Software Engineering

T.E – B

Unit 4

Faculty Incharge: Ms. Drashti Shrimal

Q.1 Characteristics of a good Software Design.

Software is treated as good software by the means of different factors. A software product is concluded as good software by what it offers and how well it can be used.

Software design is a mechanism to transform user requirements into some suitable form, which helps the programmer in software coding and implementation. It deals with representing the client's requirement, as described in SRS (Software Requirement Specification) document, into a form, i.e., easily implementable using programming language.

1) Correctness

First of all, the design of any software is evaluated for its correctness. The evaluators check the software for every kind of input and action and observe the results that the software will produce according to the proposed design. If the results are correct for every input, the design is accepted and is considered that the software produced according to this design will function correctly.

2) Understandability

The software design should be understandable so that the developers do not find any difficulty to understand it. Good software design should be self- explanatory. This is because there are hundreds and thousands of developers that develop different modules of the software, and it would be very time consuming to explain each design to each developer. So, if the design is easy and self- explanatory, it would be easy for the developers to implement it and build the same software that is represented in the design.

3) Efficiency

The software design must be efficient. The efficiency of the software can be estimated from the design phase itself, because if the design is describing software that is not efficient and useful, then the developed software would also stand on the same level of efficiency. Hence, for efficient and good quality software to be developed, care must be taken in the designing phase itself.

4) Maintainability

The software design must be in such a way that modifications can be easily made in it. This is because every software needs time to time modifications and maintenance. So, the design of the software must also be able to bear such changes. It should not be the case that after making some modifications the other features of the software start misbehaving. Any change made in the software design must not affect the other available features, and if the features are getting affected, then they must be handled properly.

Other characteristics:

1. **Completeness:** The design should have all components like data structures, modules, and external interfaces, etc.
2. **Flexibility:** Able to modify on changing needs.
3. **Consistency:** There should not be any inconsistency in the design.

Q.2 Define Coupling and Cohesion. And Describe types of both.

The purpose of Design phase in the Software Development Life Cycle is to produce a solution to a problem given in the SRS (Software Requirement Specification) document. The output of the design phase is Software Design Document (SDD).

Coupling and Cohesion are two key concepts in software engineering that are used to measure the quality of a software system's design.

Coupling refers to the degree of interdependence between software modules. High coupling means that modules are closely connected and changes in one module may affect other modules. Low coupling means that modules are independent and changes in one module have little impact on other modules.

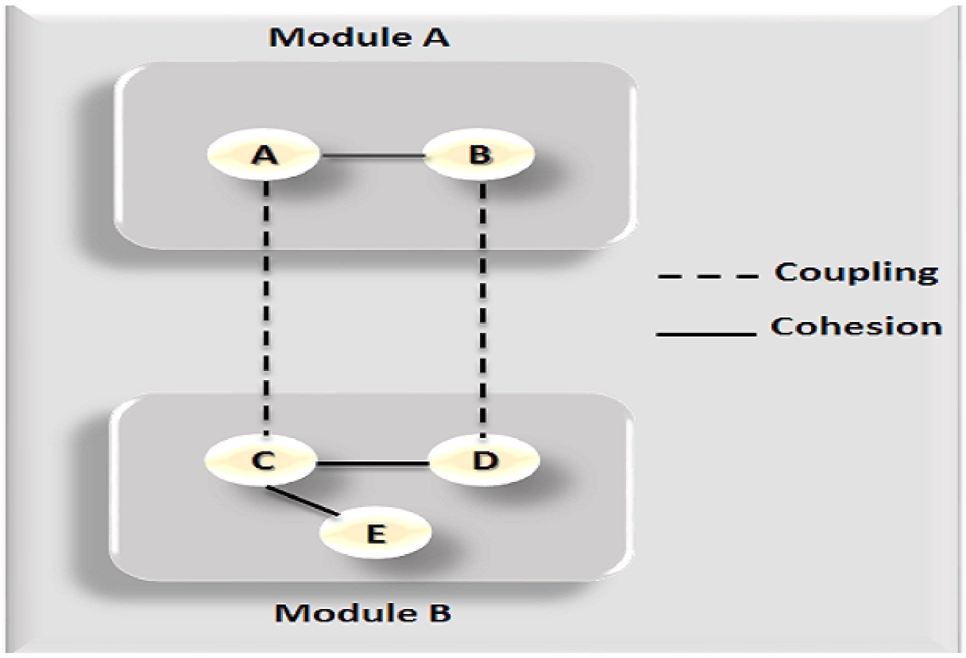
Types of Coupling:

