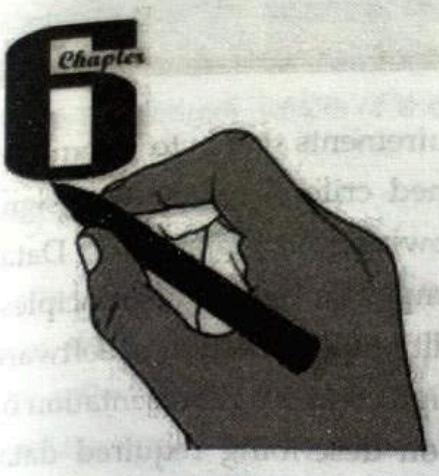
Chapter 6



SOFTWARE ARCHITECTURAL DESIGN

CHAPTER OUTLINE

After studying this chapter, the reader will be able to understand the

- Introduction
- Design Concept
- Architectural Design
- Architectural Views
- System Organization/System structuring
- Application Architectures
- Information Presentation

INTRODUCTION

Design is the meaningful way of representing the customer's requirements that is to be built. It is also the method of linking quality against a set of predefined criteria for good design. Software engineering context, design mainly focuses on the following four major areas: Data, Architecture, Interfaces and Components. Software design encompasses the set of principles, concepts and particles that lead to the development of a high quality representation of software that serve as a guide for the construction activity that follows. Design creates a representation or model of software but unlike the analysis mode that focuses on describing required data, function & behavior, the design model provides detail about software data structures, architecture, interfaces and component that are necessary to implement the system. Design allows a software engineer to model the system or product that is to be built. It is the place where software quality is established. Design depicts the software in a number of different ways. First, the architecture of product be represented. Then, the interfaces that connect the software the end users, to other systems and devices, and its own constituent components are modeled. Finally, the software components that are used to construct the system are designed.

Design begins with the requirements model. We work to transform this model into four levels of design detail: the data structure, the system architecture, the interface representation, and the component level detail. During each design activity, we apply basic concepts and principles that lead to high quality. Ultimately, a Design Specification is produced. The specification is composed of the design models that describe data, architecture, interfaces, and components. Each is a work product of the design process. At each stage, software design work products are reviewed for clarity, correctness, completeness, and consistency with the requirements and with one another.

DESIGN CONCEPT

A set of fundamental software design concepts evolved over the history of software engineering. Fundamental software design concepts provide the necessary framework for getting it right common activities in design process are:

Abstraction

An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer. While designing a system, at highest level of abstraction, a solution is stated in broad terms using the their language of problem environment. At lower level of abstraction a more detailed description of the solution is provided. There are three different types of basic abstraction in software engineering techniques i.e. procedural abstraction, data

abstraction and Control abstraction. The procedural abstraction hides the information on how the process is done; data abstraction hides the data information whereas control abstraction hides the control details of the system.

Architecture

Software architecture is defined as the overall structure of the system which provides the idea how the system is exactly implemented. Software architecture provides the overall structure of the software and the ways in which that structure provide conceptual integrity for a system. Architecture is the structure or organization of program components, the manner in which these components interact and the structure of data that are used by the components. It provides the way in which the components communicate with each other and the data travels in the system. So, we can say that this is a hierarchical structure of the system where subsystem structure and their collaboration is clearly stated and data communication is defined. The process helps engineers to reuse design level concepts. It is a method of representing the system into subsystem and their interface view so that from outside every subsystem are seen easy with only their functionality and how the collaborate. The architectural design can be represented using one or more of a number of different models. Structural models represent architecture as an organized collection of program components. Framework models increase the level of design abstraction by attempting to identify repeatable architectural design frameworks (patterns) that are encountered in similar types of applications. Dynamic models address the behavioral aspects of the program architecture, indicating how the structure or system configuration may change as a function of external events. Process models focus on the design of the business or technical process that the system must accommodate. Finally, functional models can be used to represent the functional hierarchy of a system.

Pattern

Design pattern describes a design structure that solves a particular design problem within a specific context. Design pattern describes a repeatedly presenting issue during software designing, as well the solution to it. Applying design pattern enables developers to reuse it to solve a specified designing issue. Design patterns help designers communicate architectural knowledge, help people learn a new design paradigm, and help new developers avoid traps and pitfalls that have traditionally been learned only by costly experiences. The intent of each design pattern is to provide a description that enables a designer to determine:

i. Whether the pattern is applicable to the current work.
ii. Whether the pattern can be reused.
iii. Whether the pattern can serve as a guide for developing a similar but functionally or structurally different pattern.

Modularity

The act of partitioning a program into individual components can reduce its complexity to some degree. Although partitioning a program is helpful for this reason, a more powerful justification for partitioning a program is that it creates a number of well-defined, documented boundaries within the program. Software architecture and design pattern embody modularity. Software is divided into separately named and addressable components called module, that are integrated to satisfy problem requirements. Modularity is the single attribute software that allows a program to be intellectually manageable.

Information Hiding

The principle of information hiding suggests that modules can be characterized by design decisions that each hides from all others. Modules should be specified and designed so that information (algorithm and data) contained within module is in accessible to other modules that have no need for such information. The principle of information hiding suggests that modules be "characterized by design decisions that (each) hides from all others." In other words, modules should be specified and designed so that information (procedure and data) contained within a module is inaccessible to other modules that have no need for such information. Hiding implies that effective modularity can be achieved by defining a set of independent modules that communicate with one another only that information necessary to achieve software function.

The use of information hiding as a design criterion for modular systems provides the greatest benefits when modifications are required during testing and later during maintenance.

Functional Independence

The concept of functional independence is a direct outgrowth of modularity and the concepts of abstraction and information hiding. Software with independent modules is easier to develop because function may be compartmentalized and interfaces are simplified. Independent modules are easier to maintain. Functional independence is a key to good design and design is the key to software quality. Independence is assessed using two qualitative criteria: cohesion and coupling. Cohesion is an indication of the relative functional strength of a module. It is a natured extension of the information hiding concept.

