UNIT III

3 Software Design

Syllabus

Software design - Design process - Design concepts - Coupling - Cohesion - Functional independence - Design patterns - Model-view-controller - Publish-subscribe - Adapter - Command - Strategy - Observer - Proxy - Facade - Architectural styles - Layered - Client Server - Tiered - Pipe and filter- User interface design - Case Study.

Contents

3.1 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 a vital activity during the process of software development. It is carried out irrespective of any process model used.
- During software engineering process, the first task is to identify and analyze the requirements. Based on this analysis the software design is developed. This design serves as a basis for code generation and testing. Hence software design is an intermediate process which translates the analysis model into design model.
- The four types of elements of **analysis model** are scenario based elements, class based elements, behavioral elements, and flow oriented elements.
- During the scenario based analysis various models such as use case diagrams, activity diagrams or swimlane diagrams are created. During the class based analysis the class diagrams are created. During flow oriented analysis the Data flow diagrams are created. Behavioral analysis can be done using state diagrams and sequence diagrams.
- The **class based elements** of analysis model are used to create **class diagrams.**
- The **flow oriented elements** and **class based elements** are used to create **architectural design.**

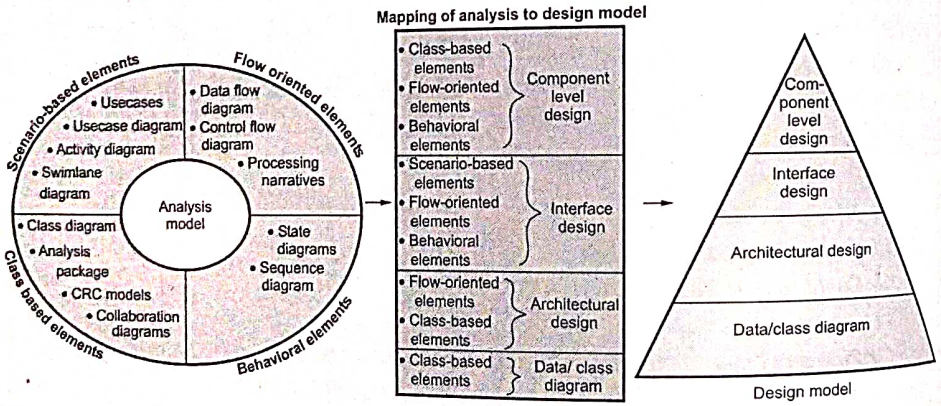


Fig. 3.1.1 Translating analysis model into the design model

- The **scenario based elements, flow oriented elements and behavioral elements** are used to create **Interface design.** The interface design describes how software communicates with systems.
- The **class based elements, flow oriented elements and behavioral elements** are used to create **component level design.** This design transforms the structural elements of software architecture into procedural description of software module. (Refer Fig. 3.1.1).
- Software design is very important for assessing the quality of software. Because design is the only way that we can accurately translate the user requirements into the finished software product.
- Without software design, it is difficult to test the software product. Not only this, but for making a small change in the software - it is the software design which helps us to make the necessary changes without disturbing other part of the software.

3.2 Design Process

AU : May-05. Marks 8

- Software design is an iterative process using which the user requirements can be translated into the software product.
- At the initial stage of the software is represented as an **abstract view.** During the sub-sequent iterations data, functional and behavioral requirements are traced in detail. That means at each iteration the refinement is made to obtain lower level details of the software product.

Characteristics of Good Design

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

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 posses **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(FURPS Model)

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.

Review Question

1. *Discuss in detail about the design process in software development process.* **AU : May-05. Marks 8**

3.3 Design Concepts

AU : May-06,08,16,19, Dec.-05,17,22. Marks 16

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

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

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

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, if a match is found then declare success !! Other wise declare 'name not found')

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

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 subproblems and then the solution to these subproblems 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

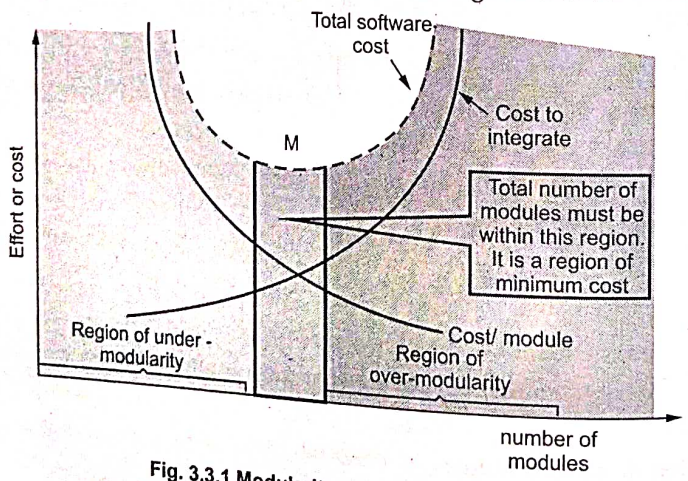


Fig. 3.3.1 Modularity and software cost

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 Fig. 3.3.1.

The Fig. 3.3.1 provides useful guideline for the modularity and that is - **overmodularity or the undermodularity 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.
- **Importance :**
- Due to modularity, each task forms **a separate, distinct program module.** At implementation time each module and its **inputs and outputs are well-defined;** there is no confusion in the intended interface with other system modules.
- The testing of modules and their integrity with other modules can be tested independently.
- Some modules may be defined by standard procedures which are used and reused in different programs or parts of the same program.
- Large projects containing modules become easier to monitor and control.

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 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 model.

3.3.4 Refinement

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

3.3.5 Pattern

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

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

- Pattern can be **reusable.**

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

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 other module or to any local data structure used by other module.

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

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 :

i) **Data coupling** - The data coupling is possible by parameter passing or data interaction.

ii) **Control coupling** - The modules share related control data in control coupling.

iii) **Common coupling** - In common coupling common data or a global data is shared among the modules.

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

Sr. No.	Coupling	Cohesion
1.	Coupling represents how the **modules are connected** with other modules or with the outside world.	In cohesion, the cohesive module performs **only one thing.**
2.	With coupling **interface complexity** is decided.	With cohesion, **data hiding** can be done.
3.	The goal of coupling is to achieve **lowest coupling.**	The goal of cohesion is to achieve **high cohesion.**
4.	Various types of couplings are - Data coupling, Control coupling, Common coupling and Content coupling.	Various types of cohesion are - Coincidental cohesion, Logical cohesion, Temporal cohesion, Procedural cohesion and Communicational cohesion.

3.3.8 Refactoring

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

Benefits of refactoring are -

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

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

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

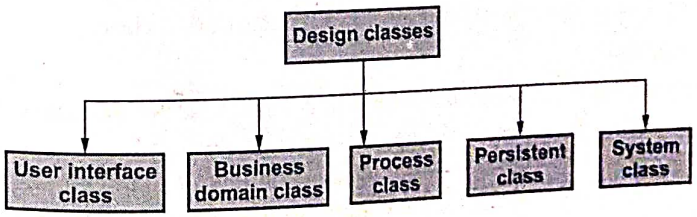


Fig. 3.3.2 Types of design classes

1. User Interface Class

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

2. Business Domain Class

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

3. Process Class

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

4. Persistent Class

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

5. System Class

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

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

- **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 posses 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 neighbouring design classes only.

Review Questions

1. *Describe decomposition levels of abstraction and modularity concepts in software design.* **AU : Dec.-05, Marks 16**
2. *Explain about the various design concepts considered during design.* **AU : May-06, Marks 12 , Dec.-17, Marks 13**
3. *Describe the concepts of cohesion and coupling. State difference between cohesion and coupling with a suitable example.* **AU : May-08,13, Dec.-22, Marks 13**
4. *What is modularity ? State its importance and explain coupling and cohesion.* **AU : May-16, Marks 8**
5. *List and explain any five fundamental software design concepts.* **AU : May-19, Marks 13**

3.4 Design Patterns

AU : Dec.-15, May-19, Marks 8

Definition of design pattern : Design pattern is general reusable solution to commonly occurring problem within a given context in software design.

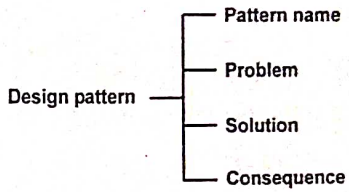


Fig. 3.4.1

In general the pattern has **four essential elements -**

Pattern name : The pattern name is used as handle to describe the design pattern. The naming of the pattern increases the design vocabulary. Due to the name of the pattern it becomes easy to think about the design and to communicate the design with the others. Finding the appropriate name for the design pattern is one of the hardest and tricky part.

Problem : The problem describes when to apply the pattern. It explains the problem and its context. It might describe various things such as algorithm for the problem, the class or the object structure of the problem or it can list down the conditions that must be followed before applying the pattern.

Solution : The solution describes the elements of the design, their relationships, responsibilities and collaborations. The solution never describes the concrete design or the implementation because the pattern is like templates and it can be applied to various problems in varied situations.

Consequence : The consequences describe the results and trade-offs of applying pattern. The consequences for software often concerned with space and time trade-offs. The listing of consequences helps in evaluation of the design pattern.

Generally the design pattern is divided into three categories - creational pattern, structural pattern and behavioral pattern.

