SOFTWARE ENGINEERING UNIT: 2

Software Design: Overview of the Design Process, How to Characterize of a Design, Cohesion and Coupling, Layered Arrangement of Modules, Approaches to Software Design

PART 2: SOFTWARE DESIGN

SOFTWARE DESIGN:

Software design is a crucial phase of the software development life cycle that focuses on solution domain of the system. It is the process of describing the blue print or sketch of final software product in the form of design model.

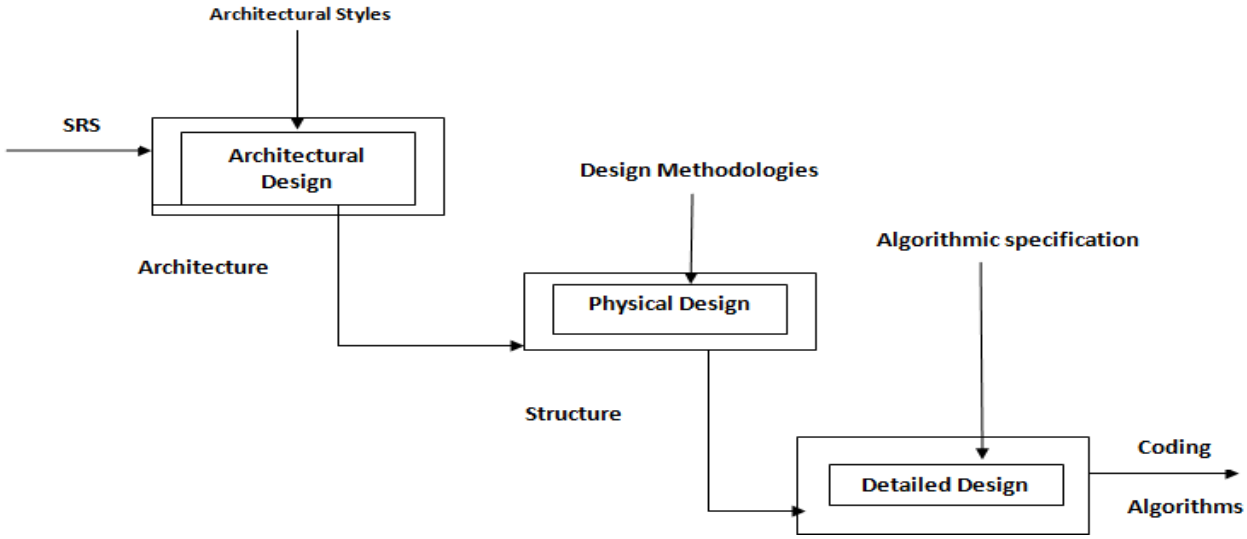
Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation. For assessing user requirements, an SRS (Software Requirement Specification) document is created whereas for coding and implementation, there is a need of more specific and detailed requirements in software terms.

OVERVIEW OF SOFTWARE DESIGN PROCESS

A software design exists between requirement engineering and programming. A software design process is a set of design activities carried out in the design phase to produce a design model from SRS.

The software design process basically consists of three design phases or design levels like-

- Architectural design
- Physical design
- Detailed design



Architectural Design:

It is an external design which considers the external behavior of the system concerned with people who control it, server, database, security aspects, and software running in it. The external design considers the architectural aspects related to business, technology, major data stores,

structure of data and modules, reports, performance criteria, and high-level process structure of the product. It represents the conceptual view in the form of schematic design.

Physical Design:

Physical design is a high-level design or structural design which is concerned with refining the conceptual view of the system; identifying the major modules; decomposing the modules into sub-modules, interconnections among modules, data structure, and data store in the system. This is also called ***system design***. Both high-level design and external design help in designing the overall software architecture. A design methodology is a systematic approach that is used to produce a design with the help of certain activities and guidelines.

Detailed Design:

Detailed design is the algorithmic design of each module in the software. It is also called ***logical design***. The detailed design concentrates on the specification of algorithms and data structures of each module, the actual interface description and data stores of the modules, and package specifications of the system.

