SOLID

Gute Software schreiben

Tomasz Lubowiecki

SOLID

Gute Software schreiben

- **S**ingle-Responsibility-Prinzip
- **O**pen-Closed-Prinzip
- **L**iskovsches Substitutionsprinzip
- **I**nterface-Segregation-Prinzip
- **D**ependency-Inversion-Prinzip

Single-Responsibility-Prinzip

„Es sollte nie mehr als einen Grund dafür geben, eine Klasse zu ändern.“

Robert C. Martin

Single-Responsibility-Prinzip

- Eine Klasse soll nur eine Verantwortlichkeit haben
- Viele kleine Klassen sind wenigen großen vorzuziehen

Open-Closed-Prinzip

„Module sollten sowohl offen (für Erweiterungen) als auch geschlossen (für Modifikationen) sein.“

Bertrand Meyer

Open-Closed-Prinzip

- Eine Klasse soll offen für Erweiterungen, aber geschlossen gegenüber Modifikationen d.h. das Verhalten einer Klasse darf erweitert, aber nicht verändert werden.
- Zwei möglichen Wege: Vererbung, Interfaces.

Liskovsches Substitutionsprinzip

„Sei q(x) eine Eigenschaft des Objektes x vom Typ T, dann sollte q(y) für alle Objekte y des Typs S gelten, wobei S ein Subtyp von T ist.“

Barbara Liskov

Liskovsches Substitutionsprinzip

- Eine Subklasse erfüllt immer alle Eigenschaften der Superklasse und ist als Objekt der Superklasse verwendbar.
- Eine Subklasse darf Erweiterungen enthalten, aber keine grundlegenden Änderungen.

Interface-Segregation-Prinzip

„Clients sollten nicht dazu gezwungen werden, von Interfaces abzuhängen, die sie nicht verwenden.“

Robert C. Martin

Interface-Segregation-Prinzip

- Client sollte nicht von den Funktionen eines Servers abhängig sein darf, die er gar nicht benötigt.
- Ein Interface darf nur die Funktionen enthalten, die eng zusammengehören.

Dependency-Inversion-Prinzip

„A. Module hoher Ebenen sollten nicht von Modulen niedriger Ebenen abhängen. Beide sollten von Abstraktionen abhängen.

B. Abstraktionen sollten nicht von Details abhängen. Details sollten von Abstraktionen abhängen.“

Robert C. Martin

Dependency-Inversion-Prinzip

- Prinzip: Module höherer Hierarchie beschreiben generelle Abläufe, die von spezielleren Modulen verwendet werden. Je niedriger die Hierarchie, desto spezifischere Probleme löst es.
- Klassen auf einem höheren Abstraktionslevel dürfen nicht von Klassen auf einem niedrigen Abstraktionslevel abhängig sein
- Auf Abhängigkeiten zwischen Klassen sollte verzichtet werden; es sollen nur noch Abhängigkeiten zu Interfaces bestehen (beidseitig).