SOLID는 함수와 데이터 구조를 클래스로 배치하는 방법이자 클래스의 결합 방법을 설명하는 소프트웨어 설계의 5가지 원칙을 말한다. SOLID는 SRP, OCP, LSP, ISP, DIP로 구성되어 있으며 프로그램에 새로운 요구사항이나 수정이 있을 때 다른 지점이 받는 영향을 최소화 시키는 것을 목표로 한다.

SRP는 단일 책임 원칙으로, 한 동작에 대해서만 책임을 질 것을 요구한다. 이를 위해서는 추상화를 할 때 역할 구분이 명확해야 한다. SRP를 준수할 경우 프로그램의 유지보수에서 이점을 가질 수 있다. 프로그램을 수정해야 할 때 수정을 최소한만 할 수 있기 때문이다. SRP의 좋은 설계는 클래스의 동작을 최소화함으로써 많은 역할을 부여하지 않는 것으로부터 시작된다. 반대로 SRP의 나쁜 설계는 다른 클래스와 많이 얽혀 있는 것이다.

OCP는 개방 폐쇄 원칙으로, 클래스의 확장 가능성은 열어두되, 변경 가능성이 있으면 안된다고 말한다. 기존의 코드를 수정하지 않고 추가함으로써 기능을 확장하는 것이 OCP이다. OCP는 재사용성을 향상시킴으로써 유지 보수에 도움을 준다. 또한 OCP를 구현하기 위해서는 매개가 되는 인터페이스를 사용하면서 환경을 만드는 것이 좋다. 클래스의 확장과 변경의 구분이 모호할 경우 코드를 재사용하지 않고 비슷한 기능을 추가로 만들 수 있는데 이는 OCP의 나쁜 설계이다.

LSP는 리스코프 원칙으로 상속에서의 가이드가 된다. LSP에 따르면 상속 관계에서는 일관성을 통해 자식 클래스가 최소한 부모 클래스의 역할을 수행할 수 있어야 한다. 즉, 프로그램에서 부모 클래스의 인스턴스를 자식 클래스의 인스턴스가 대체할 수 있어야 한다. LSP를 준수할 경우 다형성을 구현할 수 있다는 장점이 있다. 이러한 LSP는 부모 클래스를 잘 이해하고 오버라이딩을 하지 않는 설계를 함으로써 구현된다. 부모 클래스를 재정의하는 설계를 할 경우 자식 클래스가 부모 클래스의 역할을 수행할 수 없는 경우가 생긴다.

ISP는 인터페이스 분리 원칙을 말한다. ISP는 SRP의 확장으로 인터페이스 역시 단일 책임을 질 것을 요구한다. ISP를 준수하기 위해서는 한 개의 범용 인터페이스가 아니라 여러 개의 특화 인터페이스를 만들어야 한다. ISP를 준수할 경우 불필요한 기능으로 인해 프로그램이 무거워지는 것을 방지할 수 있다. 또한 프로그램에 수정이 있을 때 인터페이스와 연관 없는 오류를 막음으로써 유지 보수에 장점을 준다. ISP의 좋은 설계는 인터페이스를 목적에 맞게, 클라이언트에게 특화되도록 분리하는 것이고 나쁜 설계는 인터페이스에 사용하지 않는 기능을 넣어둠으로써 구현된다.

DIP는 의존 역전 원칙으로 의존 관계를 맺을 때 변화가 적은 곳 위주로 관계를 맺을 것을 말한다. DIP를 통해서 클래스가 다른 클래스에게 종속적이 되는 것을 막을 수 있다. 따라서 DIP는 프로그램의 유지 관리와 수정에 이점을 준다. 이를 위해서 개발자는 관계를 설정할 때 추상을 매개로 관계하는 설계를 만들 필요가 있다. 또한 상속은 신중하게 결정해야 하고 오버라이딩은 안 하는 것이 좋다. 한편 수정이 많은 클래스나 구체적인 클래스, 혹은 현재 개발 중인 클래스와 관계를 맺는 것은 DIP의 나쁜 설계이다.