysis step, the format and aesthetics of the display contents are considered.

For determining the format and the aesthetics of the contents following questions can be asked or answered.

- **a)** Is it possible for the interface to display various types of data in a consistent manner on geographic location?
- **b)** Is it possible to customize the screen location with respect to the contents?
- **c)** If the large report is partitioned properly for the sake of understanding?
- **d)** Is the summary of large amount of data available directly?
- **e)** Will the graphical output be displayed properly so that it will fit to the display device?
- **f)** Is there a use of sophisticated color combination?
- **g)** Are the errors and warning messages presented to the user in interactive manner?

Answers to these questions will help in gathering the requirements for the presentation of the contents.

Analysis of Work Environment

The analysis of work environment is very important.

In some applications the user interface for the computer based systems is placed in very user friendly environment. In such a situation, there may be interactive presentation of the information, proper lighting, good display height, easy keyboard access, proper sitting arrangement and so on. In such work environment using the application becomes an enjoyable activity.

But there are some workplaces in which there may be insufficient light, lot of noise and distractions, unavailability of keyboard or mouse interfacing and so on. Using the application in such environment becomes difficult and frustrating.

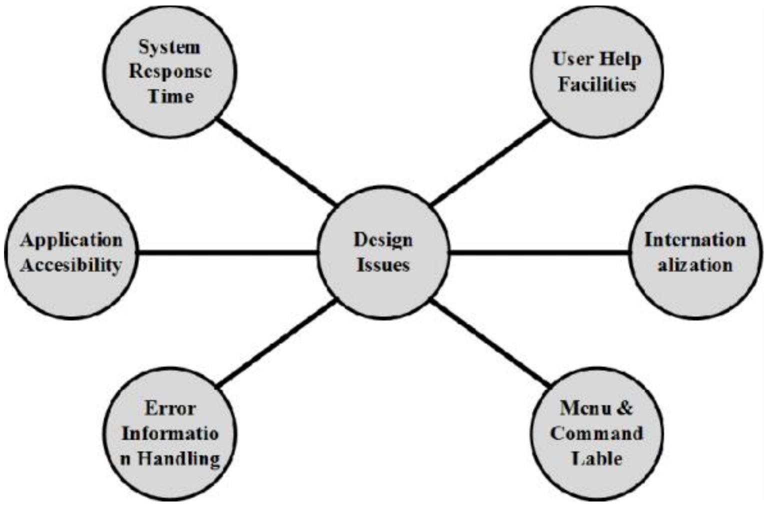
In addition to these physical factors, consideration of work place culture is extremely important. These factors are raised by following questions –

- ➪ Will more than one person access the same information before providing the input?
- ➪ Will the system interaction be measured in terms of some measure? For instance: Time required for processing of data.
- ➪ Will the system provide some kind of support to the user for handling it?

These issues must be solved before the completion of interface design phase.

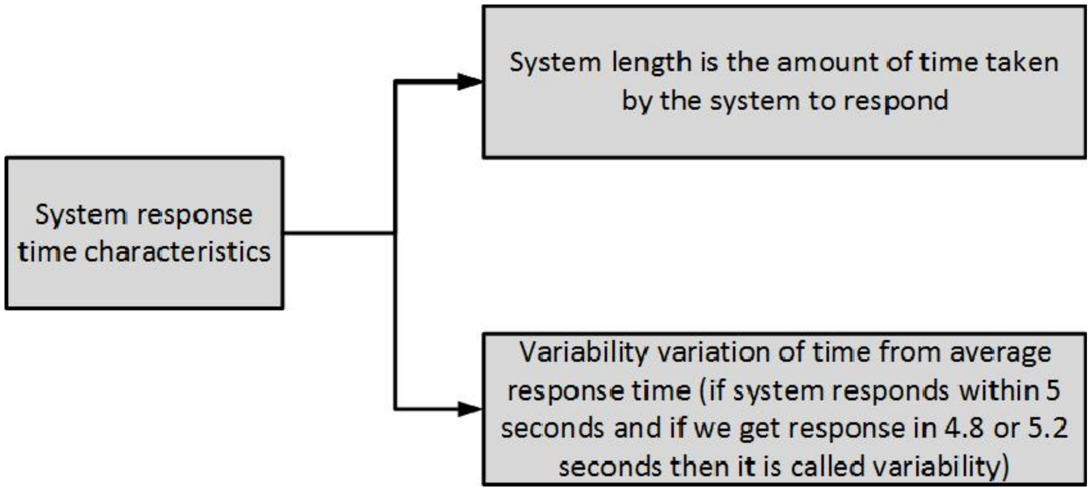
Design Issues

There are four important interface design issues that are depicted by following figure



System Response Time

Any user hate the delayed response time. Response time is the amount of time taken by the system to respond from the point at which user performs some control action (such as clicks on some point or press some key on the keyboard.


System response time characteristics
System length is the amount of time taken by the system to respond
Variability variation of time from average response time (if system responds within 5 seconds and if we get response in 4.8 or 5.2 seconds then it is called variability)

ABC has encountered error 1033. Now such type of error messages do not direct the user about what went wrong. Following are the characteristics for presenting errors.

The error message should be in a language which user can understand. For example: Remote server not responding.

There should be some useful message along with error in order to recover from the error. For example: Press enter to exit

There should be some audible or visual cue along with the error message. For I example: beep sound or highlighting back ground of the text.

There should not be any negative impact of error on the user. For example: File XYZ.SYS has been deleted such error will cause frustration on the user.

The wording of the error should be carefully used and it should not blame user because nobody likes to get criticized

Thus error messages should be so effective that they should bring qualitative interaction between the user and the system.

