# Programmierkonzepte in der Physikalischen Chemie

## 23. Single-Responsibility-Prinzip

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Dr. Till Biskup

Institut für Physikalische Chemie
Albert-Ludwigs-Universität Freiburg
Wintersemester 2016/17

# Zentrale Aspekte

UNI FREIBURG

- Eine Klasse sollte immer nur einen Grund haben, sich zu ändern.
- Symptome für den Einsatz: Zerbrechlichkeit und Unflexibilität
- Verantwortlichkeiten richtig zu trennen, rührt an den Kern aller Software-Architektur.
- Trennung der Verantwortlichkeiten ist nur dann wichtig, wenn unabhängige Änderungen real auftreten.
- Eines der einfachsten und gleichzeitig am schwersten korrekt umzusetzenden Prinzipien.

# Übersicht

UNI FREIBURG

Das Single-Responsibility-Prinzip

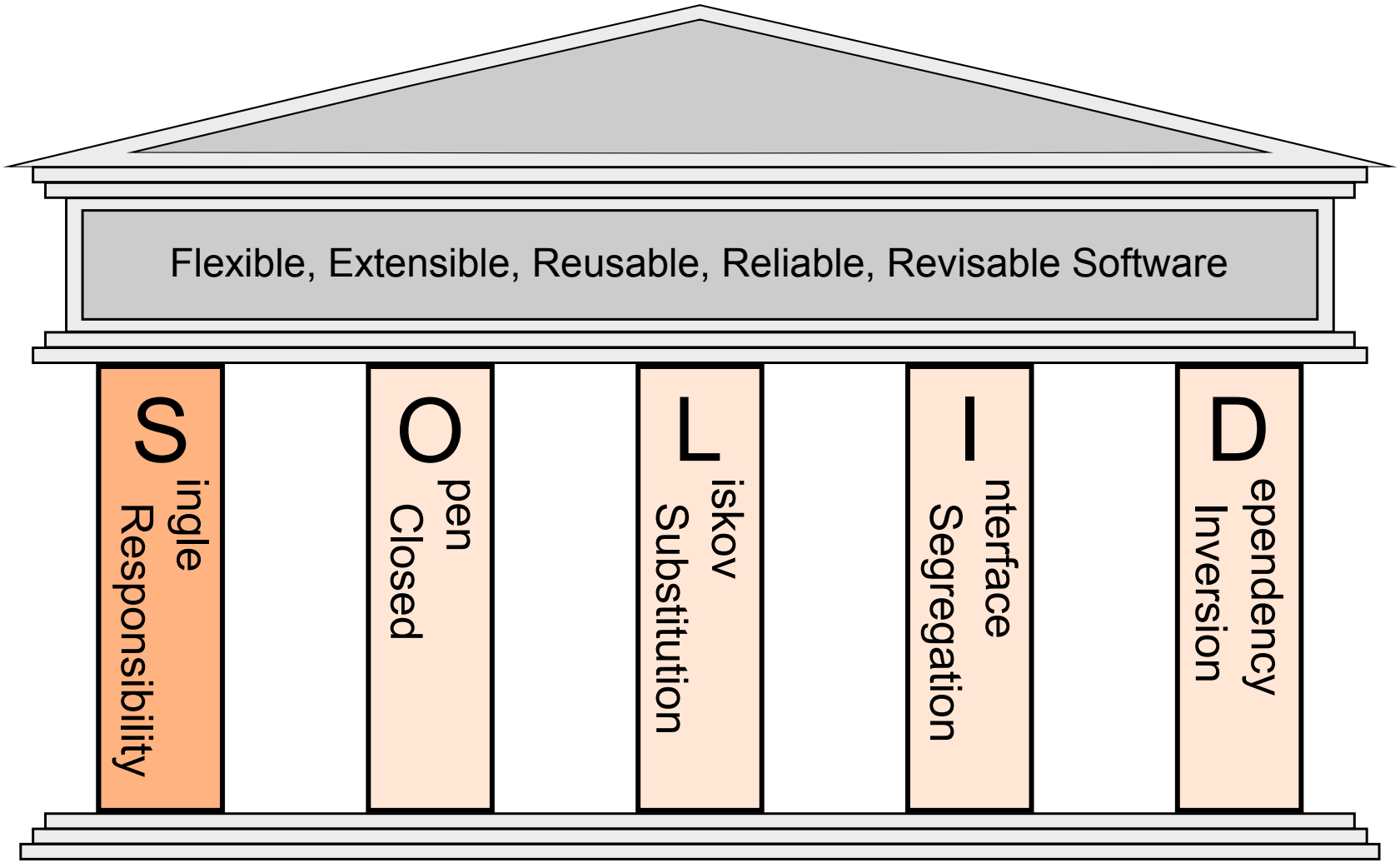
Symptome, die für seinen Einsatz sprechen

