DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

KOMPOSITUM

Lars Briem

(briem.lars@googlemail.com)

Duale Hochschule Baden Württemberg - Standort Karlsruhe

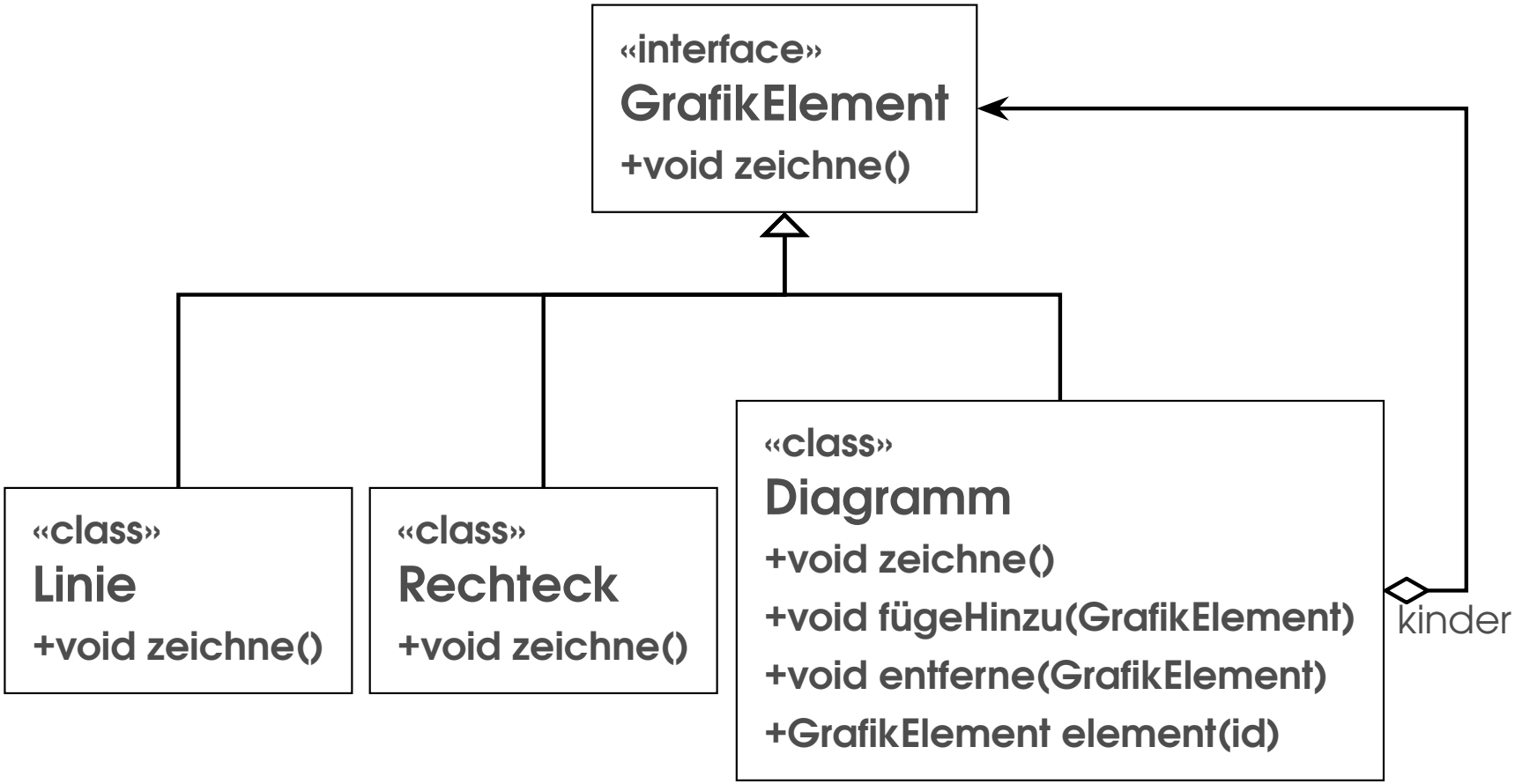
Kompositum

- Setze Objekte zu Baum-Strukturen zusammen, um Teil-Ganzes Hierarchien zu bilden
- Anwender behandeln einzelne Elemente und Komposita gleich
- Klassifikation
  - Objektbasiertes Strukturmuster
  - Datenorientiert
  - Unendliche Rekursion mit Objekten

