Software Engineering

Module 4

Software Design

By,

Ms.Swati Mahalle

Software Design

- Software Design is the process of transforming user requirements into a suitable form, which helps the programmer in software coding and implementation.
- During the software design phase, the design document is produced, based on the customer requirements as documented in the SRS document. Hence, this phase aims to transform the SRS document into a design document.

The following items are designed and documented during the design phase:

- Different modules are required.
- Control relationships among modules.
- Interface among different modules.
- Data structure among the different modules.
- Algorithms are required to be implemented among the individual modules.

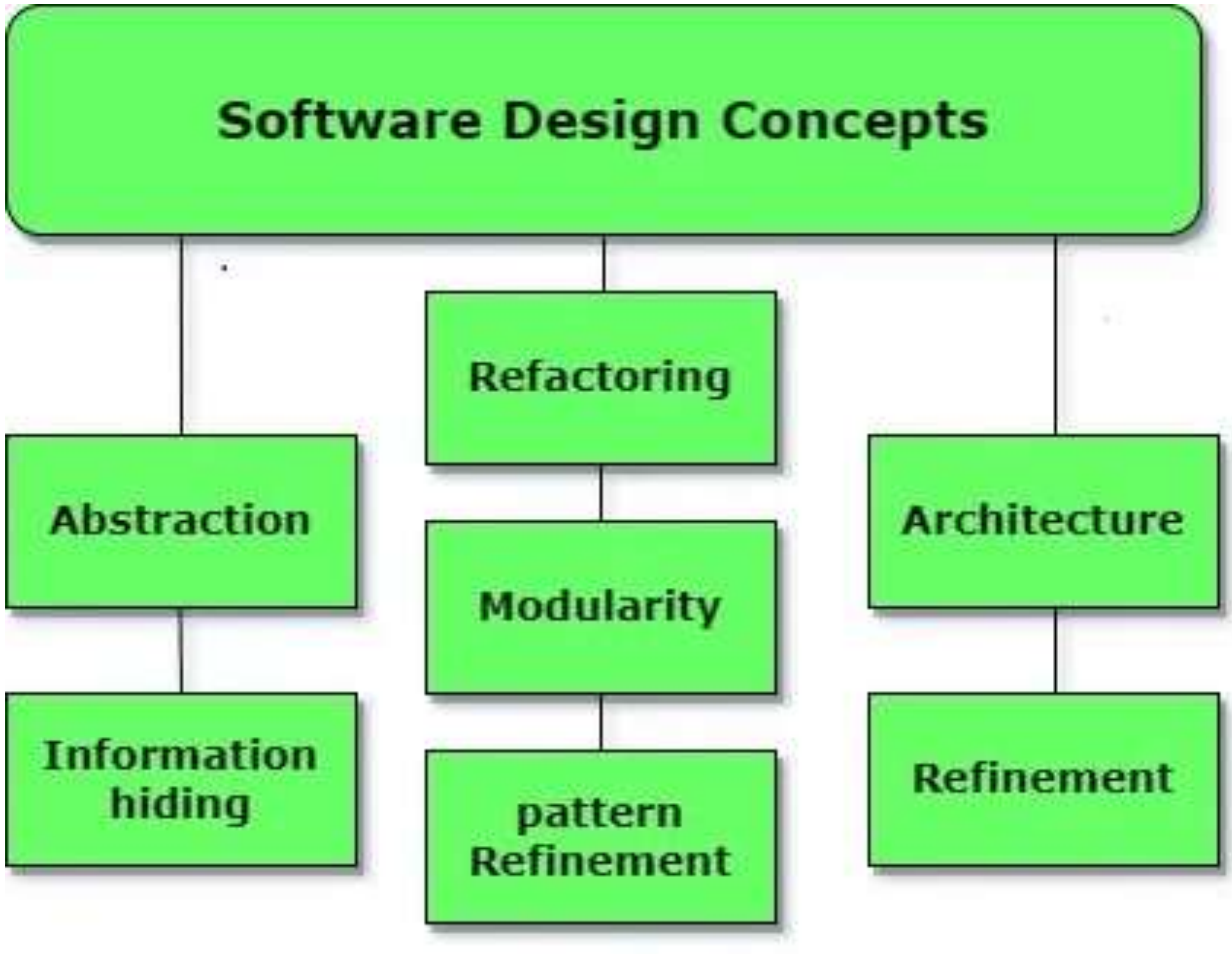
Characteristics of Good Design

1. **Correctness:** A good design should be correct i.e., it should correctly implement all the functionalities of the system.
2. **Efficiency:** A good software design should address the resources, time, and cost optimization issues.
3. **Flexibility:** A good software design should have the ability to adapt and accommodate changes easily. It includes designing the software in a way, that allows for modifications, enhancements, and scalability without requiring significant rework or causing major disruptions to the existing functionality.
4. **Understandability:** A good design should be easily understandable, it should be modular, and all the modules are arranged in layers.
5. **Completeness:** The design should have all the components like data structures, modules, external interfaces, etc.
6. **Maintainability:** A good software design aims to create a system that is easy to understand, modify, and maintain over time. This involves using modular and well-structured design principles e.g.,(employing appropriate naming conventions and providing clear documentation). Maintainability in Software and design also enables developers to fix bugs, enhance features, and adapt the software to changing requirements without excessive effort or introducing new issues.