HOW TO CHARACTERIZE OF A DESIGN

The quality of a software design can be characterized by the application domain. The desirable characteristics that a good software design should have are as follows-

- **Correctness:** A design is said to be correct if it is correctly produced according to the stated requirements of customers in the SRS. A correct design is more likely to produce accurate outcomes.
- **Efficient:** The Efficiency of a design is concerned with performance related issues; for example, optimal utilization of resources. The design should consume less memory and processor time. Software design and its implementation should be as fast as required by the user.
- **Understandability:** In a design, it should be easy to understand what the module is, how it is connected to other modules, what data structure is used, and its flow of information.
- **Maintainability:** A difficult and complex design would take a larger time to be understood and modified. The design should be easy to be modified, should include new features, should not have unnecessary parts, and it should be easy to migrate it onto another platform.
- **Simplicity:** A simple design will improve understandability and maintainability .Still designers always think to “keep it simple” rather than “make it complex”.
- **Completeness:** Completeness means that the design includes all the specifications of the SRS. A complete design may not necessarily be correct. But a correct design can be complete.
- **Verifiability:** The design should be able to be verified against the requirements documents and programs. Interfaces between the modules are necessary for integration and function prototyping.
- **Portability:** The external design mainly focuses on the interface, business, and technology architectures. This may be required when the system is to be migrated onto different platforms.

- **Modularity:** A modular design will be easy to understand and modify. Once a modular system is designed, it allows easy development and repairing of required modules independently.
- **Reliability:** The reliability factor depends on the measurement of completeness, consistency, and robustness in the software design.
- **Reusability:** The software design should be standard and generic so that it can be used for mass production of quality products with small cycle time and reduced cost. The object code, classes, design patterns, packages, etc., are the reusable parts of software.

COHESION AND COUPLING

Modular Design:

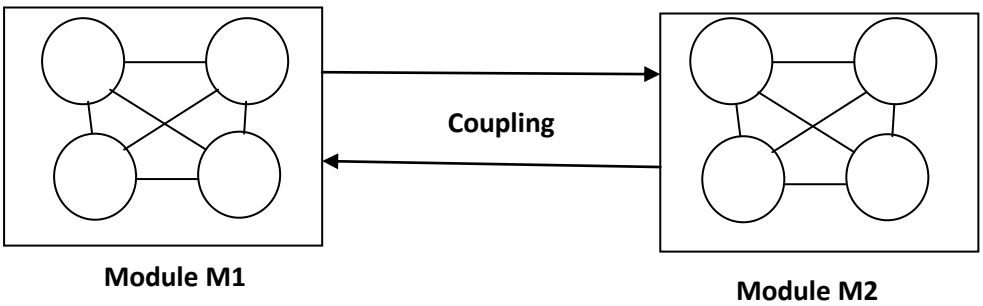
Modularization is the process of breaking a system into pieces called ***Modules***. A module is a part of a software system. A modular system consists of various modules linked via ***Interfaces*.** An Interface is a kind of link or relationship that combines two or more modules together.

Coupling and **Cohesion** are the most popular criterions used to measure the modularity in a system. In the following we discuss Coupling and Cohesion in detail.

COUPLING:

Coupling is the strength of interconnection between modules. It is the measure of the degree of interdependency between modules. Modules either loosely coupled or strongly coupled. In strong coupling, two modules are dependent each other. In loose coupling, there is weak interconnection between modules.

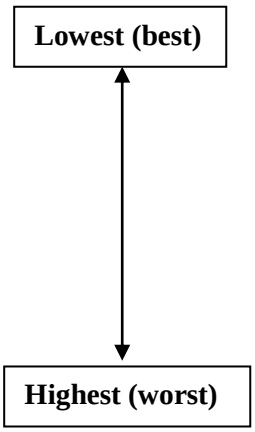
Interdependency between modules increases as coupling increases. In figure modules M1 and M2 are coupled modules. Module M1 is dependent on module M2 and vice versa.



