Solution to develop a successful application depends on

1. **Architecture:** Choosing an architecture is the first step in designing application based on the requirements. E.g., MVC, Web API, MVVM etc.
2. **Design Principles:** Application development process need to follow the design principles.
3. **Design Patterns:** We need to choose correct patterns to build the software.

SOLID Design Principle

* SOLID principles are the design principles that enable us manage most of the software design problems.
* The SOLID design principles are intended to make software designs more understandable, flexible and maintainable.

1. **Single Responsibility Principle (SRP)**
2. **Open closed Principle (OSP)**
3. **Liskov Substitution Principle (LSP)**
4. **Interface Segregation Principle (ISP)**
5. **Dependency Inversion Principle (DIP)**
6. **Single Responsibility Principle (SRP)**
7. A class should have only one reason to change.
8. Every module or class should have responsibility over a single part of the functionality provided by the software, and that responsibility should be entirely encapsulated by the class.
9. **Open closed Principle (OSP)**
10. Software entities should be open for extension but closed for modification.
11. The design and writing of the code should be done in a way that new functionality should be added with minimum changes in the existing code.
12. The design should be done in a way to allow the adding of new functionality as new classes, keeping as much as possible existing code unchanged.
13. **Liskov Substitution Principle (LSP)**
14. Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.
15. If a program module is using a Base class, then the reference to the Base class can be replaced with a Derived class without affecting the functionality of the program module.
16. We can also state that Derived types must be substitutable for their base type.
17. **Interface Segregation Principle (ISP)**

1. Many client-specific interfaces are better than one general-purpose interface.

2. We should not enforce clients to implement interfaces that they don’t use. Instead of creating one big interface we can break down it to smaller interface.

1. **Dependency Inversion Principle (OSP)**
2. One should depend upon abstractions not concretions.
3. Abstractions should not depend on the details whereas the details should depend on abstractions.
4. High-level modules should not depend on low level modules.

**If we don’t follow SOLID Principles we**

1. End up with tight or strong coupling of the code with many other modules/applications.
2. Tight coupling causes time to implement any new requirement, features or any bug fixes and sometimes it creates unknown issues.
3. End up with code which is not testable.
4. End up with duplication of code.
5. End up creating new bugs by fixing another bug.
6. End up with many unknown issues in the application development cycle.

**Advantages**

1. Achieve reduction in complexity of code.
2. Increase readability, extensibility and maintenance.
3. Reduce error and implement Reusability.
4. Achieve better testability.
5. Reduce tight coupling.