Principiile SOLID

Termenul SOLID este un acronim pentru cinci principii de design aplicate asupra design-ului software pentru a-l face mai usor de inteles, mai flexibil si mai usor de intretinut.

Principiul Singurei Responsabilitati

Responsabilitatea unui modul(ex. clasa) ar trebui sa fie doar o parte din functionalitatea oferita de software si acea responsabilitate ar trebui sa fie incapsulata in clasa respectiva.

Daca cerintele sunt modificate atunci trebuie si o restructurare la nivel de cod insemnanad faptul ca clasele trebuie modificate.Cu cat o clasa are mai multe responsabilitati, cu atat mai multe modificari vor fi necesare si mai greu de implementat. Responsabilitatiile unei clase sunt dependente una de cealalta rezultand ca o modificare a uneia se va propaga si asupra alteia.

Principiul Substitutiei Liskov

Functiile unei superclase ar trebui sa poata fi folosite de instante ale unor subclase. Principiul Substitutiei Liskov este in stransa legatura cu Principiul Singurei Responsabilitati si Principiul Segregarii Interfetei.

Principiul Deschis/Inchis

Entitatile software sa fie deschise pentru extindere dar inchise pentru modificari.Acest lucru s-ar putea traduce astfel: Codul scris sa fie realizat astfel incat sa permita adaugarea de noi functionalitati (cum ar fi adaugarea de noi clase) mentinand codul existent pe cat posibil neschimbat.

Realizarea acestui lucru se realizeaza prin abstractizare. Pentru a exdinde comporatmentul unei clase fara a fi nevoiti sa schimbam o singura linie de cod, ne folosim de abstractizari.

Principiul Segregarii Interfetei

Creearea de mai multe interfete utilizator care au roluri specifice si nu o singura interfata cu un rol general.

Presupunem ca avem o clasa Animal care are metodele eat,spleep si walk. Nu este o abstractizare perfetcta pentru clasa Animal pentru ca unele animale au si abilitatea fly.Impartind aceste trei functii in interfete am avea CanEat, CanSleep,CanWalk. Asta ar face posibil pentru o specie sa eat,sleep si spre exemplu fly. O specie ar fi o combinatie de roluri in loc sa fie caracterizata ca Animal.

Principiul Inversarii Dependentei

Abstractiile nu ar trebui sa depinda de detalii ci detaliile ar trebui sa depinda de abstractii.

Depinzand de abstractizarile de nivel mai mare putem schimbacu usurinta o instanta cu o alta instanta cu scopul de a schimba comportamentul.

Beneficii

-reducerea complexitatii codului
-codul este mai lizibil si usor de intretinut in cazul aparitiei unor erori
-reducerea posibilitatii aparitiei erorilor
-testare codului este mai eficienta

Concluzie

Exista doua concepte importante cand se vorbeste despre dezvoltare software:
Coeziune: cand diferite parti ale unui sistem lucreaza impreuna pentru a furniza rezultate mai bune decat daca fiecare parte ar lucra individual
Cuplare: poate fi vazut ca un grad de dependenta al unei clase, metode sau oricare alta enitate software.