Software Design Concepts

- Concepts are defined as a principal idea or invention that comes into our mind or in thought to understand something.
- The **software design concept** simply means the idea or principle behind the design.
- It describes how you plan to solve the problem of designing software, and the logic, or thinking behind how you will design software.
- It allows the software engineer to create the model of the system software or product that is to be developed or built.
- The software design concept provides a supporting and essential structure or model for developing the right software.

There are many concepts of software design and some of them are given below:



Points to be Considered While Designing Software

1. **Abstraction** (**Hide Irrelevant data**)**:** Abstraction simply means to hide the details to reduce complexity and increase efficiency or quality. Different levels of Abstraction are necessary and must be applied at each stage of the design process so that any error that is present can be removed to increase the efficiency of the software solution and to refine the software solution. The solution should be described in broad ways that cover a wide range of different things at a higher level of abstraction and a more detailed description of a solution of software should be given at the lower level of abstraction.

2. **Modularity (subdivide the system):** Modularity simply means dividing the system or project into smaller parts to reduce the complexity of the system or project. In the same way, modularity in design means subdividing a system into smaller parts so that these parts can be created independently and then use these parts in different systems to perform different functions. It is necessary to divide the software into components known as modules because nowadays, there are different software available like Monolithic software that is hard to grasp for software engineers. So, modularity in design has now become a trend and is also important. If the system contains fewer components then it would mean the system is complex which requires a lot of effort (cost) but if we can divide the system into components then the cost would be small.

3. **Architecture (design a structure of something):** Architecture simply means a technique to design a structure of something. Architecture in designing software is a concept that focuses on various elements and the data of the structure. These components interact with each other and use the data of the structure in architecture.

Points to be Considered While Designing Software

4.Refinement (removes impurities): Refinement simply means to refine something to remove any impurities if present and increase the quality. The refinement concept of software design is a process of developing or presenting the software or system in a detailed manner which means elaborating a system or software. Refinement is very necessary to find out any error if present and then to reduce it.

5.Pattern (a Repeated form): A pattern simply means a repeated form or design in which the same shape is repeated several times to form a pattern. The pattern in the design process means the repetition of a solution to a common recurring problem within a certain context.

6.Information Hiding (Hide the Information): Information hiding simply means to hide the information so that it cannot be accessed by an unwanted party. In software design, information hiding is achieved by designing the modules in a manner that the information gathered or contained in one module is hidden and can’t be accessed by any other modules.

7.Refactoring (Reconstruct something): Refactoring simply means reconstructing something in such a way that it does not affect the behaviour of any other features. Refactoring in software design means reconstructing the design to reduce complexity and simplify it without impacting the behaviour or its functions. Fowler has defined refactoring as “the process of changing a software system in a way that it won’t impact the behavior of the design and improves the internal structure”.

Effective Modular Design in Software Engineering

The role of effective modular design in software engineering:

- Any software comprises of many systems which contains several sub-systems and those sub-systems further contains their sub-systems.
- So, designing a complete system in one go comprising of each and every required functionality is a hectic work and the process can have many errors because of its vast size.
- Thus in order to solve this problem the developing team breakdown the complete software into various modules.
- A module is defined as the unique and addressable components of the software which can be solved and modified independently without disturbing (or affecting in very small amount) other modules of the software.
- Thus every software design should follow modularity. The process of breaking down a software into multiple independent modules where each module is developed separately is called **Modularization**.

Effective Modular Design in Software Engineering

- Effective modular design can be achieved if the partitioned modules are separately solvable, modifiable as well as compilable.
- Here separate compilable modules means that after making changes in a module there is no need of recompiling the whole software system.
- In order to build a software with effective modular design there is a factor **"Functional Independence"** which comes into play.
- The meaning of Functional Independence is that a function is atomic in nature so that it performs only a single task of the software without or with least interaction with other modules.
- Functional Independence is considered as a sign of growth in modularity i.e., presence of larger functional independence results in a software system of good design and design further affects the quality of the software.

Benefits of Independent modules/functions in a software design:

- Since the functionality of the software have been broken down into atomic levels, thus developers get a clear requirement of each and every functions and hence designing of the software becomes easy and error free.
- As the modules are independent they have limited or almost no dependency on other modules.
- So, making changes in a module without affecting the whole system is possible in this approach.
- Error propagation from one module to another and further in whole system can be neglected and it saves time during testing and debugging.
- Independence of modules of a software system can be measured using 2 criteria : Cohesion, and Coupling.