Refinement

Refinement is a stepwise process in which the software is developed by exploring the different level of abstraction, beginning from the higher levels and, incrementally refining the software through each level of abstraction by exploring more detail of each increment. Hence refinement is movement from higher levels of details to lower levels. Since it is an iterative process where to the keys factor. So the system details are explored until all details can be defined in term of

an operation in computer or programming language. Stepwise refinement is a top down design strategy that defines a process of elaboration. Refinement causes. The designer to elaborate on the original statement, providing more and more detail as each successive refinement occurs. Abstraction and refinement are complementary concept abstraction enables a designer to specify. Procedure and data and yet suppress law level detail as design progresses.

Refactoring

An important design activity suggested for many agile methods, refactoring is reorganization technique that simplifies the design of component without changing its function or behavior.

Refactoring is a process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure of the code yet improves its internal structure. When software is refactored, the existing design is examined for redundancy, unused design elements, insufficient algorithms, poorly constructed data structures or any other design failure that can be corrected to yield a better design.

Cohesion and Coupling

In software engineering, '**coupling**' is used to refer to the degree of interdependence among the different parts of a system. It is easy to see that certain systems can have chains of interdependent modules where, for example, module A depends on module B, which depends on module C, and so on. In some cases these chains may join up and create a circular dependency, which is a particular form of strong (or high) coupling.

Developers try to construct loosely coupled systems because they are easier to understand and maintain. So a good software system has low coupling, which means that changes to one part are less likely to propagate through the rest of the system. A further benefit of low coupling is that components are easy to replace and, potentially, reuse.

Whatever the level of coupling in a software system, it is important to know which modules are coupled. If there were no records of the coupling between modules, a developer would have to spend time working through the modules to determine whether or not each was affected by a change. The result would be a lot of effort spent on checking, even if no changes were needed.

Cohesion is a way of describing how closely the activities within a single module are related to each other. Cohesion is a general concept – for example, a department in an organization might have a cohesive set of responsibilities (accounts, say), or not (miscellaneous services). In software systems, a highly cohesive module performs one task or achieves a single objective – 'do one thing and do it well' is a useful motto to apply. A module should implement a single logical task or a single logical entity.

Low coupling and high cohesion are competing goals. If every module does only one thing at a low level of abstraction, we might need a complex edifice of highly coupled modules to perform an activity at higher levels of abstraction. A developer should try to achieve the best balance between the levels of coupling and cohesion for a software system.

ARCHITECTURAL DESIGN

Architectural design is the initial design process of identifying sub system is establishing a framework for sub system control and communication is called architectural design. It is the first stage in the design process and presents a critical link between the design and requirement engineering. Architectural design concerned with establishing a basic structural framework that identifies the major components of a system and the communication between these components. It is the link between design and requirement engineering and it identifies the main structural components in a system and the relationships between them. The output of architectural design process is an architectural model that describes how the system is organized as a set of communicating components. Architectural design has following three advantages:

a. **Stakeholders Communication:** The architectural design of software system is a high level presentation of the system that may be used as medium of discussion by a range of different stakeholders to understand the system in more detail and to identify the coherent picture of application.

b. **System Analysis:** Architectural design of system can be used as medium of requirement analysis. It has a profound effect on whether or not the system can meet critical requirement such as performance, reliability and maintainability.

c. **Large scale reuse:** The system architecture is often same for systems with similar requirements and so can support large scale software reuse. It can be possible to development of architecture where the same architecture is reused across a range of related systems.

Architectural Design Decisions

In architectural design, the architects have to make number of decisions that mainly affect the system and its development process. The architects have to focus on different issues such as:

- Is there a generic application architecture that can be used?
- What approach will be used to structure the system?
- What architectural styles are appropriate?
- How will the system be distributed?
- What control strategy should be used?
- How should the architecture be documented?
- How will the system be decomposed into modules?
- How will the architectural design be evaluated?

The architectural design of the system affects the performance, dependability, maintainability etc of the system. The particular architectural design and structure chosen for an application depends on the non-functional system requirements.

a. **Maintainability:** While designing architectural design, if maintainability is the critical non functional requirement the system architecture should be designed using self contained components that may be readily be changed and shared data structures should be avoided.

b. **Performance:** In architectural design if performance is the critical requirement it should be designed to localize critical operations with in small number of sub systems with little communication as possible between sub systems.

c. **Availability:** If availability is the critical requirements, this suggests that the architecture should be designed to include redundant components so that it is possible to replace and update components without stopping the system.

d. **Security:** In architectural design, if security is the critical requirement then layered architecture should be used with the most critical assets protected in innermost layer and high level of validation is performed in that layer.

e. **Safety:** If safety is the critical requirement then architecture should be designed so that safety related operations are all located either in single subsystem or small number of sun system.

ARCHITECTURAL VIEWS

Software architecture involves the high level structure of software system abstraction, by using decomposition and composition, with architectural style and quality attributes. A software architecture design must conform to the major functionality and performance requirements of the system, as well as satisfy the non-functional requirements such as reliability, scalability, portability, and availability. A view is a representation of an entire system from the perspective of a related set of concerns. It is used to describe the system from the viewpoint of different stakeholders such as end-users, developers, project managers, and testers. It provides four essential views:

a. **Physical View:** It describes the mapping of software onto hardware and reflects its distributed aspect. It focuses on how system components distributed across the processors in the system and useful for planning deployment of the system.

b. **Logical View:** It describes the object model of the design. This view provides the abstraction in the system in terms of object or object classes.

c. **Process View:** It describes the activities of the system, captures the concurrency and synchronization aspects of the design. It shows how the system is composed of interacting processes at run time. It is used to make judgment about non functional requirements of the system.

d. **Development View:** It describes the static organization or structure of the software in its development of environment. It focuses on how the software system is decomposed into components that can be developed by individual developer.