Coupling increases as interconnection between modules increases. Interconnection between modules are measured by the number of function calls, number of parameters passed, return values, data types, sharing of data files or data items etc. The strength of interconnection between modules influenced by the level of complexity of the interfaces, type of connection, and the type of communication.

There are different types of coupling between modules can be ranked lowest (best) to highest (worst) is as follows:

1. Message Coupling
2. Data Coupling
3. Stamp Coupling
4. Control Coupling
5. External Coupling
6. Common Coupling
7. Content Coupling



Message Coupling: It is the lowest (best) type of coupling which exists between modules. It can be observed in an object –oriented or a component –based system where objects or components communicate through parameters or message passing.

Data Coupling: Data coupling exists between modules when data are passed as parameters in the argument list of the function call. Each datum is primary data item (e.g., integer, character, float etc.) It is good if a small number of data items is passed in the function call.

Stamp Coupling: Stamp coupling occurs between modules when data are passed by parameters using complex data structures, which may use parts or the entire data structure by other modules.

Control Coupling: It exists when one module controls flows of another by passing control information such as flag set or switch statements. It controls the sequence of instruction execution in another module.

External Coupling: It occurs when two modules share an externally imposed data format, communication protocol, or device interface. All the modules share the same I/O device or the external environment.

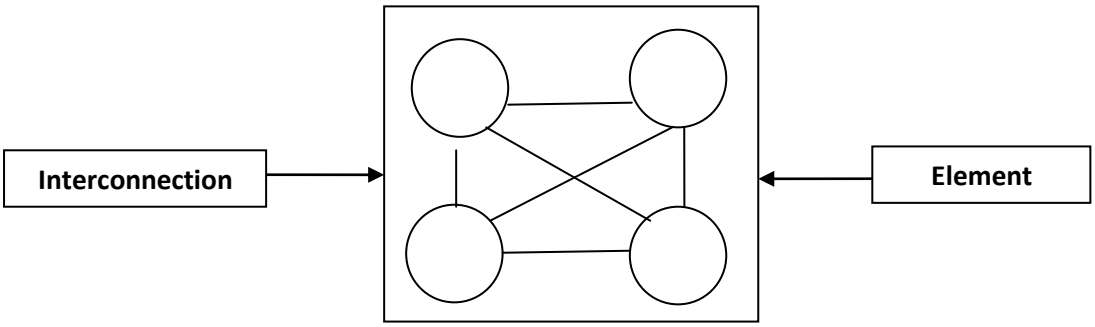
Common Coupling: It is when two modules share common data (e.g., global variable). In C language, external data items are accessed by all the modules in the program. If there is any change in the shared resource, it influences all the modules using it.

Content Coupling: It is highest coupling (worst). Content Coupling exists between modules when one module refers or shares its internal working with another module. Accessing local data items or instructions of another module is an example of content coupling.

COHESION:

Cohesion of a single module is the degree to which the elements of a single module are functionally related to achieve an objective. Module cohesion represents how tightly bound the internal elements of the module are to one another. A highly cohesive system is one which all data elements and procedures of a modules work together toward some objective. Cohesion

between the elements within the module itself. A functionally-independent module has higher cohesion as completed to dependent modules.

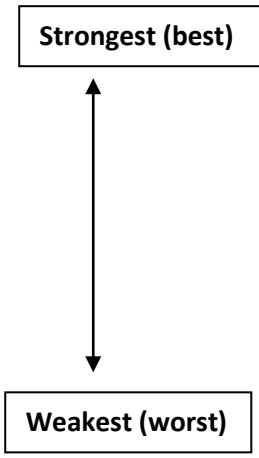


Module Cohesion

There are many different levels of cohesion used at several levels of a module, such as functional cohesion, sequential cohesion, communicational cohesion, procedural cohesion, temporal cohesion, logical cohesion, and coincidental cohesion. Functional cohesion is the strongest (best) and coincidental cohesion is the weakest (worst) level of cohesion.

