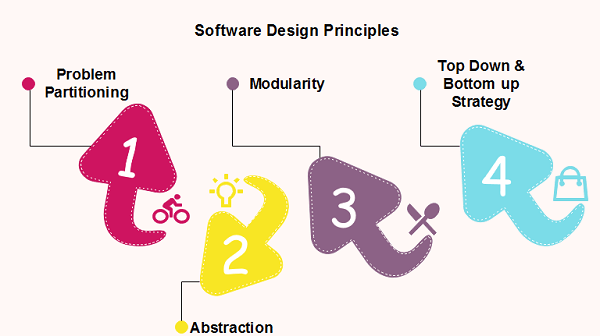
Software Design Principles

Software design principles are concerned with providing means to handle the complexity of the design process effectively. Effectively managing the complexity will not only reduce the effort needed for design but can also reduce the scope of introducing errors during design.

Following are the principles of Software Design



Problem Partitioning

For small problem, we can handle the entire problem at once but for the significant problem, divide the problems and conquer the problem it means to divide the problem into smaller pieces so that each piece can be captured separately.

For software design, the goal is to divide the problem into manageable pieces.

Benefits of Problem Partitioning

1. Software is easy to understand
2. Software becomes simple
3. Software is easy to test
4. Software is easy to modify
5. Software is easy to maintain
6. Software is easy to expand

These pieces cannot be entirely independent of each other as they together form the system. They have to cooperate and communicate to solve the problem. This communication adds complexity.

52.3M

941

Prime Ministers of India | List of Prime Minister of India (1947-2020)

**Next**

**Stay**

**Note:** As the number of partition increases = Cost of partition and complexity increases

Abstraction

An abstraction is a tool that enables a designer to consider a component at an abstract level without bothering about the internal details of the implementation. Abstraction can be used for existing element as well as the component being designed.

Here, there are two common abstraction mechanisms

1. Functional Abstraction
2. Data Abstraction

Functional Abstraction

1. A module is specified by the method it performs.
2. The details of the algorithm to accomplish the functions are not visible to the user of the function.

Functional abstraction forms the basis for **Function oriented design approaches**.

Data Abstraction

Details of the data elements are not visible to the users of data. Data Abstraction forms the basis for **Object Oriented design approaches**.

Modularity

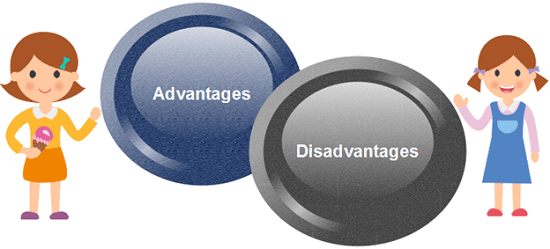
Modularity specifies to the division of software into separate modules which are differently named and addressed and are integrated later on in to obtain the completely functional software. It is the only property that allows a program to be intellectually manageable. Single large programs are difficult to understand and read due to a large number of reference variables, control paths, global variables, etc.

**The desirable properties of a modular system are:**

* Each module is a well-defined system that can be used with other applications.
* Each module has single specified objectives.
* Modules can be separately compiled and saved in the library.
* Modules should be easier to use than to build.
* Modules are simpler from outside than inside.

Advantages and Disadvantages of Modularity

In this topic, we will discuss various advantage and disadvantage of Modularity.



**Advantages of Modularity**

There are several advantages of Modularity

* It allows large programs to be written by several or different people
* It encourages the creation of commonly used routines to be placed in the library and used by other programs.
* It simplifies the overlay procedure of loading a large program into main storage.
* It provides more checkpoints to measure progress.
* It provides a framework for complete testing, more accessible to test
* It produced the well designed and more readable program.

**Disadvantages of Modularity**

There are several disadvantages of Modularity

* Execution time maybe, but not certainly, longer
* Storage size perhaps, but is not certainly, increased
* Compilation and loading time may be longer
* Inter-module communication problems may be increased
* More linkage required, run-time may be longer, more source lines must be written, and more documentation has to be done

Modular Design

Modular design reduces the design complexity and results in easier and faster implementation by allowing parallel development of various parts of a system. We discuss a different section of modular design in detail in this section:

**1. Functional Independence:** Functional independence is achieved by developing functions that perform only one kind of task and do not excessively interact with other modules. Independence is important because it makes implementation more accessible and faster. The independent modules are easier to maintain, test, and reduce error propagation and can be reused in other programs as well. Thus, functional independence is a good design feature which ensures software quality.

