Hochschule Esslingen
University of Applied Sciences

ENTWURFSPRINZIPIEN

DIE SOLID-PRINZIPIEN NACH ROBERT C. MARTIN

Agenda

Hochschule Esslingen
University of Applied Sciences

1) Einführung

2) SOLID-Prinzipien nach Robert C. Martin

3) Fazit

Mängel von Software

Hochschule Esslingen
University of Applied Sciences

1. Einführung
2. SOLID-Prinzipien
3. Fazit

- Auch Software unterliegt einem **Alterungsprozess**
- Symptome des **Alterungsprozesses**:
  - Änderungen sind schwer einzupflegen
  - Anpassungen an geänderte Programmumgebungen wie z. B. Frameworks sind schwer durchzuführen
- Robert C. Martin: "Software verrottet und stinkt"
- Grund: Zu viele Abhängigkeiten im Design
- Starke Kopplung der Software-Komponenten führt dazu, dass bei Änderungen unerwünschte Nebeneffekte auftreten

→ Laufendes Refactoring oder komplette Neuentwicklung der Software sinnvoll!

Alterungsprozess von Software

Ziele für den Softwareentwurf

Hochschule Esslingen
University of Applied Sciences

1. Einführung
2. SOLID-Prinzipien
3. Fazit

Ein System soll:

- erweiterbar,
- korrekt,
- stabil,
- so einfach wie möglich und
- verständlich dokumentiert sein.

Diese Ziele erreicht man hauptsächlich durch eine **Minimierung der Abhängigkeiten.**

Zu erreichende Ziele

Ziele für den Softwareentwurfs

1. Einführung

2. SOLID-Prinzipien

3. Fazit

Um die genannten Ziele erreichen zu können, gibt es Prinzipien und Konzepte, die beim Entwurf einzuhalten sind. Diese kann man folgendermaßen unterteilen:

- Prinzipien zum Entwurf von Systemen
- Prinzipien zum Entwurf einzelner Klassen
- Prinzipien zum Entwurf miteinander kooperierender Klassen

Einsatz von Prinzipien

1. Einführung
2. SOLID-Prinzipien
3. Fazit

Robert C. Martin fasste eine wichtige Gruppe von Prinzipien zur Erzeugung **wartbarer** und **erweiterbarer** Software unter dem Begriff **SOLID** zusammen. Dieser Begriff soll andeuten, dass diese Prinzipien für das Schreiben hochwertiger Software unabdingbar sind. Robert C. Martin erklärte diese Prinzipien zu den wichtigsten Entwurfsprinzipien.

Die **SOLID**-Prinzipien bestehen aus:

- Single Responsibility Prinzip
- Open-Closed Prinzip
- Liskovsches Substitutionsprinzip
- Interface Segregation Prinzip
- Dependency Inversion Prinzip

SOLID nach Robert C. Martin

Single Responsibility Prinzip

1. Einführung
2. SOLID-Prinzipien
3. Fazit

Das SRP "There should never be more than one reason for a class to change" (Ursprünglich nur auf Klassen bezogen, seit 2014 auf Software-Module im Allgemeinen) stammt von Robert C. Martin.
Es bedeutet:

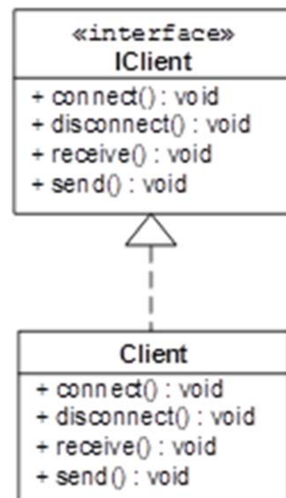
- Jedes Software-Modul sollte nur eine einzige Verantwortlichkeit realisieren
- Verantwortlichkeit = Grund für eine Änderung
- Dem Prinzip Separation of Concerns sehr ähnlich
- Mehrere Verantwortlichkeiten innerhalb eines Software-Moduls führen zu zerbrechlichem Design, da bei Änderung einer Verantwortlichkeit eine andere Verantwortlichkeit beschädigt werden kann