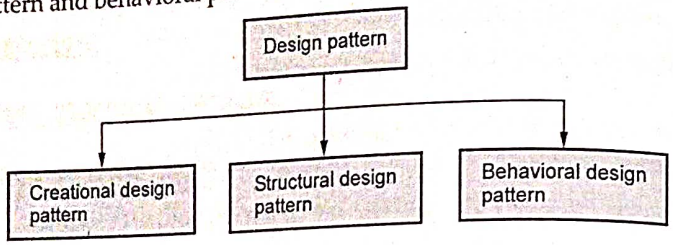


Fig. 3.4.2 Categories of design pattern

Example 3.4.1 *State the role and patterns while developing system design.*

AU : Dec.-15, Marks 8

Solution :

Sr.No	Pattern	Role
1.	Creator	The role of creator pattern is to find creator that needs to be connected to the created object in any event.
2.	Information expert	This pattern assigns the responsibilities to various objects that are participating in the system design.
3.	Singleton	This pattern ensures that a class has only one instance and provides global point of access to it.
4.	Factory method	It is responsible for creating the instances of several derived classes.
5.	Bridge	The role of this pattern is to separate out the interface from implementation without using inheritance.
6.	Adapter	The role of this pattern is to allow the interface of existing class to be used from another interface.
7.	Observer	The role of this pattern is to define the link between objects so that when one object's state changes, all dependent objects are updated automatically.
8.	Strategy	The role of this pattern is to define the family of algorithms, encapsulate these algorithms within in class and make them interchangeable.

3.4.1 Creational Pattern

Creational design pattern are the design pattern that are used for **object creation** mechanism. This type of pattern is used in the situation when basic form of object creation could result in design problems or increase complexity of a code base.

The creational patterns show how to make the software design flexible.

There are five well known design patterns that are part of creational pattern. These are -

1. **Abstract factory pattern :** This pattern is for creating the instances of several families of classes.
2. **Builder pattern :** This pattern separates the object construction from its representation.
3. **Factory method pattern :** This pattern is for creating the instances of several derived classes.
4. **Prototype pattern :** It specifies the kinds of objects to create using a prototypical instance and create new objects by copying this prototype.
5. **Singleton pattern :** It ensures that a class has only one instance and provides global point of access to it.

3.4.2 Structural Pattern

- In software engineering, structural design patterns are design patterns that ease the design by identifying a simple way **to realize relationships between entities.**
- The structural design pattern serves as a blueprint for how different classes and objects are combined to form larger structures.
- Structural patterns are just similar to data structures.
- There are various design patterns that are the part of structural design pattern -
  1. **Bridge :** This pattern separates the object interface from its implementation.
  2. **Adapter :** This pattern matches the interface of different classes.
  3. **Composite :** This pattern is based on the tree structure of simple and composite objects.
  4. **Decorator :** In this pattern the responsibilities on the object are dynamic.
  5. **Façade :** In this pattern the single class represents the entire subsystem.
  6. **Flyweight :** This pattern makes use of fine-grained instance for efficient sharing.
  7. **Private class data :** In this pattern there is restricted access to the class data.
  8. **Proxy :** In this pattern one object represents another object.

3.4.3 Behavioural Pattern

- The behavioral design patterns are design patterns that identify common communication patterns between objects and realize these patterns. By doing so, these patterns increase flexibility in carrying out this communication.
- A behavioral pattern explains how objects interact.
- It describes how different objects and classes send messages to each other to make things happen and how the various tasks are divided among different objects.
- There are various design patterns that are the part of behavioral design pattern -

1. **Chain of responsibility :** The chain of responsibility pattern is a design pattern that defines a linked list of handlers, each of which is able to process requests.
2. **Command :** The command pattern is a design pattern that enables all of the information for a request to be contained within a single object.
3. **Interpreter :** This pattern specifies how to evaluate sentences in a language. This pattern is useful when developing domain-specific languages or notations.
4. **Iterator :** This pattern is designed to sequentially access the elements of a collection.
5. **Mediator :** In this pattern the communication between objects is encapsulated with a mediator object. Instead of classes communicating directly, classes send messages to the mediator and the mediator sends these messages to the other classes.
6. **Null object :** This pattern is designed to act as a default value of an object.
7. **Memento :** This pattern captures and restores an object's internal state.
8. **Observer :** The observer pattern is a design pattern that defines a link between objects so that when one object's state changes, all dependent objects are updated automatically.
9. **State :** In this pattern it alters the object's behavior when its state changes.
10. **Strategy :** In this pattern, it encapsulates an algorithm inside a class.
11. **Template method :** This pattern defers the exact steps of an algorithm to a subclass.
12. **Visitor :** This class defines new operation to class without changes.

Review Question

1. *Write a short note on the structural patterns.* **AU : May-19, Marks 4**

3.5 Model-View-Controller

Problem : The user interface often gets changed when we extend the functionality of an application. Sometimes a customer may call for a specific user interface adaption. Different users may place conflicting requirements on the user interface. Building a system with required flexibility is expensive and error-prone if user interface is tightly interwoven with core functional logic.

Solution

The Model View Controller is also known as MVC architecture. It divides the whole application logic into three sections **model, view** and **controller.** This design pattern separates the user interface from core functional logic.

Structure

The working is as follows -

1. Suppose a user is requesting a web page from the server, then server will send this request to a specific **controller.**
2. The **controller** then asks to get the data from the **model.**
3. The **model** interacts with the database and sends the data to the controller on validation.
4. The **controller** then sends this data to the **view** in order to render it or to make a presentation of that data.
5. The **view** sends the proper presentation of data to the **controller.**
6. The **controller** then sends this presentation to the client who requested the web page. This presentation is considered a response to the client's request.

The following figure illustrates the above work.

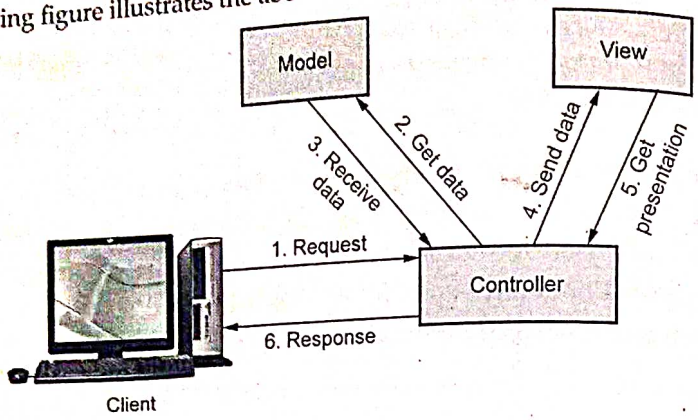


Fig. 3.5.1 Model View Controller architecture

➢ **Participating classes :**

The purpose of each component of Model-View-Controller architecture is described as follows -

Model

1) It handles the business logic or data logic.
2) It interacts with the database to get the requested data.
3) It performs the validation of that data.
4) It then sends the valid data to the controller.

View

1) It prepares the presentation of the data received from the controller.
2) It then sends the presentation to the controller.

Controller

1) It handles the request flow from **client**, to **model** and from **model** to the view. Again from **view** back to the client.
2) Sends the response to the **client**.

Known uses :

The MVC pattern is used mostly in web development applications.

The best-known example of the use of the Model-View-Controller pattern is the user-interface framework in the Smalltalk environment.

➢ **Consequences :**

Advantages

1) It is easy to make modifications in the application as the business logic is separated from the presentation logic.
2) The development of applications becomes fast.
3) It is easy for the developers to collaborate and work together.
4) Development of application becomes fast.

Disadvantages

1) It is a complex architectural pattern and difficult to learn and implement.
2) For developing simple applications MVC is not a suitable design pattern as it adversely affects the performance of such applications.

3.6 Publisher-Subscriber Pattern

Problem : A commonly arising situation is that data changes at one place and there exists many components depending on this data. For example in some GUI based application, if some internal data gets changed, then multiple views based on this data get affected.

Solution

- To solve this problem there must be some dedicated component who can inform the changes in the data to all the dependent components. Hence, in the publisher subscriber pattern, one dedicated component takes the role of the **publisher**(also called as subject). All the components dependent on changes in the publisher are called its **subscribers**(also called as observer).
- The publisher maintains a registry of currently subscribed components. Whenever a component wants to become a subscriber, it makes use of **subscribe interface** offered by publisher.
- Whenever the publisher changes the state, it sends the notification to all its subscribers.
- The subscribers then retrieve the changed data.
- Following are the freedoms used in publisher-subscriber pattern -
  - The publisher can decide which internal state changes it will notify to its subscribers.
  - An object can be subscriber to many publishers.
  - An object can take both the roles - it can be publisher or it can be a subscriber.
- Based on publisher-subscriber pattern the **push and pull models** are derived.

- In the push model, the publisher sends all the changed data to its subscribers.
- In the pull model, the publisher only sends minimal information about the change to its subscribers. It is then subscribers' responsibility to get the required data from the publisher whenever needed.
- The push model has rigid dynamic behavior whereas the pull model is flexible.

Structure :

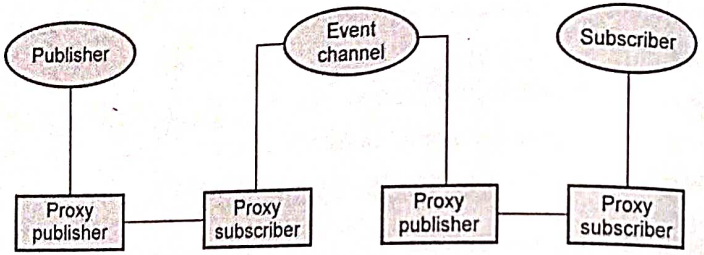


Fig. 3.6.1 Publisher-Subscriber pattern

Known uses :

Event messaging : Publisher-Subscriber pattern is widely used in delivery logistics. As we shop online more frequently for a wider variety of goods, package delivery has become commonplace. Logistics companies need to use delivery resources more efficiently. To optimize delivery, dispatching systems need up-to-date information on where their drivers are. Pub/Sub-event messaging helps logistics companies do this.

