Chapter 4 : Software Design

Designing Concepts :

- The main aim of design engineering is to generate a model which shows firmness, delight and commodity.
- Software design is an iterative process through which requirements are translated into the blueprint for building the software.

Software quality :

- A design is generated using the recognizable architectural styles and compose a good design characteristic of components and it is implemented in evolutionary manner for testing.
- A design of the software must be modular i.e the software must be logically partitioned into elements.

- In design, the representation of data , architecture, interface and components should be distinct.
- A design must carry appropriate data structure and recognizable data patterns.
- Design components must show the independent functional characteristic.
- A design creates an interface that reduce the complexity of connections between the components.
- A design must be derived using the repeatable method.
- The notations should be use in design which can effectively communicates its meaning.

Design concepts :

- **The set of fundamental software design concepts are as follows:**

1. Abstraction : A solution is stated in large terms using the language of the problem environment at the highest level abstraction.

- The lower level of abstraction provides a more detail description of the solution.
- A sequence of instruction that contain a specific and limited function refers in a procedural abstraction.
- A collection of data that describes a data object is a data abstraction.
- **2. Architecture :** The complete structure of the software is known as software architecture.
- Structure provides conceptual integrity for a system in a number of ways.
- The architecture is the structure of program modules where they interact with each other in a specialized way.
- The components use the structure of data.
- The aim of the software design is to obtain an architectural framework of a system.
- The more detailed design activities are conducted from the framework.
-

- **3. Patterns**
  A design pattern describes a design structure and that structure solves a particular design problem in a specified content.

4. Modularity : A software is separately divided into name and addressable components. Sometime they are called as modules which integrate to satisfy the problem requirements.

- Modularity is the single attribute of a software that permits a program to be managed easily.

- **5. Information hiding**
  Modules must be specified and designed so that the information like algorithm and data presented in a module is not accessible for other modules not requiring that information.

6. Functional independence : The functional independence is the concept of separation and related to the concept of modularity, abstraction and information hiding.

- The functional independence is accessed using two criteria i.e Cohesion and coupling.

- **Cohesion :** Cohesion is an extension of the information hiding concept.

- A cohesive module performs a single task and it requires a small interaction with the other components in other parts of the program.

- **Coupling :** Coupling is an indication of interconnection between modules in a structure of software.

- **7. Refinement :** Refinement is a top-down design approach.
- It is a process of elaboration.
- A program is established for refining levels of procedural details.

- A hierarchy is established by decomposing a statement of function in a stepwise manner till the programming language statement are reached.
- **8. Refactoring :** It is a reorganization technique which simplifies the design of components without changing its function behaviour.
- Refactoring is the process of changing the software system in a way that it does not change the external behaviour of the code still improves its internal structure.
- **9. Design classes :** The model of software is defined as a set of design classes.
- Every class describes the elements of problem domain and that focus on features of the problem which are user visible.

Characteristics of Good Design

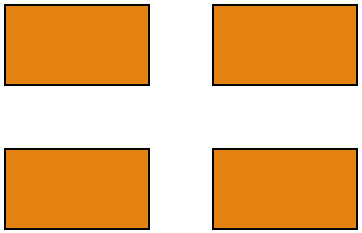
Component independence

- High cohesion
- Low coupling

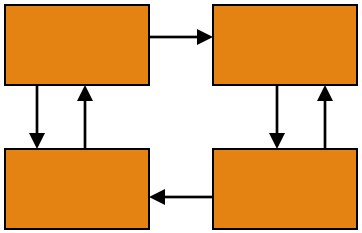
Exception identification and handling

Fault prevention and fault tolerance

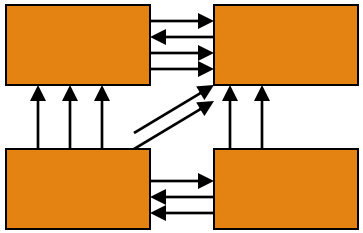
Coupling: Degree of dependence among components



No dependencies



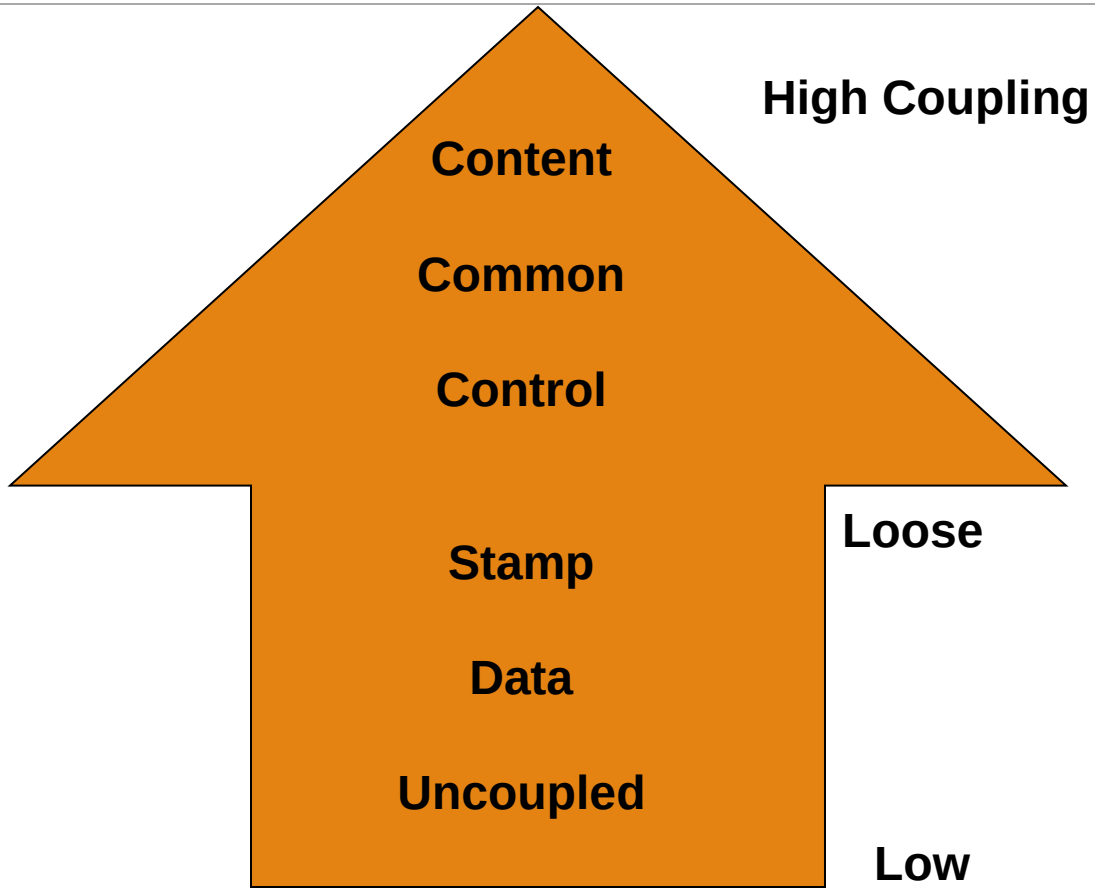
Loosely coupled-some dependencies



Highly coupled-many dependencies

High coupling makes modifying parts of the system difficult, e.g., modifying a component affects all the components to which the component is connected.