These are explained as following below.

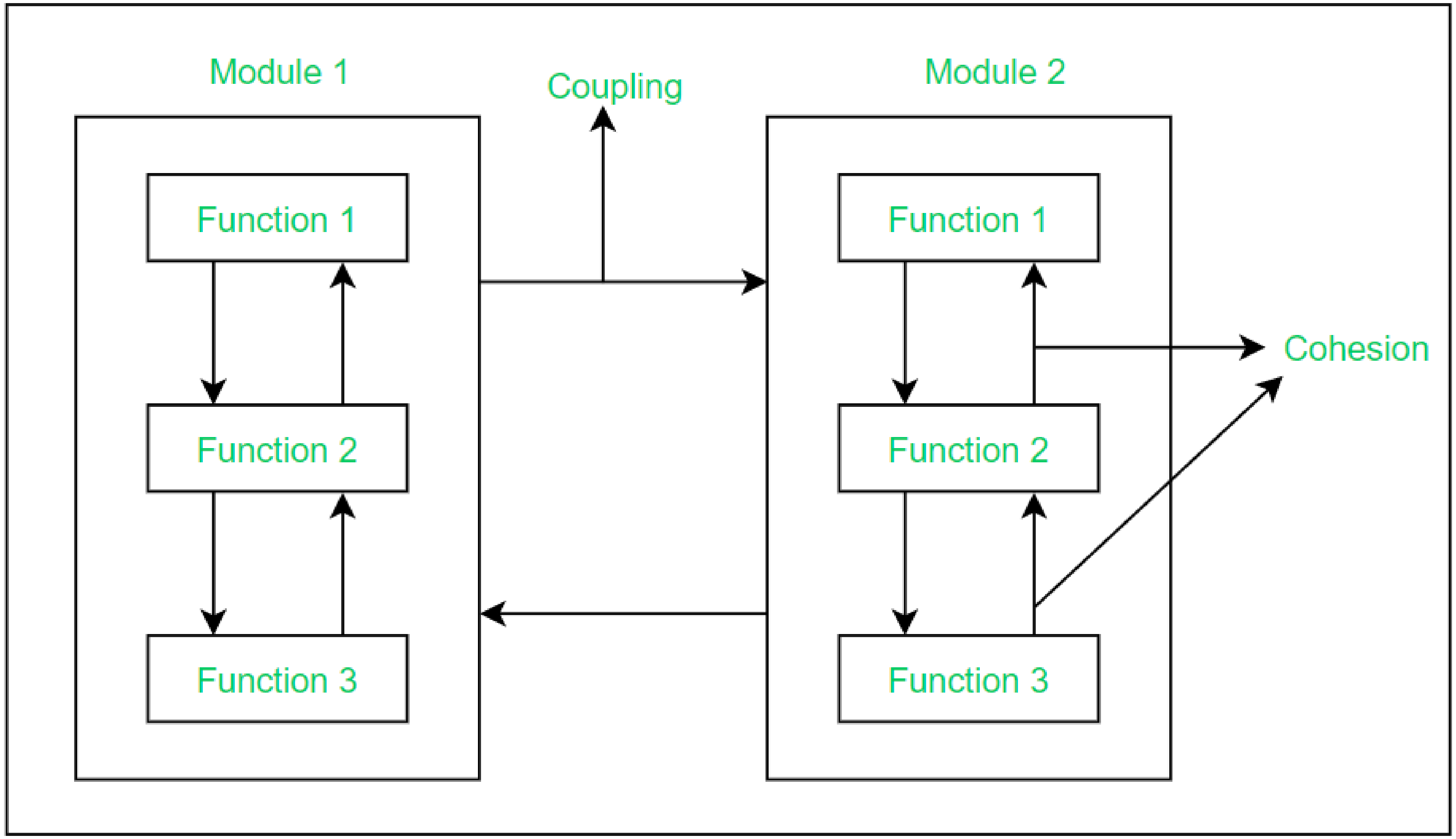


Figure – Cohesion and Coupling between 2 modules

Cohesion

- **Cohesion:** Cohesion is a measure of strength in relationship between various functions within a module. It is of 7 types which are listed below in the order of high to low cohesion:
- **1.** Functional cohesion
- **2.** Sequential cohesion
- **3.** Communicational cohesion
- **4.** Procedural cohesion
- **5.** Temporal cohesion
- **6.** Logical cohesion
- **7.** Co-incidental cohesion

Coupling

- **Coupling:** Coupling is a measure of strength in relationship between various modules within a software.
- It is of 6 types which are listed below in the order of low to high coupling:
- **1.** Data Coupling
- **2.** Stamp Coupling
- **3.** Control Coupling
- **4.** External Coupling
- **5.** Common Coupling
- **6.** Content Coupling
- A good software design requires **high cohesion** and **low coupling**.

