Learning Resource

On

Software Engineering

Chapter-8

User Interface Design

Chapter Outcomes:

After completing this chapter successfully, the students will be able to:

- Identify the purpose of user interface design.
- List characteristics of good user interface.
- Discuss the basic concepts in user interface.
- Explain different types of user interfaces.
- Compare and contrast text based interface with graphical user interface.
- Explain architectural design
- Discuss component design
- Describe web app design

Organization of the Chapter

- Introduction
- Characteristics of good user interface
- Basic Concepts in user interface
- Types of User Interfaces
- Graphical User Interface versus Text-based user interface
- Architectural design
- Component design
- Web app design

Introduction

- The user interface of a software product deals with all **interactions** between the **user and the system**.

- In early days, the software product did not have any user interface as the computers were batch systems which **did not support** user interactions.

- User interface is one of the vital component of a software product as the end-users judge the product based on their **experience** on the interface.

- **Systematic development** of user interface is very important as it takes significant portion of the total development effort.

Characteristics of good user interface

- A good user interface must have following characteristics:
  - **Speed of Learning:** A good user interface must be easy to learn. A good interface neither asks the users to remember commands nor it asks them to remember the information when they navigate from one screen to another screen.
  - **Speed of Use:** This feature indicates the time taken by the users to perform their intended tasks through the system. The interface must not take more time and effort in this regard.
  - **Speed of Recall:** Once a user understands the interface well then they must be able to recall the command issue procedure at a higher speed.

Characteristics of good user interface (contd..)

- **Error Prevention:** A good user interface should minimize the error chances while initiating different commands.
- **Attractive:** An attractive user interface is the need of the hour as it catches the attention of the users. That’s why the GUIs are more preferred over text-based interfaces.
- **Consistency:** The purpose of consistency is to allow users to generalize the knowledge about the aspects of the interface from one part to another.
- **Feedback:** A good user interface must provide feedback to various user actions. In absence of the response from the system for a longer period, a novice user may go for restart or shut down of system that may be completely unwarranted.

Basic Concepts

- **User Guidance**
  - The guidance messages in a user interface prompts the user about the current status of the system, the progress they have made so far, and the next actions they might pursue.
  - A good guidance system should have different levels for different types of users.

- **On-line help:**
  - User may seek help from the system about the operation of the software while using them. This is provided by the **On-line help** system.
  - However, this must not be confused with the guidance and error messages that may be flashed on the screen even without the user asking them.

contd..

- A mode is a collection of states in which only a subset of all user interaction tasks can be performed.
  - **Modeless Interface**
    - In this type of interface all the commands are available at one place and all the time during the operation of the software.
  - **Mode-based Interface**
    - Here, different set of commands can be invoked depending on the mode in which the system is in at present.
    - A mode-based interface can be represented by a state transition diagram, where each node would represent a mode.

Types of Interfaces

- **Command language based interfaces**
  - A command language based interface works on a **language** that a user can use to **issue the commands**.
  - The user is expected to **frame the commands** appropriately and type them whenever needed.
  - A simple command language based interface might simply assign **unique names** to the different commands. However, a more sophisticated interface may allow user to **compose a complex command** using primitive commands.
  - This interface allow **fast interaction with the system** and can be implemented using a low cost alphanumeric display terminals.
  - On the other side, this interface needs the user to **remember the commands** that can be problematic for some users.
  - As this interface does all interactions using keyboard with no use of any pointing device, it may prove **problematic for casual or inexperience users.**

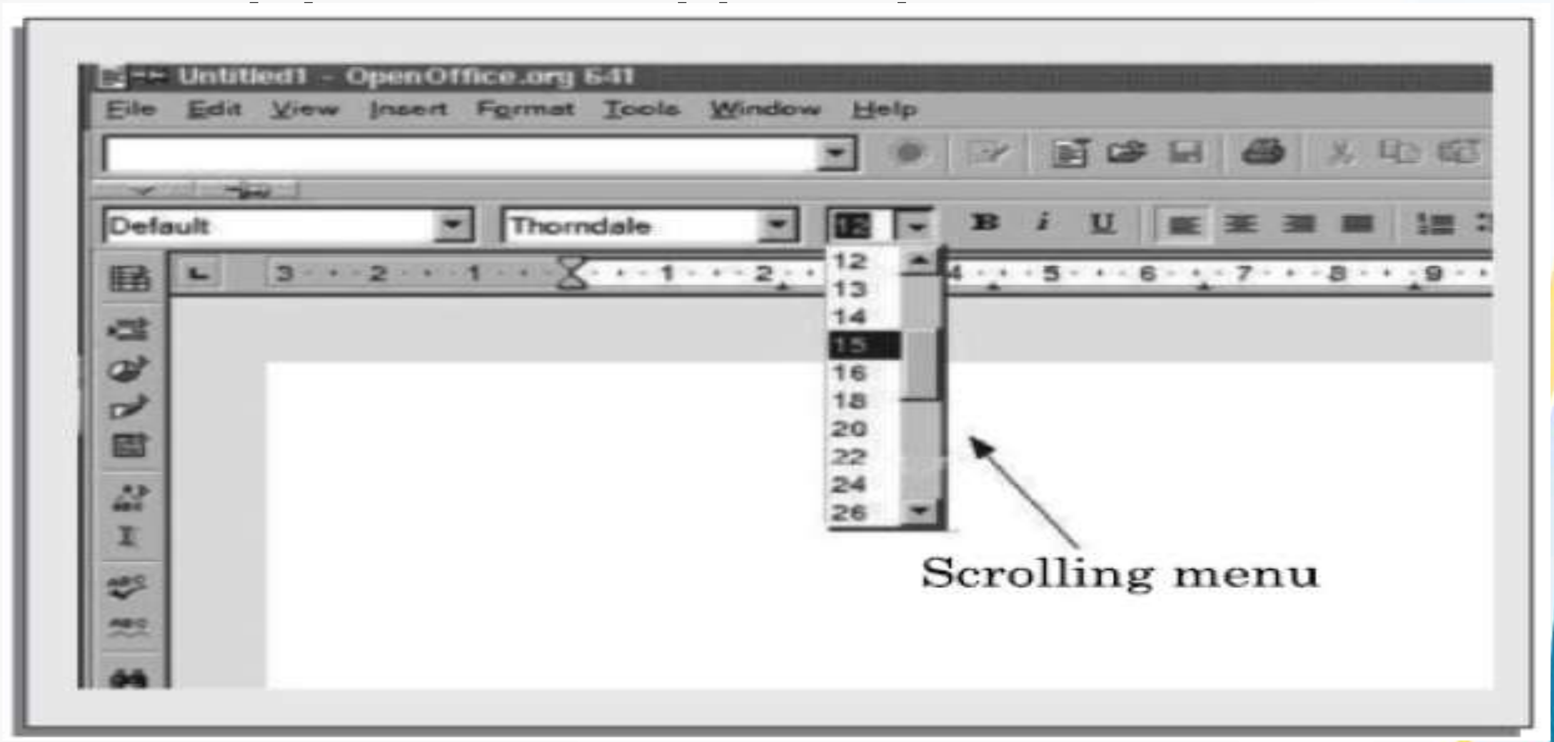
Types of Interfaces (contd..)