1. **Data Coupling:** If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent of each other and communicate through data. Module communications don't contain tramp data. Example-customer billing system.
2. **Stamp Coupling** In stamp coupling, the complete data structure is passed from one module to another module. Therefore, it involves tramp data. It may be necessary due to efficiency factors- this choice was made by the insightful designer, not a lazy programmer.
3. **Control Coupling:** If the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality. Example- sort function that takes comparison function as an argument.

4. **External Coupling:** In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware. Ex- protocol, external file, device format, etc.
5. **Common Coupling:** The modules have shared data such as global data structures. The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change. So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses, and reduced maintainability.
6. **Content Coupling:** In a content coupling, one module can modify the data of another module, or control flow is passed from one module to the other module. This is the worst form of coupling and should be avoided.

Cohesion refers to the degree to which elements within a module work together to fulfill a single, well-defined purpose. High cohesion means that elements are closely related and focused on a single purpose, while low cohesion means that elements are loosely related and serve multiple purposes.

1. **Functional Cohesion:** Functional Cohesion is said to exist if the different elements of a module, cooperate to achieve a single function.
2. **Sequential Cohesion:** A module is said to possess sequential cohesion if the element of a module form the components of the sequence, where the output from one component of the sequence is input to the next.
3. **Communicational Cohesion:** A module is said to have communicational cohesion, if all tasks of the module refer to or update the same data structure, e.g., the set of functions defined on an array or a stack.
4. **Procedural Cohesion:** A module is said to be procedural cohesion if the set of purpose of the module are all parts of a procedure in which particular sequence of steps has to be carried out for achieving a goal, e.g., the algorithm for decoding a message.
5. **Temporal Cohesion:** When a module includes functions that are associated by the fact that all the methods must be executed in the same time, the module is said to exhibit temporal cohesion.
6. **Logical Cohesion:** A module is said to be logically cohesive if all the elements of the module perform a similar operation. For example Error handling, data input and data output, etc.
7. **Coincidental Cohesion:** A module is said to have coincidental cohesion if it performs a set of tasks that are associated with each other very loosely, if at all.



Q.3 Describe and Elaborate Architectural styles.

- The architectural style shows how do we organize our code, or how the system will look like from 10000 feet helicopter view to show the highest level of abstraction of our system design.
- Furthermore, when building the architectural style of our system we focus on layers and modules and how they are communicating with each other.

1. **Dataflow systems**
   - Dataflow systems are characterized by how data moves through the system.
   - Dataflow architectures have two or more data processing components that each transform input data into output data.
   - The data processing components transform data in a sequential fashion where the output of an upstream processing component becomes the input of the next processing component.
   - This example is called a pipeline because it is limited to a linear sequence of filters.

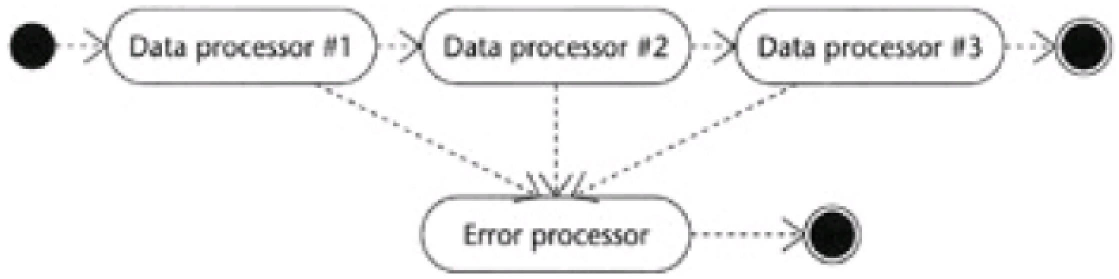


Figure 10.2: Pipes-and-filters architectural style.

Pipes and filters:

- It is common in a pipes-and-filters architecture that a processing component has two outputs, a standard output and an error output and a single input called standard input.
- Generically, the input and output mechanisms for a given processing element are called ports.
- Thus a typical filter has three ports.