Architectural Styles

- **Introduction:** The software needs the architectural design to represents the design of software. 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.” The software that is built for computer-based systems can exhibit one of these many architectural styles.

Each style will describe a system category that consists of :

- A set of components(eg: a database, computational modules) that will perform a function required by the system.
- The set of connectors will help in coordination, communication, and cooperation between the components.
- Conditions that how components can be integrated to form the system.
- Semantic models that help the designer to understand the overall properties of the system.

The use of architectural styles is to establish a structure for all the components of the system.

Taxonomy of Architectural styles:

- 1] Data centred architectures
- 2] Data flow architectures
- 3] Call and Return architectures
- 4] Object Oriented architecture
- 5] Layered architecture

1] Data centred architectures:

- A data store will reside at the centre of this architecture and is accessed frequently by the other components that update, add, delete or modify the data present within the store.

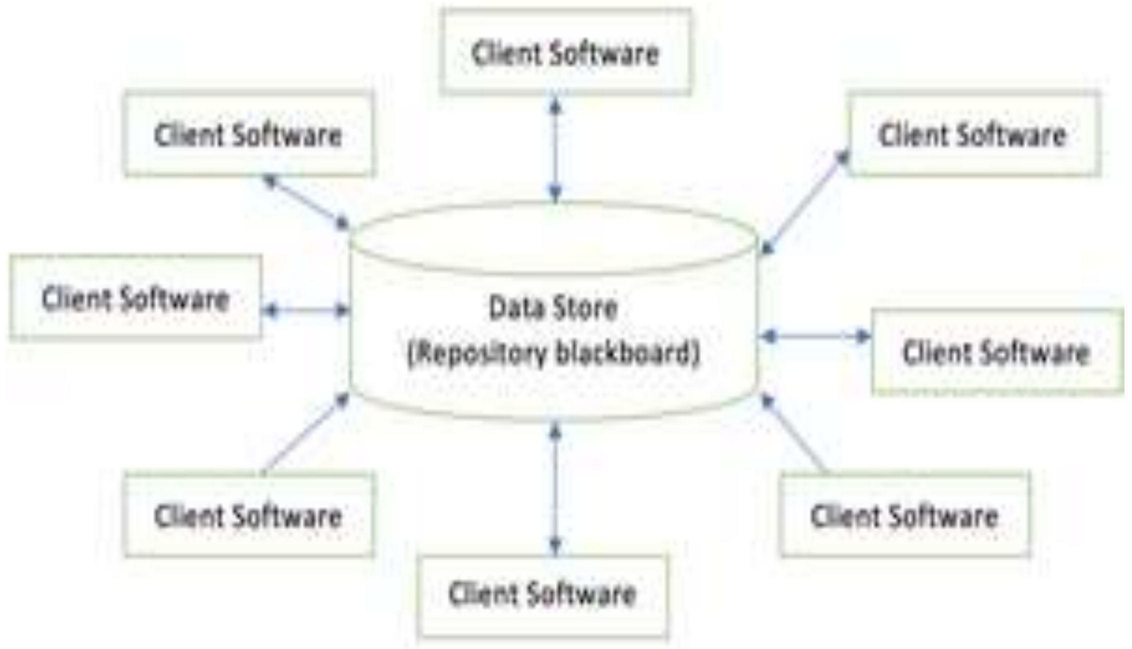
- The figure illustrates a typical data cantered style. The client software access a central repository. Variation of this approach are used to transform the repository into a blackboard when data related to client or data of interest for the client change the notifications to client software.

- This data-centered architecture will promote integrability. This means that the existing components can be changed and new client components can be added to the architecture without the permission or concern of other clients.

- Data can be passed among clients using blackboard mechanism.

- **Advantage of Data centred architecture**

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



Data centred architecture

2] Data flow architectures:

- This kind of architecture is used when input data is transformed into output data through a series of computational manipulative components.
- The figure represents pipe-and-filter architecture since it uses both pipe and filter and it has a set of components called filters connected by lines.
- Pipes are used to transmitting data from one component to the next.
- Each filter will work independently and is designed to take data input of a certain form and produces data output to the next filter of a specified form.
- The filters don’t require any knowledge of the working of neighbouring filters.
- If the data flow degenerates into a single line of transforms, then it is termed as batch sequential.
- This structure accepts the batch of data and then applies a series of sequential components to transform it.