Range of Coupling



Content coupling

Definition: One component references contents of another

Example:

occurs when one module makes use of data or control information maintained within the boundary of another module.

Secondarily, content coupling occurs when branches are made into the middle of a module.

This mode of coupling can and should be avoided.

Common Coupling

Definition: Two components share data

- Global data structures
- Common blocks

Usually a poor design choice because

- Lack of clear responsibility for the data
- Reduces readability
- Difficult to determine all the components that affect a data element (reduces maintainability)
- Difficult to reuse components
- Reduces ability to control data accesses

Control Coupling

Definition: Component passes control parameters to coupled components.

where a “control flag” (a variable that controls decisions in a subordinate or superordinate module) is passed between modules

- Not suitable - component must be aware of internal structure and logic of another module
- Good if parameters allow factoring and reuse of functionality

Stamp Coupling

Definition: when a portion of a data

structure (rather than simple arguments) is passed via a module interface.

Requires second component to know how to manipulate the data structure (e.g., needs to know about implementation)

Data Coupling

Definition: Two components are data coupled if same data are passed; a one-to-one correspondence of items exists

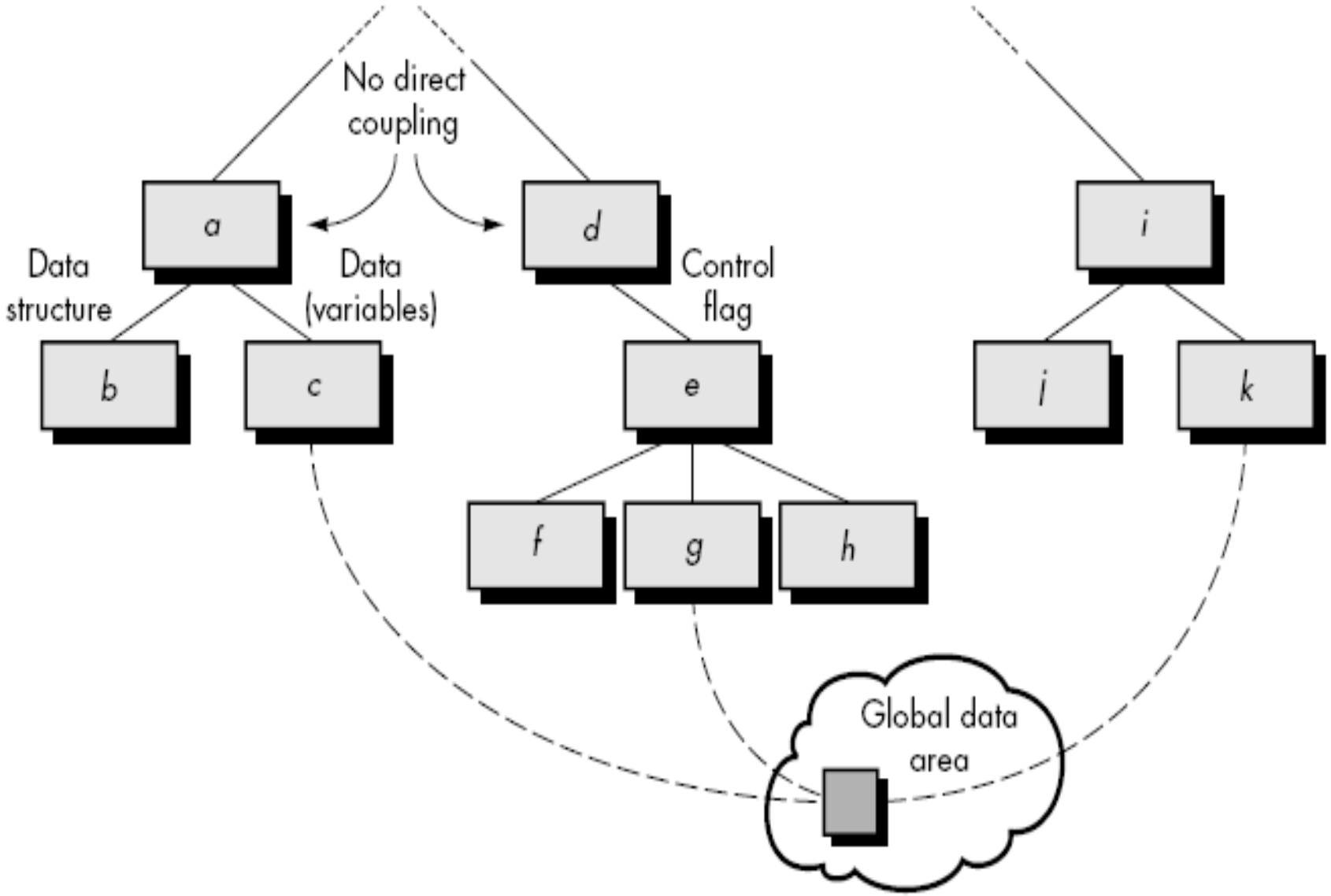
Every argument is simple argument or data structure in which all elements are used

Good, if it can be achieved.

Easy to write contracts for this and modify component independently.

No direct coupling

The modules are not related to each other.


No direct coupling
a
d
i
Data structure
Data (variables)
Control flag
b
c
e
j
k
f
g
h
Global data area

Key Idea in Object-Oriented Programming

Object-oriented designs tend to have low coupling.