The ratings of the common categories of cohesion ranked from the strongest (best) to the weakest (worst) are as follows:

1. Functional Cohesion
2. Sequential Cohesion
3. Communicational Cohesion
4. Procedural Cohesion
5. Temporal Cohesion
6. Logical Cohesion
7. Coincidental Cohesion



Functional Cohesion: It is the strongest cohesion as compared to other levels. In a functionally cohesive module, all the elements of the module perform a single function. These can be mathematical functions or functions having a well-defined goal.

Sequential Cohesion: It exists when the output from one element of a module becomes the input for some other element. It may contain several functions that may perform different tasks for a single module.

Communicational Cohesion: In Communicational cohesion, all the elements of a module operate on the same input or output data. A module may perform more than one function.

.

Procedural Cohesion: It contains the elements which belong to a common procedural unit. The functions are executed in a certain order.

Temporal Cohesion: In this cohesion, a module performs several functions in a sequence but their execution is related to a certain time. All the functions that are activated at a single time, such as start up or shut down, are brought together.

Logical Cohesion: It exists when logically related elements of a module are placed together. The elements of a module that perform similar functions such as input, error handling, etc., are put together in a single module.

Coincidental Cohesion: It is the weakest level of cohesion. It occurs when the elements within a given module have no meaningful relationship to each other. If there is a change in one part, then there will be no effect on the other part of the module.

LAYERED ARRANGEMENT OF MODULES

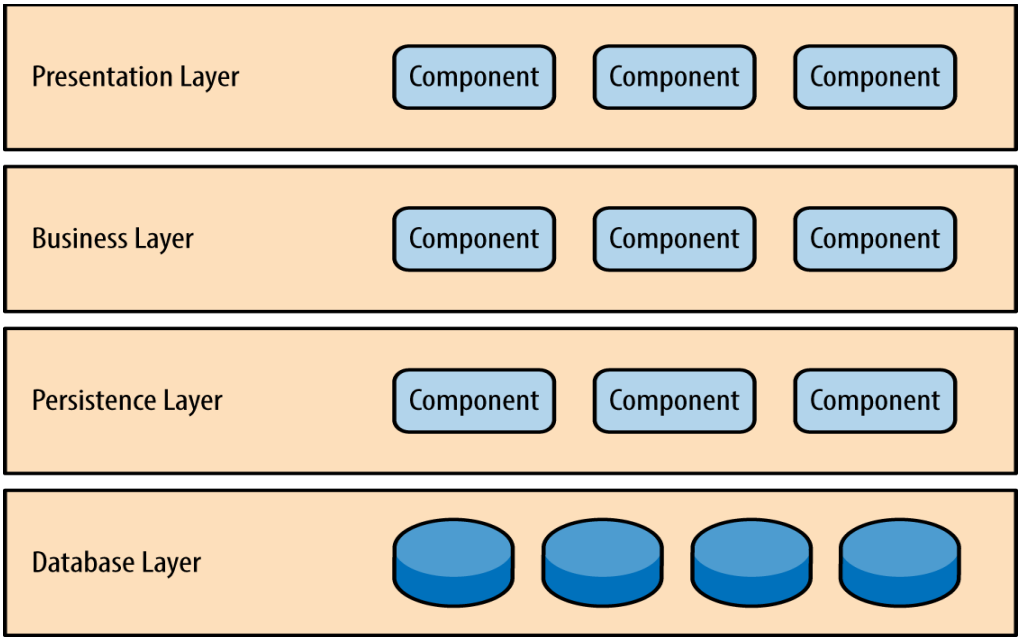
Software is a collection of modules to simplify the design complexity. Each module is connected to other to produce outcome. A connection is the relationship between modules. All the modules are represented in a control hierarchy.

In order to design software in a good architectural pattern, it follows the **Layered architectural pattern/ n-tier architectural pattern.**

Components within the layered architecture pattern are organized into horizontal layers**.** Each layer performing a specific role within the application.

Most layered architectures consist of four standard layers:

* Presentation
* Business
* Persistence
* Database



In some cases, the business layer and persistence layer are combined into a single business layer particularly when the persistence logic (e.g., SQL or HSQL) is embedded within the business layer components.

