**Design Principles**

* In SDLC, analysis is done before design.
* Analysis is deciding what needs to be done.
* Design is about how it needs to be done.
* **Why Design?**
  + To deliver faster
  + To manage changes
  + To deal with complexity
* Designs help in breaking down complex systems into simple and small systems so that there’ll be more clarity and readability.
* **Traits of Bad Design:**
  + Rigidity: Difficult to change
  + Fragility: Easily broken
  + Immobility: Can’t be moved
  + Viscosity: Highly connected internally
* **Traits of Good Design:**
  + **High Cohesion**
    - Cohesion is a measure that defines the degree of intra-dependability within elements of a module. The greater the cohesion, the better is the program design.
  + **Low Coupling**
    - Coupling is a measure that defines the level of inter-dependability among modules of a program. It tells at what level the modules interfere and interact with each other. The lower the coupling, the better the program.
* **Design Principles: 5 principles to Achieve Good Design**
  + **SOLID Principles**
    - **S**ingle Responsibility Principle
      * A class should have one, and only one, reason to change.
    - **O**pen Close Principle
      * Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.
    - **L**iskov Substitution Principle
      * Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program. (By Barbara Liskov in 1987)
    - **I**nterface-Segregation Principle
      * 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.
    - **D**ependency Inversion
      * The principle says that there should not be a tight coupling among components of software and to avoid that, the components should depend on abstraction.
  + **KISS Principle**
    - **K**eep **I**t **S**imple and **S**tupid / **S**illy
    - Designed by U.S Navy in 1960
    - Design simple systems rather than complicated systems
    - Avoid unnecessary complexity
    - Keep methods small (not more than 40-50 lines)
    - Each method should solve only one problem
  + **DRY Principle**
    - **D**on’t **R**epeat **Y**ourself
    - Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.
    - 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**
    - **Y**ou **A**in’t **G**onna **N**eed **I**t
    - Always implement things when you actually need them never implements things before you need them.
    - Do not unnecessarily write any code thinking that it’s required in future. Write only what’s needed.