Maghiș Beniamin-Dorin

Tema 1 – PSSC

S.O.L.I.D. – *"Guidelines for structures that can survive shifting tides"*

S.O.L.I.D. este unul dintre cele mai populare seturi de principii de proiectare în dezvoltarea de aplicații orientate pe obiecte. Cele cinci principii sunt principii ce fac un design bun al unei clase.

Single Responsibility Principle – Principiul Singurei Responsabilități - *"Avoid tightly coupling your tools together"*

Argumentul pentru folosirea principiul singurei responsabilități este relativ simplu: face software-ul mai ușor de implementat și împiedică efectele secundare neașteptate ale schimbărilor viitoare. O clasă ar trebui să aibă numai un motiv să se schimbe. Fără obiecte care încearcă să facă prea multe lucruri diferite.

Open/Closed Principle – Principiul Deschis/Închis - *"Brain surgery is not necessary when putting on a hat"*

Entitățile de software (clase, module, funcții etc.) ar trebui să fie deschise pentru extindere, dar închise pentru modificare. Permite ușoara adăugare de noi funcționalități. Comportamentul unei clase poate fii extins fără modificarea clasei. Este vorba despre utilizarea polimorfismului și a abstractizării pentru a construi noi funcționalități.

Liskov Substitution Principle – Principiul Substituției Liskov - *"If it looks like a duck, quacks like a duck, but needs batteries – you probably have the wrong abstraction"*

Principiul definește faptul că obiectele unei superclase trebuie să poată fi înlocuite cu obiecte din subclasele sale fără distrugerea aplicației. Ideea din spatele acestui principiu este: codul trebuie scris astfel încât, dacă se creează o nouă clasă derivată dintr-o clasă de bază, nu ar trebui modificat codul pentru a lucra cu noua clasă derivată.

Interface Segregation Principle – Principiul Segregării Interfeței - *"Tailor interfaces to individual clients' needs"*

Clienții nu ar trebui să fie obligați să depindă de interfețele pe care nu le folosesc. Interfețele trebuie să aparțină clienților, nu bibliotecilor sau ierarhiilor. Interfețe subțiri, concentrate, nu interfețe "grase" care oferă mai multă funcționalitate decât o anumită clasă sau metodă are nevoie.

Dependency Inversion Principle – Principiul Inversării Dependenței - *"Would you solder a lamp directly to the electrical wiring in a wall?"*

Modulele de nivel înalt, care oferă o logică complexă, ar trebui să fie ușor reutilizabile și neafectate de schimbările din modulele de nivel scăzut, care oferă funcții de utilitate. Construirea de clase create pentru a utiliza abstractizările obiectelor terță parte, astfel încât obiectele concrete să poată fi modificate fără nevoia rescrierii codului.