- **Menu based interfaces**
  - A menu based interface **does not require the users to remember** the commands and the interactions happen through **menu based selections** rather than recalling and typing the commands.
  - While these interfaces are **helpful** for novice and inexperienced users, the experienced ones find it **difficult to work** with menu based interface. This is because the experienced users with good typing skill can get the speedy interactions using command based interface.
  - Unlike command based interfaces, menu based interface **does not support complex interactions** as logic based interactions are difficult to represent.
  - Menu based interface is suitable for the software that come with **less number of choices** i.e. a software with reasonably large number of choices may not find menu based interfaces suitable.

Types of Menu

– Scrolling Menu

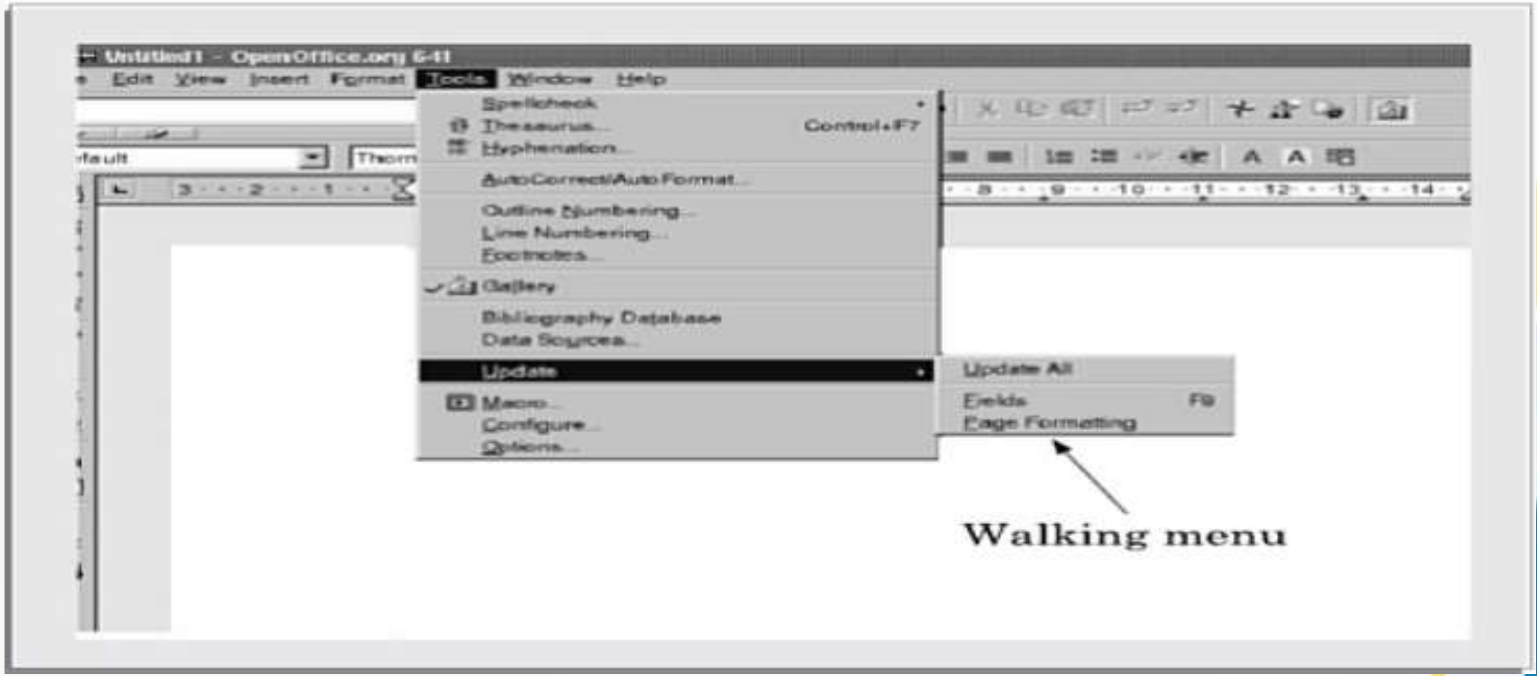
- When the full choice list is large in numbers and can not be displayed in the menu area, scrolling of the menu items is necessary.
- This enables the user to view and select the items that are not visible on the screen.
- All the commands should be highly correlated, so that the user can


