# SOLID:

**The reason behind most unsuccessful applications**

Developers build applications with good and tidy designs using their knowledge and experience. But over time, applications might develop bugs. The application design must be altered for every change request or new feature request. After some time, we might need to put in a lot of effort, even for simple tasks, and it might require a full working knowledge of the entire system.

**The following are the design flaws that cause damage to software, mostly:**

* Putting more stress on classes by assigning more responsibilities to them. (A lot of functionality not related to a class.)
* Forcing the classes to depend on each other. If classes depend on each other (in other words, tightly coupled), then a change in one will affect the other.
* Spreading duplicate code in the system/application.

**S: Single Responsibility Principle (SRP)**

Every software module should have only one Responsibility



**O: Open/Closed Principle:**

Here "Open for extension" means we must design our module/class so that the new functionality can be added only when new requirements are generated. "Closed for modification" means we have already developed a class, and it has gone through unit testing. We should then not alter it until we find bugs. As it says, a class should be open for extensions;

**L: Liskov Substitution Principle**

The Liskov Substitution Principle (LSP) states, "you should be able to use any derived class instead of a parent class and have it behave in the same manner without modification.". It ensures that a derived class does not affect the behavior of the parent class; in other words, a derived class must be substitutable for its base class.

**I: Interface Segregation Principle (ISP):**

The Interface Segregation Principle states "that clients should not be forced to implement interfaces they don't use. Instead of one fat interface, many small interfaces are preferred based on groups of methods, each serving one submodule.".

**D: Dependency Inversion Principle**

The Dependency Inversion Principle (DIP) states that high-level modules/classes should not depend on low-level modules/classes

First, both should depend upon abstractions. Secondly, abstractions should not rely upon details. Finally, details should depend upon abstractions.

**חומר קריאה מומלץ מאד:**

https://www.c-sharpcorner.com/UploadFile/damubetha/solid-principles-in-C-Sharp/