## Design Principles and Pattens Transcript:

Hello Guys,

Today we will learn about SOLID principles. Before we moving to the topic lets learn What is Software Design Principles and Design patterns and why they are important to build a good software.

Software designs is a process to plan or convert the software requirements into software systems. There are several principles are used to organize and arrange structural component of software design.

Let’s Quickly Go through each and every term in an Oval.

### Principles Of Software Design:

1. **Should not suffer from “Tunnel Vision” –**  
   While designing the process, it should not suffer from “tunnel vision” which means that is should not only focus on completing or achieving the aim but on other effects also.
2. **Traceable to analysis model –**  
   The design process should be traceable to the analysis model which means it should satisfy all the requirements that software requires to develop a high-quality product.
3. **Should not “Reinvent The Wheel” –**  
   The design process should not reinvent the wheel that means it should not waste time or effort in creating things that already exist. Due to this, the overall development will get increased.
4. **Minimize Intellectual distance –**  
   The design process should reduce the gap between real-world problems and software solutions for that problem meaning it should simply minimize intellectual distance.
5. **Exhibit uniformity and integration –**  
   The design should display uniformity which means it should be uniform throughout the process without any change. Integration means it should mix or combine all parts of software i.e. subsystems into one system.
6. **Accommodate change –**  
   The software should be designed in such a way that it accommodates the change implying that the software should adjust to the change that is required to be done as per the user’s need.
7. **Degrade gently –**  
   The software should be designed in such a way that it degrades gracefully which means it should work properly even if an error occurs during the execution.
8. **Assessed or quality –**  
   The design should be assessed or evaluated for the quality meaning that during the evaluation, the quality of the design needs to be checked and focused on.
9. **Review to discover errors –**  
   The design should be reviewed which means that the overall evaluation should be done to check if there is any error present or if it can be minimized.
10. **Design is not coding, and coding is not design –**  
    Design means describing the logic of the program to solve any problem and coding is a type of language that is used for the implementation of a design.

Courtesy: (<https://www.geeksforgeeks.org/principles-of-software-design/>)

### Design principle

Design principle is a framework for the designer to follow as good software practices.

### Revisit Interface

### Revisit Abstraction

### Solid Principles

* Maintain
* Extend
* Easier to Understand

**S – SINGLE RESPONSIBILITY PRINCINPLE**

**O – OPEN CLOSED PRINCIPLE**

**L – LISKOV SUBSTITUTION PRINCIPLE**

**I – INTERFACE SEGREGATION PRINCIPLE**

**D – DEPENDENCY INVERSION PRINCIPLE**

### SINGLE RESPONSIBILITY PRINCINPLE

Class should have one responsibility and only one reason to change.

Misinterpretation – class should perform only one task.

Example: Swiss Knife, Viru sahastra buddhe

Demo

### OPEN/CLOSED PRINCIPLE

“Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification”.

One can do refactoring or resolve errors of implementation.

The implementation is open for extension, in other words one can extend the implementation of logic and/or functionality.

Example - electric adapter

As you can see in the image:

* An adapter in the wall is always closed for modification, because we cannot change it once it is fitted or extended if we want more.
* But an adapter always provides a method of extension, so we can plug in an extension board of an adapter for more adaptation.
* So you plug in an extension board and extend an existing electric adapter fitted in wall.

Disadvantage of not following Open Closed Principle

1. Since a class or function always allows the addition of new logic, whenever new logic is added it is always necessary to test for full functionality. That requires the addition of a new test case for the added functionality and might also require the modification of an existing test case that fails because of added functionality.
2. It also breaks the Single Responsibility Principle since a class or function might end up doing multiple tasks.
3. Class or function maintenance becomes difficult since a class or function can become thousands of lines of code that is difficult to understand.

Demo

### LISKOV SUBSTITUTE PRINCIPLE

Derived classes should be able to extend their base classes without changing/affecting their behavior.

Based on Inheritance.

Example: Bulb, Square and Rectangle issue

Disadvantages:

Runtime code failure, misbehave.

Demo

### INTERFACE SEGREGATION PRINCIPLE

Client (class implementation interface) should not force to implement Interface that they don't use.

Example :

Demo

### DEPENDENCY INVERSION PRINCIPLE

High-level modules should not depend on low level modules and vice versa, both should depend upon abstraction.