Advantages :

1) There is a support for distributed system.

2) Extensibility

Disadvantages :

1) Less efficient

2) Complex pattern to implement

3.7 Adapter

AU : May-11,12,13,14, Dec.-11,12,13,14, Marks 16

Sometime we come across the situation where two components are working well functionally but they can not work together because of some reason. In such a situation we can use something like adapter to interface two components.

In object oriented design the adapter pattern is adapting between classes and objects. Like any adapter in the real world where it is used to be an interface, a bridge between two objects. In real world we have adapters for power supplies, If you can not plug in the

power cord in your laptop you can use an adapter. You plug the power cord in the adapter and the adapter in to laptop slot.

Problem : How to resolve incompatible interfaces. How to provide a stable interface to the components having different interfaces ?

Solution

Convert the original interface of component into another interface through intermediate adapter object.

This pattern is also known as **wrapper.** This is a **GoF** pattern.

There are two types of adapters -

1. Object adapters 2. Class adapters

Object adapters :

In this type of adapter pattern the adapter contains the instance of a class it wraps. When the **Client** wants some task to get fulfilled it makes a request to the **Target.** The **Adapter** inherits the **Target** and at the same time holds the instance of **Adaptee.** Hence a specific requests of **Adaptee** can be invoked by the **Apdapter.** Thus it is the **Adapter** who hides the **Adaptees's** interface from the client. It is illustrated in the Fig. 3.7.1(a).

Class adapters :

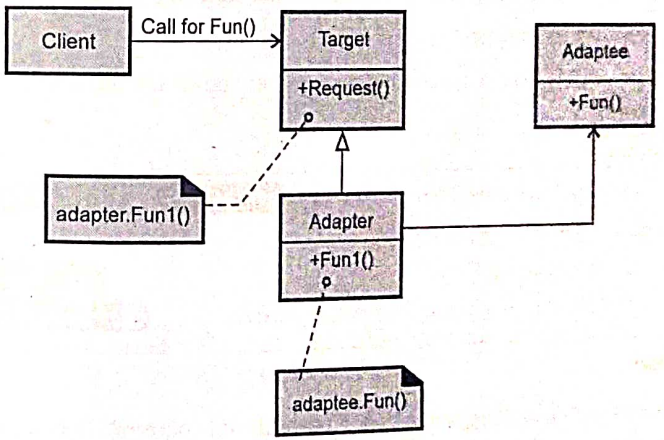


Fig. 3.7.1 (a) Object adapter pattern

In this type of adapter pattern the **Adapter** uses multiple polymorphic interfaces of **Adpatee.** Hence the number of requests of **Adaptee1,Adaptee2,..., AdapteeN** can be invoked by the **Apdapter.** It is illustrated in the Fig. 3.7.1(b).

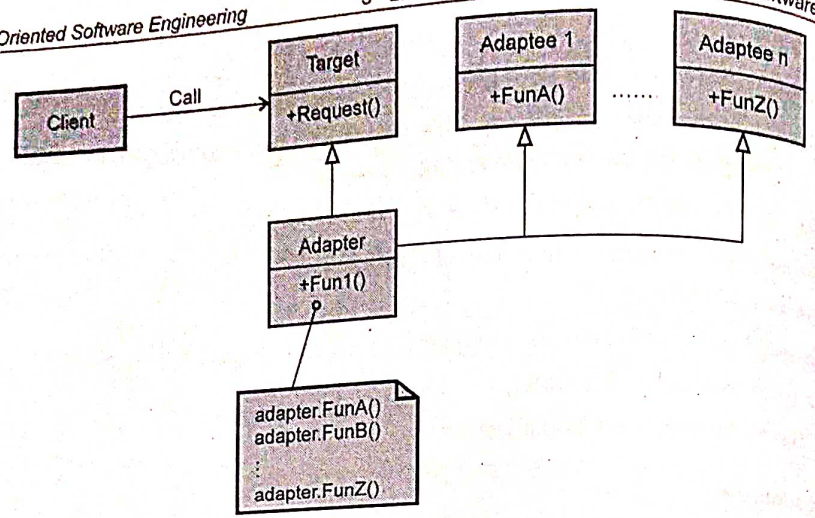


Fig. 3.7.1 (b) Class adapter pattern

Example

The typical examples of adapter pattern are WindowsAdapter in Swing API or JDBC-ODBC API.

Benefits

1. Two compatible objects or classes can be interfaced using adapter pattern.
2. Reduces coupling between the objects.
3. Due to use of polymorphism and indirection only essential behavior is focused.

Review Questions

1. *Write a short note on adapter pattern.* **AU : May-11,13,14, Dec.-13,14, Marks 4**
2. *What are the two types of adapter pattern ?*
3. *Write short note on adapter.* **AU : May-12, Marks 4**
4. *What is design pattern ? Explain the GoF design patterns.* **AU : Dec.-11,12 Marks 16**

3.8 Command

- Command pattern is one of the behavioural design pattern. This design pattern is used to implement loose coupling in a request-response model.

Intent

The command pattern converts a request or an operation into a standalone object that contains all the information about the request.

Also Known As

Action, Transaction

Motivation

There is an encapsulation of the operational request of a service into an entity called command so that the same service can be used to trigger different operations. Optionally the command can also be used for **undo, logging or queueing** operations.

Participants

Command_Interface :

Declares an interface for executing an operation.

ConcreteClass (Command_name...) :

- Defines a binding between a Receiver object and an action.
- Implements Execute by invoking the corresponding operation(s) on Receiver.

Client (Application) :

Creates a **ConcreteClass** object and sets its receiver.

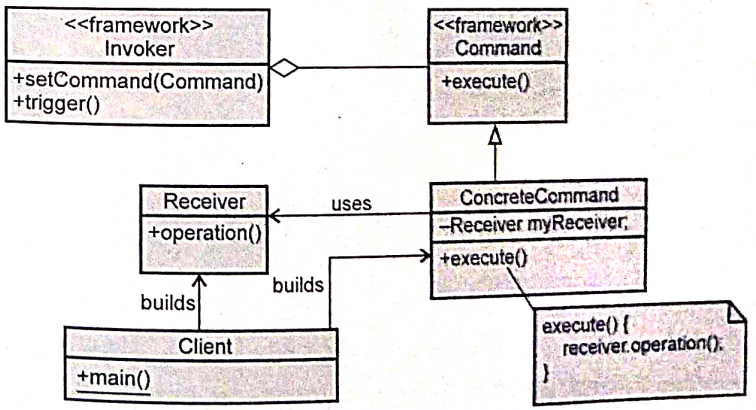
Invoker (command) :

Asks the command to carry out the request.

Receiver (Receiver_item) :

Knows how to perform the operations associated with carrying out a request. Any class may serve as a Receiver.

Structure



Implementation

```
class Invoker {
   private Command myCommand;
   public void setCommand(Command aCommand){
      myCommand = aCommand; //actual command
   }
   public void trigger(){
      myCommand.execute();
   }
}

absract class Command {
   abstract void execute();
}

class ConcreteCommand : Command {
   private Receiver myReceiver;
   public ConcreteCommand(Receiver aReceiver) {
      myReceiver = aReceiver;
   }

   public void execute() {
      myReceiver.operation();
   }
}

class Receiver {
   public operation() { // operation here }

}
class Client {
   public static void Main(String[] args) {
      Command c =
         new ConcreteCommand(new Receiver));
      Invoker i = new Invoker()
      i.setCommand(c);
      i.trigger()
   }
}
```

Example

Consider an application in which we want to perform some operations on a text file. Let us call it as **FileOperation** application. These operations can be **Read File** operation or

Write File operation using Read or Write commands. To implement **Read** and **Write** commands we need a text file. Hence the text file is called as **Receiver** over here. The **ReadFileOperation** and **WriteFileOperation** are the **concrete classes** that will invoke the commands **Read** and **Write** respectively. These methods are placed in the **Receiver**. Hence Receiver is something that will actually execute the commands. There is an **Executor** or an **Invoker** which will invoke the commands. The **Invoker** will have the **list of commands** that will be executing currently. This facility of Invoker is also useful if we want to perform undo operations. The **Client** is a command application. The **Client** calls the **Invoker** which in turn executes the desired command.

To Summarize,

Command : Read and Write operations

Receiver : File on which the operations are to be performed.

