DRY - Don’t Repeat Yourself

- Aplicabil ori de câte ori suntem nevoiți să folosim Copy/Paste la nivelul unei secțiuni de cod
- Poate fi ușor de evitat dacă mutăm codul comun într-o funcție/metodă pe care o vom apela din mai multe zone
- Aplicabil și pentru constante (șiruri de caractere, coduri de culoare, etc), ce pot fi mutate la nivel central în cadrul unei aplicații

KISS - Keep It Simple and Stupid

- De cele mai multe ori metodele simple sunt și cele mai eficiente
- Divizați funcțiile ce devin prea mari în funcții mai mici
- Nu complicați lucrurile mai mult decât e cazul

YAGNI - You Ain’t Gonna Need It

- Nu scriem cod ce nu e necesar în acest moment (în ideea că va fi necesar în viitor)
- Există o probabilitate destul de mare ca să nu mai fie necesar niciodată
- Dinamica aplicațiilor software din prezent este destul de ridicată, se testează ipoteze, se modifică specificațiile destul de des (Waterfall is dead)

S.O.L.I.D.

- Single Responsibility Principle (SRP)
- Open-Closed Principle (OCP)
- Liskov Substitution Principle (LSP)
- Interface Segregation Principle (ISP)
- Dependency Inversion Principle (DIP)

Single Responsibility Principle

- O clasă trebuie să aibă o singură responsabilitate și numai una
- Cu alte cuvinte, o clasă trebuie să aibă un singur motiv pentru care va fi modificată
- În caz contrar schimbările de specificații la nivelul unei responsabilități, vor afecta și celelalte responsabilități, facând clasa inutilizabilă sau foarte greu de actualizat în viitor

Open-Closed Principle

- O clasă trebuie să fie deschisă pentru derivări și închisă pentru modificări
- Se preferă derivarea în locul actualizării codului sursă din clase existente
- Avantajele sunt multiple, dar cel mai important este cel legat de backwards-compatibility

Liskov Substitution Principle

- Obiectele trebuie să poată fi înlocuite oricând cu instanțe ale claselor derivate fără ca acest lucru să afecteze funcționalitatea
- Vine ca o completare la relația de derivare și spune că o întrebare mai bună decât „is a?" este „can be a substitute of?"
- Pot exista situații când din punctul de vedere al lumii reale (business logic) relația de tip „is a" are sens, dar din punct de vedere al programării orientate obiect nu are (exemplul clasic pătrat vs dreptunghi)

Interface Segregation Principle

- Mai multe interfețe specializate sunt oricând de preferat unei singure interfețe generale
- Nu riscăm astfel ca prin modificarea „contractului” unui client să modificăm și contractele altor clienți
- Obiectele nu trebuie obligate sa implementeze metode care nu sunt utile
- Ușor de identificat atunci când suntem obligați să implementăm metode, dar nu le putem da un sens

Dependency Inversion Principle

- O clasă trebuie să depindă de abstractizări, nu de alte clase concrete
- Permite o cuplare slabă a diverselor componente ale unei aplicații
- Componentele de nivel jos vor depinde de componentele de nivel înalt și nu invers
- Unul dintre cele mai importante principii ce cresc mult mentenabilitatea codului sursă