Tehnici de programare

Capitolul 6

Danciu Gabriel
Decembrie 2018

Principiile SOLID

- Principiile de design software reprezintă un set de recomandări care ne ajută în a evita conceperea greșită a programului.
- Există trei elemente care trebuie evitate atunci când se concepe arhitectura unui produs:
  - 1. rigiditate - dacă un sistem este închis, este greu de schimbat
  - 2. imobilitate - reutilizarea unui modul este imposibilă atunci când există o interdependență între acesta și restul modulelor
  - 3. fragilitate - o schimbare poate aduce prăbușirea întregului sistem
- De aceea atunci când se concepe/implementează un produs acesta trebuie să respecte cele 5 principii

Single responsability

- Orice modul sau clasă trebuie să încapsuleze o singură funcționalitate.
- O clasă sau funcție, nu ar trebui să rezolve sau sa trateze mai mult decât un singur scop deoarece aceasta ar introduce cuplarea între cele două funcționalități.

Open Close

- Acest principiu exprimă faptul că o clasă sau o funcție, trebuie să fie deschisă pentru extensie dar închisă pentru modificare.
- De exemplu, dacă avem o bibliotecă ce conține un set de clase ce trebuie extinse ar fi mai corect dacă am avea inițial un set de clase abstracte. Astfel în loc să modificăm clasele concrete din cadrul bibliotecii, putem extinde clasele abstracte.
- Conform prof. Bertrand Meyer acest principiu se rezumă la 2 reguli:
  - Un modul se declara *open* dacă acesta se poate extinde. De exemplu, se pot adăuga membrii noi unei clase, prin moștenire.
  - Un modul se declară *close* dacă poate fi utilizat de alte module. Aceasta înseamnă că a fost clar definit, iar informația esențială este declarată private.

Liskov substitution

- Acest principiu spune că, într-un program dacă *S* este subtip al lui *T*, atunci obiectele de tip *T* pot fi înlocuite cu obiecte de tip *S*, fără modificarea funcționalităților esențiale.
- Acest principiu impune unele condiții în scrierea metodelor:
  - Contravarianța argumentelor metodelor unui subtip. Aceasta înseamnă impunerea unei relații inverse de ordine între părinte și copil
  - Covarianța tipurilor pe care le returnează un subtip
  - Subtipurile nu aruncă excepții. Excepția de la regulă este atunci când acele excepții sunt subtipuri ale celor aruncate de metodele tipului părinte
- Altfel spus noile clase extinse din părinte trebuie să fie capabile înlocuiască clasa de bază în toate funcțiile sale fără a fi nevoie să aducem modificări nicăieri.

Interface segregation

- Acest principiu expune ideea că niciun client nu are voie să fie forțat să depindă de metodele pe care nu le folosește
- Aceasta se realizează prin ”spargerea” interfețelor în module mai mici pe care clasele au opțiunea de a le implementa sau nu.
- Acest mod va ajuta și la ideea de decuplarea funcționalităților atunci când se dorește re-factorizarea produsului.

Dependency inversion

- Acest principiu se referă la o anumită formă de a decupla module software. Principiul spune că:
- Modulele de nivel înalt nu trebuie să depindă de modulele de nivel jos.
- Ambele module trebuie să depindă de abstractizări.
- Abstractizările nu trebuie să depindă de detalii ci detaliile trebuie să depindă de abstract.

Hands on: implementați cel puțin 2 principii în cadrul proiectului dumneavoastră