Untitled1 - OpenOffice.org 641
File Edit View Insert Format Tools Window Help
Default
Thorndale
12
12
13
14
15
16
18
20
22
24
26
Scrolling menu

Types of Menu

– Walking Menu

- In this type of menu, when an item is selected then the further menu items are displayed in a sub-menu displayed adjacent to the opened menu.
- A walking menu can successfully be used to structure commands only if there are tens rather than hundreds of choices as each adjacent menu displayed will take some space on the screen and the screen size is very much limited.



Types of Menu

- **Hierarchical Menu**
  - This type of menu is suitable for the systems with **small display area like mobile phones**.
  - In this type of menu, the menu items are organized in a **hierarchy or tree like structure**.
  - Selecting a menu item causes the current menu display to be replaced by an appropriate submenu.
  - It can be used to **manage a large number of choices**, but the users are likely to face **navigational issue** as they might lose track of where they are in the menu tree.

Graphical User Interface vs Text-based user Interface

- **Graphical User Interface (GUI)**
  - Here, multiple windows with different information can be shown on the same screen.
  - GUI uses iconic representation to show the user interactions. This symbolic representation makes GUI more user friendly.
  - GUI supports command selection using an attractive and user friendly menu selection system.
  - In GUI, a pointing device can be used to issue commands.
  - However, GUI requires special peripheral devices for implementation.
- **Text-based Interface**
  - It can be implemented using a low cost alphanumeric display terminal.
  - Mostly works on text based commands that is difficult to remember and recall.

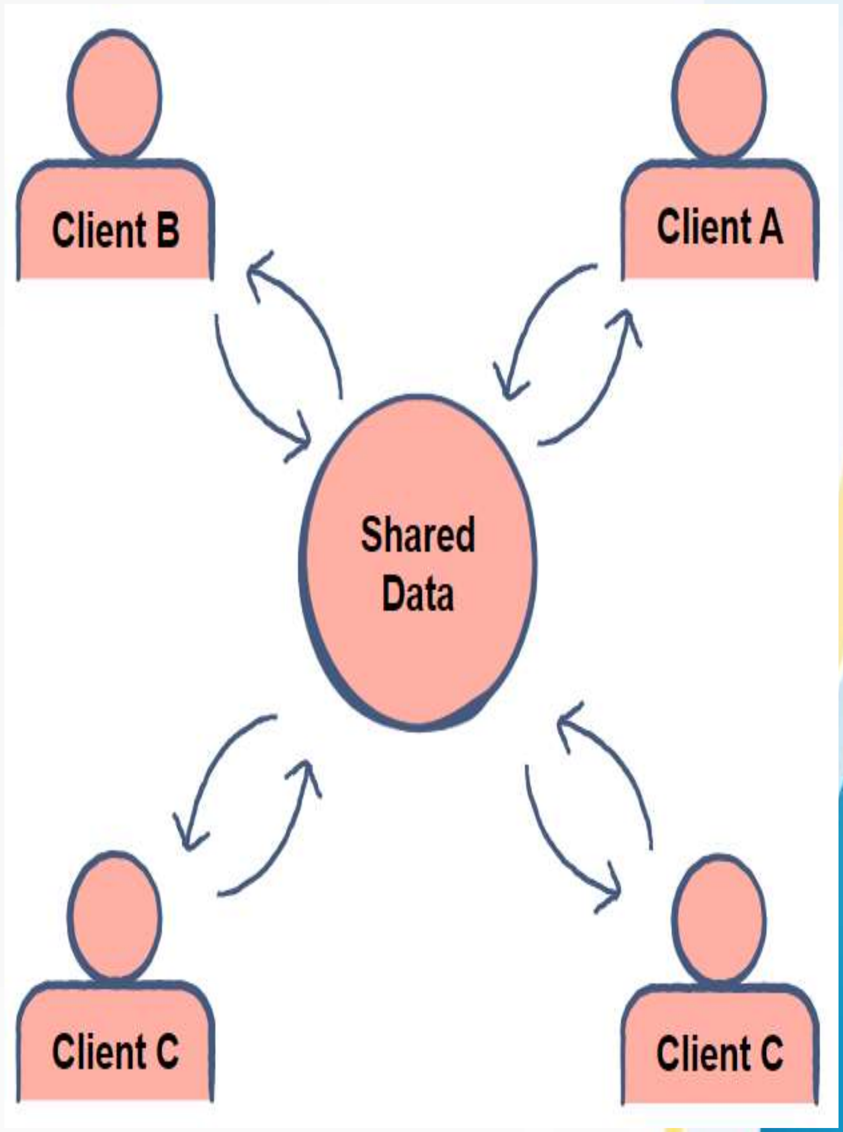
Architectural Design

- To depict the software's design, an architectural design is required.
- IEEE defines architectural design as "the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system".
- Software developed for computer-based systems may display one of these numerous architectural designs.
  - Data centered architeture
  - Data flow architecture
  - Call and return architecture
  - Client Server architecture
  - Layered architecture

Data Centered Architecture

- The core of this design will be a data store, which is regularly accessed by the other parts that add, update, remove, or change the data that is stored there.