Invoker : Contains the list of commands that are currently executing

Client : Will be using the Invoker to call the commands

The implementation for the above example is as follows -

```
//Command Interface
public interface FileOperation {
  String execute();
}
//Concrete Class1
public class ReadOperation implements FileOperation {
  private Text_File myfile;
  //Constructor
  public ReadOperation(Text_File myfile) {
    this.myfile = myfile;
  }
  @Override
  public String execute(){
    return myfile.Read();
  }
}
//Concrete Class2
public class WriteOperation implements FileOperation {
  private Text_File myfile;
  //Constructor
  public WriteOperation(Text_File myfile) {
    this.myfile = myfile;
  }
  @Override
```

```
    public String execute(){
        return myfile.Write();
    }
}
//Receiver
public class Text_File {
    private String file_name;
    //Constructor
    public Text_File(String file_name) {
        this.file_name = file_name;
    }
    //Actual Code for Read Command
    public String Read() {
        return "Reading a file: "+ file_name;
    }
    //Actual Code for Write Command
    public String Write() {
        return "Writing to a file: "+ file_name;
    }
}
//Invoker or Executor
public class MyFileOperationInvoker {
    //creating list of operations
    private final list<FileOperation> List = new ArrayList<>();
    public String executeCommand(FileOperation myOperation) {
        //Adding the currently invoked command to the ArrayList
        List.add(myOperation);
        return myOperation.execute();
    }
}
//Client
class Application_Client {
    public static void main(String args[]) {
        //client is invoking the invoker or Executor
        MyFileOperationInvoker myExecutor = new MyFileOperationInvoker();
        //Using Executor object the command for execution is called via Concrete class constructor
        //by passing the file name(Note that file name acts as a receiver)
        myExecutor.executeCommand(new ReadOperation(new Text_File("test1.txt")));
        myExecutor.executeCommand(new WriteOperation (new Text_File("test1.txt")));
    }
}
```


Merits and Demerits

Merits

1) New commands can be easily added to the existing application.
2) It is easy to perform redo and undo operations while using the commands.
3) It decouples the classes that invoke the operation from the object that knows how to execute the operation

Demerit

1) Using command design pattern may requires more effort on implementation, since each command requires a concrete command class, which will increase the number of classes significantly.

3.9 Strategy

Intent

- There are common situations when classes differ only in their behavior. For this case, is a good idea to isolate the algorithms in separate classes in order to have the ability to select different algorithms at runtime.
- This is a kind of design pattern in which a set of similar algorithms to be defined and encapsulated in their own classes.
- Thus this pattern is intended to define a family of algorithms, encapsulate each of them and make them interchangeable.
- Strategy is like **Template Method**.

Structure and Implementation

Following is an UML diagram which represents how to implement strategy design pattern.

Client class makes use of interchangeable algorithms. It maintains the reference to strategy object.

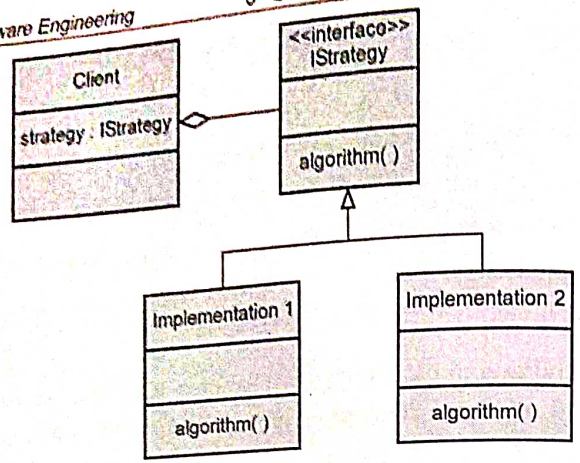


Fig. 3.9.1 Representation of strategy

IStrategy declares an interface common to all supported algorithms. The client uses this interface to call the algorithm defined by a **Implementation1**, **Implementation2** and so on.

Implementation1, **Implementation2** represent instances that provide different algorithms that can be used by the client.

The skeleton code for this can be as given below

```
static class Program
{
  public static void main(String arg[])
  {
    //invoking the algorithm() of Implementation1
    //invoking the algorithm() of Implementation2
  }
}

public class Client
{
    //declaring strategy object
}

public abstract class IStrategy
{
  public void algorithm();
}

public class Implementation1 extends IStrategy
{
  public  void algorithm()
  {
  }
```



```
}
public class Implementation1 extends IStrategy
{
public  void algorithm()
  {
  }
}
```

Example :

In online shopping system modes of payment is a typical example of strategy pattern. The online payment mode can be card payment, cash payment and so on.

➢ Merits and Demerits

Merits

1. If one needs to have several different behaviours that the object to perform, it is much simpler to keep track of them if each behaviour is a separate class. One can add, remove or change any of the behaviours very easily.
2. This strategy brings the flexibility in the code.
3. When you have several objects that are basically the same and differ only in their behaviour, it is a good idea to make use of the strategy pattern.

Demerits

1. The application must be aware of all the strategies to select the right one in right situation.
2. The memory requirement for implementing this pattern is more as **Client** has to maintain the object of **Strategy** base class, in order to choose appropriate algorithm each time.

Example 3.9.1 *For the description given below, draw the class diagram and identify the roles for a strategy design pattern. Mention the roles identified for each class and its relevant behaviour in the class diagram. Grand health club offers a scheme for membership of the health club. The options available for registering are 'yoga' and 'aerobics'. The monthly charges for aerobics membership are 2000.00. The monthly charges for yoga membership are 1000.00. The members can avail a single option out of the two options. If a person books for three months, he gets 20 % discount. If he books for six months, he gets 25 percent discount. If he books for nine months, he gets 35 % discount. If he books for one year, he gets 50 % discount.*

Solution :

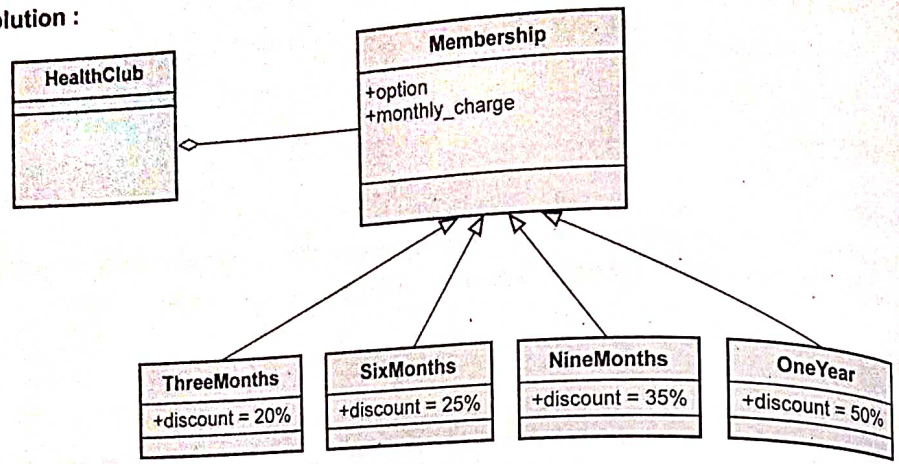


Fig. 3.9.2

Example 3.9.2 *Apply strategy design pattern to the following and draw the class diagram. A company has many employees. Each employee has a name and a performance index in the range of 1 to 5. When the index is 2, the increment is 10 percent of the previous year salary. ,3 the increment is 15 percent of the previous year salary, 4 the increment is 20 percent of the previous year salary and when 5 the increment is 25 percent of the previous year salary. Indicate the roll of each class in the class diagram.*

Solution :

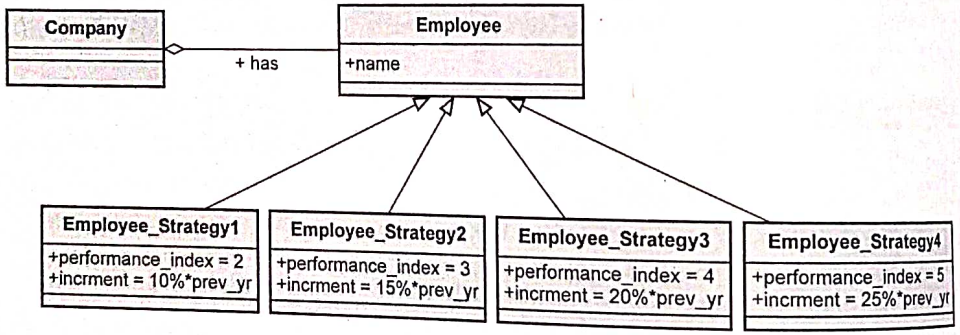


Fig. 3.9.3

3.10 Observer

Intent

The observer pattern is a design pattern that defines a link between objects so that when one object's state changes, all dependent objects are updated automatically. This pattern allows communication between objects in a loosely coupled manner.

Structure and Implementation

Following is an UML diagram which represents how to implement observer design pattern.

Observable interface or abstract class defining the operations for attaching and de-attaching observers to the client. It is also called as **subject.**

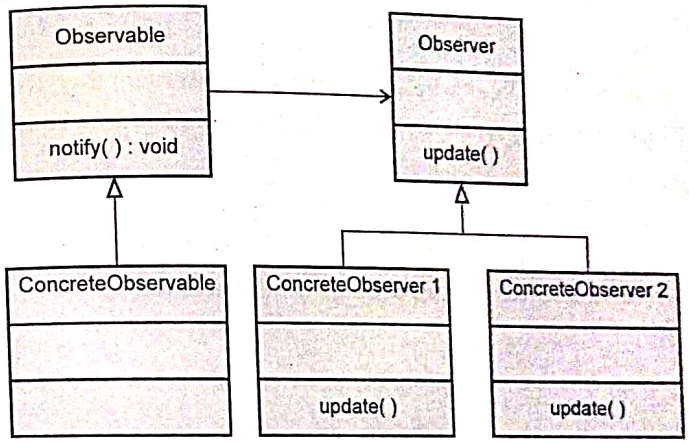


Fig. 3.10.1 Representation of observer

ConcreteObservable is a concrete Observable class. It maintain the state of the object and when a change in the state occurs it notifies the attached **Observer.**

Observer is the interface or abstract class defining the operations to be used to notify this object.

ConcreteObserver1, ConcreteObserver2 are concrete Observer implementations.

The flow of execution is as follows -

The main function instantiate the **ConcreteObservable** object. Then it instantiate and attaches the concrete observers to it using the methods defined in the **Observable** interface. Each time the on the change in state, it notifies all the attached **ConcreteObservers** using the methods defined in the **Observer** interface.