SYSTEM ORGANIZATION/SYSTEM STRUCTURING

The organization of a system reflects the basic strategy that is used to structure a system. Deciding the organization of a system is important because it affect the overall development of software. Various style or models must be adopted the develop the system architecture. These models are:

- Repository model
- Client server model
- Layered model
- MVC model

Repository Model

Repository architecture consists of a central data structure and collection of independent components which operate on the central data structure. The major system that use large amounts of data are organized around a shared database or repository. Repositories are very important for data integration, introduced in a variety of applications, management information system, including software development, CAD systems case toolset. Repository model works in two fundamental ways. In first approach all shared data is held in a central database that can be accessed by all sub systems. Similarly in another approach each sub system maintains its own database and data is interchanged with another sub system by passing messages to them. This model is applicable to applications where data is generated by one sub system and used by another sub system.

This architecture can be viewed as.

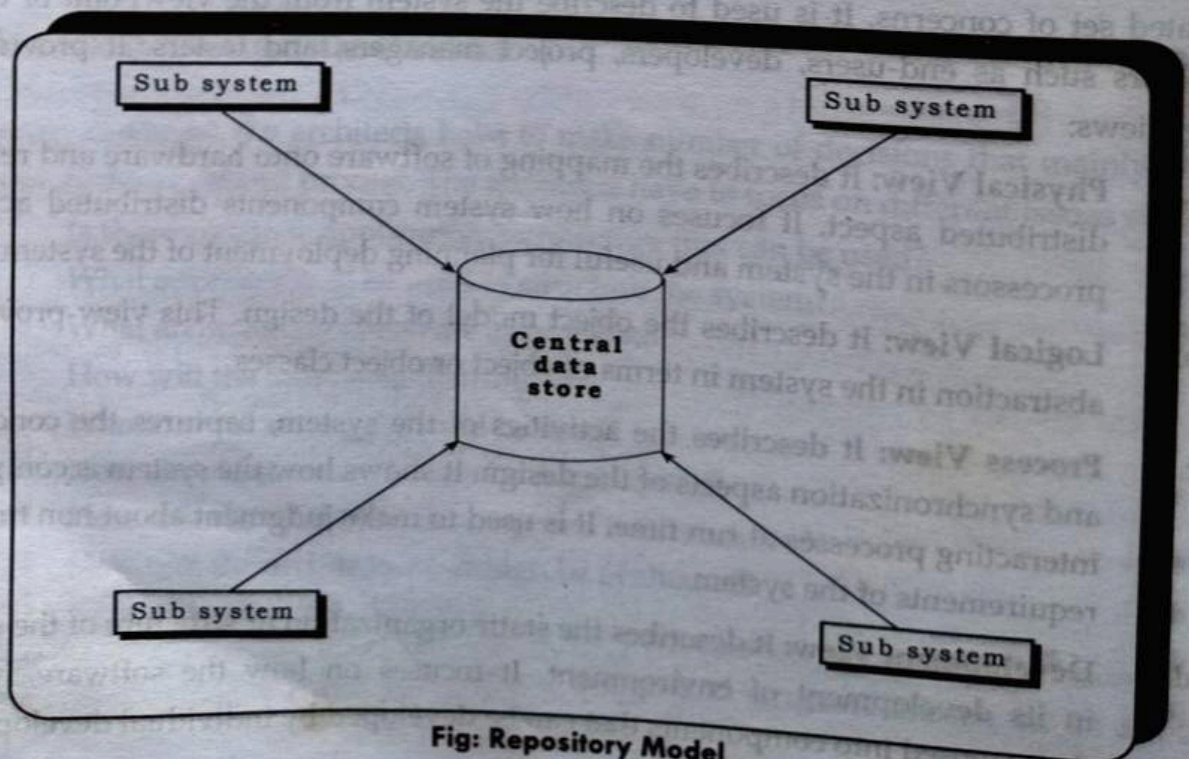


Fig: Repository Model

Advantages

- A large amount of data can be shared among all the sub systems and there is no need of separate mechanism for transmitting the information.
- Sub-systems that produce data need not be concerned with has that data is used by other sub systems.
- In centralized repository model contain problems such as data redundancy and inconsistency must be handled.

Disadvantages

- Making changes in such a model is expensive.
- Evolution may be difficult as a large volume of information is gathered according to an agreed data model.
- If may be difficult to distribute the repository over a no. of machines.

Client Server Model

The client/server architectural style describes distributed systems that involve a separate client and server system and a connecting network. The simplest form of client/server system involves a server application that is accessed directly by multiple clients, referred to as a 2-tier architectural style. It is a model where the system is organized as a set of services and associated servers and clients that access and use it. In this architecture, the functionality of the system is organized into services with each service delivered from a separate server. In this model clients are users of these services and access servers to make use of them. It is used when data in a shared database has to be accessed form a range of locations.

- **Client-Queue-client systems:** This approach allows clients to communicate with other clients to communicate with other clients through a server based queue. Clients can read data from and send data to a server that acts simply as a queue to store the data. This allows clients to distribute and synchronize files and information. This is sometimes known as a passive queue architecture.
- **Per to-peer (p2p) application:** Developed from the client-queue-client style, the p2p styles allows the client and server to swap their roles in order to distribute and synchronize files and information across clients. It extends the client/server style through multiple responses to requests, shared data, resource discovery, and resilience to removal of peers.
- **Application servers:** A specialize architectural style where the server hosts and executes applications and services that a thin client accesses through a browser or specialized client installed software. An example is a client executing on application that runs on the server through a framework such as terminal services.