Menu and command labelling

There are typically two ways by which the user handles the system i.e. keyboard and mouse. The system can be handled using commands by the power user (i.e. knowledgeable and frequent users) whereas any ordinary user makes use of GUI and he prefers not to use any command as such. There are number of design issues for typed command and menu labels.

Command related design issues	Menu label related design issues
In which form the command will be?	Will there be any menu for corresponding command?
Will the commands be difficult to learn and remember?	Are the menus self-explanatory
What to do if the command is forgotten?	Is there any consistency between menu and submenus
Is there any abbreviation for the command? Or can they be customized?	

Application accessibility

In modern era, Computer application are used by everybody and everywhere. Hence software engineers must develop a mechanism by which the most frequently required applications must be availed easily.

The software is most important entity for the users who are physically challenged. This can be used by them for moral, legal and business reasons.

Accessibility guidelines are used for while developing the applications. These guidelines are mostly used by the web applications. The guideline assist in designing the interfaces used within the application.

The accessibility guideline also provides the guideline for assistive technology that addresses the needs of those visual, hearing, speech and mobility impairments.

Internationalization

There are situations in which the user interface is developed for localized purpose and for local language. If the same interface is required in other side of the country then existing interface is used with more or less modifications.

The real challenge is to develop globalized software. That means the user interfaces should be designed to accommodate generic core functionality. This functionality can be used by any user belonging to any country.

- Localized user interface is useful to only localized market.
- There are many internationalization guidelines that are available for software developers.

- These guidelines address the design and implementation issues.
- The Unicode standard has been developed to handle the challenges of managing the natural languages with lots of characters and symbols.

Component Level Design

3.16 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.

3.17.1 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.

Basic Design Principle

There are four design principles that are used during the component level design. These principles are

1. **The Open-closed Principle**

   This principle states that - A module or a component should be open for extension and j should be closed for modification. A designer should design a component in such a) manner that some functionalities can added to it if required but in doing so there \ should not be any change in the internal design of the component itself.

2. **The Liskov Substitution Principle**

   This principle states that - subclasses should be substitutable for their base classes. This • principle is proposed by Barbara Liskov. A component that contains the base class and if that component works properly then the same component should work properly even if the derived class of that base class is used by the component as a substitute.

3. **Dependency Inversion Principle**

   The components should be dependant on the other abstract components and not on the concrete component. This is because if some component has to be extended then it becomes easy to enhance it using other abstract component instead of concrete component.

4. **The interface Segregation Principle**

   Many client specific interfaces are better than one general purpose interface. According to this principle, designer should create specialized interfaces for each major category. So that the clients can access the interfaces based on the category. If a general purpose interface is created then multiple clients may try to access it for different operations at the same time.

Component Level Design Guideline

Ambler suggested following guideline for conducting component level design -

1. **Components**

   Components are the part of architectural model. The naming conventions should be established for components. The component names should be 2 specified from the problem domain. These names should be meaningful.

2. **Interfaces**

   Interfaces serve important role in communication and collaboration. The interface should be drawn as a circle with a solid line connecting it to another element. This indicates that the element to which it is connected offers the interface. The interface should flow from left side of component. Only important interfaces must be drawn.

3. **Dependencies and interfaces**

   The dependencies must be shown from left to right. The inheritance should be shown from bottom to top i.e. from derived to base class. The component interdependencies should be represented via interfaces.

Cohesion

In component level design for object oriented systems, the cohesion means a class or a component that consists of data and operations that are closely related to each other and related to the class itself. Various types of cohesions are

1. **Functional**

   In this type of cohesion the each module performs only one component.

2. **Layer**

   This type of cohesion is used in packages, components and classes. In this type of cohesion the higher level layer access the functionalities of the lower levels but the lower level layers do not access the functionalities of the higher level layers.

3. **Communicational**

   When all the operations of the class access the same set of data then this type of cohesion occurs.

4. **Sequential**

   In this type of cohesion, the components are grouped together in such a manner that the output of one operation is provided as input to another operation. So that all the operations execute in some specific sequence.

5. **Procedural**

   In this cohesion the procedures are invoked immediately one after the other.

6. **Temporal**

   The cohesion in which the operations specify specific behavior is called temporal cohesion.

7. **Utility**

   The components, classes or operations are grouped together if they perform some specific operation otherwise they are not related with each other.

Coupling

Coupling is the manner and degree of interdependence between software modules; a measure of how closely connected two routines or modules are; the strength of the relationships between modules.

Coupling is usually contrasted with cohesion. Low coupling often correlates with high cohesion, and vice versa. Low coupling is often a sign of a well-structured computer system and a good design, and when combined with high cohesion, supports the general goals of high readability and maintainability.

Coupling is a mechanism of degree to which the classes are connected to one another.

Various types of coupling are

Types of coupling

Coupling can be "low" (also "loose" and "weak") or "high" (also "tight" and "strong"). Some types of coupling, in order of highest to lowest coupling, are as follows:

Procedural programming

A module here refers to a subroutine of any kind, i.e. a set of one or more statements having a name and preferably its own set of variable names.

[1] Content coupling (high)

Content coupling (also known as **Pathological coupling**) occurs when one module modifies or relies on the internal workings of another module (e.g., accessing local data of another module).

Therefore changing the way the second module produces data (location, type, timing) will lead to changing the dependent module.

[2] Common coupling

Common coupling (also known as **Global coupling**) occurs when two modules share the same global data (e.g., a global variable).

Changing the shared resource implies changing all the modules using it.

[3] External coupling

External coupling occurs when two modules share an externally imposed data format, communication protocol, or device interface. This is basically related to the communication to external tools and devices.

[4] Control coupling

Control coupling is one module controlling the flow of another, by passing it information on what to do (e.g., passing a what-to-do flag).

