**SOLID**

* Softwares devem ser construídos de modo que possibilitem evoluções graduais, ou seja, mudanças nos requisitos do sistema;
* Os principais motivos destas mudanças são:
  + Mudanças no negócio;
  + Regulamentos;
  + Falhas no sistema.
* É normal softwares apresentarem falhas durante seu desenvolvimento, não somente durante a manutenção;
* Resolver os erros apresentados rapidamente e sem prejudicar o funcionamento do sistema é extremamente importante;
* Um software de qualidade é construído de modo que evite soluções complexas e difíceis de decifrar. Ex:
  + Grande interdependência entre as classes;
  + Classe sobrecarregada;
  + Métodos ociosos e/ou inutilizados;
* SOLID é o conjunto de práticas e princípios de projeto orientado a objeto que resulta em várias características desejáveis na construção de um software;
* Significado de SOLID:
  + SRP: Single Responsability Principle
    - Uma classe não deve atender mais que um objeto.
  + OCP: Open Closed Principle
    - Entidades de um software devem estar abertas para extensão e fechadas para alteração.
  + LSP: Liskov Substitution Principle
    - Classes filhas devem ser capazes de utilizarem funções da classe pai sem necessidade de saberem que são classes filhas.
  + ISP: Inteface Segregation Principle
    - Clientes não devem ser forçados a dependerem de interfaces que não necessitam.
  + DIP: Dependence Inversion Principle
    - Módulos de alto nível não devem depender de módulos de baixo nível;
    - Abstrações não devem depender de detalhes. Detalhes devem depender de abstrações.
* **SRP**:
  + Evite classes extensas que realizem uma grande quantidade de funções;
  + Cada objeto deve realizar apenas uma função:
    - Um método deve realizar apenas uma ação;
    - Um objeto de domínio deve representar somente um domínio.
  + Objetivos:
    - Objetos concisos;
    - Facilitar a criação de testes automatizados;
    - Melhorar a legibilidade do código fonte;
    - Facilitar a manutenção.
  + O software é dividido em várias pequenas partes especializadas que encapsulam estado e comportamento.
* **OCP**:
  + Priorizar a extensão de funcionalidades (herança e polimorfismo);
  + Alterar o comportamento de uma entidade sem alterar o seu código fonte;
  + Objetivos:
    - Diminuir a chance de que um novo comportamento quebre as funcionalidades já existentes;
    - Estimular a criação de classes mais precisas;
    - Facilitar a manutenção.
* **LSP**:
  + A classe que referência deve ser capaz de utilizar as funções da classe referenciada sem a necessidade de instanciar a classe referenciada antes;
  + Objetivos:
    - Auxiliar na escolha correto do uso de herança;
    - Evitar a verificação de tipos de objeto;
    - Evitar métodos com muitas condicionais;
    - Evitar comportamentos inesperados durante a execução.
* **ISP**:
  + Clientes não devem depender de interfaces que não usam;
  + Interfaces complexas podem ser divididas em interfaces mais simples, que não exigem implementação de funcionalidades indesejadas;
  + Objetivos:
    - Auxiliar no uso correto de superclasses e interfaces;
    - Evitar erros causados por métodos ociosos ou não implementados;
    - Aumentar o reuso.
* **DIP**:
  + Utilizar interfaces e composições para possibilitar a substituição de funcionalidades através de comportamento polimórfico;
  + Objetivos:
    - Facilitar o reaproveitamento de código;
    - Facilitar a manutenção (diminui a quebra de funcionalidade);
    - Facilitar os testes automatizados.