Beispiele für seinen Einsatz

Bedeutung im Gesamtkontext der Software-Architektur

# Das Single-Responsibility-Prinzip

## Übersicht über die fünf Prinzipien



# Das Single-Responsibility-Prinzip

## Ein Modul sollte nur eine Verantwortlichkeit haben

UNI FREIBURG

> *A class should have only one reason to change.*
>
> – Robert C. Martin

- Kommt historisch aus dem Kontext der Kohäsion
  - Kohäsion: funktionale Zusammengehörigkeit der Elemente eines Moduls
- hier leicht anderer Blickwinkel
  - Welche Kräfte bringen ein Modul (bzw. eine Klasse) dazu, sich zu verändern?

☛ Eng verwandt mit dem Unix-Prinzip:
Ein Modul sollte genau *eine* Aufgabe erfüllen.

Robert C. Martin: Agile Software Development. Prentice Hall, Upper Saddle River 2003, S. 95

# Was ist eine Verantwortlichkeit?

## Eine kontextspezifische Antwort

UNI FREIBURG

- Im Kontext des SRP: ein Grund für Veränderung
  - Wenn man sich mehr als einen Grund vorstellen kann, dann hat das Modul mehr als eine Verantwortlichkeit.
- Warum sollten Veränderungen gekapselt werden?
  - Voraussetzung für Modularität
  - Veränderungen können sich gegenseitig stören.
  - Verringert die Auswirkungen bei kompilierten Sprachen: Weniger Module müssen neu kompiliert werden.
- Problem mit Verantwortlichkeiten
  - Wir sind gewohnt, Verantwortlichkeiten zu bündeln.
  - „Grund für Veränderung“ ist ein gutes und greifbares Maß.

# Symptome, die für seinen Einsatz sprechen

## Zerbrechlichkeit und Unflexibilität

UNI FREIBURG

### Zerbrechlichkeit (*rigidity*)

Änderungen führen zu Fehlern in Bereichen,
die konzeptionell getrennt von den Änderungen sind.