Bedeutung SRP

Single Responsibility Prinzip

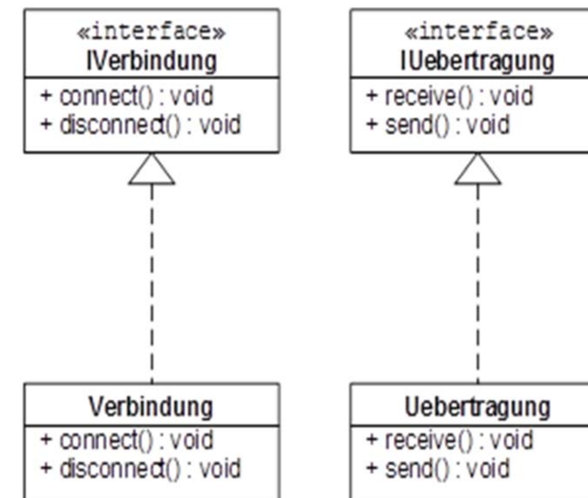
1. Einführung
2. SOLID-Prinzipien
3. Fazit

Verletzung des SRP:



Die Klassen besitzen zwei Verantwortlichkeiten: Verbindung sowie Nachrichtenaustausch.

Anwendung des SRP:



Aufteilung der Verantwortlichkeiten auf verschiedene Klassen.

Beispiel SRP

Open-Closed Prinzip

Das OCP lautet: "Module sollten offen sein und geschlossen."
Es stammt von Bertrand Meyer und fordert:

- Module sollen offen für Erweiterungen sein
- Erweiterungen sollen durch das Hinzufügen von Code durchgeführt werden können
- Gleichzeitig sollen Module geschlossen gegenüber Veränderungen sein, damit sie im Rahmen anderer Architekturen wiederverwendet werden können

Bedeutung OCP

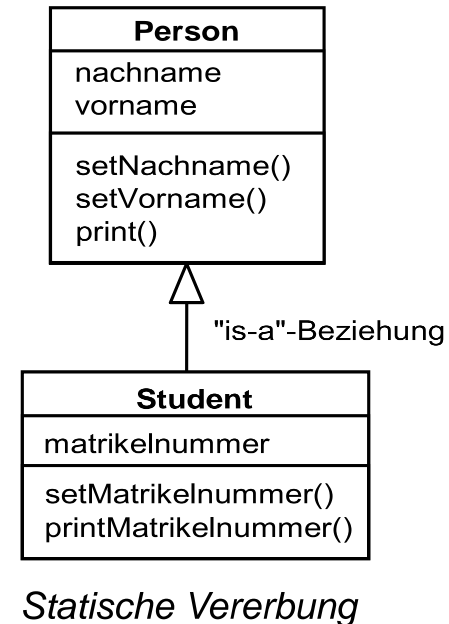
Open-Closed Prinzip

1. Einführung
2. SOLID-Prinzipien
3. Fazit

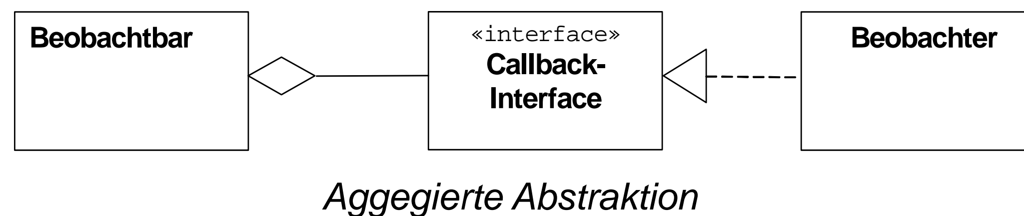
Erweiterungen können in Form von

- statischer Vererbung oder
- aggregierter Abstraktion

erfolgen. Dabei sollte nach dem Prinzip *"Favor composition over inheritance"* von Erich Gamma letzteres favorisiert werden.