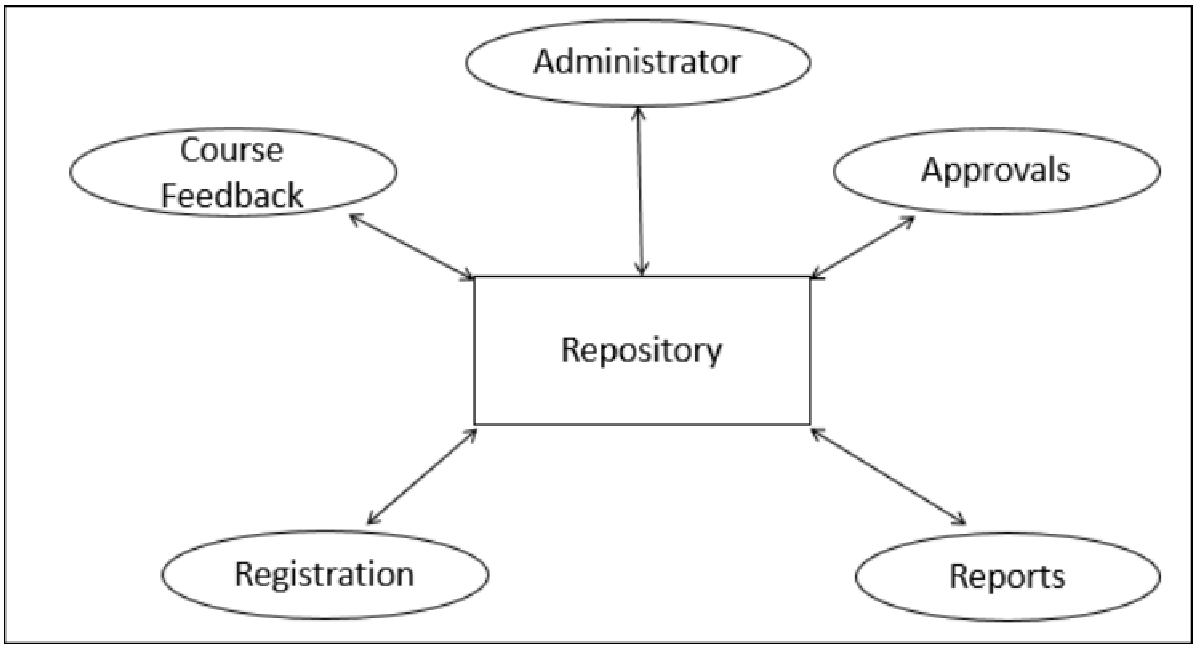
2. Repositories/ Data centered architecture

- In data-centered architecture, the data is centralized and accessed frequently by other components, which modify data. The main purpose of this style is to achieve integrality of data. Data-centered architecture consists of different components that communicate through shared data repositories. The components access a shared data structure and are relatively independent, in that, they interact only through the data store.
- The most well-known examples of the data-centered architecture is a database architecture, in which the common database schema is created with data definition protocol – for example, a set of related tables with fields and data types in an RDBMS.

Types:

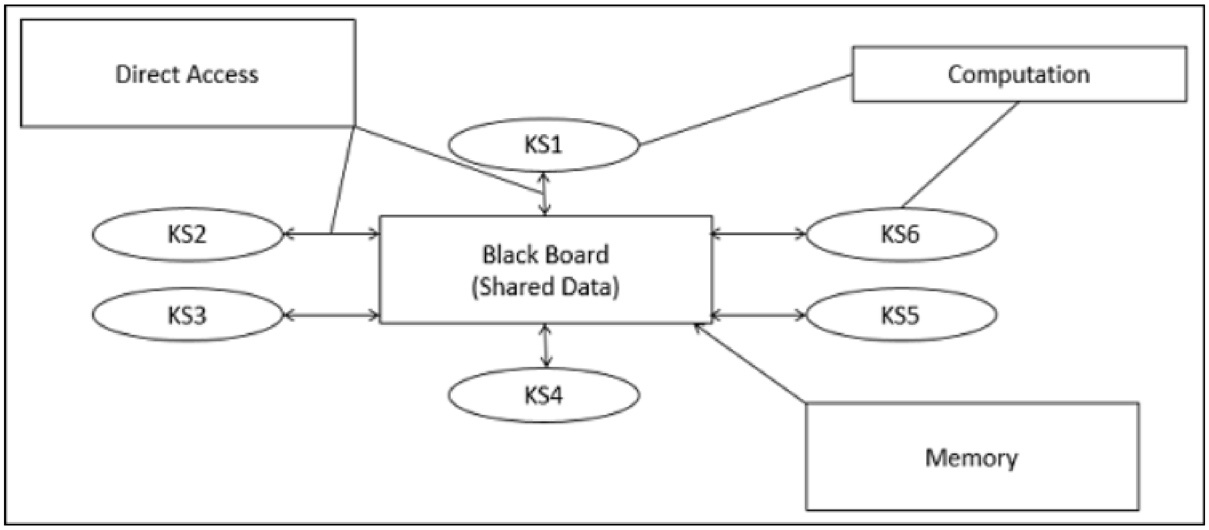
- Repository Architecture Style

In Repository Architecture Style, the data store is passive and the clients (software components or agents) of the data store are active, which control the logic flow. The participating components check the data-store for changes.

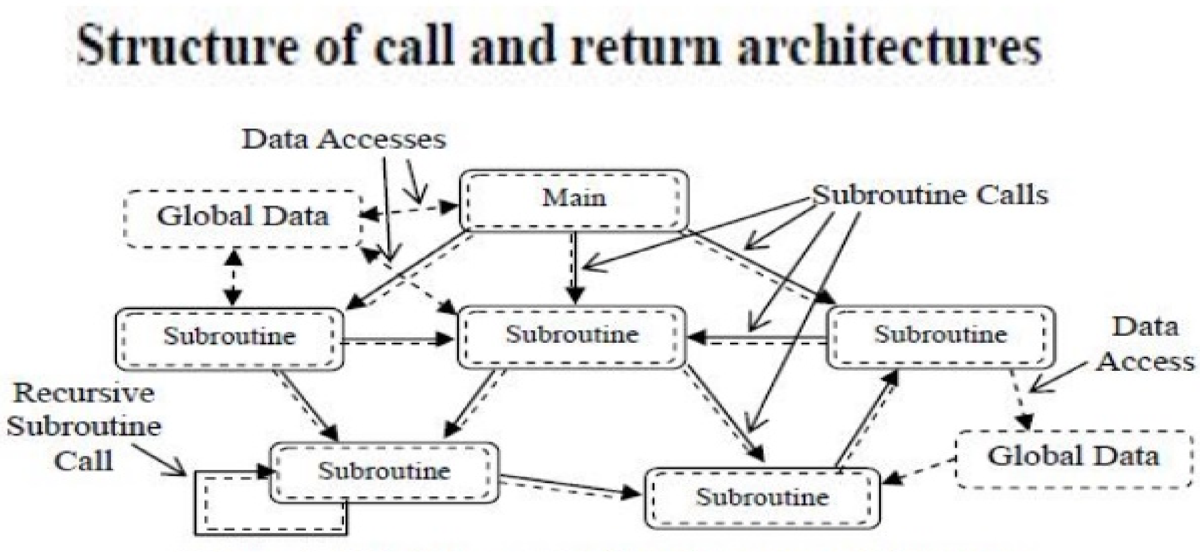


- Blackboard Architecture Style

In Blackboard Architecture Style, the data store is active and its clients are passive. Therefore the logical flow is determined by the current data status in data store. It has a blackboard component, acting as a central data repository, and an internal representation is built and acted upon by different computational elements.



3. **Call-and-return systems**
    - Call-and-return systems are characterized by an activation model that involves a main thread of control that performs operation invocations.
    - The classic system architecture is the main program and subroutine.
    - This architectural style enables a software designer to achieve a program structure that is relatively easy to modify and scale.



Types:

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.