[5] Stamp coupling (Data-structured coupling)

Stamp coupling occurs when modules share a composite data structure and use only a part of it, possibly a different part (e.g., passing a whole record to a function that only needs one field of it).

This may lead to changing the way a module reads a record because a field that the module does not need has been modified.

[6] Data coupling

Data coupling occurs when modules share data through, for example, parameters. Each datum is an elementary piece, and these are the only data shared (e.g., passing an integer to a function that computes a square root).

[7] Message coupling (low)

This is the loosest type of coupling. It can be achieved by state decentralization (as in objects) and component communication is done via parameters or message passing

[8] No coupling

Modules do not communicate at all with one another.

Object-oriented programming

[1] Subclass Coupling

Describes the relationship between a child and its parent. The child is connected to its parent, but the parent is not connected to the child.

[2] Temporal coupling

When two actions are bundled together into one module just because they happen to occur at the same time.

In recent work various other coupling concepts have been investigated and used as indicators for different modularization principles used in practice

Disadvantages of Coupling

Tightly coupled systems tend to exhibit the following developmental characteristics, which are often seen as disadvantages:

1. A change in one module usually forces a ripple effect of changes in other modules.
2. Assembly of modules might require more effort and/or time due to the increased inter-module dependency.
3. A particular module might be harder to reuse and/or test because dependent modules must be included.

3.17.2 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.

Structured Programming

There are three constructs of structured programming

1. Sequence
2. Condition
3. Repetition

Sequence denotes a linear processing of the statements. Condition provides the facility to test the logical conditions (For example if-then-else conditions). Repetition is used to denote the looping.

Following are the advantages of using structured programming constructs -

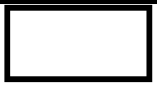
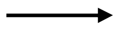
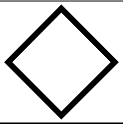
The structured programming constructs reduces program complexity.

It enhances readability, testability and maintainability of the procedure.

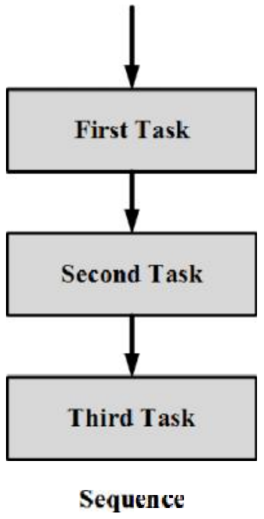
The structured programming constructs are the logical chunks that allow a reader to recognize procedural element from each programming module. This ultimately enhances the readability of the program.

Graphical Design Notations

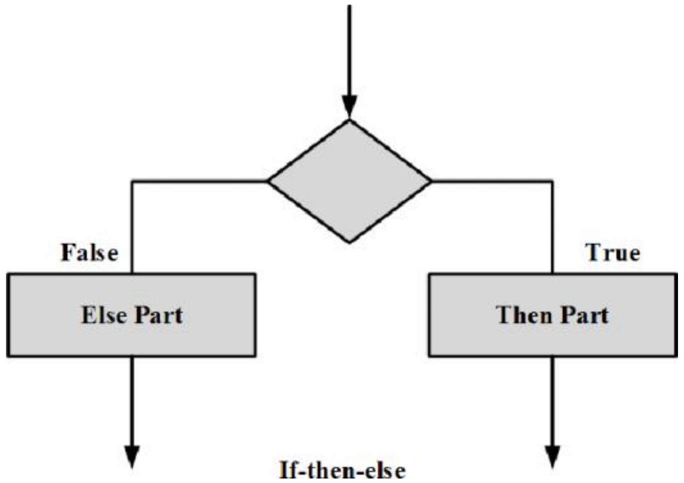
The structured programming constructs can be represented by graphical notations. These graphical notations are called flow chart. These notations are as follows –

Graphic Notations	Name	Meaning
	Box	It is used for processing step
	Flow of control	It is used to represent the flow of control from one construct to another.
	Diamond	It is used for representing the conditions, such as if-then-else or repeat until.

With help of above graphical notations the three programming constructs can be represented as follows

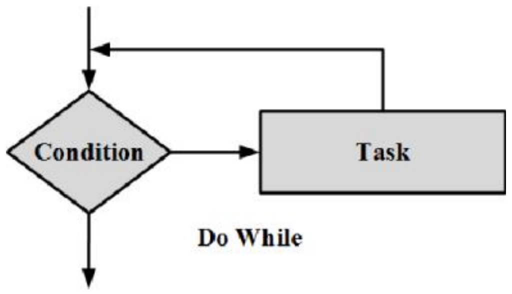


Sequence

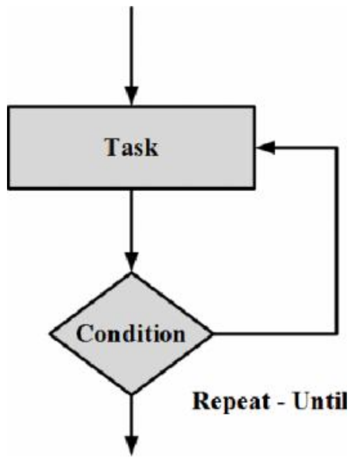


If-then-else Condition

These programming constructs can be nested one. That means one programming construct can be within another construct.