Cohesion

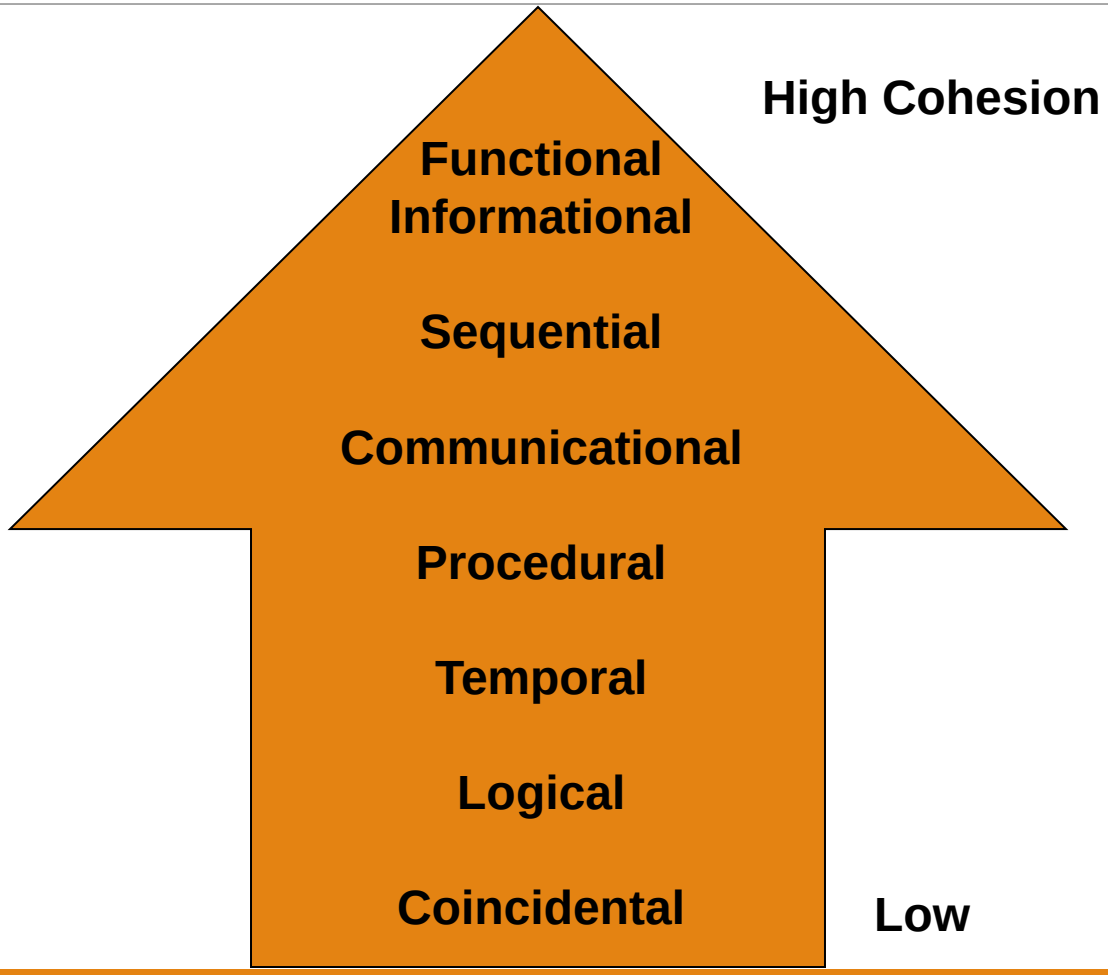
Definition: A cohesive module performs a single task within a software procedure,requiring little interaction with procedures being performed in other parts of a program.

Stated simply, a cohesive module should (ideally) do just one thing.

All elements of component are directed toward and essential for performing the same task

High is good

Range of Cohesion



Coincidental Cohesion

Definition: Parts of the component are only related by their location in source code

Elements needed to achieve some functionality are scattered throughout the system.

Accidental

Worst form

Example

Print next line

Reverse string of characters in second argument

Add 7 to 5th argument

Convert 4th argument to float

Logical Cohesion

Definition: Elements of component are related logically and not functionally.

Several logically related elements are in the same component and one of the elements is selected by the client component.

Example-1

A component reads inputs from tape, disk, and network. All the code for these functions are in the same component.

Operations are related, but the functions are significantly different.

Example-2

A component reads inputs from tape, disk, and network. All the code for these functions are in the same component. Operations are related, but the functions are significantly different.

Improvement

A device component has a read operation that is overridden by sub-class components. The tape sub-class reads from tape. The disk sub-class reads from disk. The network sub-class reads from the network.

Temporal Cohesion

Definition: Elements of a component are related by timing.

Difficult to change because you may have to look at numerous components when a change in a data structure is made.

Component unlikely to be reusable.

Example-1

system initialization routine: this routine contains all of the code for initializing all of the parts of the system. Lots of different activities occur, all at init time.

Example-2

A system initialization routine: this routine contains all of the code for initializing all of the parts of the system. Lots of different activities occur, all at init time.

Improvement

A system initialization routine sends an initialization message to each component.

Each component initializes itself at component instantiation time.

Procedural Cohesion

Definition: Elements of a component are related only to ensure a particular order of execution.

Actions are still weakly connected and unlikely to be reusable

When processing elements of a module are related and must be executed in a specific order

Example

...

Read part number from data base

update repair record on maintenance file.

...

May be useful to abstract the intent of this sequence. Make the data base and repair record components handle reading and updating. Make component that handles more abstract operation.

Communicational Cohesion

Definition: A module is said to have communicational cohesion, if all functions of the module refer to or update the same data structure.

Example

Update record in data base and send it to the printer.

database.Update (record).

record.Print().

Module determines customer details like use customer account no to find and return customer name and loan balance.

Sequential Cohesion

The output of one component is the input to another.

Occurs naturally in functional programming languages

Good situation

In a TPS, the get-input, validate-input, sort-input functions are grouped into one module.

Informational Cohesion

Definition: Module performs a number of actions, each with its own entry point, with independent code for each action, all performed on the same data.

Different from logical cohesion

- Each piece of code has single entry and single exit
- In logical cohesion, actions of module intertwined

ADT and object-oriented paradigm promote

Functional Cohesion

Definition: Every essential element to a single computation is contained in the component.

Every element in the component is essential to the computation.

Ideal situation.

Cohesion

Cohesion: Cohesion can be defined as the degree of the closeness of the relationship between its components. In general, it measures the relationship strength between the pieces of functionality within a given module in the software programming. It is an ordinal type of measurement, which is described as low cohesion or high cohesion.