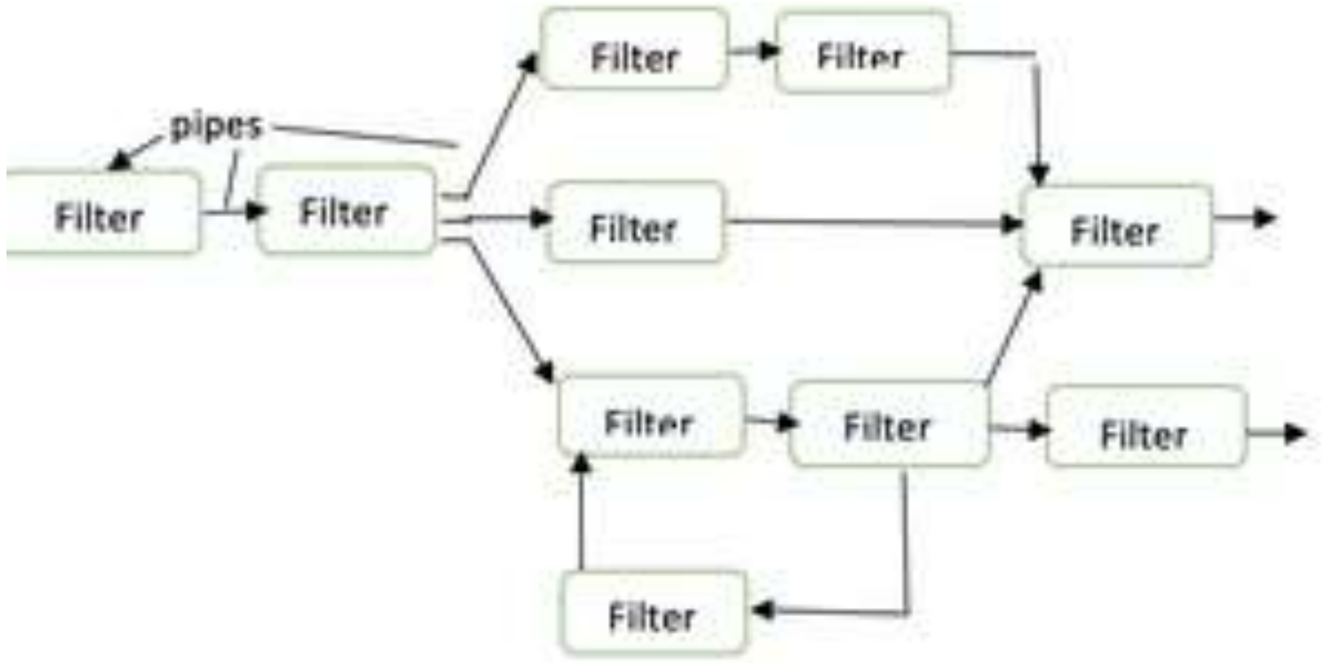
2] Data flow architectures:

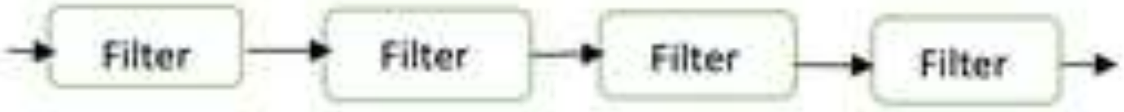
Advantages of Data Flow architecture

- It encourages upkeep, repurposing, and modification.
- With this design, concurrent execution is supported.

The disadvantage of Data Flow architecture

- It frequently degenerates to batch sequential system
- Data flow architecture does not allow applications that require greater user engagement.
- It is not easy to coordinate two different but related streams