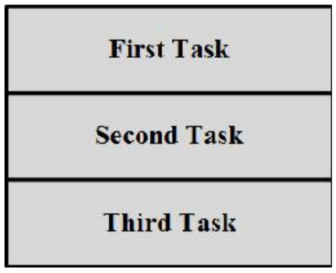


Do While

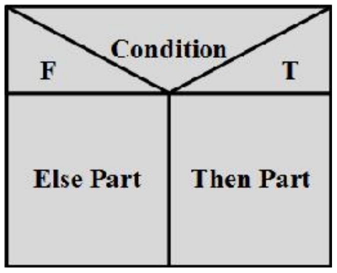


Repeat - Until

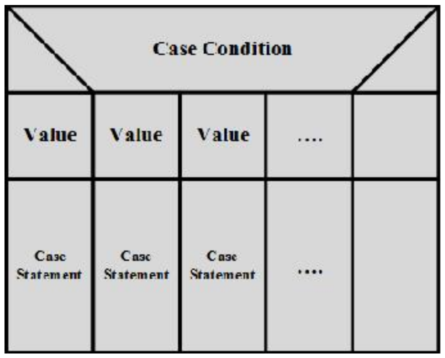
Another graphical notation is box diagram. This notation is also called as Nassi and Shneiderman chart (Nassi and Shneiderman are the names of the developer of this- notations) or NS chart. These notations are shown in the figure below



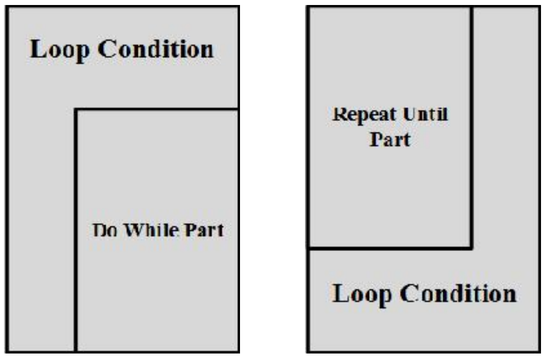
Sequence



If-then-else



Section



Repetition

Following are the characteristics of the NS chart

1. The scope of the programming constructs such as repetition, if-then-else is well defined.
2. Arbitrary transfer of the control is not possible using this notation.
3. Recursion can be represented conveniently
4. The scope of local and global data can be defined systematically.

Tabular Design Notations

Programming constructs can be represented by tabular design notation. This ai alternative representation to the graphical notations.

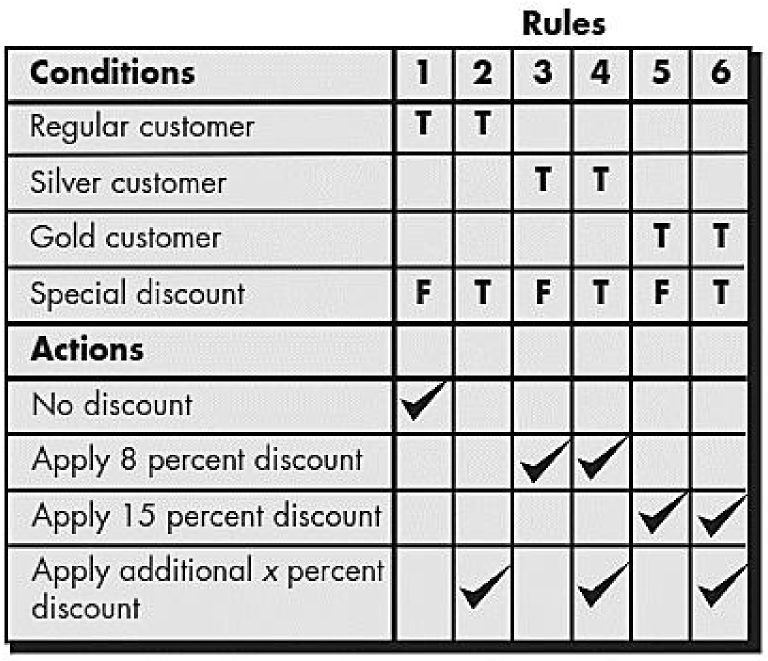
There are four sections in the tabular representation method -

1. The first section is at the upper left corner. It consists of list of conditions.
2. The second section is the lower left hand corner. It consists of list of actions.
3. The right- hand portion is a matrix which represents the combination of conditions and actions. The combination of condition and actions together form processing rules for that particular procedure.

Following steps are applied to develop a decision table -

- ➪ List all the actions associated with particular procedure.
- ➪ List all the conditions associated with particular procedure.
- ➪ Associate each condition with each corresponding action. Any impossible permutation must be eliminated. All possible permutations of conditions and corresponding actions must be represented in the table.
- ➪ Create the processing rule indicating that particular action exists on particular condition.

For example -



	Rules					
Conditions	**1**	**2**	**3**	**4**	**5**	**6**
Regular customer	T	T				
Silver customer			T	T		
Gold customer					T	T
Special discount	F	T	F	T	F	T
Actions						
No discount	✓					
Apply 8 percent discount			✓	✓		
Apply 15 percent discount					✓	✓
Apply additional *x* percent discount		✓		✓		✓

Program Design Language (PDL)

A program design language is also called pseudo code or structured English. This is similar to English but is used as a generic reference for design language. The PDL is not compiled. It is used to translate the design into the programming language.

A basic PDL syntax should possess the programming constructs for interface description, procedure definition, data declaration, condition constructs, repetition constructs and I/O constructs.

For example: Following is a PDL which demonstrates the procedure for searching the name John from the table. IF the name is found then index of the table is returned

Part-A

Design process

1. **State the guidelines for modular design.**

The guideline for modular design can be given as:

- Achieve the functional independence.
- High level of cohesion should be achieved.
- There should be minimum number of coupling.

2. **How do you evaluate user interface?**

The user interface can be evaluated by assessing the usability of interface and checking that it meets the user requirements.

After preparing the preliminary design a first level prototype is created. This prototype is evaluated by the user.

Then a formal technical review is conducted.