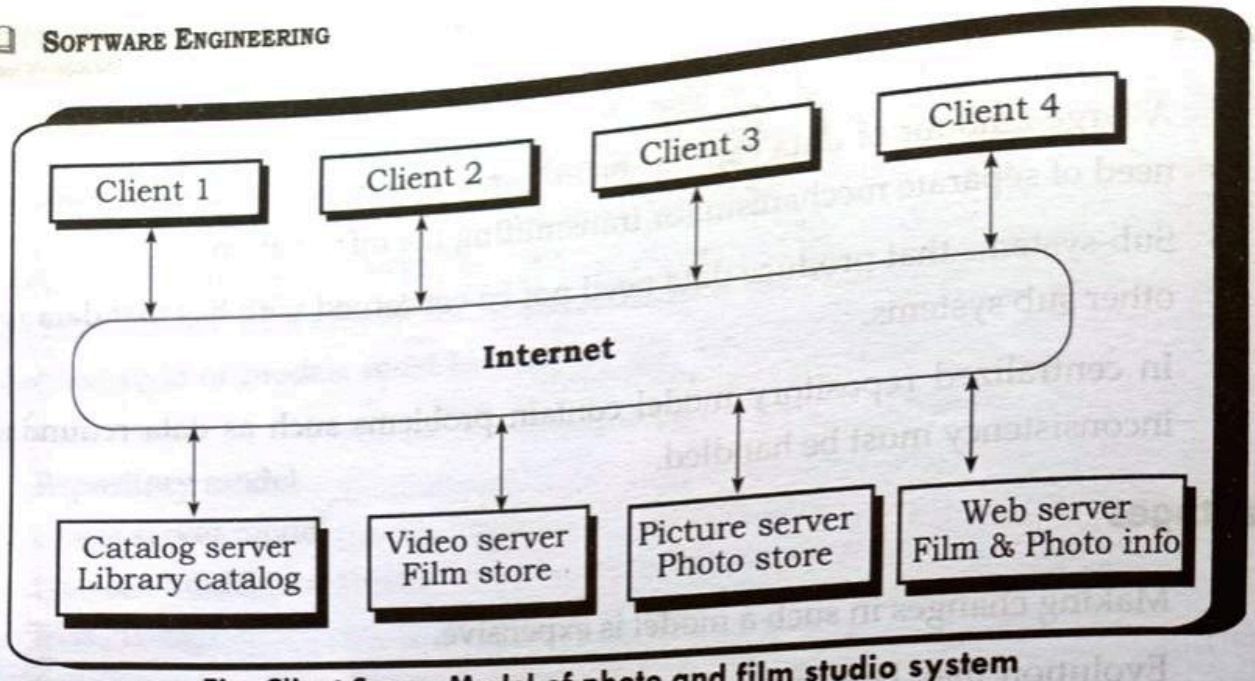


Fig: Client Server Model of photo and film studio system

This model is used when data in a shared database has to be accessed from a range of locations because servers can be replicated may also be used when the load on a system is variable. The major advantage of this model is that servers can be distributed across a network and major disadvantage is each service is a single point of failure so susceptible to denial of service attacks or server failure. Similarly performance may be unpredictable because it depends on the network as well as the system.

Layered Architectural Style

Layered architecture is an architectural model that is used to model the interfacing of sub-systems. Layered architecture focuses on the grouping of related functionality within an application into distinct layers that are stacked vertically on top of each others. The layered model of architecture organizes the systems into a set of layers (or abstract machine) each of which provides a set of services. It is used when building new facilitates on the top of existing systems, when the development is spread across several teams with each team responsibility for a layer of functionality and when there is requirement for multi-level security. It supports the incremental development of sub systems in different layers. When a layer interface changes, only the adjacent layer is affected. It is used when building new facilities on the top of existing systems and when the development is spread across several teams and requirement for multilevel security.

The major advantage of this model is that It allows replacement of entire layers so long as the interface is maintained. Redundant facilities (e.g. authentication) can be providing in each layer to increase the dependability of system. This architecture is changeable and portable. As layered system localizes machine dependencies in inner layers, this makes it easier to provide multiplatform implementations of an application system.

The major disadvantage of this model is that Structuring system in this way can be difficult. Providing a clean separation between layers is often difficult and high level layer may have to interact directly with lower level layers rather than through the layer immediately below it. Sometimes Performance can be problem because of multiple levels of interpretation of a services request as it is processed at each layer.