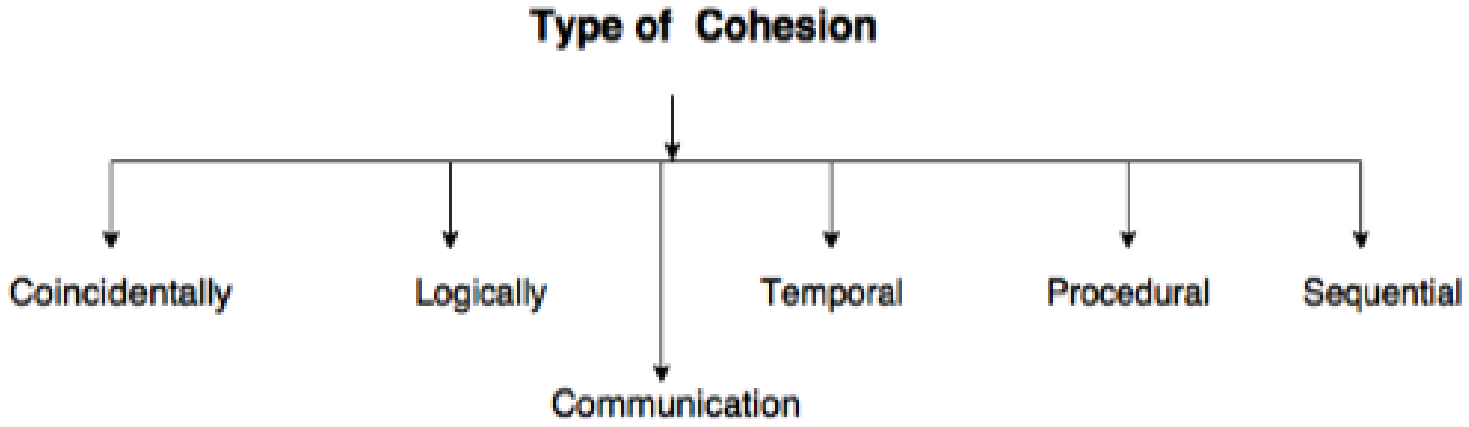
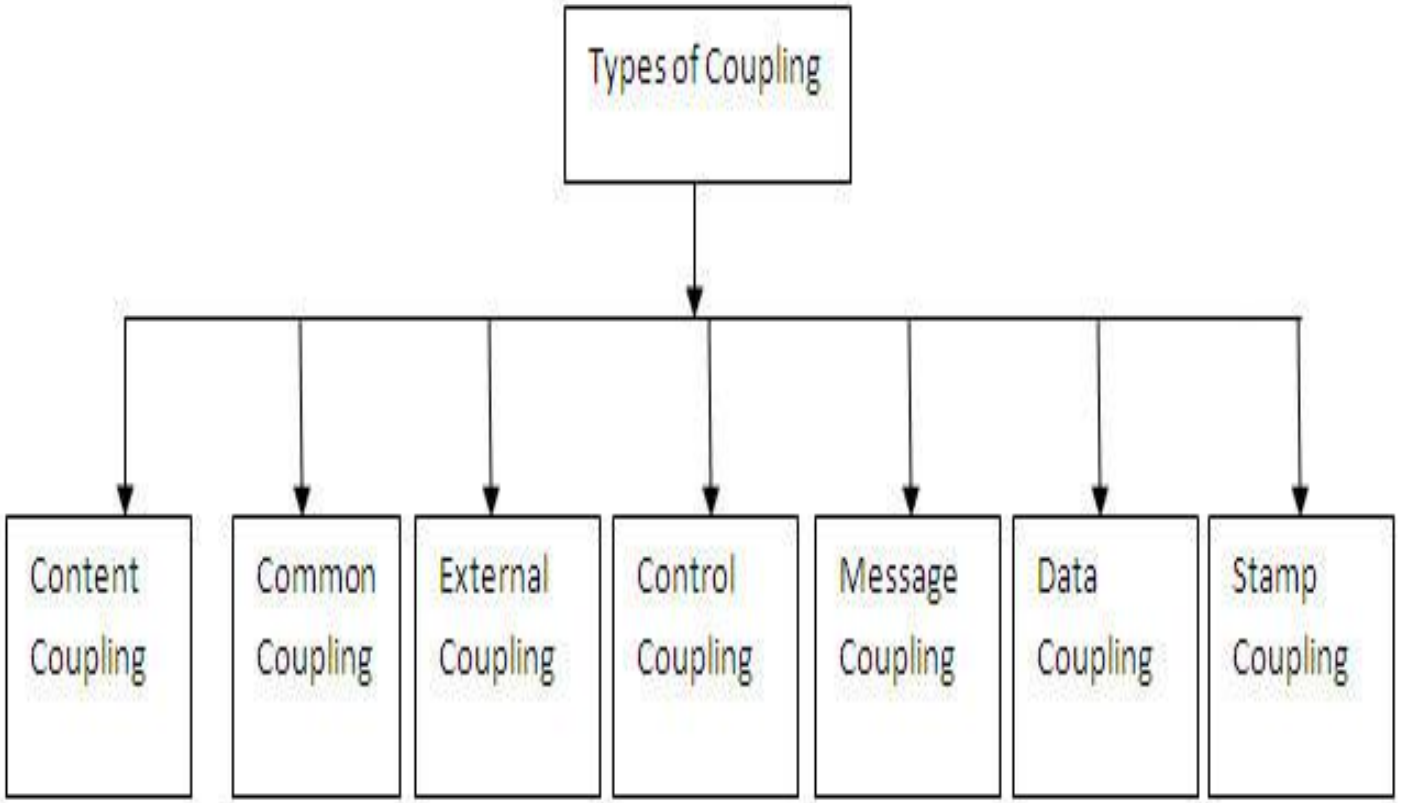