➢ Merits and Demerits

Merits

1. This pattern is useful when there is a change of state in one object that must be reflected in another object.
2. The new observers can be added easily with the minimal changes in the entire code.
3. It allows to send the data to many other objects in very efficient manner.

Demerits

When there are several **Observabels(subjects)** and **Observers** then there are multiple relations that need to be maintained. Due to which the overall code becomes complex.

3.11 Proxy

Intent

Provides the placeholder for another object to control access to it.

Also Known as

Surrogate

Motivation

- In this pattern the class represents the functionality of another pattern.
- This type of pattern belongs to structural design pattern.
- The intention of this pattern is to provide a client for an object to control access to it. Using extra level of indirection controlled and intelligent access can be provided.
- The proxy object gives the client that actual request is handled by the proxy.

Structure

Following UML diagram represents how to implement proxy design pattern

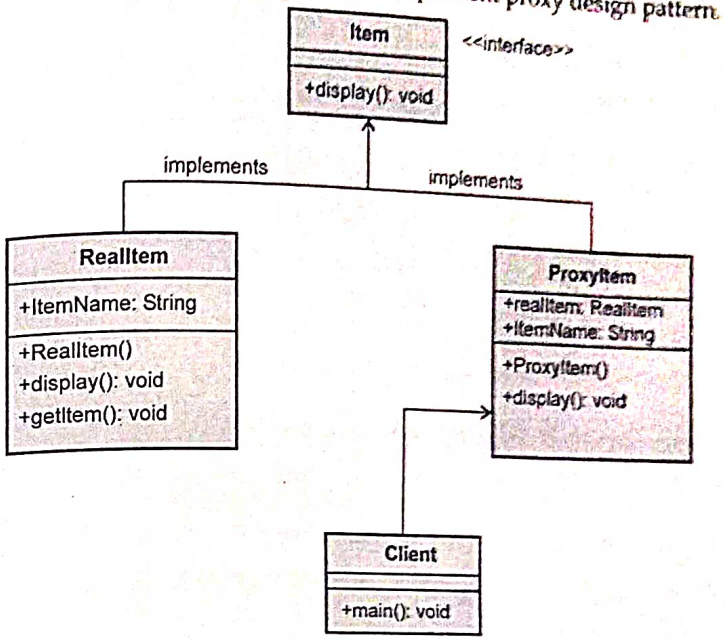


Fig. 3.11.1 Representation of proxy

Implementation

When client needs some item, it will actually use the ProxyItem to get an objet for Item. The **Item** is an interface which is implemented by the concrete classes **RealItem** and **ProxyItem.** The **ProxyItem** is a proxy class which uses the instance of **RealItem.**

The skeleton code for the above representation is as given below

The interface is created as follows

Item.java

```
public interface Item
{
   void display();
}
```

The concrete classes **RealItem** and **ProxyItem** are as follows -

RealItem.java

```
public class RealItem implements Item
{
   private String ItemName;
   public RealItem(String ItemName)
   {
     this.ItemName = ItemName;
     getItem(ItemName);
   }
   public void display()
   {
     System.out.println("Displaying Item : " +ItemName);
   }
   private void getItem(String ItemName)
   {
     System.out.println("Item: " +ItemName+" is obtained");
   }
}
```

ProxyItem.java

```
public class ProxyItem implements Item
{
   private RealItem myItem;
   private String ItemName;
   public ProxyItem(String ItemName)
   {
     this.ItemName = ItemName;
   }
   public void display()
   {
     if(myItem == null)
     {
       myItem = new RealItem(ItemName);
     }
     myItem.display();
   }
}
```

Client.java

```
public class Client
{
  public static void main(String[] args) {
  Item item = new ProxyItem("MyFavorite Item");
//item will be obtained using getItem first and then will be displayed
```



```
item.display();
  }
}
```

Application

The **payment** made by the customer by **cheque** or by bank **demand draft** is proxy for the actual cash present in the account. A **cheque or DD can be used in place of cash** for making purchases and ultimately controls access to cash in the issuer's account.

➢ Merits and Demerits

Merits

1. Security is the most primary advantage of this design pattern. The main object remains protected from the client access by providing the proxy instance.
2. The duplication of the object can be avoided, this saves the system's memory space and also increases the performance of application.

Demerits

The conflicts in the behavior of the system may occur if one client directly accesses the real object and another client accesses the proxy object.

Review Question

1. *Explain the proxy designer pattern with suitable example.*

3.12 Façade

The Façade design pattern is a structural design pattern. It is evolved from the French word façade which means "frontage" or "face".

It defines the high-level interface that makes the subsystem easier to use.

Intent

- Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.
- Wrap a complicated subsystem with a simpler interface.

Motivation

To simplify the complex system architecture by unifying it with different interfaces, the Façade object is used. The Façade shields the complexity of the system from the client.

Structure

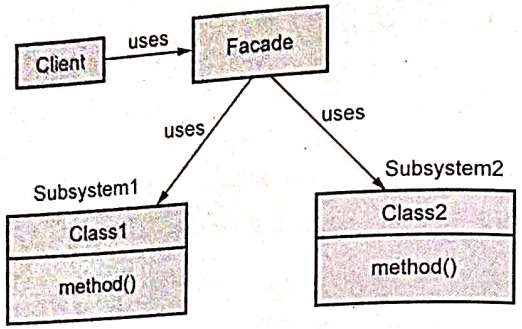


Fig. 3.12.1

Example

Consider a scenario where a customer has some issues with the product he/she has purchased. So the customer calls makes a call on customer care number and speaks with a customer service representative. The customer care service person acts as a Facade, providing an interface to the order fulfilment. The underlying system may include a team of service engineers. The corresponding service engineer of that region is then assigned a duty to look after the complaint raised by the customer and solve the complaint. The order of execution will be -

1) Customer calls the **ProductServiceSystem**
2) The customer care service person then hands it over to a subsystem that assigns the appropriate service engineer.
3) The service engineer visits the customer and solves the issue raised by the customer.

Implementation

```
public class ProductServiceSystem(){
   public void assignServiceEngineer(){

   }
}
public class ServiceEngineer() {
   public void getRequest(){

   }
   public void fulfillRequest(){

   }
}
```



```
//Facade
public class CustomerCareServicePerson(){
   private ProductServiceSystem service;
   private ServiceEngineer engineer;
   service.assignServiceEngineer();
   engineer.getRequest();
   engineer.fulfillRequest();
}
```

➢ Merits and Demerits

Merits

1) It hides much of the complexity and makes the subsystem easy to use.
2) It helps to have the principle of loose coupling, so changes can be made to the system without affecting the clients.

Demerits

1) It incorporates the additional level of indirection.
2) There is high degree of dependence on façade interface.
3) Clients cannot access underlying classes and certain functionalities might be unavailable to clients.

Review Question

1. *Explain the intent, motivation, structure, implementation, merits and demerits of Facade design pattern.*

3.13 Role of Architecture

- Architecture is a design of a system which gives a high level view of the parts of the systems and focus on the relationship among these subsystems.
- There is no unique structure of the system which can be described by the architecture. In fact there are various types of structures. Hence the architecture can be formally defined as

Definition : Software architecture of the system is a structure or structures of the system which consists of software elements and the externally visible properties of those elements and relationships among them.

- According to this definition the elements that have some important properties are considered as the main elements. The behaviour of these elements is a part of architecture.
- The uses of the software architecture descriptions are -

1. Understanding and communication

- The software architecture description is for communicating the architecture of the system to its users, developers and to the architects. This description helps these stakeholders(users, developers and architects) to understand the micro properties of the system and how the system satisfies the functional and quality requirements.
- The architecture description of proposed system will describe how the system will be composed and what the parts of it are. This will help in understanding the proposed system.
- The architecture description is also useful for understanding the existing system.

2. Reuse

- Software architecture helps in reusing the software. The software reuse help in increasing the productivity and decreasing the cost of software.
- Software engineers are working for developing such software components that will be useful for building any desired software system. Then architectures are chosen in such a manner that these components will be reused effectively.
- If there are similar types of products then architectures suggest the components that can be reused in similar products.

3. Construction and evolution

- For construction of the system, the **partitioning** is done.
- Architecture partitions the system into parts. Suitable partitioning in the architecture helps the project to build the system part by part. Then each part can be independently developed and built by separate group of developers. Thus architecture guides the development process.
- After construction of the complete system sometimes **new features** need to be **added** or some **changes** must be done in the existing system.
- It is the architecture which helps in deciding where to add the new features with minimum complexity and effort.
- If some changes need to be incorporated in the system then architecture decides which part of the system will get affected by the changes. The changes should be made carefully without disturbing the overall functionality of the system.

4. Analysis

- Before building the actual system some important properties of the system can be determined. The software engineers use the models to analyze the design of a product for its cost, reliability and performance. Architecture helps in determining such models.
- The properties of the system that has to be build can be predicted if the system is developed from the architecture. Thus architecture description helps in analyzing the system before its actual development.