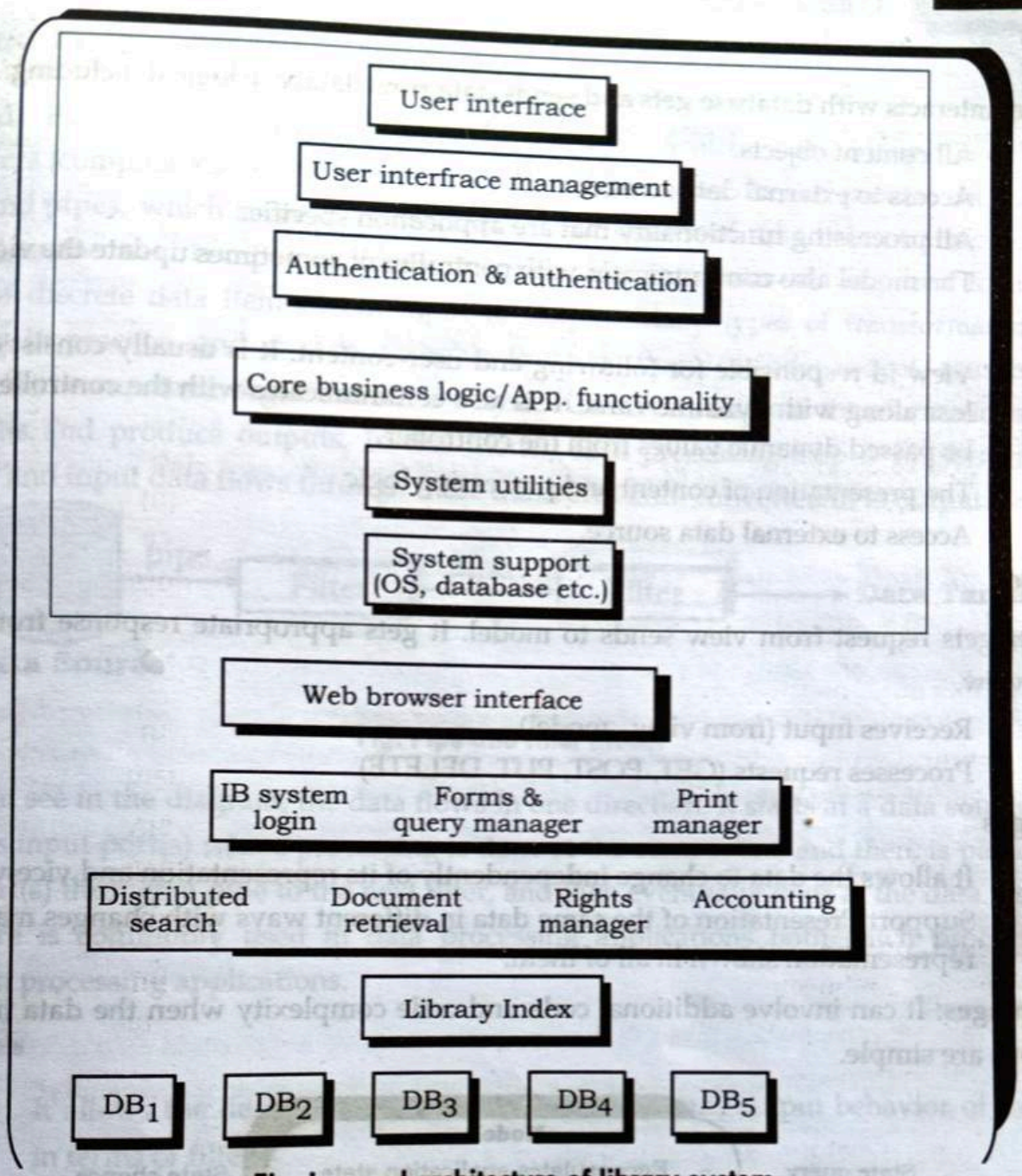


Fig: Layered architecture of library system

MVC Architecture

MVC stands for model view controller. It is a concept or architectural design pattern and basis of interaction management in many web based systems. It separates presentation and interaction from the system data. In this model system is structured into three logical components that interact with each other. The model components manage the system data and associated operations on that data. The view component defines and manages how the data is presented to the user and the controller component manages user interaction and passes these interactions to the view and model. This model is used when there are multiple ways to view and interact with data as well as when future requirement for interaction and presentation of data are unknown. The MVC pattern separates the element of system allowing them to change independently. For example adding new view or changing an existing view can be done without any changes to the underlying data in the model.

Model

The model interacts with database gets and sends data runs database logical including:

- All content objects.
- Access to external data sources.
- All processing functionality that are application specific.
- The model also communicates with controller of sometimes update the view.

View

- View id responsible for following end user content. It is usually consist of HTML loss along with dynamic value. It is also communicates with the controller and can be passed dynamic values from the controller.
- The presentation of content and processing logic
- Access to external data source.

Controller

Controller gets request from view sends to model. It gets appropriate response from models sends to view.

- Receives input (from view, model)
- Processes requests (GET, POST, PUT, DELETE)

Advantages

- It allows the data to change independently of its representation and vice-versa.
- Support Presentation of the same data in different ways with changes made in one representation shown in all of them.

Disadvantages: It can involve additional code and code complexity when the data model and interactions are simple.