- This design has centralised data that is regularly accessible by other components for data modification. The attainment of data integrality is the primary goal of data-centered architecture.



Source: www.educative.io

contd..

Advantage

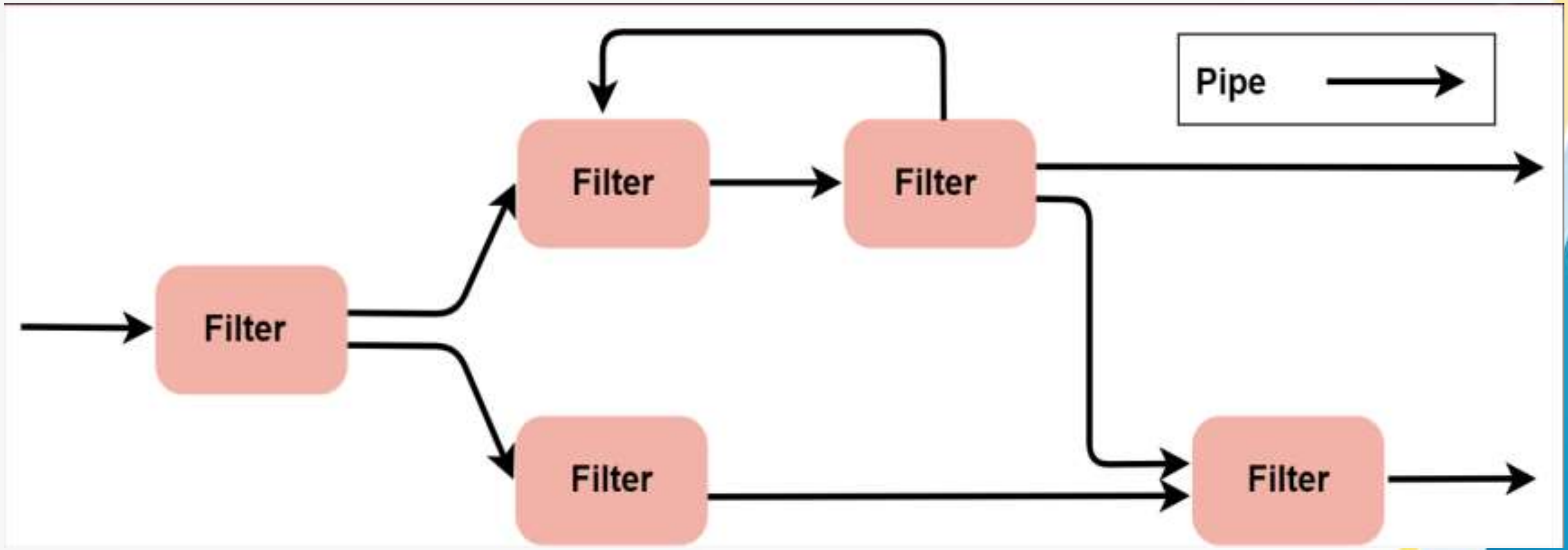
- Repository of data is independent of clients
- Client work independent of each other
- It may be simple to add additional clients.
- Modification can be very easy

Disadvantage

- As the data is centralized, any damage to the data repository will bring the entire architecture at halt.

Data flow architectures

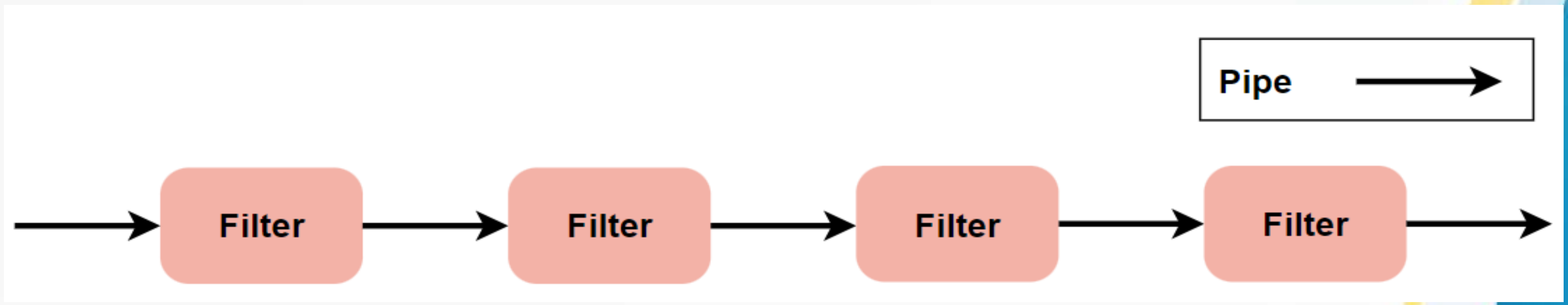
- This kind of architecture is used when input data is transformed into output data through a series of computational manipulative components.
- Given that it incorporates both **pipes and filters**, as well as a collection of parts connected by lines to form filters, the figure exemplifies pipe-and-filter architecture.



Source: www.educative.io

contd..

- Data transmission between components is accomplished via **pipes**.
- Every filter is made to receive a specific kind of data input, process it separately, and output the results in a different form to the filter after it. It is not necessary for the filters to understand how nearby filters operate.
- **Batch sequential data flow** is defined as one in which each transform line becomes a single line. This structure takes the batch of data and transforms it using sequential components one after the other.