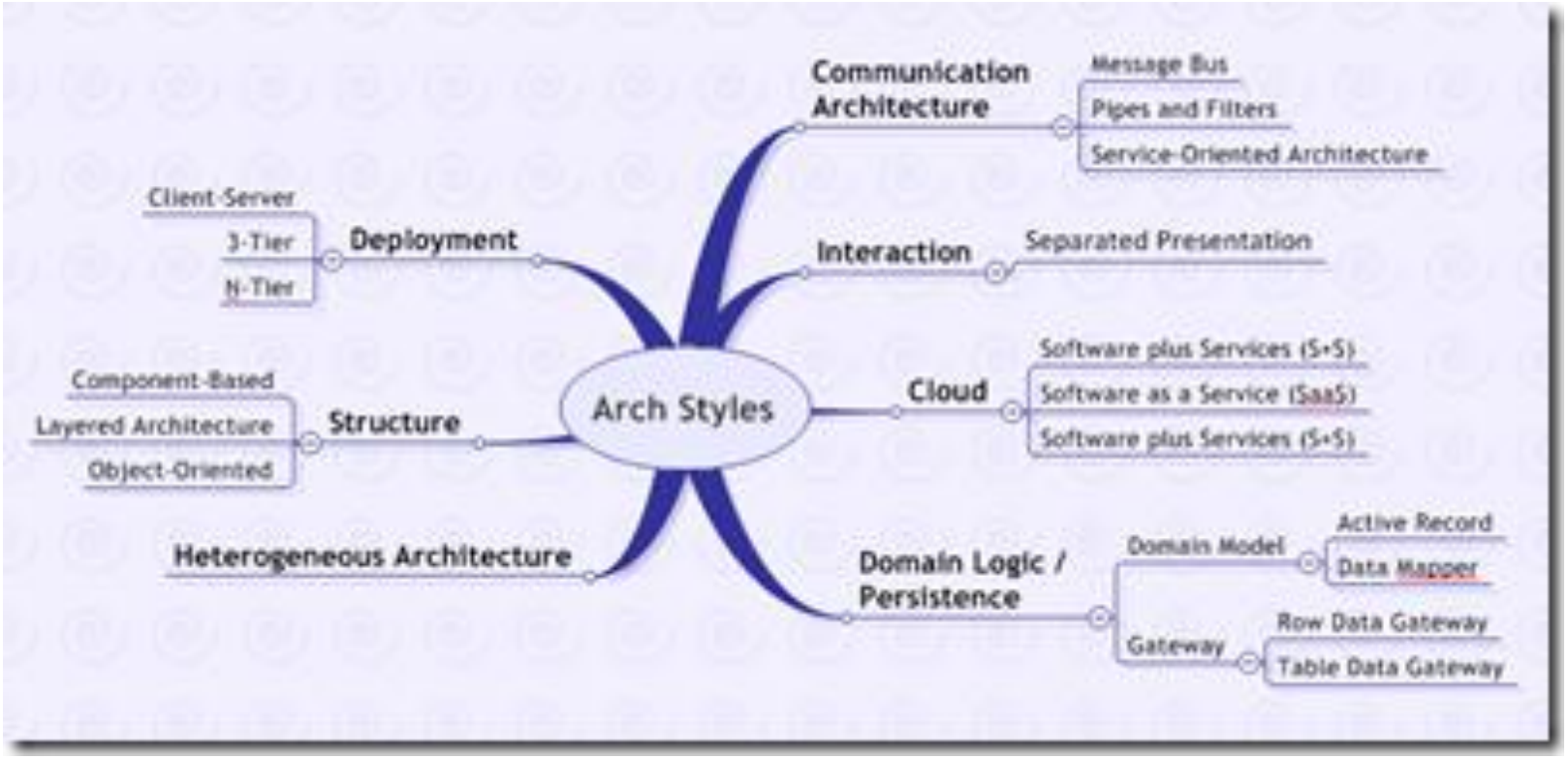
Fig. Different types of Cohesion

Coupling: In software engineering, the coupling can be defined as the measurement to which the components of the software depend upon each other. Normally, the coupling is contrasted with the cohesion. If the system has a low coupling, it is a sign of a well-structured computer system and a great design.

https://www.ques10.com/p/37939/effective-modular-design-cohesion-and-coupling-1/


Types of Coupling
Content Coupling
Common Coupling
External Coupling
Control Coupling
Message Coupling
Data Coupling
Stamp Coupling

Architectural Design :



Architectural Styles:

Each style describes a system category that encompasses

1)A Set of Components (eg database ,computational model) that perform a function required by a system

2) A Set of connectors that enable "communication, cordination & cooperation among Components.

3) Constraints that define how components can be integrated to form the system and

4)Semantic models that enable a designer by analyse the overall properties of a system by analysing the known properties of its constituent parts-

