**DESIGN PRINCIPLES**

Analysis is about what to do it and

design is about how to do it.

These are the basic requirements for building a project

* Why do we need the design is the basic question?
* Here are few needs of why we need a design.

1. To manage the changes
2. To reduce the complexity
3. To deliver faster

* In this world the performance of everything is defined by one of its qualities.

For some mobile apps it is the response time to open and use the app.

* Likewise for the Design, it is “WTF” it means work that frustrates. The lesser the WTF the better the design is.

**SYMPTOMS OF BAD DESIGN**

1. Rigidity-how inflexible the code is.

2.fragile- the quality of easily broken or damaged

3.Immobile- In ability to reuse the software in other systems

4.viscous

The reason for the above symptoms is the bad dependencies b/w the modules

While writing a code base we should be concentrating on high cohesion and low coupling.

**Cohesion** represents the degree to which a part of a code base forms a logically single, atomic unit.

**Coupling**, on the other hand, represents the degree to which a single unit is independent from others. In other words, it is the number of connections between two or more units. The fewer the number, the lower the coupling.

* high cohesion means **keeping parts of a code base that are related to each other in a single place**. Low coupling, at the same time, is about **separating unrelated parts of the code base as much as possible**.
* The cohesion and coupling are the measures by which the quality of design of modules and the interaction between them

The cohesion is basically inside the module. (

The coupling is basically between 2 or more modules.

There are seven types of cohesion.

**Coincidental**

**Logical**

**Temporal**

**Procedural**

**Communicational**

**Sequential**

**Functional Cohesion**

There are five levels of Coupling.

**Content coupling**

**Common coupling**

**Control coupling**

**Stamp coupling**

**Data coupling**

How can we achieve the good principles for design

* SOLID principles
* KISS principles
* DRY principles
* YAGNI principles

**SOLID PRINCIPLES**

Single Responsibility principle

Open closed principle:(open for extension, closed for modify)the system can be extended without having to modify its existing implementation. i.e. New features should be implemented using the new code, but not by changing existing code

Liskov substitution principle: It means that we must make sure that new derived classes are extending the base classes without changing their behaviour.

Interface segregation: Clients should not be forced to implement any methods they don’t use. Rather than one fat interface, numerous little interfaces are preferred based on groups of methods with each interface serving one submodule

Dependency Inversion: Dependency Injection (DI) is a software design pattern that allows us to develop loosely coupled code.

Types of dependency injection:

Constructor injection

Method Injection

Getter /setter Injection

**KISS PRINCIPLE(**Keep it simple stupid**)**

* Most systems work best if they are kept simple rather than making complicated (simplicity is the key goal in design)
* Make sure a method should not contain more than 40-50 lines.
* Each method should only solve one problem if there are many functionalities to be implemented then. Break these into small methods.

**DRY PRINCIPLE(Don’t repeat yourself)**

This principle states that each small pieces of knowledge (code) may only occur exactly once in the entire system. This helps us to write scalable, maintainable and reusable code.

**YAGNI PRINCIPLE (YOU AINT GONNA USE IT)**

This principle states that always implement things when you need them never implements things before you need them.