Source: www.educative.io

contd..

Advantages of Data Flow architecture:

- It provides modularity in the code.
- The same filters can be reused to create different configurations, which allows for reusability.
- Parallel processing can be done in the pipe and filter architecture.

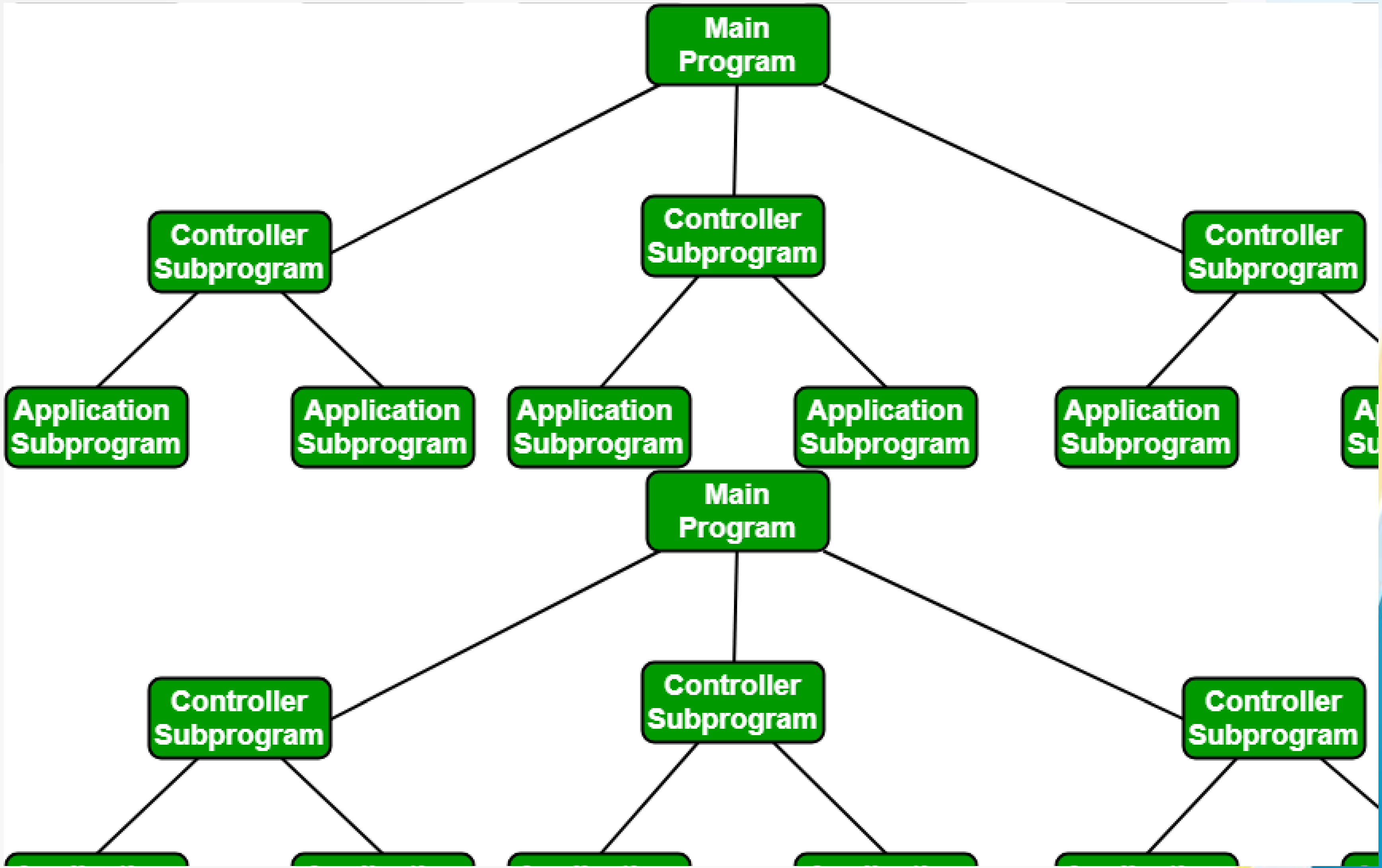
Disadvantages of Data Flow architecture:

- It is not suitable for interactive applications since the output is obtained after a series of transformations.
- Extra filters may be needed to make independent components compatible with one another: i.e., to transform the output of one component to make it compatible with the input of the next.
- If components are not designed independently, it may lead to tight data coupling.

Call and Return architectures

- It is employed to produce programs that are simple to scale and alter. This category contains a wide variety of sub-styles. Below is an explanation of two of them.
  - **Architecture for remote procedure calls:** This component is used to display data in a main program or subprogram that is dispersed over several networked machines.
  - **Main program or subprogram architectures:** A control hierarchy is formed when a number of subprograms or functions break out from the main program structure. Numerous subprograms in the main program have the ability to call other components.

contd..



Source: www.geeksforgeeks.org

contd..

Advantages

- They enable code modularity.
- They are easy to scale and modify.