**It is measured using two criteria:**

* **Cohesion:** It measures the relative function strength of a module.
* **Coupling:** It measures the relative interdependence among modules.

**2. Information hiding:** The fundamental of Information hiding suggests that modules can be characterized by the design decisions that protect from the others, i.e., In other words, modules should be specified that data include within a module is inaccessible to other modules that do not need for such information.

The use of information hiding as design criteria for modular system provides the most significant benefits when modifications are required during testing's and later during software maintenance. This is because as most data and procedures are hidden from other parts of the software, inadvertent errors introduced during modifications are less likely to propagate to different locations within the software.

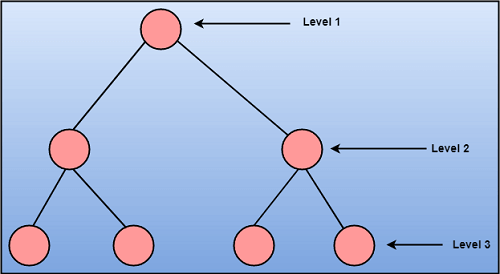
Strategy of Design

A good system design strategy is to organize the program modules in such a method that are easy to develop and latter too, change. Structured design methods help developers to deal with the size and complexity of programs. Analysts generate instructions for the developers about how code should be composed and how pieces of code should fit together to form a program.

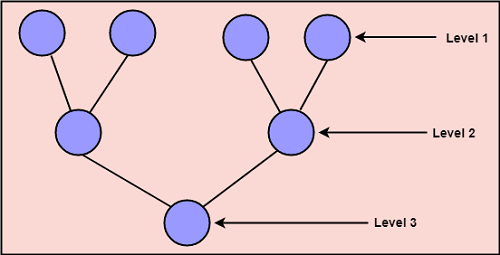
To design a system, there are two possible approaches:

1. Top-down Approach
2. Bottom-up Approach

**1. Top-down Approach:** This approach starts with the identification of the main components and then decomposing them into their more detailed sub-components.



**2. Bottom-up Approach:** A bottom-up approach begins with the lower details and moves towards up the hierarchy, as shown in fig. This approach is suitable in case of an existing system.



**1. High Level Design :**   
High Level Design in short HLD is the general system design means it refers to the overall system design. It describes the overall description/architecture of the application. It includes the description of system architecture, data base design, brief description on systems, services, platforms and relationship among modules. It is also known as macro level/system design. It is created by solution architect. It converts the Business/client requirement into High Level Solution. It is created first means before Low Level Design.

**2. Low Level Design :**   
Low Level Design in short LLD is like detailing HLD means it refers to component-level design process. It describes detailed description of each and every module means it includes actual logic for every system component and it goes deep into each modules specification. It is also known as micro level/detailed design. It is created by designers and developers. It converts the High Level Solution into Detailed solution. It is created second means after High Level Design.

**Difference between High Level Design and Low Level Design :**

| **S.No.** | **HIGH LEVEL DESIGN** | **LOW LEVEL DESIGN** |
| --- | --- | --- |
| 01. | High Level Design is the general system design means it refers to the overall system design. | Low Level Design is like detailing HLD means it refers to component-level design process. |
| 02. | High Level Design in short called as HLD. | Low Level Design in short called as LLD. |
| 03. | It is also known as macro level/system design. | It is also known as micro level/detailed design. |
| 04. | It describes the overall description/architecture of the application. | It describes detailed description of each and every module. |
| 05. | High Level Design expresses the brief functionality of each module. | Low Level Design expresses details functional logic of the module. |
| 06. | It is created by solution architect. | It is created by designers and developers. |
| 07. | Here in High Level Design the participants are design team, review team, and client team. | Here in Low Level Design participants are design team, Operation Teams, and Implementers. |
| 08. | It is created first means before Low Level Design. | It is created second means after High Level Design. |
| 09. | In HLD the input criteria is Software Requirement Specification (SRS). | In LLD the input criteria is reviewed High Level Design (HLD). |
| 10. | High Level Solution converts the Business/client requirement into High Level Solution. | Low Level Design converts the High Level Solution into Detailed solution. |
| 11. | In HLD the output criteria is data base design, functional design and review record. | In LLD the output criteria is program specification and unit test plan. |