The presentation layer would be responsible for handling all user interface and browser communication logic, whereas a business layer would be responsible for executing specific business rules associated with the request.

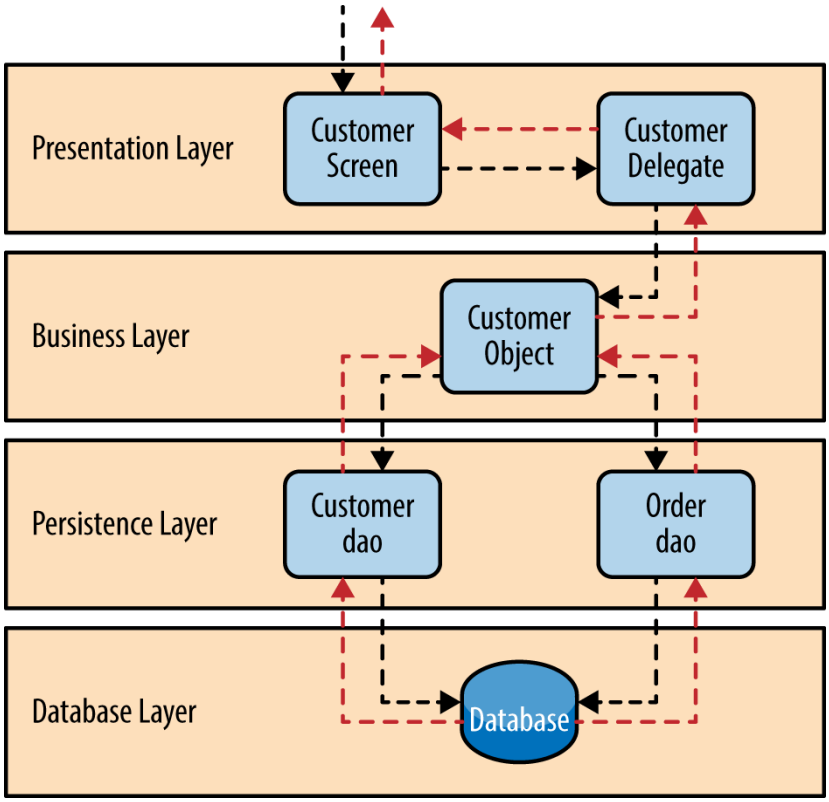
One of the powerful features of the layered architecture pattern is the *separation of concerns* among components. Components within a specific layer deal only with logic that pertains to that layer.

Example:

In this example, the customer information consists of both customer data and order data (orders placed by the customer). The black arrows show the request flowing down to the database to retrieve the customer data, and the red arrows show the response flowing back up to the screen to display the data.

The *customer screen* is responsible for accepting the request and displaying the customer information. Once the customer screen receives a request to get customer information for a particular individual, it then forwards that request onto the *customer delegate* module.

This module is responsible for knowing which modules in the business layer can process that request and also how to get to that module and what data it needs (the contract). The *customer object* in the business layer is responsible for aggregating all of the information needed by the business request (in this case to get customer information).



APPROACHES TO SOFTWARE DESIGN

Design approaches are used to propose a solution for the system in a conceptual manner once all the requirements are available. The design approaches are also referred as Design Methodologies.

A design methodology provides the techniques and guidelines for design process of the system. The design process consists of various design activities.

The most popular design methodologies are-

- Function-oriented design (Top-down approach)
- Object-oriented design (Bottom-up approach)

Function-oriented design:

It begins with requirement document ie., SRS to understand different modules. It follows the **top-down design** strategy which focuses on overall system. Thereafter, the system is **decomposed** into subsystems using top down strategy. During decomposition, the design decisions are reconsidered to avoid any mismatch or wrong module design. It works well for small and understandable problems.

Object-oriented design:

It has become a popular design methodology in the recent years. It follows the **bottom-up strategy** for design. It deals with real world **entities** of the environment for problem solving. These entities are characterized by **objects**, where similar objects are combined into a group called a **class.** It focuses on objects where services are distributed on different objects that can be passed to other objects. Here the complexity is reduced and the program structure becomes very clear.