The feedback of both the above mentioned reviews is given to designer to make appropriate design modifications.

This evaluation cycle is continued no further modifications are suggested in the interface design.

3. **What do you mean by horizontal and vertical 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.

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

4. **Brief the importance of user interface.**

An inexperienced user can use the system easily with the user interface.

The user can switch from one task to another very easily. He can also interact with many applications simultaneously. The application information remains visible in its own window.

Fast and full screen interaction is possible with user.

5. **What is the work product of software design process and who does it ? [Nov/Dec 2012]**

The work product of software design is the design specification. This specification consists of design models that describe data, architecture, interfaces and components. Software engineers can do it but while designing the complex systems specialized system engineers are required.

Design Concepts

6. **Define the term software architecture.**

The software architecture is the hierarchical structure of software components and their interactions. In software architecture the software model is designed. The structure of that model is partitioned horizontally or vertically.

7. What is meant by transaction mapping? How it is used in software design?

In transaction mapping the user interaction subsystem is considered and DFD is j mapped into transaction processing structure.

In transaction mapping technique, the user command which is given as input, flows j into the system and produces more information flows, ultimately causes the output flow I from the DFD.

8. Distinguish between horizontal partitioning and vertical partitioning.

Horizontal partitioning	Vertical partitioning
Horizontal partitioning can be done by partitioning system into: input, data transformation (processing) and output.	Vertical partitioning suggests the control and work should be distributed top-down in program structure.
This kind of partitioning have fewer side effects in. change propagation or error propagation	Vertical partitioning define separate branches of the module hierarchy for each major function Hence these are easy to maintain the changes

9. What are the various models produced by the software design process ?

Various models produced during design process are -

Data design model used to transform the information domain model of analysis phase into the data structures.

The architectural design model is used to represent the relationship between major structural elements with the help of some "design patterns."

The interface design model describes how software interacts within itself.

In the component-level design model the structural elements of software architecture into procedural description of software components.

10. What are the quality parameters considered for effective modular design?

[OR]

State different criteria's applied to evaluate an effective modular system.

[May/June 2014]

Various parameters considered for effective modular design are -

Functional independence - By using functional independence functions may be compartmentalized and interfaces are simplified. Independent modules are easier to maintain with reduced error propagation.

Cohesion - 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.

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.

11. In what way abstraction differs from refinement?

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.

12. List out at least four design principles of a good design.

- The design process should not suffer from "tunnel vision".
- The design should be traceable to analysis model.
- The design should exhibit the uniformity and integrity.
- Design is not coding and coding is not design.

Design Model

13. "Modularity is the single attribute of the software that allows a program to be intellectually manageable" - How this is true?

This is quoted by Glenford Mayers. Consider that there is a large program composed of single module. Such a program cannot be grasped by reader. The control paths, span of reference, number of variables and overall complexity of such a program is beyond understanding. On the other hand if the program is divided into certain number of modules then overall complexity of the program gets reduced. The error detection and handling can be done effectively. Also changes made during testing and maintenance become manageable. Hence it is true that "Modularity is the single attribute of the software that allows a program to be intellectually manageable".

14. List out the elements of analysis model.

The elements of analysis model are :

i) Data dictionary
ii) Entity relationship diagram
iii) Data flow diagram
iv) State transition diagram
v) Control specification
vi) Process specification

15. Name the three levels of abstraction, which are in practice for the design.

Abstraction is the process of picking out common features of objects and procedures. For example, a programmer would use abstraction to note that two functions perform almost the same task and can be combined into a single function.

Abstraction is one of the most important techniques in software engineering and is closely related to two other important techniques -- encapsulation and information hiding. All three techniques are used to reduce complexity.

16. What are the steps involved in design stage of a software?

Three fundamental steps in the design stage are,

- ❖ Defining the Data and Control Flow
- ❖ Developing the logic
- ❖ Architecture Document
- ❖ Implementation Plan
- ❖ Critical Priority Analysis
- ❖ Performance Analysis
- ❖ Test Plan

17. What are various supporting documents to be prepared for the software ?

Stage	Software Document	Purpose Of The Document
System feasibility study	Feasibility report	To check the feasibility of the software development is checked in this stage

Requirement analysis and project planning	• Software requirement specification(SRS) • Project plan (it includes RMMM plan also)	The requirement gathering and analysis is made in this document. The purpose of this document is to identify scope of the project, effort estimation and determine the project schedule.
Design	Design document	The detailed system design is given in this document.
Code	Programs	Algorithms using suitable programming languages are implemented.
Testing	Test plan, test report	This document typically contains various test cases and test suits.
Installation	Installation manual/user guide	This document contain the, specific system requirements that are must for installing the software system being developed. User guide/manual helps the end user to operate the system. .

Design Heuristic

18. What is an architectural style?

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.

19. What are the types of interface design?

Following are the types of interface design.

- User interface.
- External interface to other systems, networks and devices.
- Internal interfaces between various design components.

20. Why modularity is important in software projects?

Modularity is important in software projects because of following reasons - Modularity reduces complexity and helps in easier implementation. The parallel development of different parts of the system is possible due to modular design. Changes made during testing and maintenance become manageable and they do not affect the other modules.

21. Why it is necessary to design the system architecture before specifications are completed?

The system architecture defines the role of hardware, software, people, database procedures and other system elements. Designing of system architecture will help the developer to understand the system as a whole. Hence system architecture must be built before specifications are completed.

22. If a module has logical cohesion what kind of coupling is this module likely to have with others?

When a module that performs a tasks that are logically related with each other is called logically cohesive. For such module content coupling can be suitable for coupling with other modules. The content coupling is a kind of coupling when one module makes use of data or control information maintained in other module.