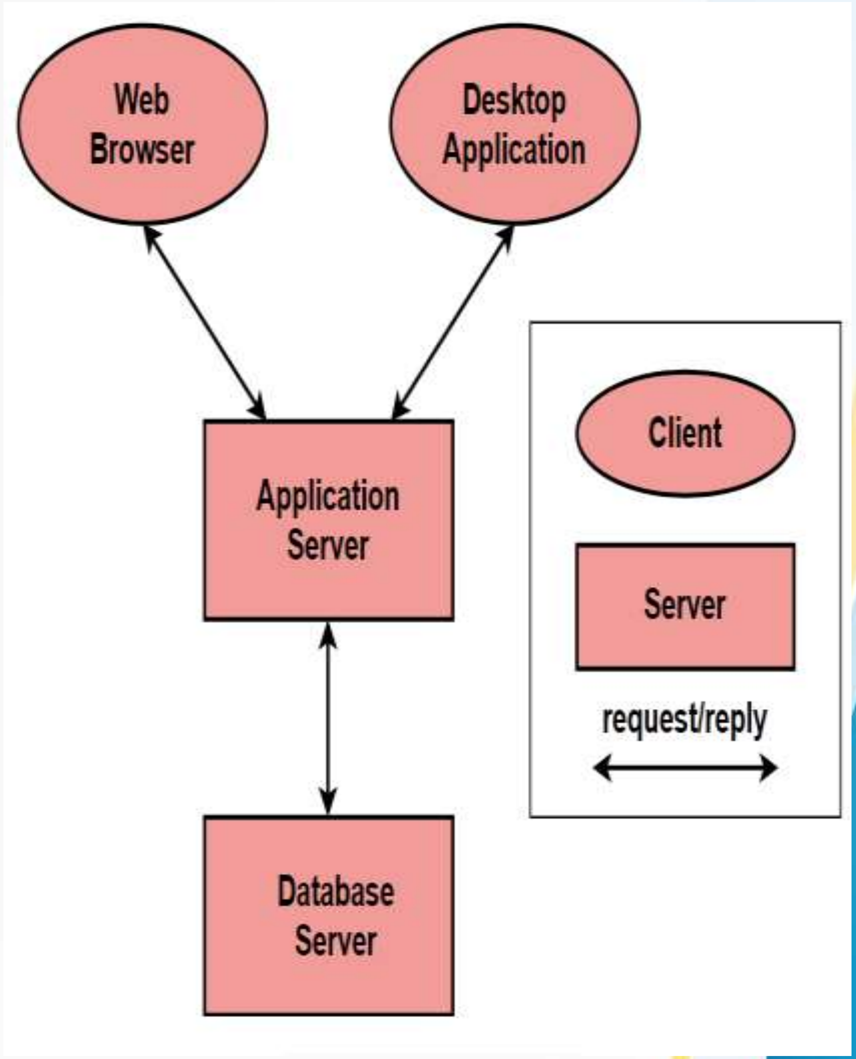
Disadvantages

- If interfaces are not well-defined, it may lead to tight data coupling.
- If internal data structures change, it may result in ripples to other modules.

Client-server architectures

- In client-server architectures, the system is divided into the following two components:-
  - Client
  - Server

- The server component is the service provider while clients are the service requesters.



Source: www.educative.io

contd..

Advantages

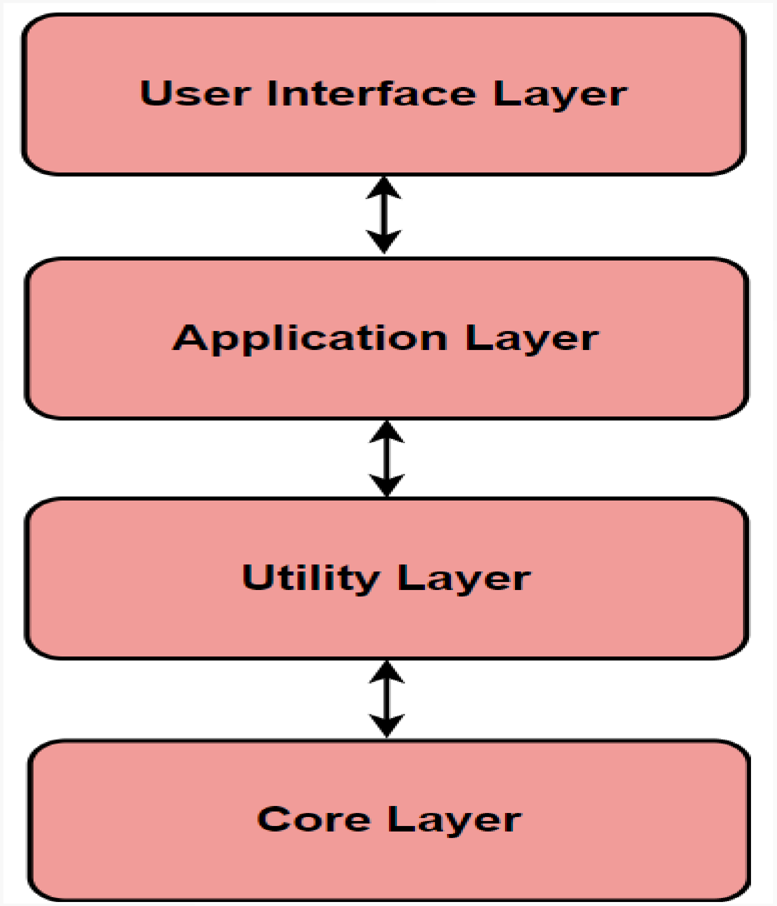
- It provides ease in adding new servers or updating the existing ones.
- Supports parallel implementation.
- This architecture is secure.
- It is easily accessible irrespective of the platform or the location.