Statische Vererbung



Aggegierte Abstraktion

Erweiterung

Liskovsches Substitutionsprinzip

1. Einführung
2. SOLID-Prinzipien
3. Fazit

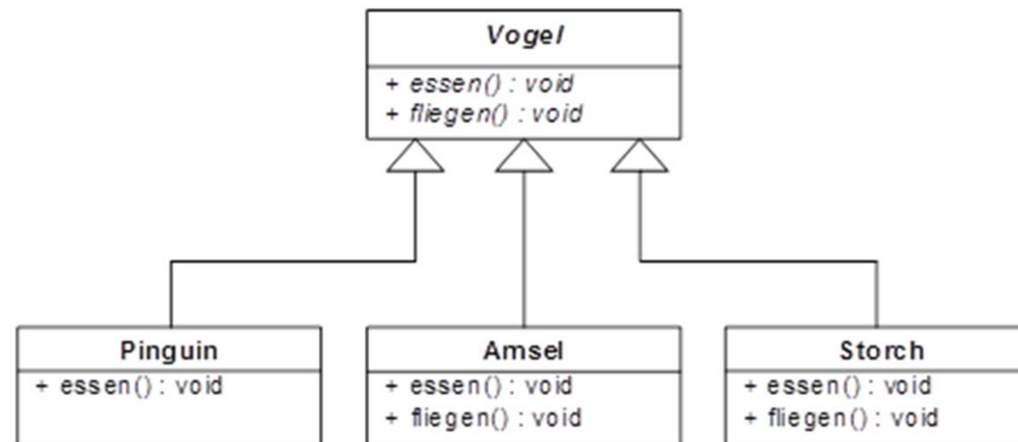
Das LSP von Barbara Liskov formuliert Bedingungen, damit Polymorphie gefahrlos eingesetzt werden kann:

- Ein Objekt einer abgeleiteten Klasse muss an die Stelle eines Objekts seiner Basisklasse treten können, ohne dass ein Client dies merkt
- Vor- und Nachbedingungen müssen eingehalten werden, Klasseninvarianten dürfen nicht gebrochen werden (Design by Contract)

Bedeutung LSP

1. Einführung
2. SOLID-Prinzipien
3. Fazit

Entwurf einer Vererbungshierarchie:



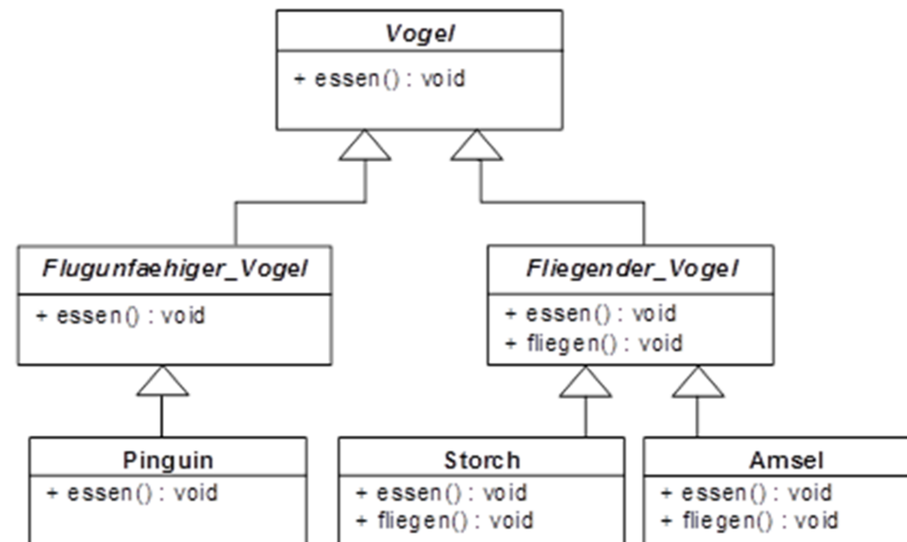
Die Klasse `Pinguin` verstößt gegen das LSP, da die Methode `fliegen()` nicht oder nur mit leerem Methodenrumpf implementiert ist.

Beispiel LSP

Hochschule Esslingen
University of Applied Sciences

1. Einführung
2. SOLID-Prinzipien
3. Fazit

Neue Einteilung:



Jede abgeleitete Klasse kann nun an die Stelle ihrer Basisklasse treten.