Review Question

1. *What is software architecture ? Also explain the uses of software architecture description.*

3.14 Architectural Styles

- Architectural style is a **pattern** for creating the system architecture for the given problem. Most of the large systems are **heterogeneous** and do not follow a single architectural style.
- These styles can **provide ideas** for creating an architectural view for the given problem.
- Styles can be combined to form **richer views**.
- Many systems use multiple styles and different parts of a system may use different styles.
- Various architectural styles are -
  - Layered architectural style
  - Client Server style
  - Tiered architecture
  - Pipe and Filter style

3.15 Layered Architectural Style

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

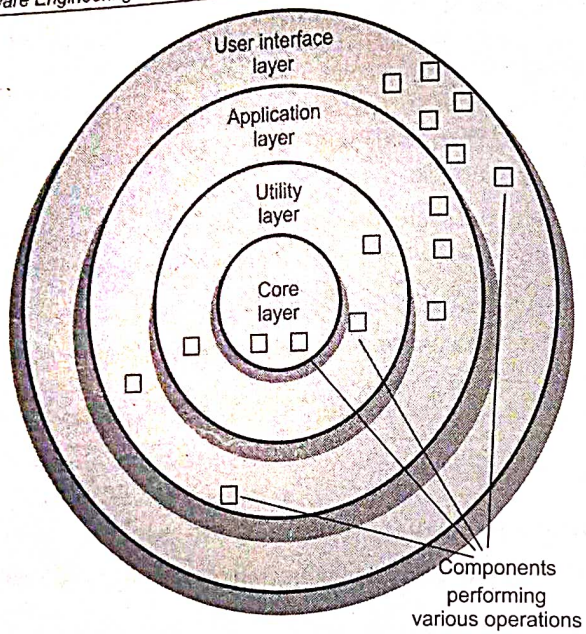


Fig. 3.15.1 Layer architecture of components

- The outer layer is responsible for performing the user interface operations while the components in the inner layer perform operating system interfaces.
- The components in intermediate layer perform utility services and application software functions.

3.16 Client Server Style

- In this style, there are two component types **client** and **server**.
- The client can communicate to the server but not to another client.
- The communication between the client and server is initiated by the client. The client sends the request to the server and for service and the server supports.
- The server receives any request from the client on a specific port. Then it builds the requested service and then the result is returned to the client.
- There is one type of connector used in this style and that is **request and reply** type. The task of this connector is to connect the client with the server. This type of connector is **asymmetric**. The client end of the connector makes the request and the server end of connector the response is sent. The communication is synchronous in nature. That means client makes a request and wait for the response and it gets blocked until the request gets a replay.

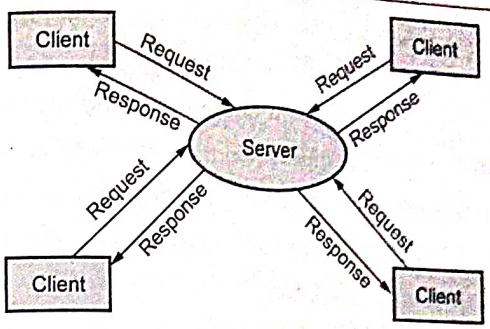


Fig. 3.16.1 Client Server architecture

Review Question

1. *Explain the client server architecture.*

3.17 Tiered Architecture

- The general form of client server architecture is **n-tier** structure. The client sends a request to the server and in order to serve the request the server further sends this request to some another server.
- The most commonly used structure is **3-tier** architecture. In this architecture the client makes a request to the server and receives the response from the server. This is a **presentation tier**. Then the server processes this request and computes the result. This is a **middle tier** which is called **business logic tier**. The third tier is the **database tier** in which the data resides. The business tier interacts with the database tier for accessing the required data.

  Refer Fig. 3.17.1.

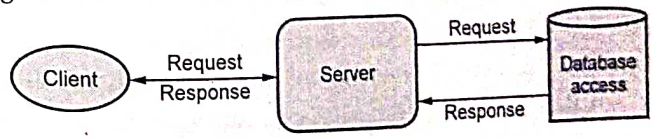


Fig. 3.17.1 Client Server architecture

For the client server communication usually the read write connector is used but many client-server systems today support the TCP ports as the connectors. For the Web applications HTTP protocol is used to support the connector.

3.18 Pipe and Filter

AU : Dec.-20, May-22, Marks 13

- In this architecture series of transformations are applied to produce the output data.
- There is only one type of component used in this architectural style and that is

Filter which is connected to only one type of connector and that is **Pipe**. The data is transformed from one filter to another using the pipes.

- The filters work independently without bothering the other filter.
- A filter may have more than one inputs and more than one outputs.
- Filters are only concerned with the data coming to them as input through pipe. A filter need not know the identity of the filter that sent the input data or the identity of the filter that will consume the data they produce.
- Pipe does not change the data which travel through them. These are **unidirectional** entities which transfers the data received at one end to another.

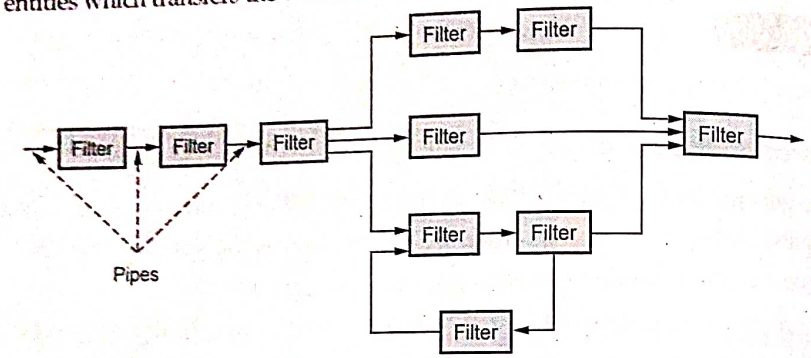


Fig. 3.18.1 Pipes and Filters

- There are some constraints on this architectural style imposes -
  1. The filters should work without the knowledge of producer (the filter who sends the data) or consumer (the filter who receives the data).
  2. A pipe is a **two-way connector**. It must connect an output port of a filter to an input port of another filter.
  3. In pure pipe and filter structure a filter has independent thread of control which process the data and to support a pipe mechanism which transforms the data the **synchronization** needed.
- Pipe and filter architectural style is well suited for **data processing** and **transformation**.
- In **text processing applications** and **Signal processing applications** this style proves to be well suited. Also it is useful in the applications that typically perform encoding, error correction and other transformations on the data.

Review Questions

1. *Explain pipe and filter architectural pattern.*
2. *What are the different types of architectural styles exist for software and explain any software architecture in detail ?* **AU : Dec.-20. Marks 13**
3. *Bring out the importance of architectural design. Explain any two software architectural styles with suitable example.* **AU : May-22. Marks 13**

3.19 User Interface Design

AU : May-16,13,22. Dec.-08,09,17,15,16. Marks 16

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.

Advantages :

Following are the advantages of having user interface system.

1. The user interface systems provide fast and intuitive interactions to the user. The new user can easily operate the system.
2. As in user interface design, menus are provided, it avoids typing mistakes and very little typing is required.
3. The user interface helps in simple data entry.
4. It provides the flexibility in switching from one task to another.

All the above mentioned advantages of user interface design helps in building the flexible software product. And therefore the sale for software product can be increased.

3.19.1 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.

Let us discuss each rule in detail -

3.19.1.1 Place the User in Control

While analysing 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 'interruption' 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.

3.19.1.2 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.

1. **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.

2. **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.

3. **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.

4. **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.

5. **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.

3.19.1.3 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.

1. **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.

2. **Maintain consistency across family of product.**

   If an application come 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 product 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.

3. **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.

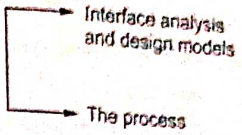
3.19.2 User Interface Analysis and Design

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

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

These steps can be broadly categorized in two classes.

Let us discuss each of them in detail –



3.19.2.1 Interface Analysis and Design Model

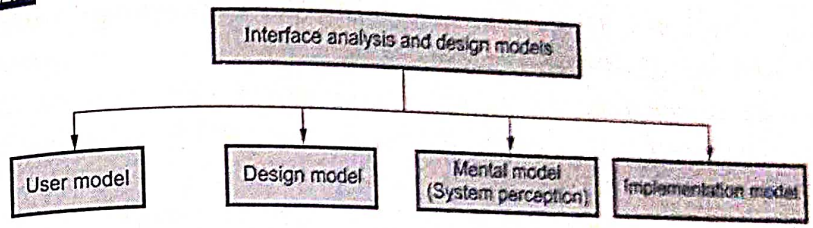


Fig. 3.19.1 Analysis and design model

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, economical 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 dependant 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!

3.19.2.2 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 below 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.

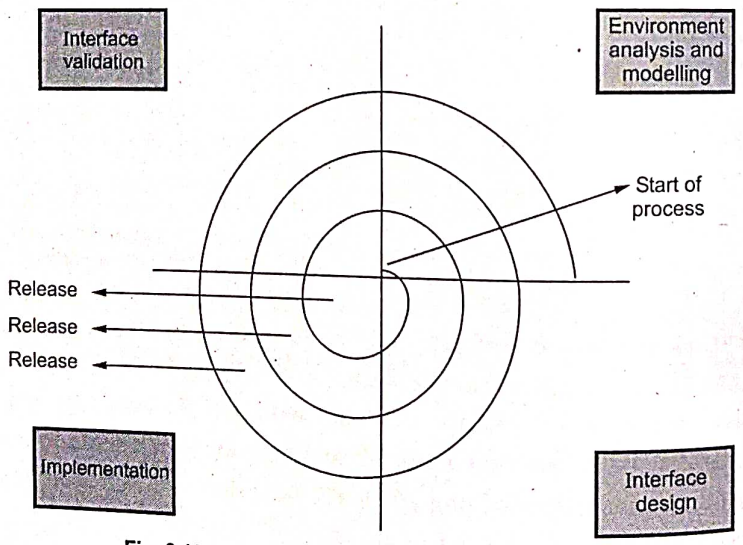


Fig. 3.19.2 Interface analysis and design process

Environment analysis and modelling - In this phase three major factors are focused i.e. user, task and environment. First of all the user profile is analysed to understand the user and to elicit the requirements, then the tasks that are required to carry out desired functionality are identified and analysed. 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 design - The interface design is a phase in which all the interface objects and corresponding actions of each task are defined.