Disadvantages

- Management problems if servers are owned by others.
- Performance depends on the system and the network thus, performance is unpredictable.
- Servers may get overloaded with requests.
- Single point of failure, which means if the main server fails, the whole system will be down.

Layered Architecture

- In layered architectures, the system is designed in a stack of layers. Each layer hides the layers below it. The functionality of a system is organized into layers, with each layer only dependent on the layer below it.



Source: www.educative.io

contd..

Advantages

- It supports incremental development.
- It allows the replacement of layers as long as the interface of the layer does not change.
- A series of layers can help partition complex problems.
- It is secure and portable.
- Layers can be reused.

Disadvantages

- Clean separation between layers is hard to achieve.
- Multiple layers of processing may lead to the degradation of performance.
- It is difficult to structure some systems into layers.

Component Design

- The modularised approach to software development exhibited by **component design** places the pieces in a systematic order to help control complexity and improve manageability.

- **Elements**, as the essential components of software architecture, are primarily in charge of enhancing the program's functionality or providing its services.

- **Modularity, reusability, and maintainability** are guaranteed when several features are carried out as autonomous units known as **components**.

- By encouraging **adaptability and scalability**, this method makes it possible to create dynamic systems made up of interlocking, more fundamental components.

Types of Components

- **UI components:** Combining visible and presentational elements like **buttons, forms, and widgets**, user interface components offer a simpler and more practical way to encapsulate logic.

- **Service components:** The foundation of business logic or application services are service components, which act as the means of carrying out tasks including **data processing, authentication**, and **system communication** with external parties.

- **Data components:** Data components handle **database interaction problems** and provide data structures for accessing, updating, and storing data through data abstraction and the provision of interfaces for data access.

contd..

- **Infrastructure components:** The hardware components view as essential resources or services that a software system need, such as **security, caching, logging, and communication protocols.**

- **Integration components: Protocol translation, workflow orchestration, and data interchange** are made possible by integration components, which facilitate data transmission and exchange between various systems or modules.

- **Reusable components:** Reusable components encourage **code reuse and uniformity** by encapsulating common functionality or methods that may be used to various projects and domains.

Principles of Component Design

- **Single Responsibility Principle (SRP):** To guarantee that the system is readily comprehensible, cohesive, and stable, each component should have a thorough and well-defined task.

- The **Open/Closed Principle (OCP)** states that while module extendibility is crucial, developers should be able to customise or expand functionality without having to alter the current codebase in order to maintain stability.

- The **Interface Segregation Principle (ISP)** states that components should have no unnecessary dependencies and should offer consistent interfaces that serve the unique demands of customers.

contd..

- **Dependency Inversion Principle (DIP):** In order to maintain functionality and exchangeability, certain components must be derived interfaces rather than implementation-specific ones.

- **Separation of Concerns (SoC):** To make the functionality obvious, maintainable, and reusable, components should encompass many areas of the task, such as display, business logic, and data access. These blocks also divide the entire operation.

Component Design Patterns

- When building component-based architectures, design architects frequently encounter difficulties that can be resolved by using component design patterns.
- Some popular component design patterns include:
  - **Composite Pattern:** This pattern treats individual objects and combinations of things consistently and enables the composition of objects into tree structures.
  - **Decorator Pattern:** By encapsulating decorators that are added to already-existing features without altering the interface, the decorator pattern offers the capability of dynamic augmentation in the object behaviour.
  - **Adapter Pattern:** By creating a bridge that uses this interface to transform requests from one to another, adapter pattern addresses interface incompatibility problems.

contd..

- **Factory Pattern:** Factory pattern understands the object preparation process and allows users to generate things without having to specify their concrete classes. This promotes flexibility and decoupling.
- The **Observer pattern** is a one-to-many dependence that forms a unidirectional link between objects. As one object's state changes, other objects are automatically notified and updated.
- The **Facade pattern** encapsulates the fine features of a complicated system and presents a single interface to clients, thereby simplifying the system's interface.
- **Singleton Pattern:** Control centralised access to shared objects and data with the Singleton pattern, which guarantees that a class has a single instance and provides global access to that object.

Component Lifecycle Management

- A methodical technique to controlling the lifecycle of software components from creation to decommissioning is called component lifecycle management.
  - **Creation:** Components are created staically or dynamically.
  - **Initialization:** Set up, configured, and prepared up for use.
  - **Utilization:** utilized within the software system, fulfilling their designated functionalities and interacting with other components as necessary.
  - **Maintenance:** Components undergo maintenance activities such as updates, bug fixes, and optimizations to ensure continued functionality and performance.
  - **Decommissioning:** Components are decommissioned, or disposed, when they are no longer needed

Challenges of Component Design

- **Granularity:** Designers must select the right amount of granularity, as too finely grained components can become cluttered and raise overhead, while too coarsely grained components are unnecessarily rigid and have limited reusability.
- **Interoperability:** It can be challenging to keep components from different suppliers or technologies working together seamlessly.
- **Dependency management:** Maintaining dependencies between components, including versions, compatibility, and conflicts, may be quite challenging, particularly in larger systems with many dependencies.
- **Performance Overhead:** Due to the additional levels of abstraction, communication, and runtime dependency that component-based designs entail, as well as the need for careful optimisation and profiling, runtime performance concerns may arise.

Real world Examples of Component Design

- **E-commerce Platforms:** Component-based architecture is used by e-commerce platforms like Amazon, eBay, and Shopify to meet the various needs of their products.

- **Content Management Systems (CMS):** With component-based design, content management systems like as WordPress, Drupal, and Joomla provide modular functionality through plugins, themes, and extensions that let users create, organise, and publish content, among other things.