23. What is system design?

System design is process of translating customer's requirements into a software product layout. In this process a system design model is created.

Architectural Design

24. List the architectural models that can be developed.

Following are the architectural models that can be developed

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

25. List out design methods.

The two software design methods are -

- Object oriented design
- Function oriented design

26. What are the design quality attributes FURPS mean? **[ND 2017/2015]**

The design attributes FURPS stands for

- ❖ **Functionality**: Functionality can be checked by assessing the set of features and capabilities of the functions.
- ❖ **Usability**: The usability can be assessed by knowing the usefulness of the system
- ❖ **Reliability**: Reliability is a measure of frequency and severity of failure.
- ❖ **Performance**: It is a measure that represents the response of the system.
- ❖ **Supportability**: It is also called maintainability. It is the ability to adopt the enhancement or changes made in the software.

27. Define data abstraction.

Data abstraction is a kind of representation of data in which the implementation details are hidden.

Architectural styles

28. What is software architecture?

Software architecture is a structure of systems which consists of various components, externally visible properties of these components and inter-relationships among these components.

29. Name some of the commonly used architectural styles. **[MJ 2015]**

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.

30. Define Software Quality.

Software quality is the degree of conformance to explicit or implicit requirements and expectations.

The degree to which a component, system or process meets specified requirements and/or user/customer needs and expectations.

31. Differentiate between Cohesion and Coupling.

Cohesion	Coupling
Cohesion is the indication of the relationship within **module**.	**Coupling** is the indication of the relationships between modules.
Cohesion shows the module's relative **functional** strength.	Coupling shows the relative **independence** among the modules.
Cohesion is a degree (quality) to which a component / module focuses on the **single** thing.	Coupling is a degree to which a component / module is connected to the **other** modules.

32. What are the various types of Cohesion?
Different types of cohesion are:

[1] Coincidentally cohesive
[2] Logically cohesive
[3] Temporal cohesion
[4] Procedural cohesion
[5] Communicational cohesion

33. What are the various types of coupling?
Various types of coupling are:

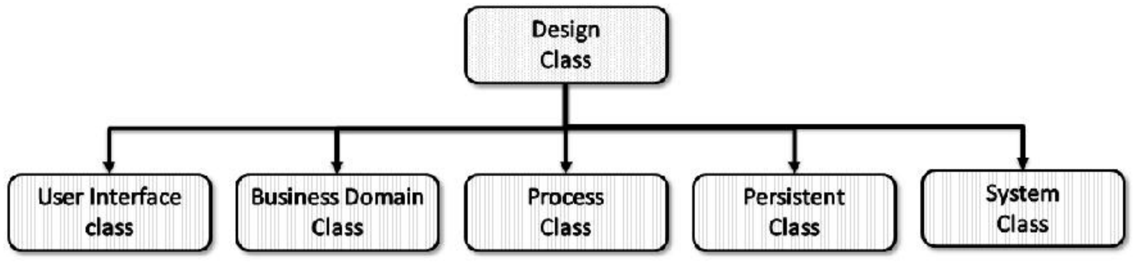
[1] Data coupling
[2] Control coupling
[3] Common coupling
[4] Content coupling

34. If a module has a logical cohesion, what kind of coupling in this module likely to have?
[MJ 2016]
If a module has a logical cohesion, the coupling in this module likely to have is data coupling

35. List out the goals of the design classes.
The goal of design classes are:

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.

36. What are the types of design class?
There are five different types of design classes



37. What are the sources used in building Architectural model?
Architectural model can be built using following sources -

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

38. What is the need for architectural mapping using data flow? [MJ 2016]

Architectural Mapping Using Data Flow

A mapping technique, called structured design, is often characterized as a data flow-oriented design method because it provides a convenient transition from a data flow diagram to software architecture.

- The transition from information flow to program structure is accomplished as part of a six step process:
  - (1) The type of information flow is established,
  - (2) Flow boundaries are indicated,
  - (3) The DFD is mapped into the program structure,
  - (4) Control hierarchy is defined,
  - (5) The resultant structure is refined using design measures.
  - (6) The architectural description is refined and elaborated.
- Example of data flow mapping, a step-by-step "transform" mapping for a small part of the SafeHome security function.
- In order to perform the mapping,
- The type of information flow must be determined. It is called***transform flow and exhibits a linear quality.***
- Data flows into the system along an incoming flow path where it is transformed from an external world representation into internalized form. Once it has been internalized, **it is processed at a transform center.**
- Finally, it flows out of the system along an outgoing flow path that transforms the data into external world form.

User Interface Analysis and Design

39. Define interface design.

Interface is the component of the computer system with which an user interacts, Interface design is the process of developing an Interface for the users to interact with the software in order to get the desired output.

40. What are the golden rules of the interface design? [ND 2015]

Thao Mandel has proposed three golden rules for user interface design -

1) Place the user in control
2) Reduce the user's memory load
3) Make the interface consistent.

41. What is user interface? [ND 2016]

The user interface (UI) is everything designed into an information device with which a human being may interact, including display screen, keyboard, mouse, light pen, the appearance of a desktop, illuminated characters, help messages, and how an application program or a Web site invites interaction and responds to it

42. List down the steps to be followed for User Interface Design. [MJ 2015 /2017]

Following are the commonly used interface design steps -

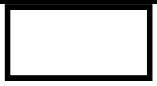
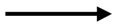
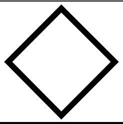
1. **1.** During the interface analysis step define interface objects and corresponding actions or operations.
2. **2.** Define the major events in the interface. These events depict the user actions. Finally model these events.
3. **3.** Analyze how the interface will look like from user's point of view.
4. **4.** Identify how the user understands the interface with the information provided along with it.