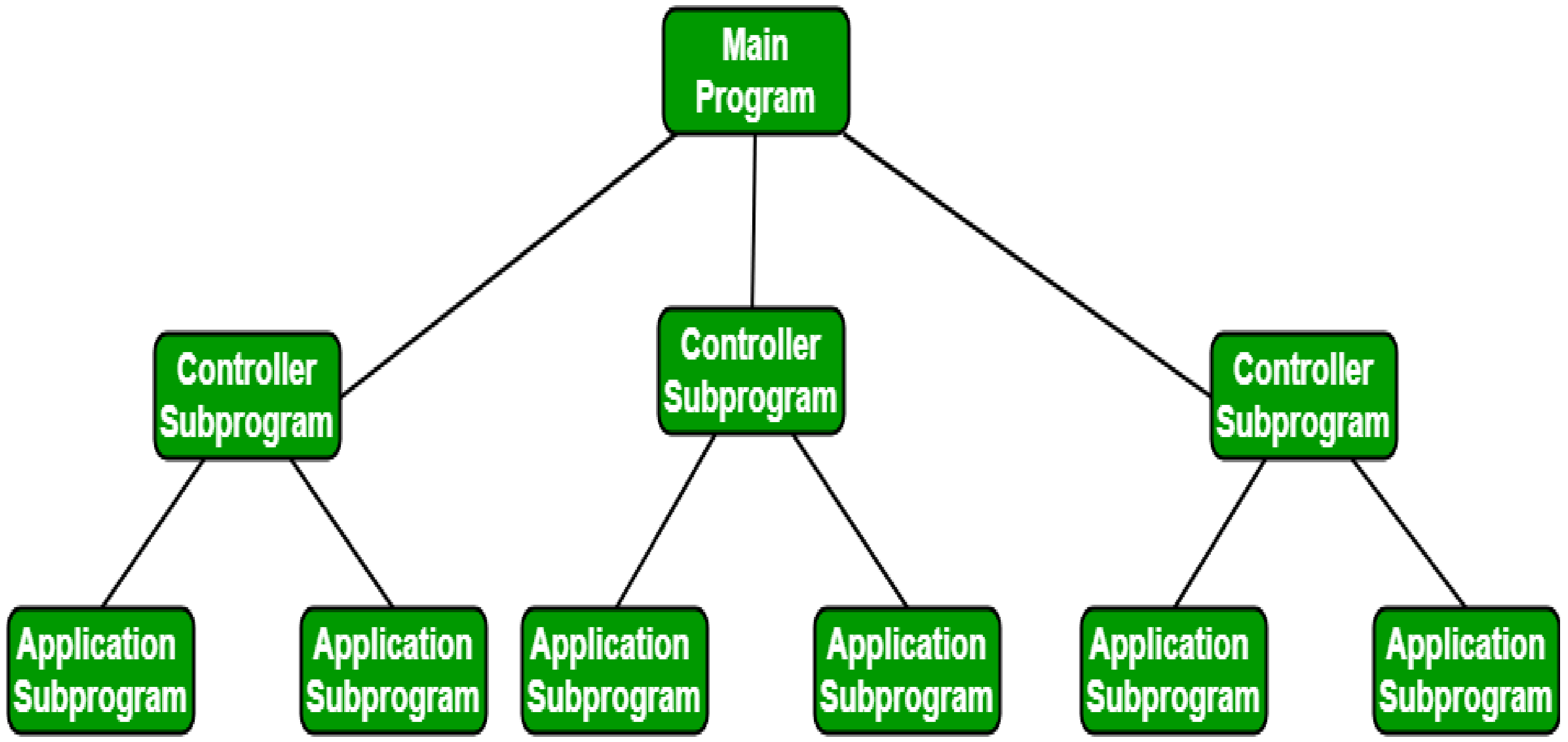
Data Flow architecture

3] Call and Return architectures:

- It is used to create a program that is easy to scale and modify. Many sub-styles exist within this category. Two of them are explained below.

- **Remote procedure call architecture:** This components is used to present in a main program or sub program architecture distributed among multiple computers on a network.

- **Main program or Subprogram architectures:** The main program structure decomposes into number of subprograms or function into a control hierarchy. Main program contains number of subprograms that can invoke other components.



Call and Return architectures

4] Object Oriented architecture:

- The components of a system encapsulate data and the operations that must be applied to manipulate the data.
- The coordination and communication between the components are established via the message passing.

Characteristics of Object Oriented architecture

- Object protect the system’s integrity.
- An object is unaware of the depiction of other items.

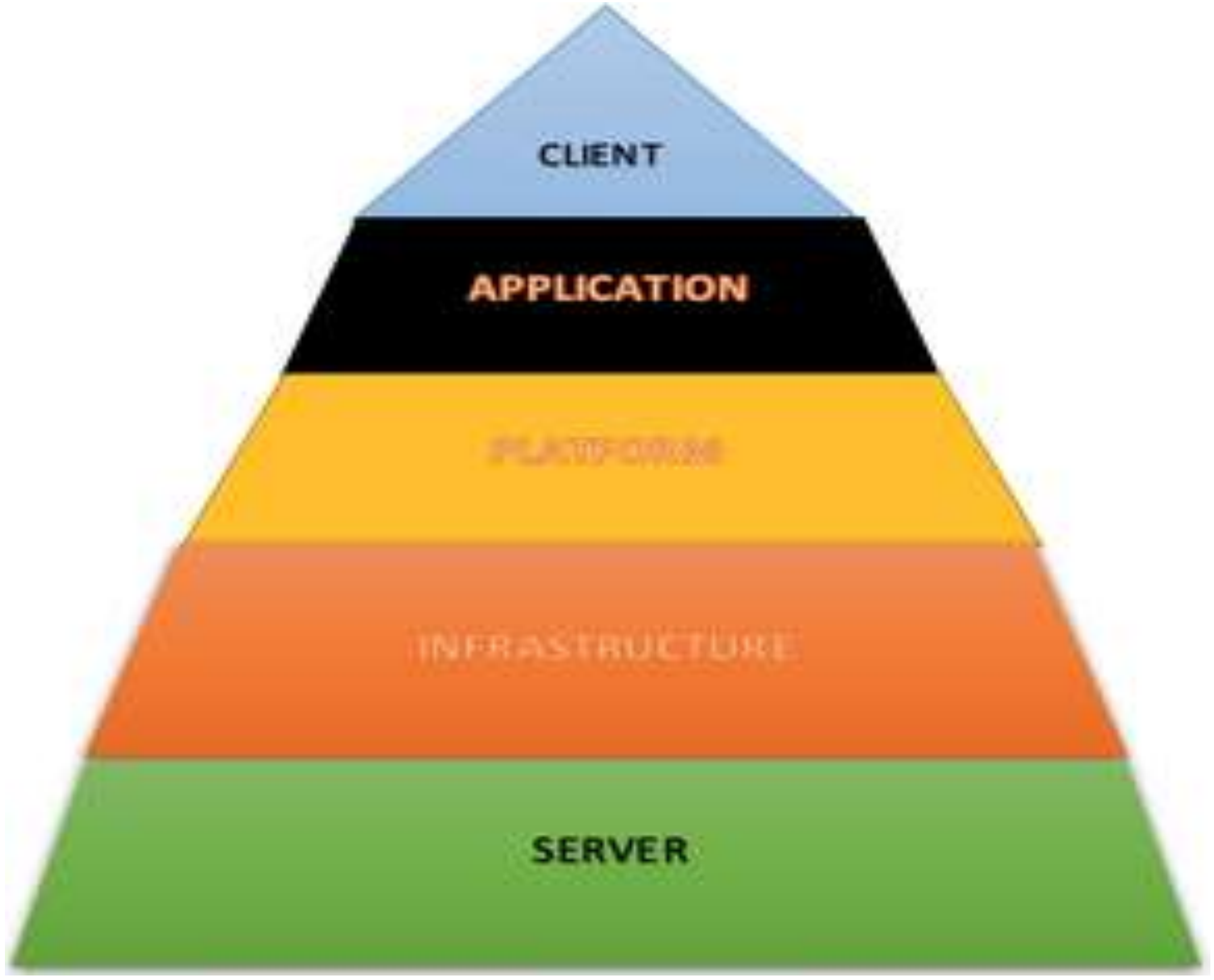
4] Object Oriented architecture:

Advantage of Object Oriented architecture

- It enables the designer to separate a challenge into a collection of autonomous objects.
- Other objects are aware of the implementation details of the object, allowing changes to be made without having an impact on other objects.

5] Layered architecture:

- A number of different layers are defined with each layer performing a well-defined set of operations. Each layer will do some operations that becomes closer to machine instruction set progressively.
- At the outer layer, components will receive the user interface operations and at the inner layers, components will perform the operating system interfacing(communication and coordination with OS)
- Intermediate layers to utility services and application software functions.
- One common example of this architectural style is OSI-ISO (Open Systems Interconnection-International Organisation for Standardisation) communication system.



Layered architecture

UI Design

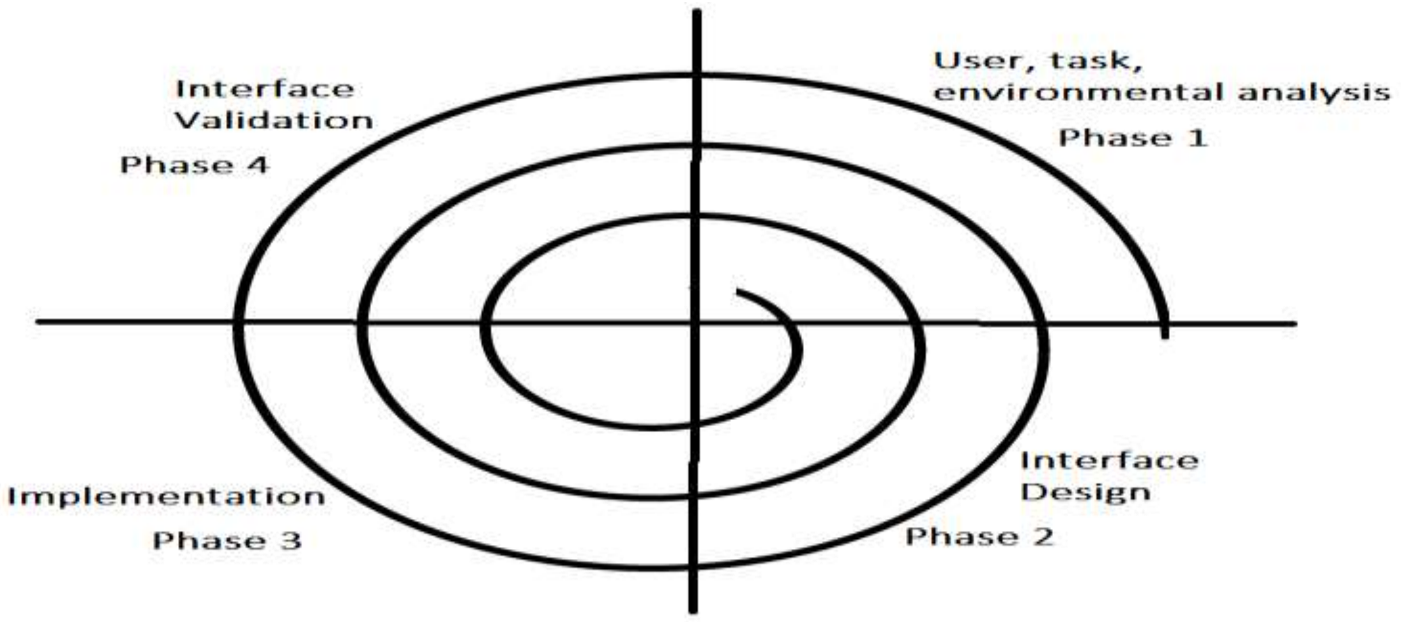
- The user interface is the front-end application view to which the user interacts to use the software. The software becomes more popular if its user interface is:
- **Attractive**
- **Simple to use**
- **Responsive in a short time**
- **Clear to understand**
- **Consistent on all interface screens**

