**Software Design Principle**

What are Software design principles? They are widely applicable guidelines, biases and design considerations which designers apply as per there use.

These principles are ideal pieces of advice for you to make easy-to-use, presentable designs. You apply them when you are creating, organising features in your work. These principles represent the accumulated experience of practitioners in design and related fields. These rules help developers to create software as per industry standards and which are scalable and robust in most of the situations.

Now discussing various Software Design Principles.

1) DRY – Don’t Repeat Yourself

2) KISS – Keep it Simple Silly

**DRY (Don’t Repeat Yourself)**

This is aimed at reducing repetitions of information. Every piece of information and logic must be single and unambiguous. Most of the time code developers tend to repeat the code which violates DRY principle. So to ensure that DRY logic does not fails following steps to follow:-

1. Don’t write length codes.
2. Divide your logic into pieces which can be used in other places also.
3. You can use [Inheritance](https://www.w3schools.com/java/java_inheritance.asp), [Polymorphism](https://www.tutorialspoint.com/java/java_polymorphism.htm#:~:text=Polymorphism%20is%20the%20ability%20of,is%20considered%20to%20be%20polymorphic.) from OOPs concept to ensure code reusability.

Less code is easy to maintain and debug.

**KISS (Keep it Simple Silly)**

This principle is aimed to keep the code simple and clear, making it easy to understand. Also, programming languages are for humans to understand — computers can only understand 0 and 1. Keep your methods small. Method length should never be more than 40-50 lines.

Each method should only solve one small problem. If you have a lot of conditions in the method, break these out into smaller methods. It will not only be easier to read and maintain, but it can help find bugs a lot faster. Typically when a developer is faced with a problem, they break it down into smaller pieces. Many developers make the mistake that they don't break down the problem into small enough or understandable enough pieces.

This code clutter is a result of the developer realizing exception cases to his original code while he is in initial development of code. These exception cases would have been solved if the developer had broken down the problem further.

Benefits of KISS:

1. Faster Problem Solving.
2. Less number of code lines to solve complex problems.
3. High quality code as the result.
4. Easy maintainability of codes.
5. You're code base will be more flexible, easier to extend, modify or refactor when new requirements arrive.

**YAGNI (You Aren’t Gonna Need It)**

Some time may be due to overthinking or maybe as an effort to make your code robust you tend to include those cases into your code that are not needed.

YAGNI as a principle is a part of XP (Extreme Programming). By implementing the ideals of “You Aren’t Gonna Need It” programming, you will save yourself time and be able to move forward with projects efficiently.

When you feel an unexplained anxiety to code some extra features that in the moment are not necessary, but might be useful in the future, just calm down and see all the pending work you have at this moment. Include them only if you have time and does not hamper the timeline