43. What are the three types of Interface design objects?

There are three types of objects

4 **Target object -** The target object is an object in which some object can be merged.
5 **Source object -** The source object is an object which can be dragged and dropped to some other object
6 **Application object -** The object which represents the application specific data.

44. List out the graphical notations used by structured programming. [May/June 2015]

Graphic Notations	Name	Meaning
	Box	It is used for processing step
	Flow of control	It is used to represent the flow of control from one construct to another.
	Diamond	It is used for representing the conditions, such as if-then-else or repeat until.

Component Level Design

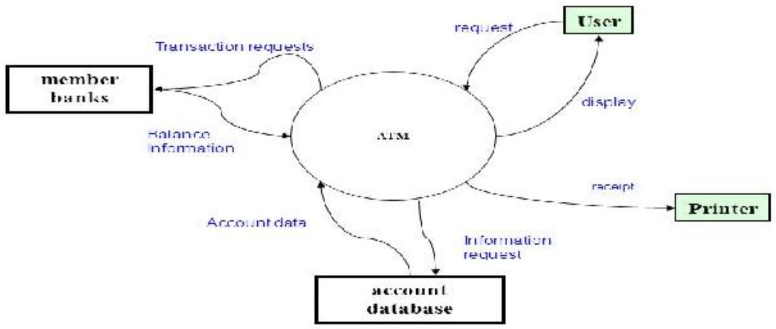
45. What is abstraction?

Abstraction is the act of representing essential features without including the background details or explanations. In software engineering, the abstraction principle is used to reduce complexity and allow efficient design and implementation of complex software systems.

46. What does refinement of software product means?

Refinement is a term of software engineering that encompasses various approaches for producing correct software products and simplifying existing programs to enable their formal verification.

47. Draw the context flow graph of a ATM automation system. **[ND 2017]**



48. Difference between Function Oriented Design and Object Oriented design. [MJ 2016]

Function Oriented Design	Object Oriented Design
The basic abstractions, which are given to the user, are real world functions	The basic abstractions are not the real world functions but are the data abstraction where the real world entities are represented.
Functions are grouped together by which a higher level function is obtained.an eg of this technique is SA/SD.	Functions are grouped together on the basis of the data they operate since the classes are associated with their methods
In this approach the state information is often represented in a centralized shared memory	In this approach the state information is not represented in a centralized memory but is implemented or distributed among the objects of the system.
Approach is mainly used for computation sensitive application,	Whereas OOD approach is mainly used for evolving system which mimics a business process or business case.
We decompose in function/procedure level	We decompose in class level
Top down Approach	Bottom up approach
It views system as Black Box that performs high level function and later decompose it detailed function so to be mapped to modules	Object-oriented design is the discipline of defining the objects and their interactions to solve a problem that was identified and documented during object-oriented analysis.
Begins by considering the use case diagrams and Scenarios.	Begins by identifying objects and classes.

49. What UI design patterns are used for the following [MJ 2017/ND 2016]

- Navigation Through Menus And Web Page-**Navigation**
- Shopping cart-**Miscellaneous**
- Tables-**Dealing with data**
- Page layout-**Getting Input**

50. If a module has logical cohesion, what kind of coupling is this module likely to have?

When a module that performs a tasks that are logically related with each other is called logically cohesive. For such module content coupling can be suitable for coupling with other modules. The content coupling is a kind of coupling when one module makes use of data or control information maintained in other module.

51. What is the need of architectural mapping using data flow? M/J 2016

Transform mapping and transaction mapping is used for architectural mapping using data flow diagrams.

52. What architectural styles are preferred for the following systems? Why? N/D 2016

a. Networking
b. Web based systems
c. Banking system.

a) Pipe and filter can be used for networking system. The pipe and filters pattern has set of components called filters, connected by pipes which transmit data from one component to next.

b) The layered architecture can be used for web based systems. In this architecture
   a. At outer layer: There are component service user interface operations
   b. At inner layer: Components perform operating system interfacing
   c. At intermediate layer: There are utility service and application software functions.

c) The object oriented architecture can be used for banking system. In this architecture, each object encapsulates data and operations. The object are identified and corresponding operations are performed.

53. What UI design patterns are used for the following? N/D 2016

a. Page layout
b. Tables
c. Navigation through menus and web pages
d. Shopping cart

a) Page layout – Card stack
b) Tables – Sorted tables
c) Navigation through menus and pages – Edit-in place
d) Shopping cart – Shopping cart which provide list of items selected for purchase.

54. What is the purpose of Petri Net? A/M 2017

Petrinets are rigorously used to define the system. There are three types of components in petrinets. These are

i. **Places**: Represent possible states of the system
ii. **Transitions**: Causes the change in the states.
iii. **Arc**: Connects a place with transition or transition with places.

55. What UI design pattern are used for the following? A/M 2017

a. Page layout
b. Navigations through menus and web pages.

Ref. Question No 53

56. Write a note on FURPS model. N/D 2017

FURPS stands for Functionality, Usability, Reliability, Performance and Supportability. These are the quality attributed used to measure design quality.

57. Draw the context flow graph of ATM automation system. N/D 2017

Refer Question no 47

58. List the principle of software design. A/M 2018

- The design process should not suffer from "tunnel vision".
- The design should be traceable to the analysis model.
- The design should exhibit uniformity and integration. iv. Design is not coding.
- The design should not reinvent the wheel.

59. How does the data flow diagram help in design of software system? [May 2019]

- Data flow diagrams are used to graphically represent the flow of data in a business information system. DFD describes the processes that are involved in a system to transfer data from the input to the file storage and reports generation.
- Data flow diagrams can be divided into logical and physical. The logical data flow diagram describes flow of data through a system to perform certain functionality of a business. The physical data flow diagram describes the implementation of the logical data flow.