Kompositum – Motivation

- Kombination einfacher Elemente zur Erzeugung komplexer Strukturen
- Gleichbehandlung von
  - Elementen, die etwas ausführen
  - Containern, die Elemente aufnehmen
- Anwender soll Elemente nicht unterscheiden müssen
  - Implementierungsdetails verbergen

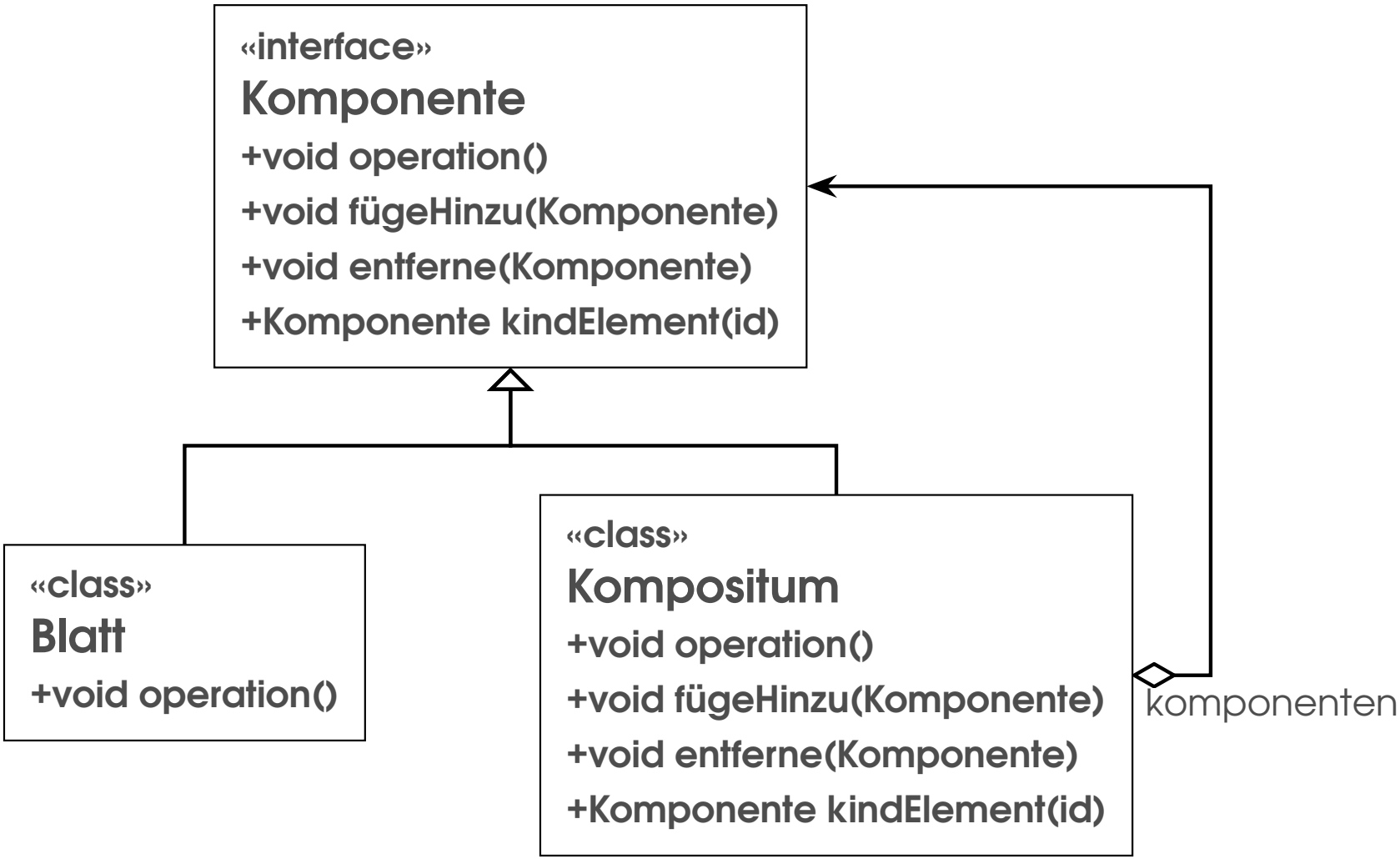
Kompositum – Beispiel



Kompositum – Anwendung

- Teil-Ganzes Hierarchien sollen repräsentiert werden
- Für Anwender soll es egal sein, ob einzelne oder mehrere Elemente vorhanden sind
- Anwender behandeln alle Elemente der Struktur gleich