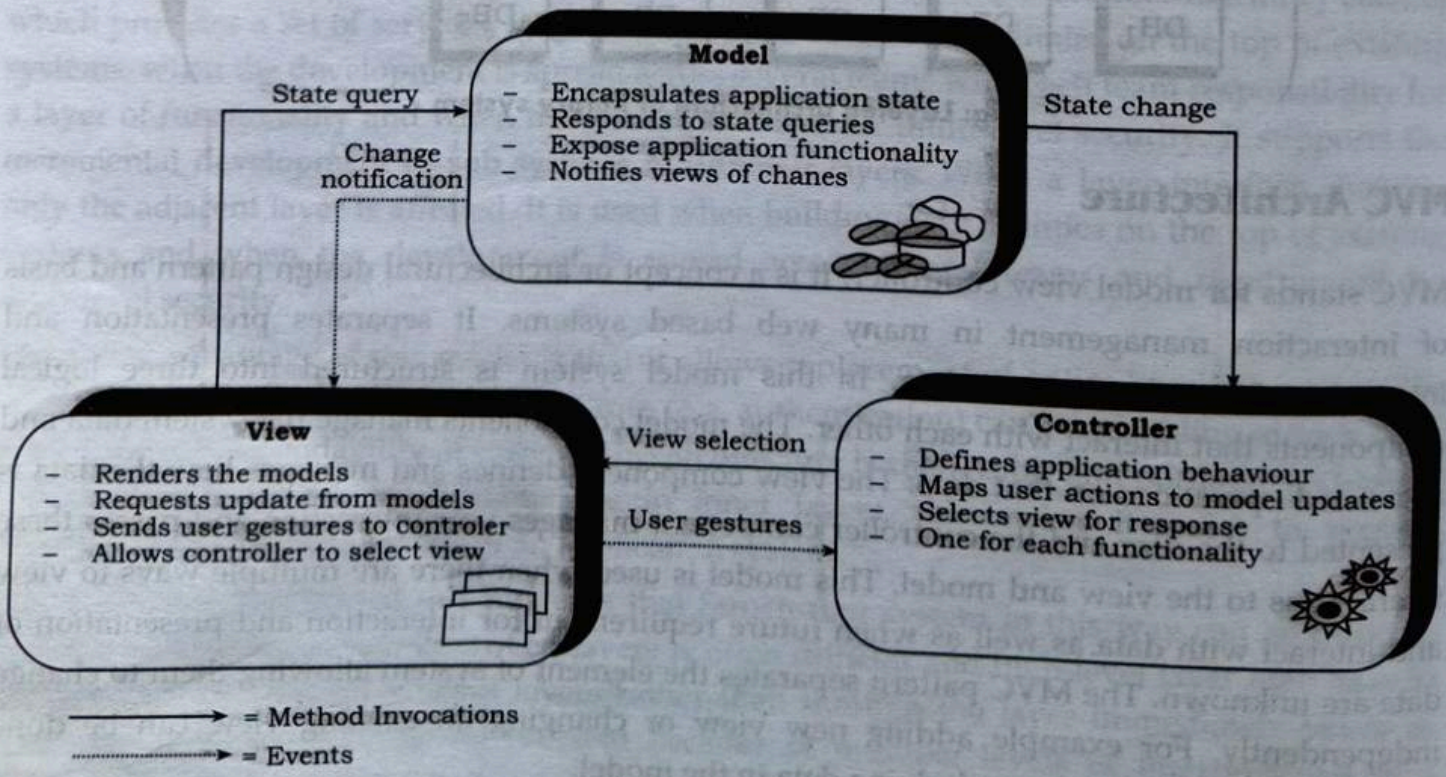


Fig: MVC Model

Pipe and Filter Model

Pipe and Filter is another architectural pattern, which has independent entities called **filters** (components) which perform transformations on data and process the input they receive, and **pipes**, which serve as connectors for the stream of data being transformed, each connected to the next component in the pipeline. Many systems are required to transform streams of discrete data items, from input to output. Many types of transformations occur repeatedly in practice, and so it is desirable to create these as independent, reusable parts, Filters. It is the run time organization of a system where functional transformations process their inputs and produce outputs. In this model, each processing step is implemented as a transform and input data flows through these transform until converted in to output.

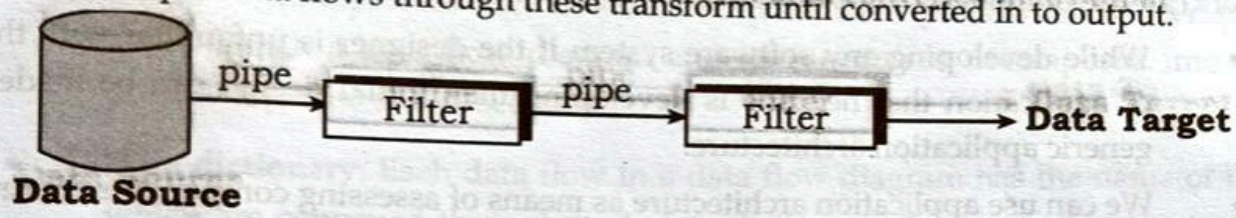


Fig: Pipe and filter model

As you can see in the diagram, the data flows in one direction. It starts at a data source, arrives at a filter's input port(s) where processing is done at the component, and then, is passed via its output port(s) through a pipe to the next filter, and then eventually ends at the data target. This architecture is commonly used in data processing applications both batch processing and transaction processing applications.

Advantages

- It allows the designer to understand overall input/output behavior of the system in terms of filter.
- Easy to understand and support transformation reuse.
- Easy maintenance and reuse.
- Can be implemented as either a sequential system or concurrent system.
- Each filter can be implemented as a separate task and be executed in parallel with other filters.

Disadvantages

- Interactive transformations are difficult because filters being independent entities and designers have to think of each filter as providing a complete transformation of input data to output.
- Each transformation must parse its input and unparsed its output to the agreed form.
- No filter cooperation.

APPLICATION ARCHITECTURES

Application architecture describes the patterns and techniques used to design and build an application. The architecture gives you a roadmap and best practices to follow when building an application, so that you end up with a well-structured app. Application Architecture is the process of defining the framework of an organization's application solutions against business requirements. It involves the definition of the application landscape, aiming to optimize this landscape against the ideal blueprint. An application architecture helps ensure that applications are scalable and reliable, and assist enterprises identifies gaps in functionality. Application architecture encapsulates the principal characteristics of a class of system and designers can use of models of application architecture in number of ways:

- While developing any software system if the designer is unfamiliar with the type of application that he/ she is developing then initial design can be made using generic application architecture.
- We can use application architecture as means of assessing components for reuse.
- If an architecture design of an application has been already developed then we can compare this with generic application architecture to identify whether design is consistent or not.
- Application architecture can be used to compare the applications of same type.

Procedural Design

Data, architectural, and interface design must be translated into operational software. To accomplish this, the design must be represented at a level of abstraction that is close to code. Component-level design establishes the algorithmic detail required to manipulate data structures, effect communication between software components via their interfaces, and implement the processing algorithms allocated to each component. A software engineer performs component-level design.

You have to be able to determine whether the program will work before you build it. The component-level design represents the software in a way that allows you to review the details of the design for correctness and consistency with earlier design representations (i.e., the data, architectural, and interface designs). It provides a means for assessing whether data structures, interfaces, and algorithms will work. Design representations of data, architecture, and interfaces form the foundation for component-level design. The processing narrative for each component is translated into a procedural design model using a set of structured programming constructs. Graphical, tabular, or text-based notation is used to represent the design.

The procedural design for each component, represented in graphical, tabular, or text-based notation, is the primary work product produced during component-level design. A design walkthrough or inspection is conducted. The design is examined to determine whether data structures, interfaces, processing sequences, and logical conditions are correct and will produce the appropriate data or control transformation allocated to the component during earlier design steps.

Using Structured Methods

The typical examples of structured methods are the well-known Structure Analysis and Design. Structured Analysis (SA), supports the phase of requirements analysis, while Structured Design is for design phase. Structured Analysis is a technique for capturing the system requirements with the system's functions and the data flows among them. It guides the developers in producing the following artifacts:

- **Data flow diagram**: a directed graph whose nodes denote the interfaces from/to external environments (called external entities or source & sinks), the functions as data transformations (called processes or bubbles), or data stores holding data, and whose edges represent data flows between nodes. A process can be hierarchically defined with a data flow diagram and its function is decomposed into smaller processes (called sub-processes).
- **Data dictionary**: Each data flow in a data flow diagram has the name of the data which are conveyed through it. A data dictionary represents the structure of the data, i.e. a data type
- **Mini-spec**: As a process is repeatedly decomposed, we can get sub-processes each of which comprises atomic operations only. Mini-specs describe the contents of an atomic operation with structured natural language.

User Interface Design

User interface is the medium of creating effective communication between the human and the computer system. System users often judge a system by its interface rather than its functionality. A poorly designed interface can cause a user to make catastrophic errors. Poor user interface design is the reason why so many software systems are never used. Most users of business systems interact with these systems through graphical interfaces although, in some cases, legacy text-based interfaces are still used. The set of interface design principles, design identifiers, interface objects and actions are identified and then they are used to create a screen layout leads us to a human interface for the communication mechanism with the system to be developed i.e. we can say a user interface prototype. This task is done by the software engineer by following iterative process by applying different designing principles. User-centered design is an approach to UI design where the needs of the user are paramount and where the user is involved in the design process. UI design always involves the development of prototype interfaces.