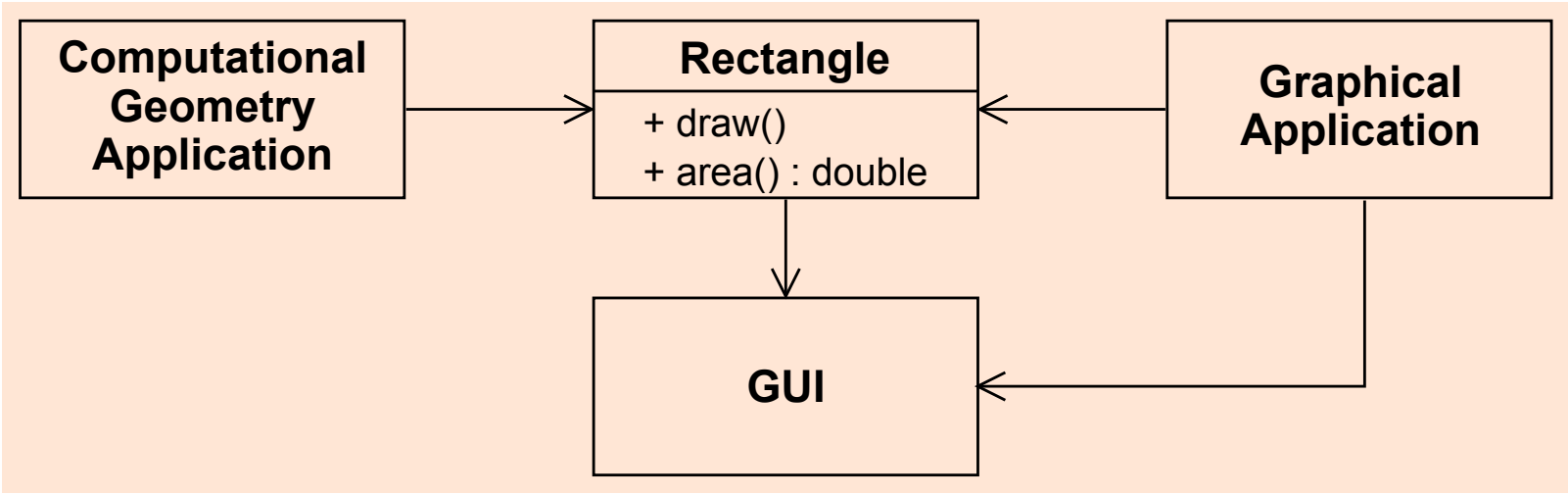
### Unflexibilität (*inflexibility*)

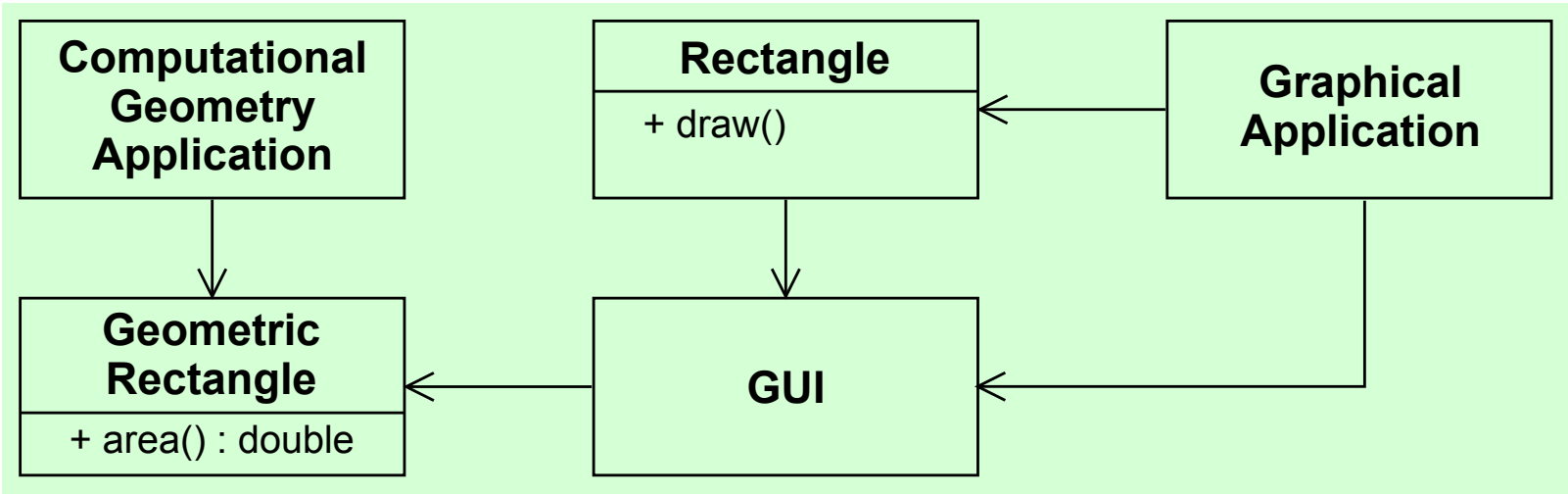
Jede Änderung zieht viele Änderungen
in anderen Teilen des Programms nach sich.