Beispiel LSP

Interface Segregation Prinzip

1. Einführung

2. SOLID-
Prinzipien

3. Fazit

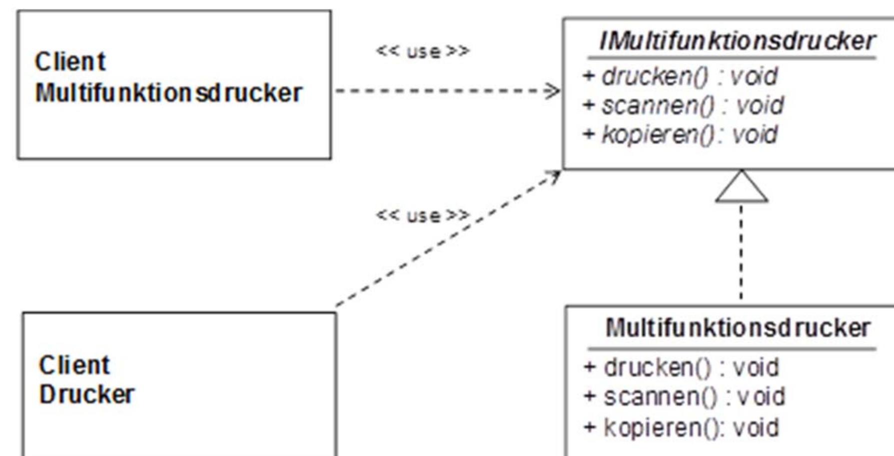
Das ISP "Clients should not be forced to depend upon methods that they do not use" stammt von Robert C. Martin und fordert:

- Interfaces sollen nur Methodenschnittstellen beinhalten, die den Anforderungen eines Clients oder einer Gruppe von Clients genügen
- Änderungen an nicht benötigten Schnittstellen sollen sich nicht auf Clients auswirken, die diese nicht benutzen
- "Fat interfaces" sollen vermieden werden

Bedeutung ISP

1. Einführung
2. SOLID-Prinzipien
3. Fazit

Beispiel Verletzung des ISP:

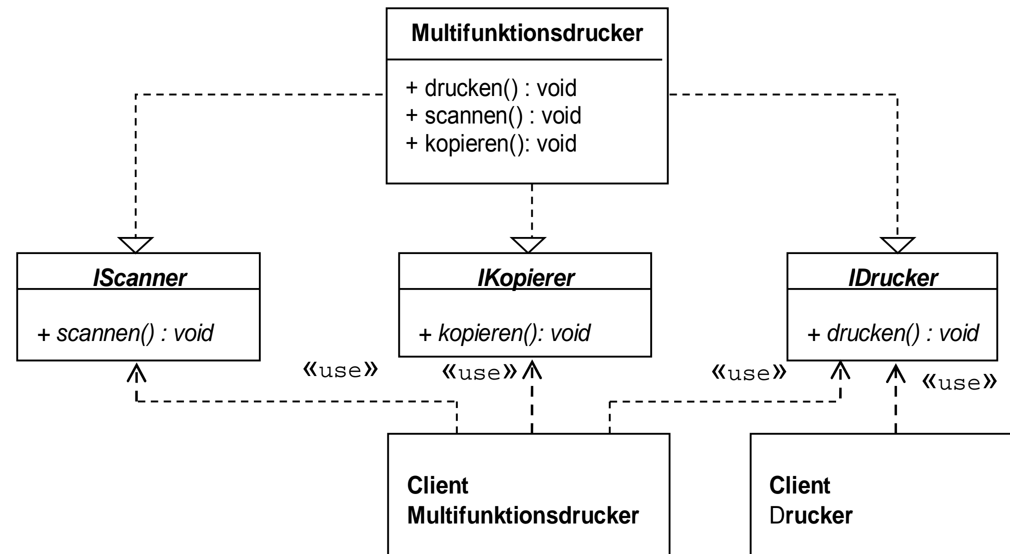


Der Druckerclient ist von Methoden abhängig, die er nicht nutzt. Lösung: Aufteilen des Interface

Beispiel ISP

1. Einführung
2. SOLID-Prinzipien
3. Fazit

Lösung nach ISP:

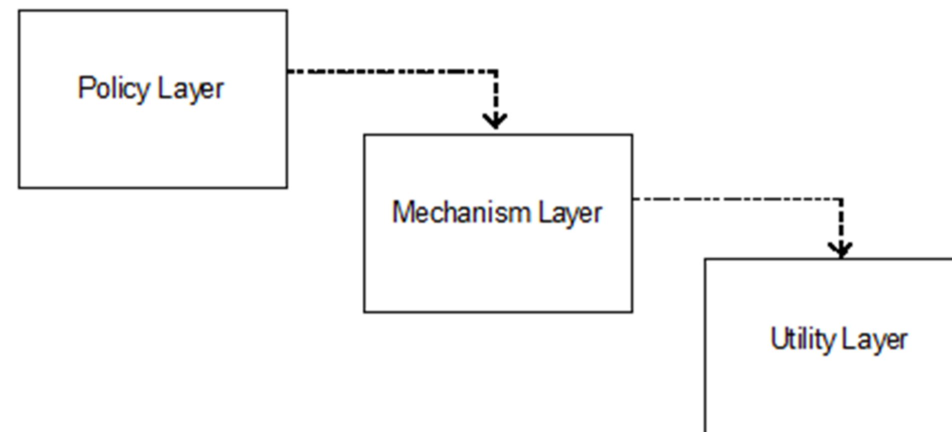


Clients hängen nur von ihren benötigten Schnittstellen ab.

Beispiel ISP

1. Einführung
2. SOLID-Prinzipien
3. Fazit

Problem: Klassisches hierarchisches System nach Grady Booch [Boo95]:



Die Klassen der höheren Ebenen sind jeweils von den Klassen einer darunterliegenden Ebene abhängig.

Problem

Dependency Inversion Prinzip (DIP)

1. Einführung
2. SOLID-Prinzipien
3. Fazit

Das DIP "High-level modules should not depend on low-level modules. Both should depend on abstractions. Abstractions should not depend upon details. Details should depend upon abstractions" stammt von Robert C. Martin und fordert:

- Eine Klasse einer höheren Ebene soll nicht von einer Klasse einer tieferen Ebene abhängig sein
- Hingegen soll eine Klasse einer tieferen Ebene wie auch die Klasse der höheren Ebene von einer Abstraktion abhängen

Bedeutung DIP

Dependency Inversion Prinzip (DIP)

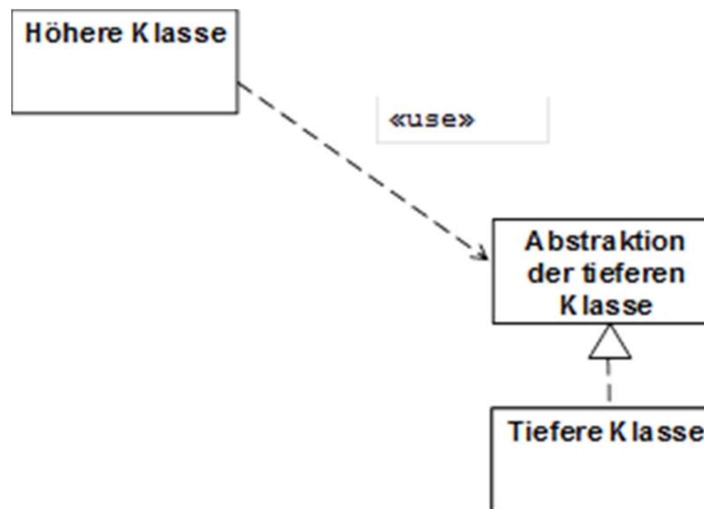
Hochschule Esslingen
University of Applied Sciences

1. Einführung

2. SOLID-Prinzipien

3. Fazit

Einführung einer Abstraktionsschicht:



Statt einer Abhängigkeit der höheren Klassen zur tieferen Klasse sind beide Klassen nur noch von der Abstraktion abhängig. Dies erlaubt es, hinter der Abstraktion beispielsweise Mock-Objekte zu verwenden.

Beispiel DIP

Fazit

1. Einführung
2. SOLID-Prinzipien
3. Fazit

SOLID-Prinzipien sind förderlich für:

- Verlangsamung des Alterungsprozesses durch Reduktion von Abhängigkeiten
- Wartbarkeit
- Erweiterbarkeit
- Korrektheit

Aber:

- Einhalten der Prinzipien erfordert Erfahrung
- Verstoß kommt häufig erst bei auftretenden Problemen zum Vorschein

Durch rechtzeitiges **Refactoring** des Systems können zukünftige Probleme vermieden werden und die **Lebensdauer** eines Software-Systems kann um ein Vielfaches **erhöht** werden.

Einhalten von Prinzipien

Hochschule Esslingen
University of Applied Sciences

DANKE FÜR IHRE AUFMERKSAMKEIT!