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

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

3.19.3 Interface Analysis

- Before preceding 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.

3.19.3.1 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.

3.19.3.2 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 swimlane diagram. In swimlane diagram, the modelling is done for the classes who are responsible for carrying out the activities.

See Fig. 3.19.3 on next page.

As shown in Fig. 3.19.3 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.
- The finance service will receive the payment.
- The booking service will issue the tickets.

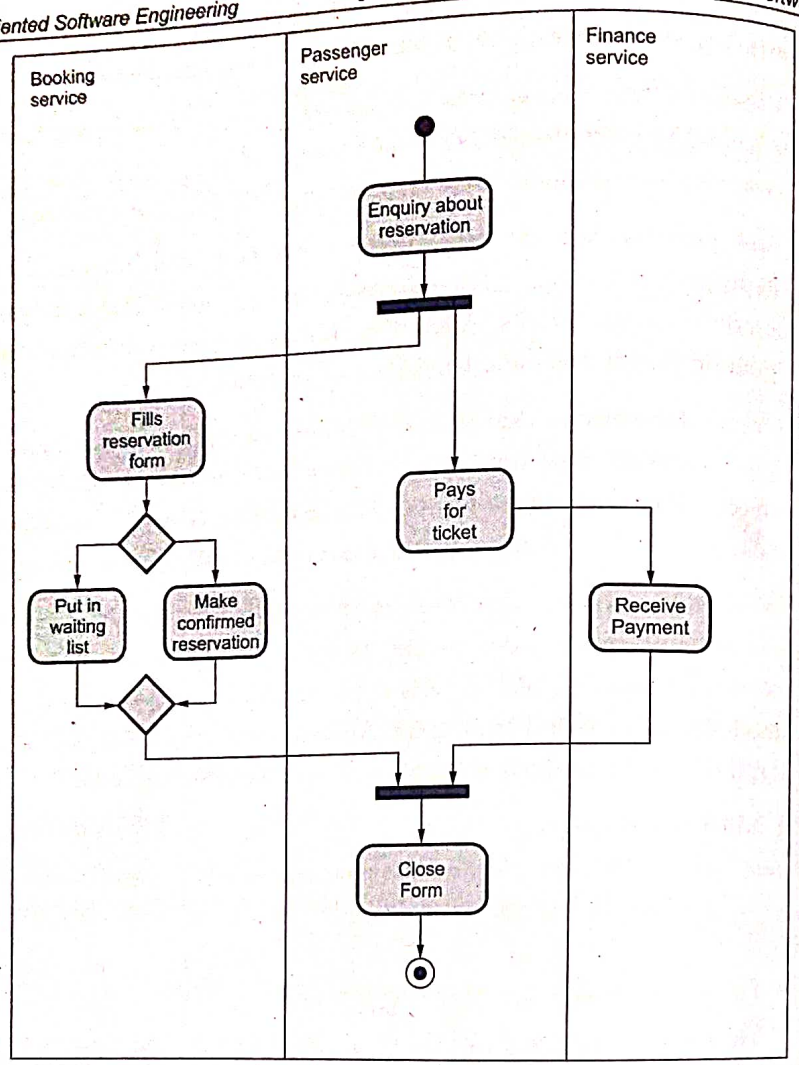


Fig. 3.19.3

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**
  - **i. Specify name(s)**

ii. Specify age

iii. Desired source city

iv. Desired destination city

v. Date and time

Reservation details

i. Name(s)

ii. Age

iii. Source city

iv. Destination city

v. Date and time of train

vi. Seat numbers

vii. Type of reservation

viii. Name of the train

3.19.3.3 Analysis of Display Content

- The contents that get displayed as an 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.

3.19.3.4 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.

3.19.4 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. **Analyse** 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 it 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
- Analyse the working environment

Let us now discuss with some example how to apply these design steps -

3.19.4.1 Application of Interface Design Steps

As we know the first step in any interface design is to **identify** all the necessary **objects and corresponding actions**. The use cases are used for this purpose. That is, the use case description is parsed and the nouns and verbs from this description is identified. The nouns correspond to objects and verbs correspond to actions. Thus a list of objects and actions is prepared.

There are three types of objects

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

After identifying all the necessary objects and actions the screen layout can be prepared. The creation of screen layout includes placing of useful icons, descriptive text, menus and windows. This layout should resemble the real world description of the application.

For example : Online student registration

Some part of problem description is given below for online student registration system.

"A **student** will be typical user of this system. This user has to fill up an online **registration form**. In this form he has to submit the **student information** and then he has to select the **courses** which he/she wants to adopt. Then the **Time Table** of the corresponding course will be displayed to the user. There should be a facility that student can get a print of time table. Then student will be given some **Student ID**, he then has to set the **password** for this ID. If a **teacher** wants to find the particular student teacher will put the student ID and can get the complete information of him."

In above problem description the boldface words indicate the objects and the underline words represent the verbs using this information we can design the user interface as shown below

"A **student** will be typical user of this system. This user has to fill up an online **registration form**. In this form he has to submit the **student information** and then he has to select the **courses** which he/she wants to adopt."

Then the **Time Table** of the corresponding course will be displayed to the user. There should be a facility that student can get a print of time table.

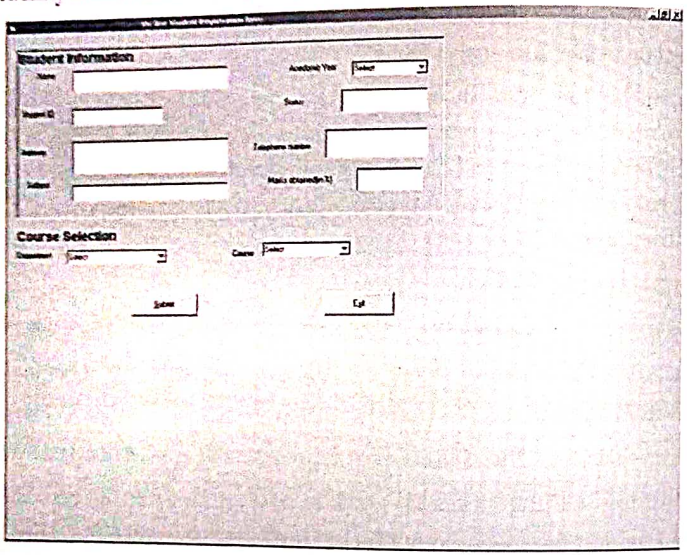


Fig. 3.19.4

If a **teacher** wants to find the particular student teacher will put the student ID and can get the complete information of him.

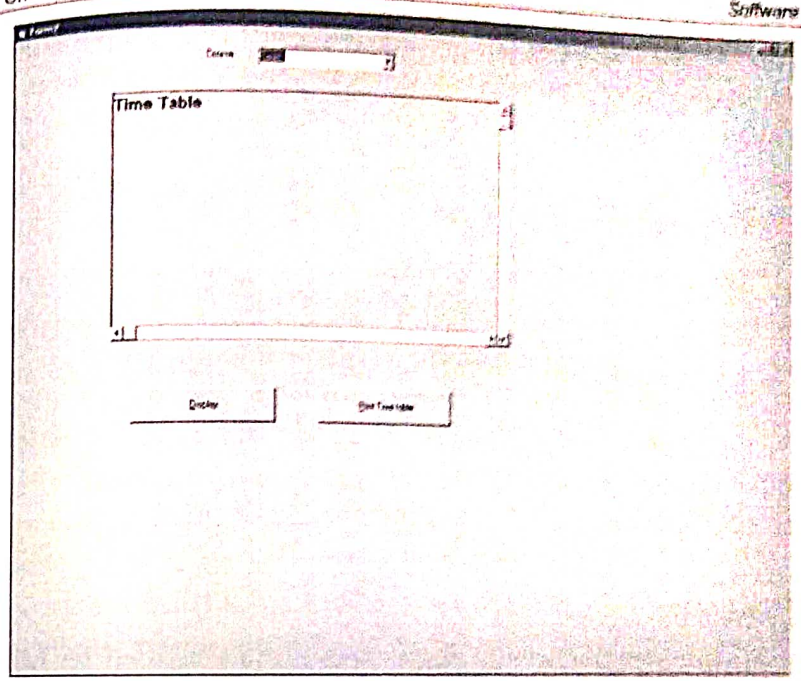


Fig. 3.19.5

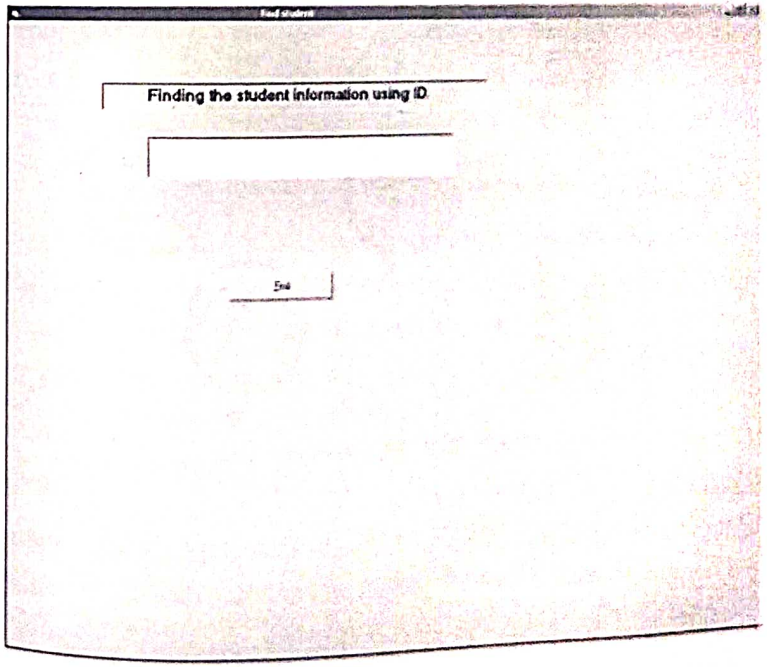


Fig. 3.19.6

3.19.4.2 User Interface Design Pattern

Today it is a common practice to include some user interface design patterns. These design patterns serve as a solution to specific design problem.