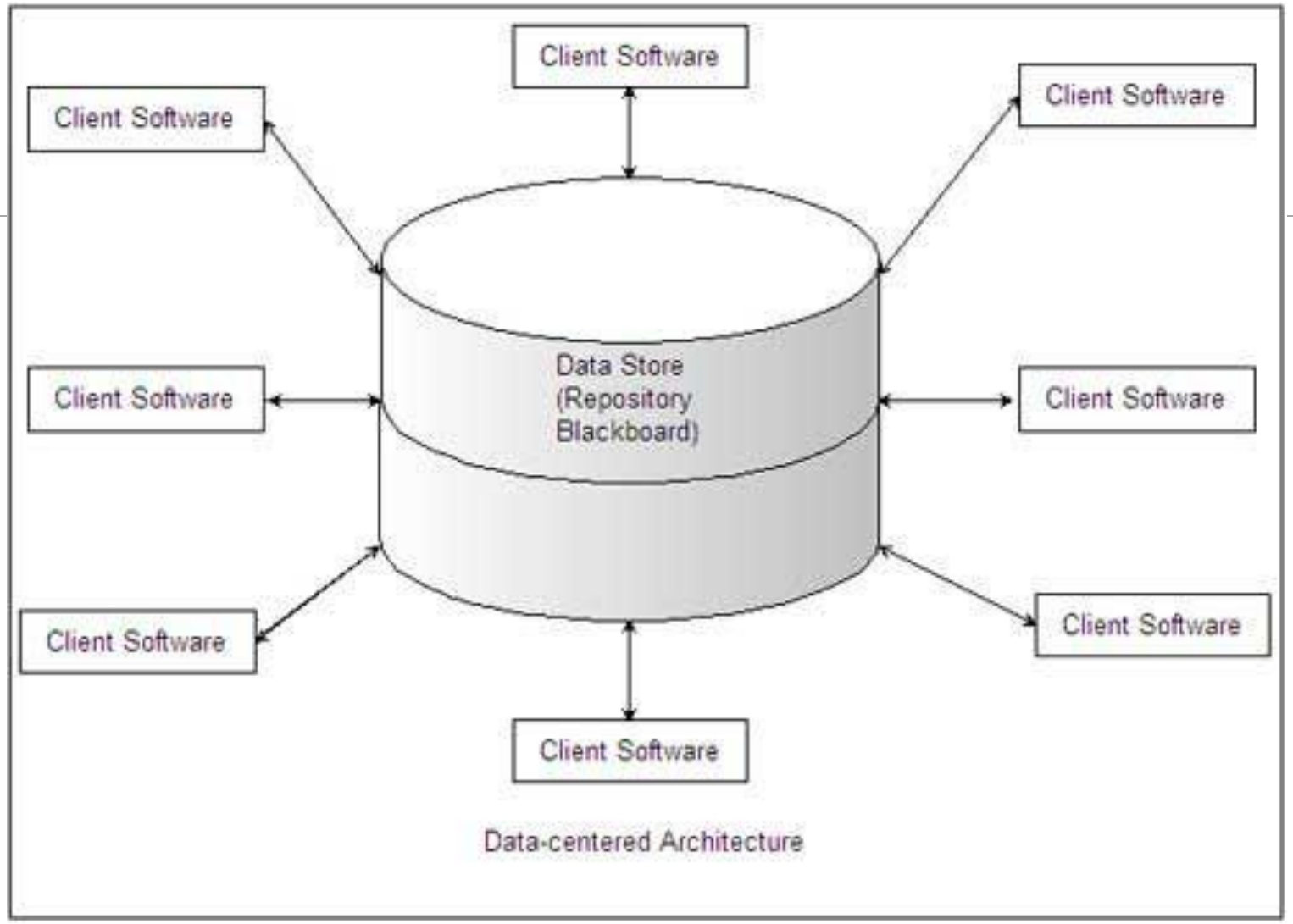
1. Data-centered architecture

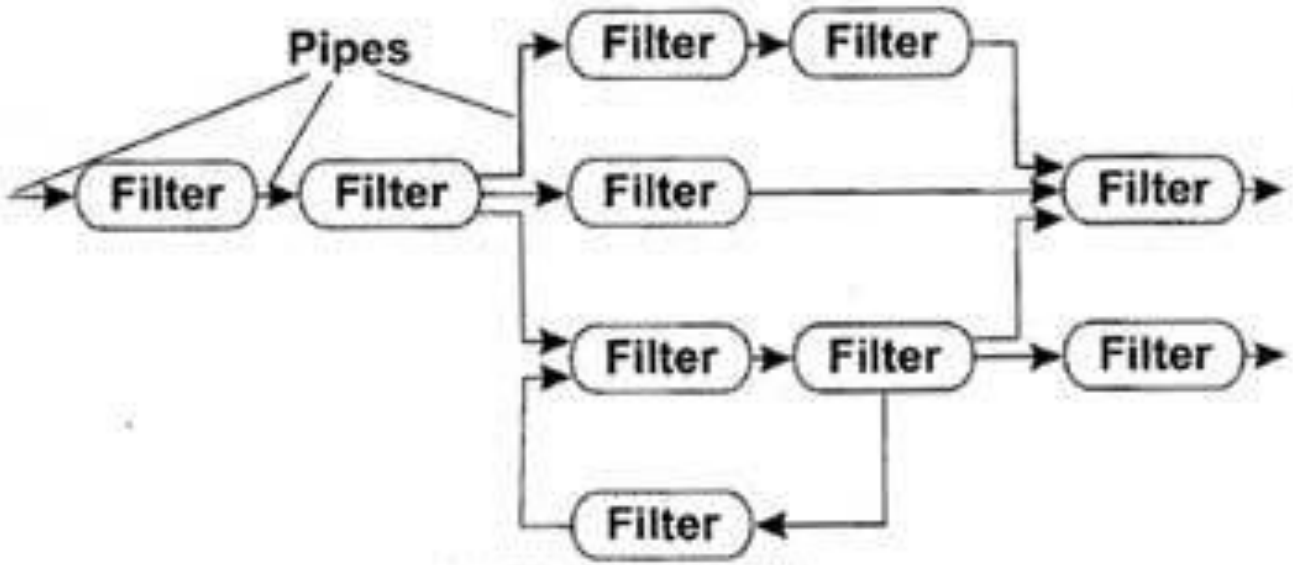
- The data store in the file or database is occupying at the center of the architecture.
- Store data is access continuously by the other components like an update, delete, add, modify from the data store.

- Data-centered architecture helps integrity.
- Pass data between clients using the blackboard mechanism.
- The processes are independently executed by the client components.

2. Data-flow architecture

- This architecture is applied when the input data is converted into a series of manipulative components into output data.
- A pipe and filter pattern is a set of components called as filters.
- Filters are connected through pipes and transfer data from one component to the next component.
- The flow of data degenerates into a single line of transform then it is known as batch sequential.


Client Software
Client Software
Client Software
Client Software
Data Store
(Repository
Blackboard)
Client Software
Client Software
Client Software
Client Software
Data-centered Architecture



Data-flow Architecture

3. Call and return architectures :

- This architecture style allows to achieve a program structure which is easy to modify.

Following are the sub styles exist in this category:

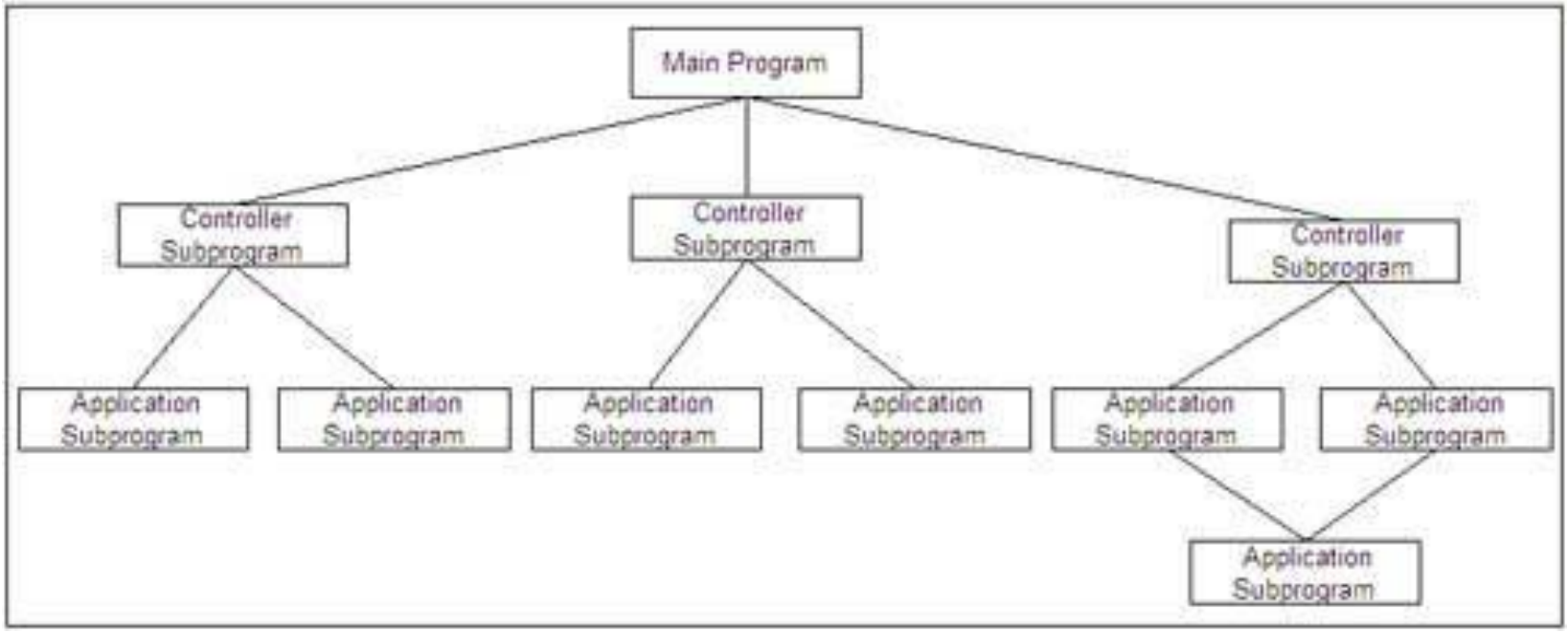
---

1. Main program or subprogram architecture The program is divided into smaller pieces hierarchically.

- The main program invokes many of program components in the hierarchy that program components are divided into subprogram.
- **2. Remote procedure call architecture**The main program or subprogram components are distributed in network of multiple computers.
- The main aim is to increase the performance.