- Zur Erinnerung: Änderungen sind der Regelfall...

# Beispiele für seinen Einsatz

## Ein Rechteck mit zwei Verantwortlichkeiten

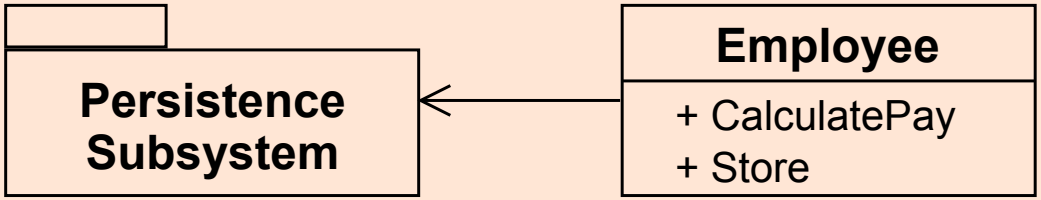



Robert C. Martin: Agile Software Development. Prentice Hall, Upper Saddle River 2003, S. 96

# Beispiele für seinen Einsatz

## Der Klassiker: direkter Zugriff auf die Persistenzschicht




- Die Employee-Klasse hat zwei Verantwortlichkeiten
  - Geschäftsregeln (*business rules*)
  - Persistenz (Speicherung)
- Die Verantwortlichkeiten ändern sich unabhängig
  - Geschäftsregeln ändern sich meist häufiger.
  - Die Ursachen sind grundsätzlich verschieden.
- Mögliche Lösung
  - separate Schichten und Abstraktion
  - Dependency-Inversion-Prinzip

# Bedeutung im Gesamtkontext

## Das einfachste Prinzip – und eines der schwersten

UNI FREIBURG

- eines der einfachsten Prinzipien
  - konzeptionell einfach verständlich
- eines der am schwierigsten richtig anzuwendenden
  - erfordert viel Erfahrung und Überblick
  - Zu viel ist genauso schlecht wie zu wenig.
- Manchmal lassen sich Kopplungen nicht ganz vermeiden.
  - wichtig: alle Abhängigkeiten weisen weg von dieser Klasse

☛ Verantwortlichkeiten korrekt voneinander zu trennen, ist sehr nah am Kern aller Software-Architektur.

☛ Das Konzept wird uns auch bei den anderen Prinzipien immer wieder begegnen.

# Grenzen des Einsatzes

Prinzipien sollten nicht aus Prinzip angewendet werden.

> *An axis of change is an axis of change only if the changes actually occur.*
>
> – Robert C. Martin

- nur anwenden, wenn die Verantwortlichkeiten sich tatsächlich unabhängig voneinander ändern
  - hängt immer vom gegebenen Kontext ab
  - erfordert nötigen Weitblick, Kenntnis und Vertrautheit
- Prinzipien generell immer nur anwenden, wenn das zugehörige Symptom auftritt
  - gilt für alle Prinzipien
  - führt ansonsten oft zu unnötig komplexem Code

Robert C. Martin: Agile Software Development. Prentice Hall, Upper Saddle River 2003, S. 97

# Zentrale Aspekte

- Eine Klasse sollte immer nur einen Grund haben, sich zu ändern.
- Symptome für den Einsatz: Zerbrechlichkeit und Unflexibilität
- Verantwortlichkeiten richtig zu trennen, rührt an den Kern aller Software-Architektur.
- Trennung der Verantwortlichkeiten ist nur dann wichtig, wenn unabhängige Änderungen real auftreten.
- Eines der einfachsten und gleichzeitig am schwersten korrekt umzusetzenden Prinzipien.