For example

Pattern	Meaning
Progress indicator	We found this type of pattern in any installation program. By this pattern user gets a feel that some process is progressing continuously behind the screen.
Wizard	This pattern serves as a guideline for completion of task through various windows.
Shopping cart	It is typically used in on-line purchasing system. It provides the list of items that can be purchased.

3.19.4.3 Design Issues

There are four important interface design issues that are depicted by following Fig. 3.19.7.

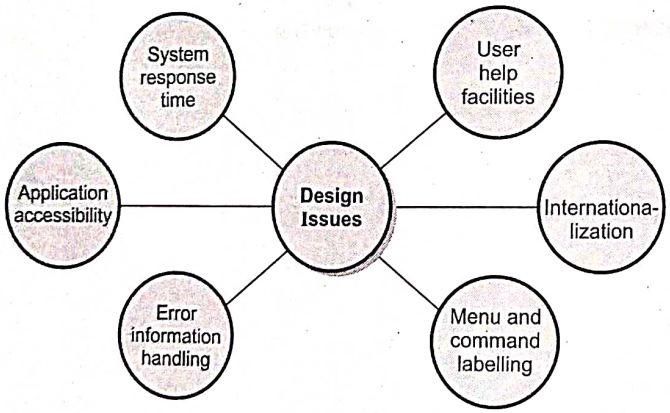


Fig. 3.19.7 Design issues

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).

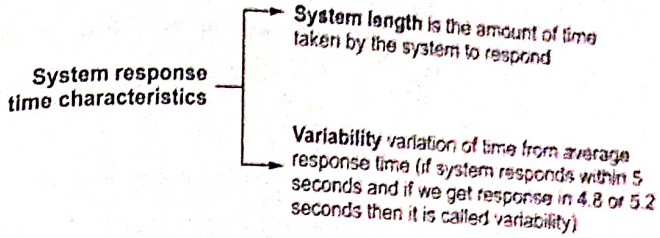


Fig. 3.19.8

Help facilities : This is the most essential criteria for any user interface this makes the system more interactive. The help can be **on-line help** or it can be in the form of **user manual**. Even some software provide the help in the **assistance** form i.e. the question may be asked by the user and the answer given by the system will serve as a help to the user.

Error handling : Errors and warning cause the frustrations to the user. Sometimes the error messages are in very vague form. For example : Application

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.

1. The error message should be in a language which user can understand.
   For example : Remote server not responding.
2. There should be some useful message along with error in order to recover from the error. For example : Press enter to exit
3. There should be some audible or visual cue along with the error message. For example: beep sound or highlighting back ground of the text.
4. 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.
5. 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 every where. 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 localised 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 **globalised** software. That means the user interfaces should be designed to accommodate a generic core functionality. This functionality can be used by any user belonging to any country.
- Localized user interface is useful to only localised 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.

Review Questions

1. *List the activities of user interface design process.* **AU : May-13, Dec.-08, Marks 8**
2. *Discuss about user interface design of a software with an example and neat sketch.* **AU : Dec.-15, Marks 16, Dec.-17, Marks 13**
3. *Describe the golden rules for interface design.* **AU : Dec.-16, Marks 8**
4. *Write short note -User-interface design.* **AU : May-16, Marks 4**
5. *Explain the core activities involved in user interface design process with necessary block diagram.* **AU : Dec.-20, Marks 13**
6. *Describe the golden rules for designing user interface design.* **AU : May 22, Marks 7**
7. *Explain the three golden rules of Theo Mandel in user interface design.* **AU : Dec.-22, Marks 13**

3.20 Two Marks Questions with Answers

Q.1 State the guidelines for modular design. **AU : IT, Dec.-03**

Ans. : The guideline for modular design can be given as :

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

Q.2 How do you evaluate user interface ? **AU : IT, May-03**

Ans. : 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.

Q.3 Brief the importance of user interface. **AU : IT, Dec.-03**

Ans. :

1. An inexperienced user can use the system easily with the user interface.
2. 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.
3. Fast and full screen interaction is possible with user.

Q.4 What is the work product of software design process and who does it ? AU : IT, May-04

Ans. : 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.

Q.5 Define the term software architecture. AU : IT, May-04; CSE, May-08

Ans. : 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.

Q.6 What are the various models produced by the software design process ? AU : IT, May-06

Ans. : Various models produced during design process are -

1. Data design model used to transform the information domain model of analysis phase into the data structures.
2. The architectural design model is used to represent the relationship between major structural elements with the help of some "design patterns."
3. The interface design model describes how software interacts within itself.
4. In the component-level design model the structural elements of software architecture into procedural description of software components.

Q.7 What are the quality parameters considered for effective modular design ?

OR

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

AU : CSE, IT, May-06

Ans. : Various parameters considered for effective modular design are -

1. Functional independence - By using functional independence functions may be compartmentalized and interfaces are simplified. Independent modules are easier to maintain with reduced error propagation.
2. 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.
3. 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.

Q.8 In what way abstraction differs from refinement ? AU : CSE, May-06

Ans. : 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.

Q.9 List out atleast four design principles of a good design. AU : IT, Dec.-06 , May-09,18

Ans. :

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

Q.10 "Modularity is the single attribute of the software that allows a program to be intellectually manageable" - How this is true ? AU : IT, Dec.-06

Ans. : 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".

Q.11 What are the types of coupling ? (Refer section 3.3.7.2) AU : IT, May-07

Q.12 Name the three levels of abstraction, which are in practice for the design. (Refer section 3.3.1) AU : IT, May-07

Q.13 What are the steps involved in design stage of a software ? (Refer section 3.2) AU : CSE, May-07

Q.14 What are various supporting documents to be prepared for the software ? Explain. AU : CSE, May-07

Ans. :

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.

Q.15 What is an architectural style ? (Refer section 3.14) **AU : CSE, Dec.-07**

Q.16 What are the types of interface design ? **AU : CSE, May-08**

Ans. : Following are the types of interface design.

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

Q.17 Why modularity is important in software projects ? **AU : IT, Dec.-07**

Ans. : 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.

Q.18 Why it is necessary to design the system architecture before specifications are completed ? **AU : IT, May-08**

Ans. : 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.

Q.19 How can we evaluate a design method to determine if it will lead to effective modularity ? **AU : CSE, Dec.-08**

Ans. : Meyer has defined five criteria for evaluating design method to determine if it will lead to effective modularity.

Refer section 3.3.2 for these criteria.

Q.20 If a module has logical cohesion what kind of coupling is this module likely to have with others. **AU : CSE, May-09, May-16**

Ans. : 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.

Q.21 What is system design ? **AU : IT, May-09**

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

Q.22 Define data abstraction. **AU : May-13**

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

Q.23 What is software architecture ? **AU : May 14**

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

Q.24 Define : Modularity. **AU : Dec.-14**

Ans. : The modularity is an approach used during the designing of the software system. In this approach, the software is divided into separately named and addressable components called **modules.** Due to modularity, the program becomes manageable.

Q.25 A system must be loosely coupled and highly cohesive. Justify. **AU : Dec.-14**

Ans. : Loose coupling is an approach to interconnecting the components in a system or network so that those components, are less dependant upon each other. High cohesiveness indicate that module perform only one task. Ideally the components in the system must be least dependent on each other and every component must perform only one task at particular instance. Hence it is said that the system must be loosely coupled and highly cohesive.

Q.26 List down the steps to be followed for user Interface Design. **AU : May-15**

Ans. : 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.

Q.27 What are the golden rules for an interface design ? **AU : Dec.-15**

Ans. : There are 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.

Q.28 How can refactoring be made more effective ? AU : May-16

Ans. : After careful analysis of design (or code) separate out the components that can be refactored. Then refactor them and finally integrate and test them.

Q.29 What architectural styles are preferred for the following systems ? Why ?

a) Networking b) Web based systems c) Banking system AU : Dec.-16

Ans. : a) Pipe and filter can be used for networking system. The pipe and filter 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

- **At outer layer :** There are component service user interface operations.
- **At inner layer :** Components perform operating system interfacing
- **At intermediate layer :** There are utility services 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 objects are identified and corresponding operations are performed.

Q.30 What UI design patterns are used for the following ?

a) Page layout b) Tables

c) Navigation through menus and web pages

d) Shopping cart AU : Dec.-16, May-17,18

Ans. :

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

Q.31 List out the various types of cohesion and coupling. AU : Dec.-20

Ans. : Different types of cohesion are -

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

Different types of coupling are -

1) Data coupling
2) Control coupling
3) Common coupling
4) Content coupling

Q.32 Define cohesion and coupling. AU : May-22

Ans. : 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.

Q.33 What is software design ? AU : Dec.-22

Ans. : 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.

❑❑❑