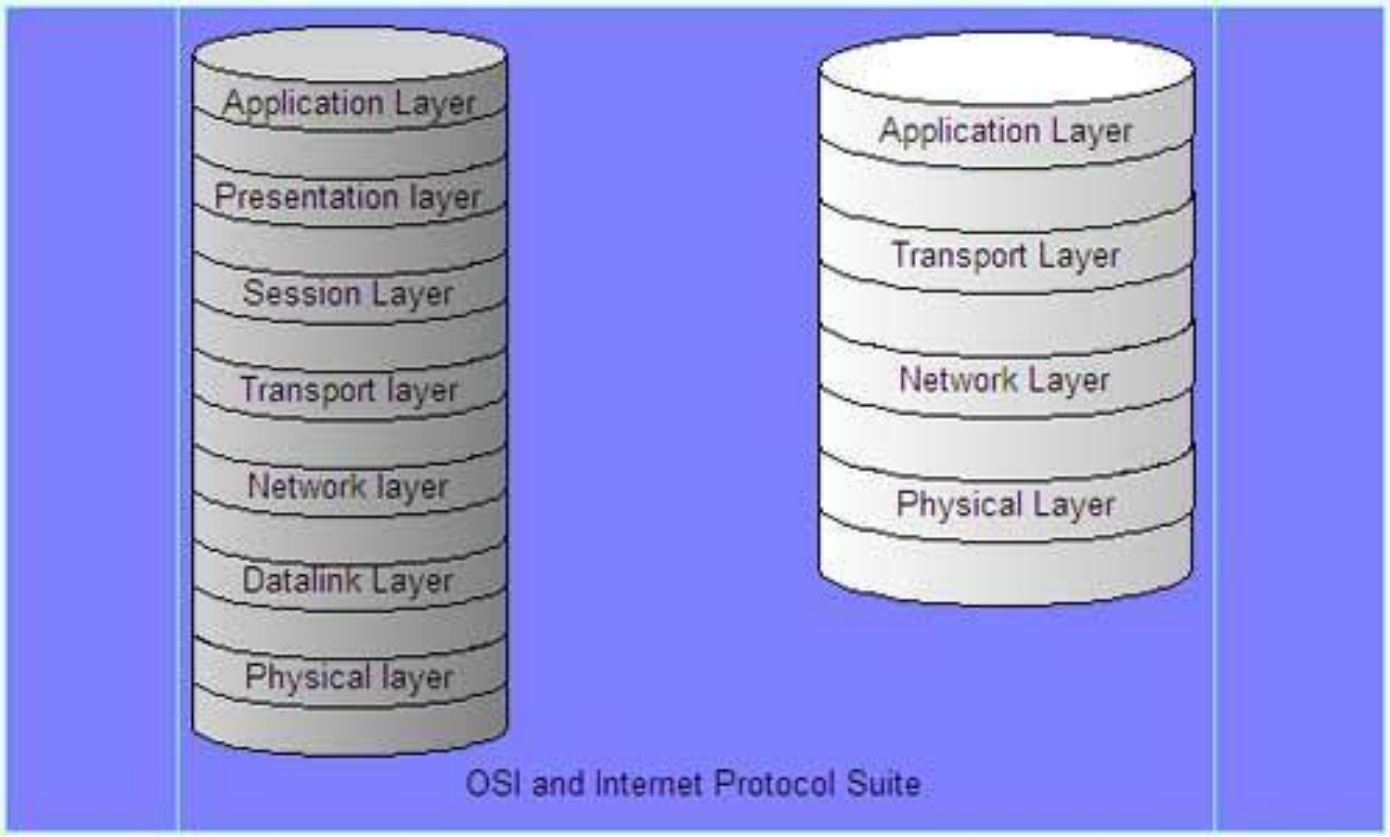
4. Object-oriented architectures :

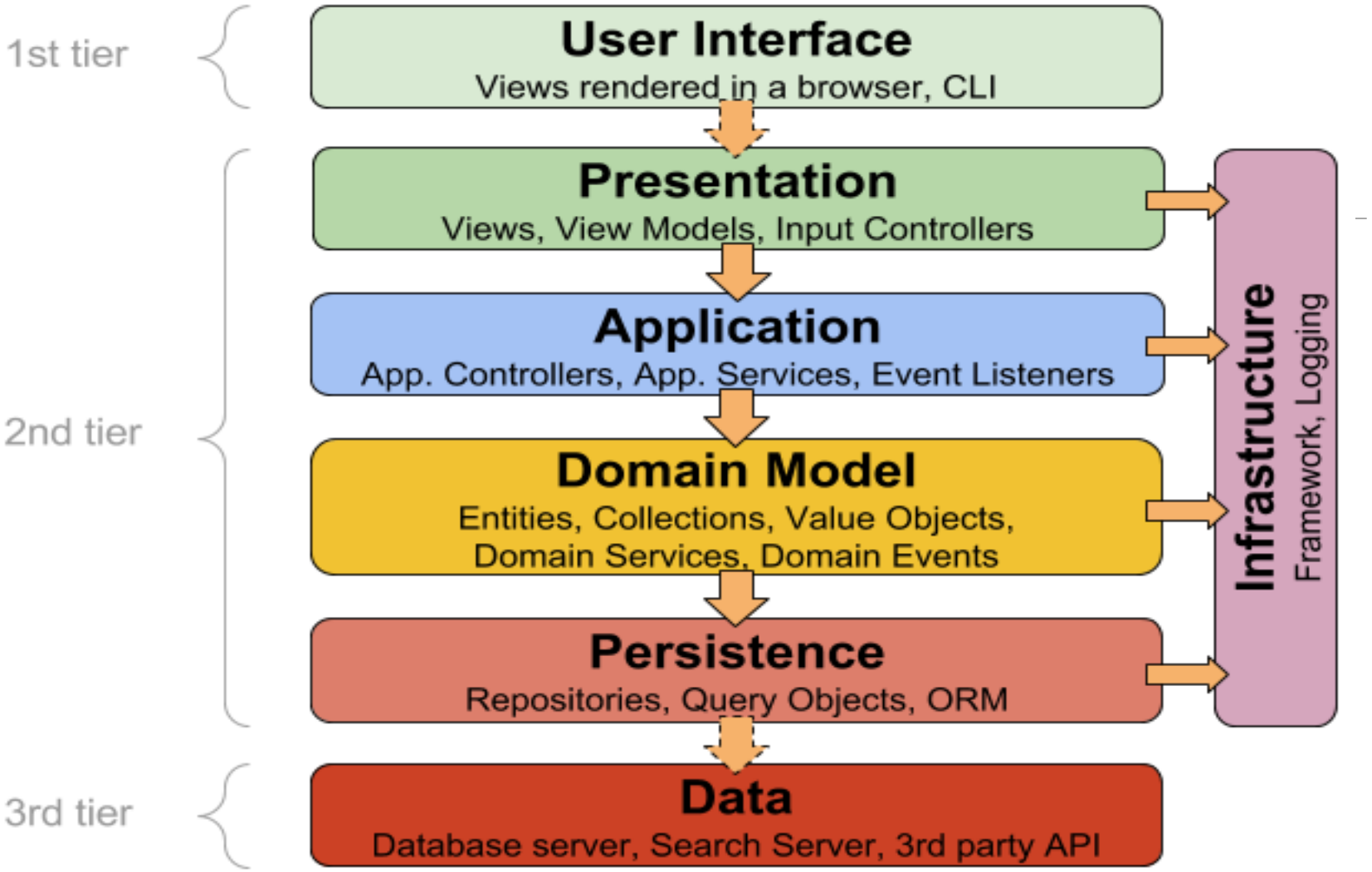
- This architecture is the latest version of call-and-return architecture.
- It consist of the bundling of data and methods.