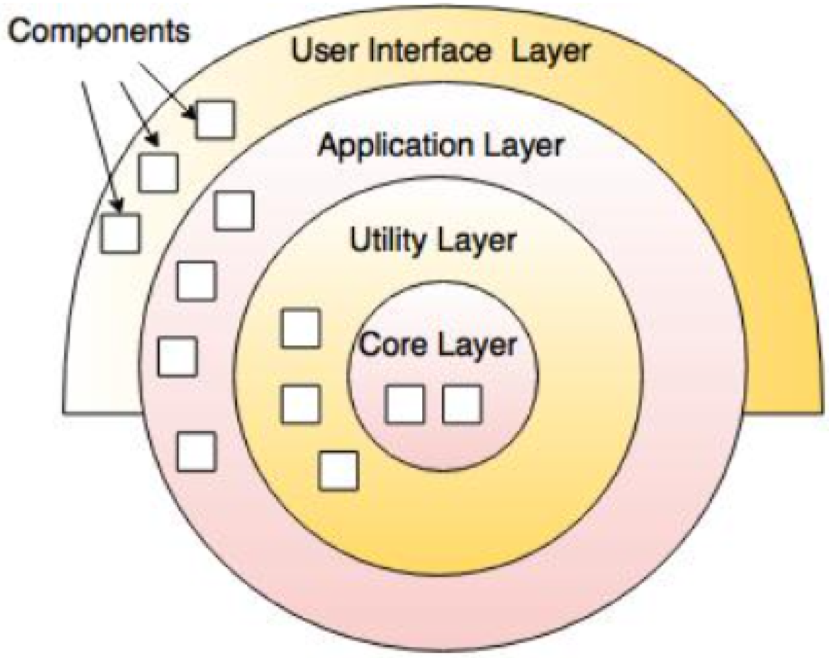
2. Remote procedure call architecture
    - The main program or subprogram components are distributed in network of multiple computers.
    - The main aim is to increase the performance.

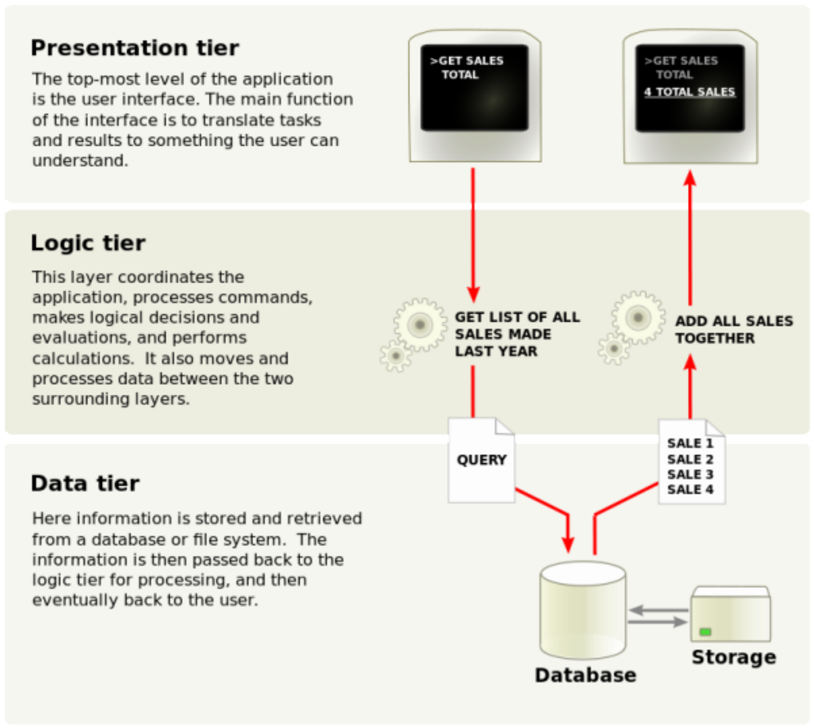
4. **Object-oriented systems**
    - The components of a system encapsulate data and the operations that must be applied to manipulation the data. Communication and coordination between components is accomplished via message passing.
    - Basic features include:

a. Encapsulation
b. Information hiding
c. Inheritance
d. Polymorphism
e. Message passing

5. Layered Architecture

- The layered architecture style is one of the most common architectural styles. The idea behind Layered Architecture is that modules or components with similar functionalities are organized into horizontal layers. As a result, each layer performs a specific role within the application.
- The layered architecture style does not have a restriction on the number of layers that the application can have, as the purpose is to have layers that promote the concept of separation of concerns. The layered architecture style abstracts the view of the system as a whole while providing enough detail to understand the roles and responsibilities of individual layers and the relationship between them.





Presentation tier

The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Data tier

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.