***Software Design principles***

***Tactical principles:***

Do not repeat youeself (DRY)

Duplication in logic should be eliminated via abstraction; duplication in process should be eliminated via automation.

Keep it stupid simple (KISS)

Most systems work best if they are kept simple rather than made complicated; therefore, simplicty should be a key goal in design, and unnecessary complexity should be avoided.

You aren’t gonna need it (YAGNI)

Features should only be added when required. As a part of the extreme programming philosophy.

Uncle Bob clean code tips:

* Function names should be verbs.
* Functions should contain one level of abstraction.
* Extract functions until you can’t extract them anymore.
* Functions should contain a maximum of two indents.
* Functions should have 3 arguments at maximum.
* You should NOT pass booleans as arguments to a function except in setters.
* Avoid switch statements.
* Comments should be used only when you can’t express yourself in code.
* Variables should have a length proportional to the size of thier scope.
* Function and class names should be inversely proportional to the size of thier scope.
* Avoid Hungarian notation.

***Strategic principles:***

## Single Responsibility Principle (SRP)

## **A class should have one and only one reason to change, meaning that a class should have only one job.**

## Open Closed Principle (OCP)

## **Objects or entities should be open for extension, but closed for modification.**

## Liscov Substitution Principle (LSP)

## Functions that use pointers or references to base classes must be able to use objects of derived classes without causing any problem in the system.

## Interface Segregation Principle (ISP)

## **A client should never be forced to implement an interface that it doesn’t use or clients shouldn’t be forced to depend on methods they do not use.**

## Dependency Inversion Principle (DIP)

High level modules should not depend upon low level modules. Both should depend upon abstractions and abstractions should not depend upon details. Details should depend upon abstractions.

Robustness principle:

Be conservative in what you send, be liberal in what you accept.

# Encapsulate what changes:

Encapsulate or isolate the parts of your system that change more often.

### Composite Reuse principle:

Classes should achieve [polymorphic](https://en.wikipedia.org/wiki/Polymorphism_(computer_science)) behavior and [code reuse](https://en.wikipedia.org/wiki/Code_reuse) by their [composition](https://en.wikipedia.org/wiki/Object_composition) rather than [inheritance](https://en.wikipedia.org/wiki/Inheritance_(computer_science)) from a base or parent class.

### Delegation principles:

Don’t do all stuff by yourself, delegate it to the respective class.