Kompositum – Struktur



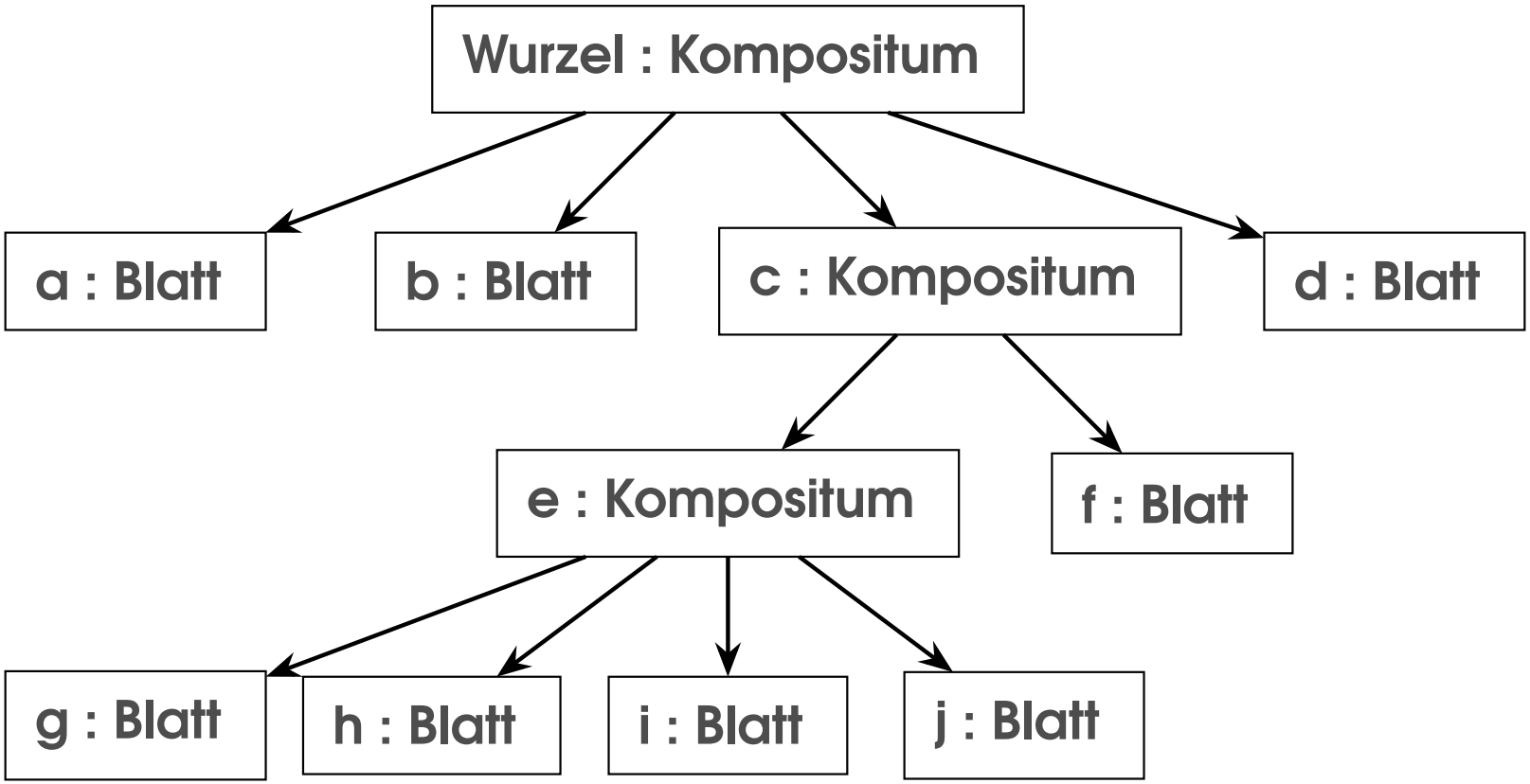
Kompositum – Akteure

- Anwender
  - Manipuliert Objekte im Kompositum nur über Interface
- Komponente
  - Definiert das Interface der Objekte im Kompositum
  - Implementiert Standardverhalten für alle Fälle
  - Definiert Interface zur Verwaltung der Kinder
  - Optional: Definiert und implementiert Schnittstelle für Zugriff auf Eltern, wenn notwendig