60. Define a Component. Give example. [Nov 2019]

- Component level design is the definition and design of components and modules after the architectural design phase. Component-level design defines the data structures, algorithms, interface characteristics, and communication mechanisms allocated to each component for the system development. A complete set of software components is defined during architectural design. But the internal data structures and processing details of each component are not represented at a level of abstraction that is close to code. Component-level design defines the data structures, algorithms, interface characteristics, and communication mechanisms allocated to each component. According to OMG UML specification component is expressed as, "A modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces."

61. What is inheritance? [Nov 2019]

- Inheritance enables new objects to take on the properties of existing objects. A class that is used as the basis for inheritance is called a *superclass* or *base class.* A class that inherits from a superclass is called a *subclass* or *derived class*. The terms *parent class* and *child class* are also acceptable terms to use respectively. A child inherits visible properties and methods from its parent while adding additional properties and methods of its own.

PART B Questions Bank

Q.No	Questions	Univ. QP
1	Explain in detail how the software designing is done in the context of Software Engineering.	[MJ 2013]
2	Elaborate the design process along with the guidelines for achieving better quality and what are the attributes used for it.	
3	Explain in detail why design is important for implementing the right software and discuss about Abstraction and Modularity.	
4	Write a short note on Functional Independence constraint in software design. [OR] Explain various Coupling and Cohesion models in software engineering	[MJ/ND 2015] [MJ 2017]
5	Write a short note on Design Classes and their role in software design.	
6	Discuss in detail the role of a design model in the software engineering process with an example.	
7	Explain the role of Structural Partitioning in Architectural Design of computer software.	[MJ 2015]
8	Explain with an example how Data Flow Diagram is used in Architectural Mapping	[MJ/ND 2017] [ND 2016]
9	Write a short note on Transform Mapping and what are the steps involved in it.	

10	Outline the utilization of User Interface Analysis and what are the various models of user interface design. [Nov/Dec 2012 – 8M]	[ND 2012] [ND 2017] [ND 2015] [MJ 2016]
11	Explain in detail the various techniques used in task analysis and modelling	
12	Explain the various interface design steps in detail and how work environment analysis	
13	Discuss in detail about commonly known design issues.	
14	Briefly explain the basic component level design principals and provide guidelines for the same.	[MJ 2016] [ND 2016]
15	Explain in detail about Cohesion and Coupling.	[ND 2015] [MJ 2016]
16	Explain in detail various constructs of structured programming along with the notations used.	
17	Discuss about the design concepts in a Software development process	[ND 2017]
18	What is software Architecture? Describe in detail different types of software architectures with Illustrations	[MJ 2017]
19	Explain Design Heuristics and the process involved in Heuristic Evaluation of a software product	[ND 2014] [MJ 2016]
20	List out and explain various elements of data design and the guidelines to be followed in the architectural design of a software	[MJ 2015]
21	Write a short note on Architectural style and patterns and explain some of the most popularly used styles.	[ND 2013]
22	Briefly outline the golden rules of User Interface design.	[ND 2016]
23	List and explain any five fundamental software design concepts.	[MJ 2019]
24	Define Software architecture (2 Marks) Explain and compare the following architectural styles. 1.Call and return architecture 2.object –oriented architecture 3.Layered architecture	[MJ 2019]
25	What is software architecture? Outline the architectural styles with an example.	[ND 2019]
26	Outline the steps in designing class based components with an example	[ND 2019]

University Exam Questions

Part A

1. If a module has logical cohesion, what kind of coupling is this module likely to have? **M/J 2016**
2. What is the need of architectural mapping using data flow? **M/J 2016**
3. What architectural styles are preferred for the following systems? Why? **N/D 2016**
    a. Networking
    b. Web based systems
    c. Banking system.
4. What UI design patterns are used for the following? **N/D 2016**
    a. Page layout
    b. Tables
    c. Navigation through menus and web pages
    d. Shopping cart
5. What is the purpose of Petri Net? **A/M 2017**
6. What UI design pattern are used for the following? **A/M 2017**
    a. Page layout
    b. Navigations through menus and web pages.
7. Write a note on FURPS model. **N/D 2017**
8. Draw the context flow graph of ATM automation system. **N/D 2017**
9. List the principle of software design. **A/M 2018**
10. What UI design patterns are used for the following? **A/M 2018**

Part B

A/M 2015

1. Explain the various coupling and cohesion used in Software design.
2. For a Case study of your choice show the architectural and component design. (16)

N/D 2015

3. Explain the cohesion and coupling types with examples. (8+8)
4. Discuss about User Interface Design of a Software with an example and neat sketch. (16)

M/J 2016

5. Write short notes on the following
    i) Design heuristics
    ii) User-interface design
    iii) Component level design
    iv) Data/Class design
6. What is modularity? State its importance and explain coupling and cohesion. (8)
7. Discuss the difference between object oriented and function oriented design. (8)

N/D 2016

8. What is structured design? Illustrate the structured design process from DFD to structured chart with a case study. (16)
9. Describe the golden rules for interface design. (8)
10. Explain component level design with suitable examples. (8)

A/M 2017

11. What is cohesion? How is it related to coupling? Discuss in detail different types of cohesion and coupling with suitable examples. (13)
12. What is software architecture? Describe in detail different types of software architecture with illustrations. (13)

N/D 2017

13. Discuss about the design concepts in a software development process. (13)
14. Discuss about User Interface Design of a software with an example and neat sketch. (13)

A/M 2018

15. What is software architecture? Describe the different software architectural styles with examples.(13

16. Explain in detail types of cohesion and coupling with examples. (13)