- **Enterprise Resource Planning (ERP) Systems:** To connect corporate processes and functionalities from a range of sectors, such as finance, human resources, supply chain management, and customer relationship management, the ERP systems SAP, Oracle, and Microsoft Dynamics have components.

Web App Design

- The process of developing a website application that fulfils users' basic demands and provides a smooth user experience (UX) with an appealing user interface (UI) is known as **web app design.**
  - **Example:** Consider Airbnb. Airbnb recognised that users desired a flexible method of looking for vacation rentals based on many parameters such as cost and location.
    - They developed an interactive web application with a basic user interface that has clickable map functionality and advanced filters.
- **Websites** and **web applications** share many similarities, such as being accessible through browsers and emphasising good user interface (UI) and user experience (UX) design to draw in visitors and give them a seamless navigation experience across various platforms.
- **Web apps** and **webpages** don't require users to install any software on their device, in contrast to native mobile apps.

Web App Design vs Website Design

- The **main purpose of websites** is to present users with information that motivates them to take specific activities. On the other hand, **web apps** are dynamic online programs that enable the user to carry out actions and modify elements in a highly interactive manner.
- Many aspects of web app design overlap with website design. But web app design requires a much sharper focus on responsive site elements, performance across multiple devices, and a streamlined user journey.
- Web app design teams must take into account additional factors because web apps are more complex to develop and design than websites.
  - For example, they must decide if the app will provide offline functionality, which would allow users to finish certain tasks without a reliable internet connection. Google Docs, for instance, allows users to edit offline.

Why Web App Design Process???

- Effective web app design processes assist you in:
  - Address the most important user issues.
  - Remain current in a changing market.
  - Provide exceptional usability and intuitive navigational structures to enable people to accomplish their goals.
  - Ensure user engagement with a visually appealing and user-friendly interface.
  - Boost user happiness and conversions
  - Organise your entire team around your design backlog.
  - Obtain support for your ideas from relevant parties.

Web App Design Process

- Here are five stages to help you organise the design of your web app.
  - **Discover key user and market needs:** Start by learning about your target market and users. Important web app design questions to ask yourself include:
    - Who will use this web app?
    - What objectives do consumers have?
    - What primary organisational objectives does this online application serve?
    - How can our online application differentiate itself from rivals?

contd..

- **Define potential solutions:**
  - Come up with solutions for the main issues that you have identified with users.
  - To generate ideas for web app features and components that will meet your users' most pressing demands, employ brainstorming strategies. You'll probably think of a few different possible answers.
  - Utilise prioritising strategies such as the **'MoSCoW method,'** which involves classifying ideas into categories such as must-have, should-have, could-have, and won't-need.
  - During the remainder of the design phase, this will assist you in determining where to allocate your resources.

contd..

- **Create a backlog for your web app design and collaborate**
  - Once you've defined the solutions your web app will offer, and have a clear sense of your priorities, you can plan out the web app design process by creating a backlog: a list of ideas for new design features, updates to existing features, bug fixes, and infrastructure optimizations.
  - Include a plan for the next stages of the product design process, including development, testing, and further user discovery.
  - The best backlogs are structured but also flexible and collaborative.
  - Getting input from different perspectives will strengthen your web app solutions and plan to realize them.

contd..

- **Build and iterate:**
  - Create solutions that you can implement and test on actual consumers. Start by modelling the fundamental web abb navigation architecture and UX features with mockups, wireframes, and prototypes.
  - After that, test them on actual or potential consumers, and make necessary revisions based on their input.
- **Launch and test:**
  - You have the chance to test and improve your design even more when you launch your web app or new features.
  - If you want to make modifications and correct bugs before releasing a feature to the public, you might want to think about doing a soft launch, which involves rolling out to a small number of users first.

Web App Design Patterns

- **Grid:** A **grid** pattern solves users' needs for organized, easily-scannable content. Organizing key content snippets in a grid makes it easy for users to view and navigate content-heavy web apps. Grids also offer more options for dynamic viewing and scrolling than a simple list structure. **For example: Pinterest** presents its web app content in a grid.

- **Cards:** The card web app design pattern lets users interact with content and access controls without encountering cluttered screens. This pattern presents information and options in small 'cards' that can be manipulated by the user. **For example: Twitter** uses cards to categorize information or large image thumbnails into bite-sized pieces to streamline its user interface.

contd..

- **Split screen:** Split-screens solve the problem of providing two or more primary screen focuses. With split-screen design patterns, you showcase two contrasting ideas, giving them equal visual footing on your web app. **For example: Hiristic** shows how a split-screen layout can help users navigate signup and login for different roles.

- **Single page:** Single-page patterns let users complete multiple tasks and actions all on one page without them having to waste time navigating between several different pages. **For example: Gmail** uses the single-page web app design pattern to let users read and compose emails, chat with coworkers, and separate emails into categories without leaving the main page.

contd..

- **Direct Messaging:** Direct messaging design patterns let users send private messages from within your web app alongside their other interactions. Direct messaging can be combined with other web app design patterns to cater to multiple user needs in-app. **For example: Instagram** uses direct messaging to let users chat or direct message as an integral part of their experience.

- **Draggable objects:** Drag and drop design patterns help users sort and organize items in ways that make sense to them. This pattern lets users pick up and rearrange content, or simply drag it to perform an action. **For example: Google Drive** lets users drag and drop images into the drive, helping them complete their tasks faster.