**Software Design**

## **What is software design?**

Software design is the most important step in software development. It decides how the whole development will take place, which tools and strategies will be used, what the deadlines are and how the product will be tested, delivered, integrated, and launched.

In short, it is a whole set of activities for a software development project from start to the end.

According to Pressman Software Life-cycle Model, a software development project undergoes three main phases: definition, development, and maintenance. Software design comes under the development phase, putting forward how the project will be executed.

## **Software design methodology**

It is important to note there that software design is a skill; it requires a structure to offer the methodology for development. That’s where software design methodology or approaches come in.

Each software design approach comes with unique characteristics. It can be functional, behavioral, or structural, depending on the needs and objectives. Software methodology uses different diagrams and models to ensure the right sequence of activities.

## **Role and significance of software design methodology**

Design methodology is crucial for large projects that involve an array of programming languages being coded by different software developers.

Using an effective methodology offers the right communication channels so that all the developers keep end goals and objectives into consideration. Also, it helps translates design into codes, reducing the chances of errors.

## **Major software design methods**

Also known as software design approaches, the methods of software design are mainly classified into the following:

### **Level-oriented design**

The level-oriented design follows two common strategies: Step-Wise refinement and Design by Composition.

Step-wise refinement is also known as the top-down process. It begins at the topmost level and has the following significant features:

* It undergoes functional decomposition to create small modules which are easier to analyze, design and code
* It is dependent on the decisions made in the defining stage of the development process
* Each module is further decomposed into more detailed and refined small modules, known as the atomic level of a module
* All the modules are divided among software developers as per their skills and expertise.

### **2.      Data flow-oriented design**

Software development companies often use data flow-oriented design process when detailed information is available. It helps derive the design as per provided information so that clients end up with the desired custom software.

This method uses different techniques such as **[Data Flow Diagram](https://en.wikipedia.org/wiki/Data-flow_diagram)** (DFD), Structured Analysis and Design Technique (SADT), and Structured Design (SD), etc.

Each of these techniques comes with a unique process and tools and follows a unique structure. Custom software developers are well acquainted with such concepts.

### **3.      Data structure-oriented design**

The data structure-oriented design dedicates its focus upon the object, which is usually the available information. This structured information determines the efficiency of algorithms used for the process.

There can be different types of data such as alternative data, repetitive data, and hierarchal data. The data is represented using hierarchal diagrams. Software developers look into the characteristics of data structure, map the data into control hierarchy, refine it and come up with procedure of the software.

### **4.      Object-oriented design**

This approach works upon certain entities classified as objects. This involves identification of objects and classes. The object has a private data structure and so, developers choose the operations accordingly.

All objects include data, and the entire data is encapsulated for use and re-use.

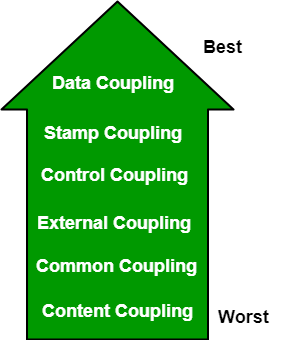
This is a rather new method of software design and is still evolving. It focuses on data design, architecture, and procedural design.

**Cohesion and Coupling**

**Modularization**: Modularization is the process of dividing a software system into multiple independent modules where each module works independently. There are many advantages of Modularization in software engineering. Some of these are given below:

* Easy to understand the system.
* System maintenance is easy.
* A module can be used many times as their requirements. No need to write it again and again.

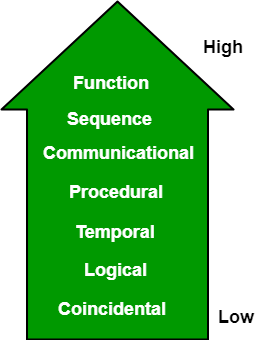
**Coupling**: Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling.



**Types of Coupling:**

* 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 to each other and communicating through data. Module communications don’t contain tramp data. Example-customer billing system.
* 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 made by the insightful designer, not a lazy programmer.
* 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.
* 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.
* 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.
* 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**: Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion.



**Types of Cohesion:**

* Functional Cohesion: Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.
* Sequential Cohesion: An element outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional programming languages.
* Communicational Cohesion: Two elements operate on the same input data or contribute towards the same output data. Example- update record int the database and send it to the printer.
* Procedural Cohesion: Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.
* Temporal Cohesion: The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time-span. This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at init time.
* Logical Cohesion: The elements are logically related and not functionally. Ex- A component reads inputs from tape, disk, and network. All the code for these functions is in the same component. Operations are related, but the functions are significantly different.
* Coincidental Cohesion: The elements are not related(unrelated). The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion. Ex- print next line and reverse the characters of a string in a single component.