Kompositum – Akteure

- Blatt
  - Repräsentiert Blatt-Objekt
  - Hat keine Kinder
  - Definiert Verhalten einfacher Objekte
- Kompositum
  - Definiert Verhalten für Objekte mit Kindern
  - Verwaltet Kinder
  - Implementiert Verhalten bezogen auf Kinder

Kompositum – Interaktion der Akteure



Kompositum – Auswirkungen

+ Einfache Elemente können beliebig zusammengebaut werden
  + Rekursive Verschachtelung notwendig
+ Vereinfacht die Logik beim Anwender
  + Behandelt alle Objekte gleich
+ Neue Komponenten können einfach definiert werden
  + Bestehende arbeiten problemlos mit neuen zusammen

Kompositum – Auswirkungen

- Design zu generell
  - Schwieriger Komponenten einzuschränken
  - Wenn nur bestimmte Elemente erwünscht sind, kann das Typsystem nicht helfen, Überprüfung nur zur Laufzeit

- Referenz zu Eltern kann bei Verarbeitung hilfreich sein
  - Komponente enthält die Logik zur Verwaltung der Elternbeziehung
  - Sorgt für die Einhaltung der Invarianzen

Transparenz vs. Typsicherheit

- Wo soll die Verwaltung der Kinder definiert und implementiert werden?
- Definition in der Wurzel (Komponente)
  - Maximale Transparenz (alle gleich behandeln)
  - Geringere Typsicherheit, weil Anwender sinnlose Aufrufe machen kann
  - Bereitstellung von Standard Add/Remove (Leere Methoden)
    - Anwender erhält keinen Fehler in Blatt, obwohl nichts passiert
    - Anstatt leer, lieber Exception schmeißen bei Add/Remove –> Laufzeitfehler statt Typsicherheit

Transparenz vs. Typsicherheit

- Definition am Kompositum
  - Maximale Typsicherheit
  - Elemente können nur zu "sinnvollen" Klassen hinzugefügt werden
  - Geringere Transparenz, da unterschiedliche Interfaces
  - – Konvertierung zu Kompositum bei Bedarf notwendig (cast)
  - – Verhalten des Anwenders koppelt sich an Kompositum anstatt Komponente