Main Program
Controller Subprogram
Controller Subprogram
Controller Subprogram
Application Subprogram
Application Subprogram
Application Subprogram
Application Subprogram
Application Subprogram
Application Subprogram
Application Subprogram

5. Layered architectures

- The different layers are defined in the architecture. It consists of outer and inner layer.
- The components of outer layer manage the user interface operations.
- Components execute the operating system interfacing at the inner layer.
- The inner layers are application layer, utility layer and the core layer.
- In many cases, It is possible that more than one pattern is suitable and the alternate architectural style can be designed and evaluated.


Application Layer
Presentation layer
Session Layer
Transport layer
Network layer
Datalink Layer
Physical layer
Application Layer
Transport Layer
Network Layer
Physical Layer
OSI and Internet Protocol Suite


1st tier
User Interface
Views rendered in a browser, CLI
2nd tier
Presentation
Views, View Models, Input Controllers
Application
App. Controllers, App. Services, Event Listeners
Domain Model
Entities, Collections, Value Objects,
Domain Services, Domain Events
Persistence
Repositories, Query Objects, ORM
Infrastructure
Framework, Logging
3rd tier
Data
Database server, Search Server, 3rd party API

UI Design :

User interface (UI) design is the process designers use to build interfaces in software or computerized devices, focusing on looks or style. Designers aim to create interfaces which users find easy to use and pleasurable. UI design refers to graphical user interfaces and other forms.

Types of User Interface

There are two main types of User Interface:

- Text-Based User Interface or Command Line Interface
- Graphical User Interface (GUI)

UI Design :

- 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

1. **Command Line Interface:** Command Line Interface provides a command prompt, where the user types the command and feeds to the system. The user needs to remember the syntax of the command and its use.
2. **Graphical User Interface:** Graphical User Interface provides the simple interactive interface to interact with the system. GUI can be a combination of both hardware and software. Using GUI, user interprets the software.

Designing User Interfaces for Users

User interfaces are the access points where users interact with designs. They come in three formats:

1. **Graphical user interfaces (GUIs)**—Users interact with visual representations on digital control panels. A computer's desktop is a GUI.
2. **Voice-controlled interfaces (VUIs)—**Users interact with these through their voices. Most smart assistants—e.g., Siri on iPhone and Alexa on Amazon devices—are VUIs.
3. **Gesture-based interfaces**—Users engage with 3D design spaces through bodily motions: e.g., in virtual reality (VR) games.

GUI Characteristics

Characteristic	Description
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 others, icons represent 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 interest in a window.
Graphics	Graphical elements can be mixed with text on the same display.

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

Design principles

- 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

User interface design principles

Principle	Description
User familiarity	The interface should use terms and concepts which are drawn from the experience of the people who will make most use of the system.
Consistency	The interface should be consistent in that, wherever possible, comparable operations should be activated in the same way.
Minimal surprise	Users should never be surprised by the behaviour of a system.
Recoverability	The interface should include mechanisms to allow users to recover from errors.
User guidance	The interface should provide meaningful feedback when errors occur and provide context-sensitive user help facilities.
User diversity	The interface should provide appropriate interaction facilities for different types of system user.

User-system interaction

- Two problems must be addressed in interactive systems design
  - How should information from the user be provided to the computer system?
  - How should information from the computer system be presented to the user?
- User interaction and information presentation may be integrated through a coherent framework such as a user interface metaphor

Interaction styles

- Direct manipulation
- Menu selection
- Form fill-in
- Command language
- Natural language

Direct manipulation advantages

- Users feel in control of the computer and are less likely to be intimidated by it
- User learning time is relatively short
- Users get immediate feedback on their actions so mistakes can be quickly detected and corrected

Direct manipulation problems

- The derivation of an appropriate information space model can be very difficult
- Given that users have a large information space, what facilities for navigating around that space should be provided?
- Direct manipulation interfaces can be complex to program and make heavy demands on the computer system

Menu systems

- Users make a selection from a list of possibilities presented to them by the system
- The selection may be made by pointing and clicking with a mouse, using cursor keys or by typing the name of the selection
- May make use of simple-to-use terminals such as touchscreens

Advantages of menu systems

- Users need not remember command names as they are always presented with a list of valid commands
- Typing effort is minimal
- User errors are trapped by the interface
- Context-dependent help can be provided. The user's context is indicated by the current menu selection

Problems with menu systems

- Actions which involve logical conjunction (and) or disjunction (or) are awkward to represent
- Menu systems are best suited to presenting a small number of choices. If there are many choices, some menu structuring facility must be used
- Experienced users find menus slower than command language

Command interfaces

- User types commands to give instructions to the system e.g. UNIX
- May be implemented using cheap terminals.
- Easy to process using compiler techniques
- Commands of arbitrary complexity can be created by command combination
- Concise interfaces requiring minimal typing can be created

Problems with command interfaces

- Users have to learn and remember a command language. Command interfaces are therefore unsuitable for occasional users
- Users make errors in command. An error detection and recovery system is required
- System interaction is through a keyboard so typing ability is required

Command languages

- Often preferred by experienced users because they allow for faster interaction with the system
- Not suitable for casual or inexperienced users
- May be provided as an alternative to menu commands (keyboard shortcuts). In some cases, a command language interface and a menu-based interface are supported at the same time

Natural language interfaces

- The user types a command in a natural language. Generally, the vocabulary is limited and these systems are confined to specific application domains (e.g. timetable enquiries)
- NL processing technology is now good enough to make these interfaces effective for casual users but experienced users find that they require too much typing

Interaction style	Main advantages	Main disadvantages	Application examples
Direct manipulation	Fast and intuitive interaction Easy to learn	May be hard to implement Only suitable where there is a visual metaphor for tasks and objects	Video games CAD systems
Menu selection	Avoids user error Little typing required	Slow for experienced users Can become complex if many menu options	Most general-purpose systems
Form fill-in	Simple data entry Easy to learn	Takes up a lot of screen space	Stock control, Personal loan processing
Command language	Powerful and flexible	Hard to learn Poor error management	Operating systems, Library information retrieval systems
Natural language	Accessible to casual users Easily extended	Requires more typing Natural language understanding systems are unreliable	Timetable systems WWW information retrieval systems