Tema PSSC

În programarea informatică orientată pe obiecte, termenul SOLID este un acronim pentru cinci principii de proiectare menite să facă modelele de software mai ușor de înțeles, mai flexibile și mai ușor de întreținut. Este strâns legată de principiile GRASP de proiectare a software. Principiile sunt un subset al mai multor principii promovate de Robert C. Martin. Deși se aplică în cazul oricărui proiect orientat pe obiecte, principiile SOLID pot de asemenea să formeze o filozofie de bază pentru metodologii cum ar fi dezvoltarea Agile sau Adaptive Software Development (ASD). Teoria principiilor SOLID a fost introdusă de Martin în lucrarea „Design Principles and Design Patterns", deși acronimul SOLID însuși a fost introdus mai târziu de Michael Feathers.

S - Single Responsibility Principle: prevede că fiecare clasă ar trebui să aibă o singură responsabilitate. Nu trebuie să existe mai mult de un motiv pentru schimbarea unei clase. Atunci când cerințele se schimbă, acest lucru implică faptul că codul trebuie să fie supus unei anumite reconstrucții, ceea ce înseamnă că clasele trebuie să fie modificate. Cu cât există mai multe funcționalități într-o clasă, cu cât mai multe solicitări de schimbare vor fi cerute și cu atât mai grele vor fi aceste schimbări. Responsabilitățile unei clase sunt legate între ele, deoarece schimbările uneia dintre responsabilități pot duce la modificări suplimentare pentru ca celelalte responsabilități să fie gestionate corespunzător de acea clasă.

O – Open – Closed Principle: afirmă că entitățile software trebuie să fie deschise pentru extensie, dar închise pentru modificare. „Dechise pentru extensie" transmite faptul că comportamentul clasei poate fi extins. Pe măsură ce cerințele se schimbă, ar trebui să putem face o clasă să se comporte în moduri noi și diferite, pentru a satisface noilor cerințe. „Închise pentru modificare" spune că nimănui nu i se permite să modifice codul (pentru a nu cauza probleme de funcționalitate).

L – Liskov Substitution Principle: spune că obiectele dintr-un program ar trebui să poată fi înlocuite cu instanțe ale subtipurilor lor, fără a modifica corectitudinea programului. Ceea ce înseamnă cu adevarat este că dacă se trece o subclasă a unei abstractizări, trebuie să ne asigurăm că nu schimbăm nici un comportament sau semantica abstractizarii părinte.

I – Interface Segregation Principle: prevede că mai multe interfețe specifice clientului sunt mai bune decât o interfață cu scop general. Acest principiu este similar principiului „Single Responsibility Principle". O interfață este un „contract" care satisface o nevoie. Este bine să ai o clasă care implementează interfețe diferite, dar trebuie să fim atenți pentru a nu încălca SRP.

D – Dependency Inversion Principle: are două puncte cheie: Primul: **i)** modulele de nivel înalt nu trebuie să depindă de module de nivel scăzut. Ambele ar trebui să depindă de abstractizări. Al doilea: **ii)** Abstractizările nu trebuie să depindă de detalii. Detaliile ar trebui să depindă de abstractizări. Depinzând de abstractizările de nivel superior, putem schimba ușor o instanță cu o altă instanță pentru a schimba comportamentul. Inversiunea dependenței mărește reutilizarea și flexibilitatea codului nostru.