Types of User Interface

- **Command Line Interface:** The Command Line Interface provides a command prompt, where the user types the command and feeds it to the system. The user needs to remember the syntax of the command and its use.

- **Graphical User Interface:** Graphical User Interface provides a simple interactive interface to interact with the system. GUI can be a combination of both hardware and software. Using GUI, the user interprets the software.

User Interface Design Process



User Interface Design Process

The analysis and design process of a user interface is iterative and can be represented by a spiral model. The analysis and design process of user interface consists of four framework activities.

1. User, Task, Environmental Analysis, and Modelling

- Initially, the focus is based on the profile of users who will interact with the system, i.e., understanding, skill and knowledge, type of user, etc., based on the user’s profile users are made into categories.
- From each category requirements are gathered.
- Based on the requirement’s developer understand how to develop the interface.
- Once all the requirements are gathered a detailed analysis is conducted. In the analysis part, the tasks that the user performs to establish the goals of the system are identified, described and elaborated.
- The analysis of the user environment focuses on the physical work environment. Among the questions to be asked are:
- Where will the interface be located physically?
- Will the user be sitting, standing, or performing other tasks unrelated to the interface?
- Does the interface hardware accommodate space, light, or noise constraints?
- Are there special human factors considerations driven by environmental factors?

2. Interface Design

- The goal of this phase is to define the set of interface objects and actions i.e., control mechanisms that enable the user to perform desired tasks.
- Indicate how these control mechanisms affect the system.
- Specify the action sequence of tasks and subtasks, also called a user scenario. Indicate the state of the system when the user performs a particular task.
- Always follow the three golden rules stated by Theo Mandel. Design issues such as response time, command and action structure, error handling, and help facilities are considered as the design model is refined. This phase serves as the foundation for the implementation phase.

3. Interface Construction and Implementation

- The implementation activity begins with the creation of a prototype (model) that enables usage scenarios to be evaluated.
- As iterative design process continues a User Interface toolkit that allows the creation of windows, menus, device interaction, error messages, commands, and many other elements of an interactive environment can be used for completing the construction of an interface.

4. Interface Validation

- This phase focuses on testing the interface.
- The interface should be in such a way that it should be able to perform tasks correctly, and it should be able to handle a variety of tasks.
- It should achieve all the user’s requirements. It should be easy to use and easy to learn.
- Users should accept the interface as a useful one in their work.

Key Principles for Designing User Interfaces

1. **User-centered design:** User interface design should be focused on the needs and preferences of the user. This involves understanding the user’s goals, tasks, and context of use, and designing interfaces that meet their needs and expectations.
2. **Consistency:** Consistency is important in user interface design, as it helps users to understand and learn how to use an application. Consistent design elements such as icons, color schemes, and navigation menus should be used throughout the application.
3. **Simplicity:** User interfaces should be designed to be simple and easy to use, with clear and concise language and intuitive navigation. Users should be able to accomplish their tasks without being overwhelmed by unnecessary complexity.
4. **Feedback:** Feedback is significant in user interface design, as it helps users to understand the results of their actions and confirms that they are making progress towards their goals. Feedback can take the form of visual cues, messages, or sounds.
5. **Accessibility:** User interfaces should be designed to be accessible to all users, regardless of their abilities. This involves considering factors such as color contrast, font size, and assistive technologies such as screen readers.
6. **Flexibility:** User interfaces should be designed to be flexible and customizable, allowing users to tailor the interface to their own preferences and needs.

GUI Characteristics

Characteristics	Descriptions
Windows	Multiple windows allow different information to be displayed simultaneously on the user's screen.
Icons	Icons different types of information. On some systems, icons represent files. On other icons describes processes.
Menus	Commands are selected from a menu rather than typed in a command language.
Pointing	A pointing device such as a mouse is used for selecting choices from a menu or indicating items of interests in a window.
Graphics	Graphics elements can be mixed with text or the same display.

Thank You!