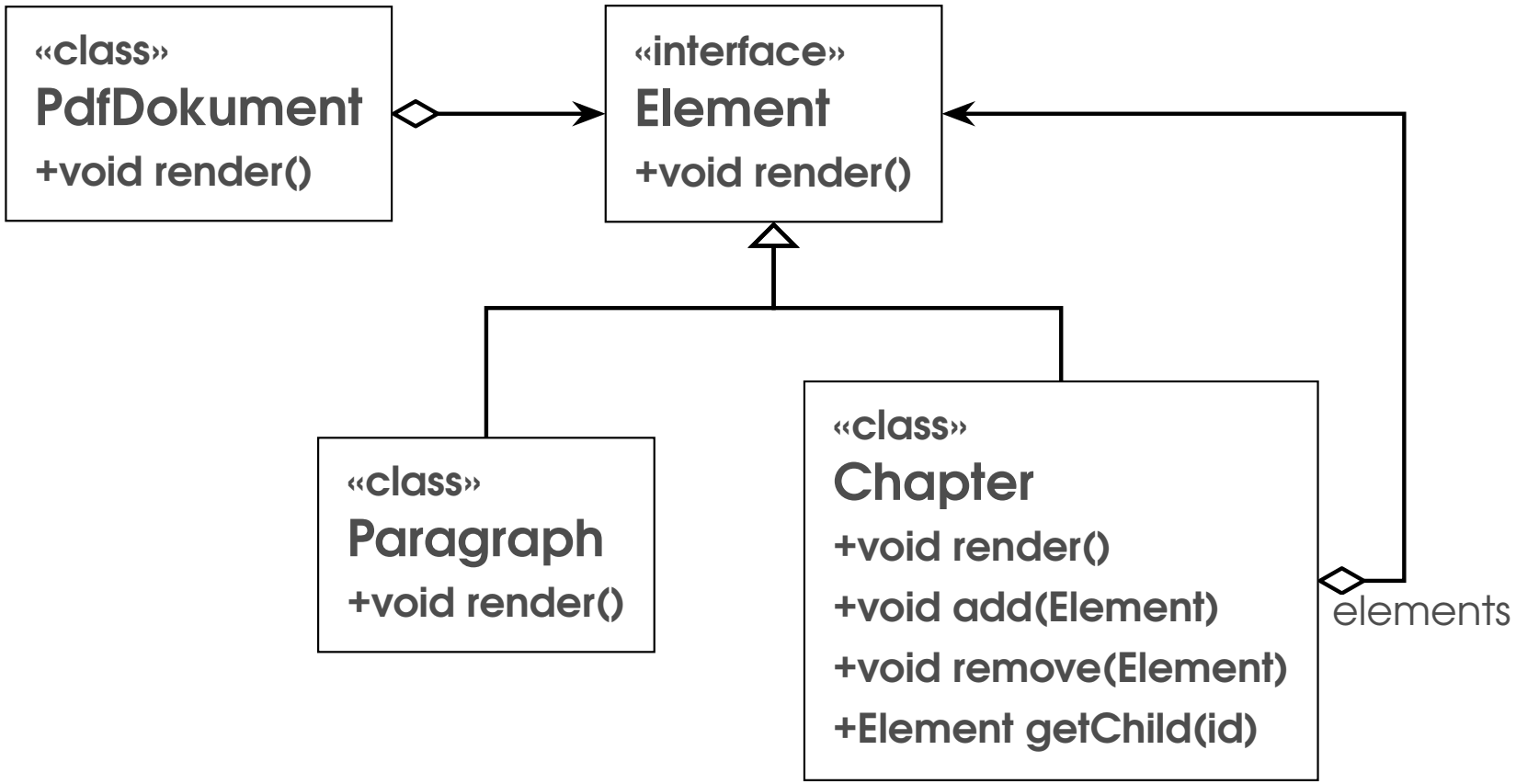
Transparenz vs. Typsicherheit

- Implementierung von Add/Remove in Komponente inklusive Liste der Kinder
  - Overhead für Blatt Elemente
- Alternative
  - Definition einer "getComposite" Methode
  - Bei Kompositum liefert sie dieses zurück
  - Bei Blatt liefert sie `null`

  $\Rightarrow$ `null`-Überprüfung notwendig

  $\Rightarrow$ Besser Optional

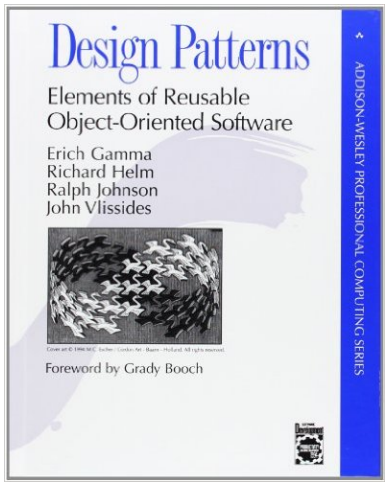
Kompositum – Beispiel - PDF Dokument



Kompositum – Zusammenfassung

- Versteckt einfache und komplexe Objekt-Hierarchien hinter einer Schnittstelle
- Kombiniert einfache Elemente zu komplexen Strukturen
- Anwender kann alle Elemente gleich behandeln

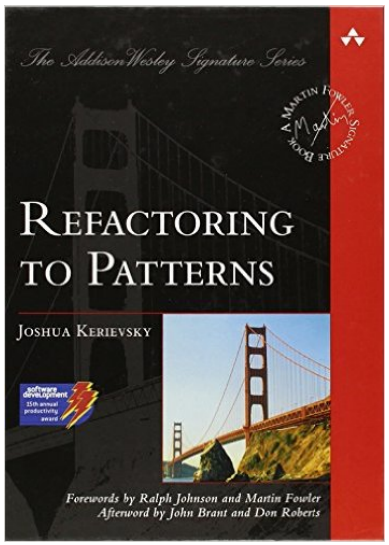
Literatur



- Design Patterns
  - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
  - Addison-Wesley
  - ISBN: 978-0201633610

Weitere Infos

- Entwurfsmuster auf YouTube
  - John Lindquist erklärt Entwurfsmuster mit StarCraft II
  - `https://www.youtube.com/playlist?list=PL8B19C3040F6381A2`



- Refactoring to Patterns
  - Joshua Kerievsky
  - Addison-Weslay
  - ISBN: 978-0321213358