Following are the common user interface design principles.

- **User familiarity:** The interface should be based on user-oriented terms and concepts rather than computer concepts. For example, an office system should use concepts such as letters, documents, folders etc. rather than directories, file identifiers, etc.

- **Consistency:** The system should display an appropriate level of consistency. Commands and menus should have the same format, command punctuation should be similar, etc.
- **Minimal surprise:** If a command operates in a known way, the user should be able to predict the operation of comparable commands
- **Recoverability:** The system should provide some resilience to user errors and allow the user to recover from errors. This might include an undo facility, confirmation of destructive actions, 'soft' deletes, etc.
- **User guidance:** Some user guidance such as help systems, on-line manuals, etc. should be supplied.
- **User diversity:** Interaction facilities for different types of user should be supported. For example, some users have seeing difficulties and so larger text should be available

Human Computer Interaction (User Interaction)

Human-computer interaction (HCI) is a cross-disciplinary area that deals with the theory, design, implementation, and evaluation of the ways that humans use and interact with computing devices. User interaction means issuing commands and associated data to the computer system.

The designer of a user interface to a computer mainly focused on how the information from the user be provided to the computer system and how can information from the computer system is presented to the user. So, a coherent user interface must integrate user interaction and information presentation.

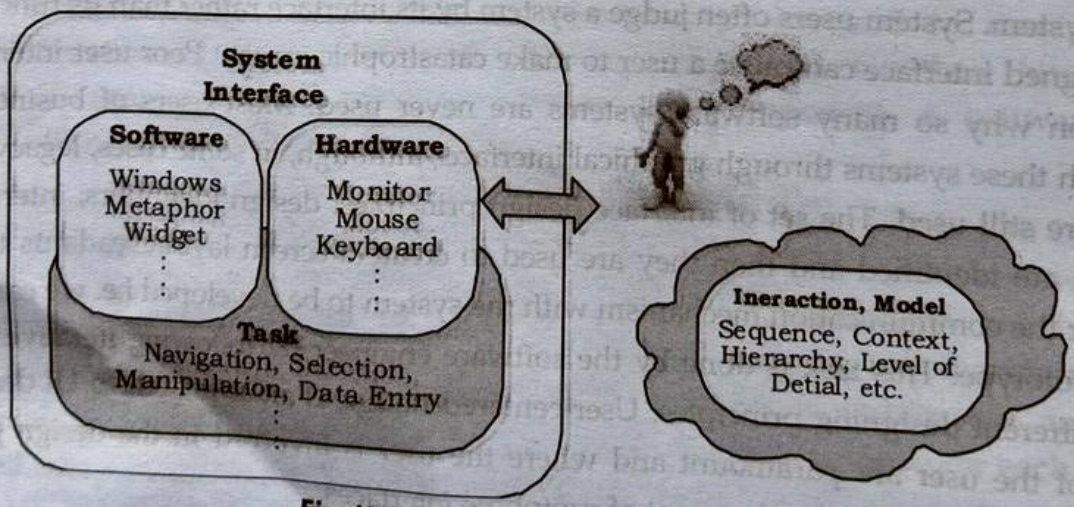


Fig: Human computer Interaction

There are five basic methods of user interactions:

a. **Direct Manipulation:** In this method the user interacts directly with objects on the screen. It is fast and intuitive interaction and easy to learn. But it may be hard to implement, only suitable where there is a visual metaphor for tasks and objects.

b. **Menu Selection:** In this method a user selects a command from the list of possible men. It is often the case that another screen object is selected at the same time and the command operates on that objects. For example: to delete a file, user selects the file then selects the delete command.

c. **Form fill-in:** Here a user fills in the fields of a for. Some fields may have associated menus and the form may have action buttons that when pressed cause some action to be initiated. This is simple data entry and easy to learn but it takes up a lot of screen space.

d. **Command Language:** In this method the user issues a special command and associated parameters to instruct the system what to do. For example to delete a file, the user issues a delete command with the filename as a parameter.

e. **Natural Language:** In this the user issues a command in natural language. It is accessible to causal users and requires more typing. Natural language understanding systems are unreliable. For example: WWW information retrieval systems.

INFORMATION PRESENTATION

Information presentation is concerned with presenting information to system users. The information may be presented directly or may be transformed in some way for presenting such as graphical form. It is good system design practice to keep the software required for information presentation separate from the information itself. By separating the presentation from data, the representation on the users screen can be change without having to change the underlying computational system. Different information presentation techniques can be used such as:

- Text based presentation can be used. It takes less screen space but cannot be read at a glance.
- Dynamically varying numeric information is usually best presented graphically using analogue representation. It gives a user some sense of value.
- When precise alphanumeric information is presented, graphics can be used to pick out the information from its background, it can be displayed in a box or indicated using icon.
- Graphical highlighting can also be used to draw user's attention to changes part of display.
- When large amount of data have to presented, abstract visualization that link related data may be used.
- Graphical information display should be used when it is intended to present trends and approximate values.
- Digital display should be only be used when precision is required.

Interface Evaluation

User interfaces are becoming critical to the success of software products. However, designing and evaluating user interfaces is difficult and time -consuming. There are many reasons why a focus on the user interface is important and why user interface design and evaluation are inherently difficult tasks. The process of finding whether the needs of a user are addressed or not by the user interface prototype is called the interface evaluation. The goal of user interface evaluations is to make products and services more usable, easy to learn and intuitive for the user. It should be the part of the normal verification and validation process for software systems. Learners create an evaluation strategy and validate user interfaces using evaluation techniques and usability testing. Evaluation can span a formality spectrum that ranges from an informal test drive, in which a user provides important feedback to a formally designed study that uses statistical methods for the evaluation of questionnaires completed by a population of end-users.

The evaluation process can be executed as the process illustrated by the figure below.

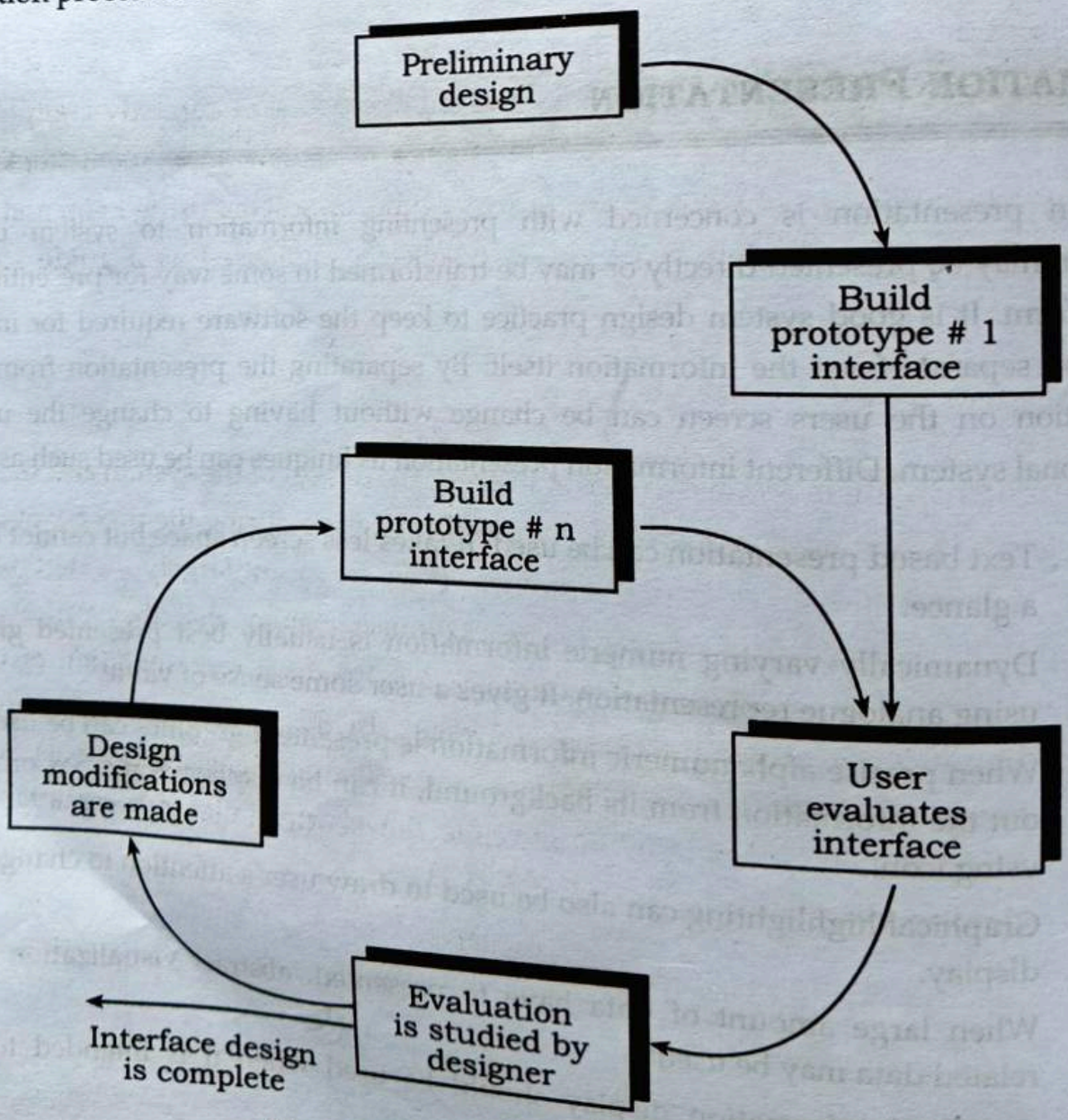


Fig: Interface design evaluation cycle

The prototyping approach is effective, but is it possible to evaluate the quality of a user interface before a prototype is built? If potential problems can be uncovered and corrected early, the number of loops through the evaluation cycle will be reduced and development time will shorten. If a design model of the interface has been created, a number of evaluation criteria can be applied during early design reviews:

1. The length and complexity of the written specification of the system and its interface provide an indication of the amount of learning required by users of the system.
2. The number of user tasks specified and the average number of actions per task provide an indication of interaction time and the overall efficiency of the system.
3. The number of actions, tasks, and system states indicated by the design model imply the memory load on users of the system.
4. Interface style, help facilities, and error handling protocol provide a general indication of the complexity of the interface and the degree to which it will be accepted by the user.

If quantitative data are desired, a form of time study analysis can be conducted. Users are observed during interaction, and data—such as number of tasks correctly completed over a standard time period, frequency of actions, sequence of actions, time spent "looking" at the display, number and types of errors, error recovery time, time spent using help, and number of help references per standard time period—are collected and used as a guide for interface modification.

Design Notation

During design phase there are two things of interest: the design of the system and the process of designing itself. Design notations are largely mean to be used during the process of design and are used to represent design or design decisions. They are meant largely for designers so that they can quickly represent their decisions in a compact manner that they can evaluate and modify. These notations are frequently graphical. It is pictorial representation of the system without any formal use of programming languages. In this method we will represent the system in the form of self-explanatory diagrams using different signs and symbols. Some of the notations used are: data flow diagram, pseudo code, flowchart, tabular representation, structure charts, procedure specification, etc.

1. What is software design? Explain the different software design concepts.
2. Define architectural design. Explain the advantages of architectural design.
3. What is repository architecture? Explain its advantages and disadvantages.
4. Explain client server model and abstract machine model with their advantages and disadvantages.
5. What are control styles? Explain centralized control and event based control.
6. What do you mean by modular decomposition? Explain object oriented decomposition with example.
7. What do you mean by user interface design? Explain user interface design principles.
8. What do you mean by human computer interaction? Explain different methods of user interaction.

❑❑❑