Programmierkonzepte in der Physikalischen Chemie

22. Architektur

UNI FREIBURG

Albert-Ludwigs-Universität Freiburg

Dr. Till Biskup

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

Zentrale Aspekte

- Software ist mehr als die Summe der Einzelteile.
- Architektur schlägt die Brücke vom sauberen Code einzelner Module zur eigentlichen Anwendung.
- Intuitive Lösungen widersprechen oft dem großen Ziel flexibler, erweiterbarer und wiederverwendbarer Software.
- Gute Architektur fokussiert auf die Funktionalität, nicht auf konkrete Umsetzungen.
- Für objektorientierte Architektur gibt es eine Reihe bewährter Prinzipien für „saubere Architektur“.

Standortbestimmung

UNI FREIBURG

Grobgliederung der Vorlesung „Programmierkonzepte“

1. Motivation
2. Infrastruktur
3. Code
4. **Architektur**
5. Datenauswertung in der PC

☛ Architektur vermittelt vom Abstraktionsniveau zwischen Code und Anforderungen an das Programm.

Übersicht

UNI FREIBURG

Zur spezifischen Situation in den Naturwissenschaften

Ziel: Flexibler, wiederverwendbarer, wartbarer Code

Symptome für schlechte Software-Architektur

Fünf Prinzipien guter Software-Architektur

Das Problem mit der Literatur

Software-Architektur ist das Feld professioneller Entwickler

UNI FREIBURG

- Fokus auf große Systeme
  - nicht von einzelnen Entwicklern/kleinen Teams zu stemmen
  - erfordern entsprechende Planung und Organisation
- Teams professioneller Entwickler
  - Austausch über das Problem möglich
  - professionell: entsprechende Kenntnis und Beherrschung von Strategien und Werkzeugen
- (relativ) klare, externe Anforderungen
  - Der Kunde hat die Abläufe (hoffentlich) verstanden.
  - Änderungen und Erweiterungen kommen trotzdem vor.

☛ Lässt sich nur schwer auf die Situation im akademischen Kontext übertragen.

Die Situation in den Naturwissenschaften

Softwareentwicklung hat einen zu geringen Stellenwert.

UNI FREIBURG

- Einzelkämpfer
  - Softwareentwicklung ist meist Mittel zum Zweck.
  - Vorteile eines Teams gehen verloren.
- Anforderungen entwickeln sich mit dem Verständnis
  - Das Forschungsgebiet wird i.d.R. erst erarbeitet.
  - Anforderungen entwickeln sich parallel.
  - Eine Anforderungsanalyse vorweg ist selten möglich.
- mangelndes Bewusstsein für den Wert guter Software
  - Software hat einen viel zu geringen Stellenwert.
  - Das Rad wird viel zu oft neu erfunden.
- fehlendes Wissen
  - Entwicklung komplexerer Software ist nichttrivial.

Agile Softwareentwicklung

- Kern
  - unbürokratischer Entwicklungsprozess ohne viele Regeln
  - iterative Vorgehensweise
- Fokus
  - reine Entwurfsphase minimieren
  - schnell zu ausführbarer Software gelangen, die sich in der Praxis bewähren kann

☛ Setzt trotzdem solide Kenntnis von Konzepten und Prinzipien für gute Software voraus.

☛ Mangels Teamgröße nur in Teilaspekten umsetzbar.

Flexibler, wiederverwendbarer, wartbarer Code

Eine Frage der Ökonomie

UNI FREIBURG

- flexibel
  - Anforderungen an Software ändern sich grundsätzlich.
  - Anforderungen entwickeln sich parallel zum Verständnis.
- wiederverwendbar
  - Baukasten: Viele elementare Operationen werden immer wieder in unterschiedlichem Kontext benötigt.
  - Modularität und saubere Schnittstellen sind Trumpf.
- wartbar
  - Software sollte ihren Erstentwickler überdauern können.
  - stellt Ansprüche an Softwarequalität und Wissenstransfer

☛ Gleiche Anforderungen wie an „Clean Code“ – nur auf einem höheren Abstraktionsniveau

Grundsätzliche Strategien

Weitestgehende Entkopplung auf allen Ebenen

UNI FREIBURG

- Trennung der Zuständigkeiten (*Separation of Concerns*)
  - grundlegendes Prinzip auf vielen Ebenen
  - Bsp.: Funktionen sollten genau eine Aufgabe erfüllen.
  - Kapselung und damit „Verstecken“ von Information
  - Schichten eines komplexeren Programms (s.u.)
  - Führt zu Modularität und Austauschbarkeit von Codeteilen.
- „Gesetz von Demeter“
  - Objekte sollten nur mit Objekten in ihrer unmittelbaren Umgebung kommunizieren.
  - Verringert die Abhängigkeiten und erhöht die Wartbarkeit.
  - als Regel für objektorientierte Programmierung entwickelt

☛ Änderungen eines Aspekts sollten lokal klar begrenzte Auswirkungen auf den Code des Gesamtprojekts haben.

Modularität führt zu mehreren Schichten

Schichten sollten so unabhängig wie möglich voneinander sein.

UNI FREIBURG

- normalerweise (mindestens) drei Schichten
  - Präsentation
  - Verarbeitung
  - Daten
- je nach Komplexitätsgrad weitere Schichten
  - Zwischenschichten zwischen den oben genannten
  - Persistenzschicht für die Daten (Speicherung)
- Die Schichten sollten so unabhängig wie möglich sein.
  - hilft bei Austauschbarkeit und Wiederverwendbarkeit
  - Änderungen sollten immer nur eine Schicht betreffen.

☛ Kontrollfluss und Abhängigkeiten zeigen nicht zwingend in die gleiche Richtung.

Fokus auf Funktionalität

Entscheidend sind Funktion und Verhalten, nicht Implementierung

UNI FREIBURG

- Die Verarbeitungsschicht sollte im Zentrum stehen.
  - Weder Präsentation noch Datenschicht sind zentral.
  - Präsentation und Datenschicht sollten austauschbar sein.
- Abhängigkeiten und Kontrollfluss trennen
  - Eine Präsentation muss letztlich auf Daten zugreifen.
  - Eine Präsentation sollte deshalb nicht (direkt oder indirekt) von der Datenschicht abhängen.
  - Trennung erlaubt hochgradige Modularität.
  - Strategien werden später separat vorgestellt.

☛ Dieser Perspektivenwechsel führt mitunter zu überraschenden Ergebnissen – und eleganter Architektur.

Symptome für schlechte Software-Architektur

Hinweise auf Handlungsbedarf für bessere Qualität

UNI FREIBURG

- Unflexibilität (*rigidity*)
  - Jede Änderung zieht viele Änderungen in anderen Teilen des Programms nach sich.
- Zerbrechlichkeit (*fragility*)
  - Änderungen führen zu Fehlern in Bereichen, die konzeptionell getrennt von den Änderungen sind.
- Unbeweglichkeit (*immobility*)
  - Das System lässt sich schwer in wiederverwendbare Komponenten aufteilen.
- Zähigkeit (*viscosity*)
  - Dinge richtig zu machen ist schwieriger, als sie falsch zu machen.

Symptome für schlechte Software-Architektur

Hinweise auf Handlungsbedarf für bessere Qualität

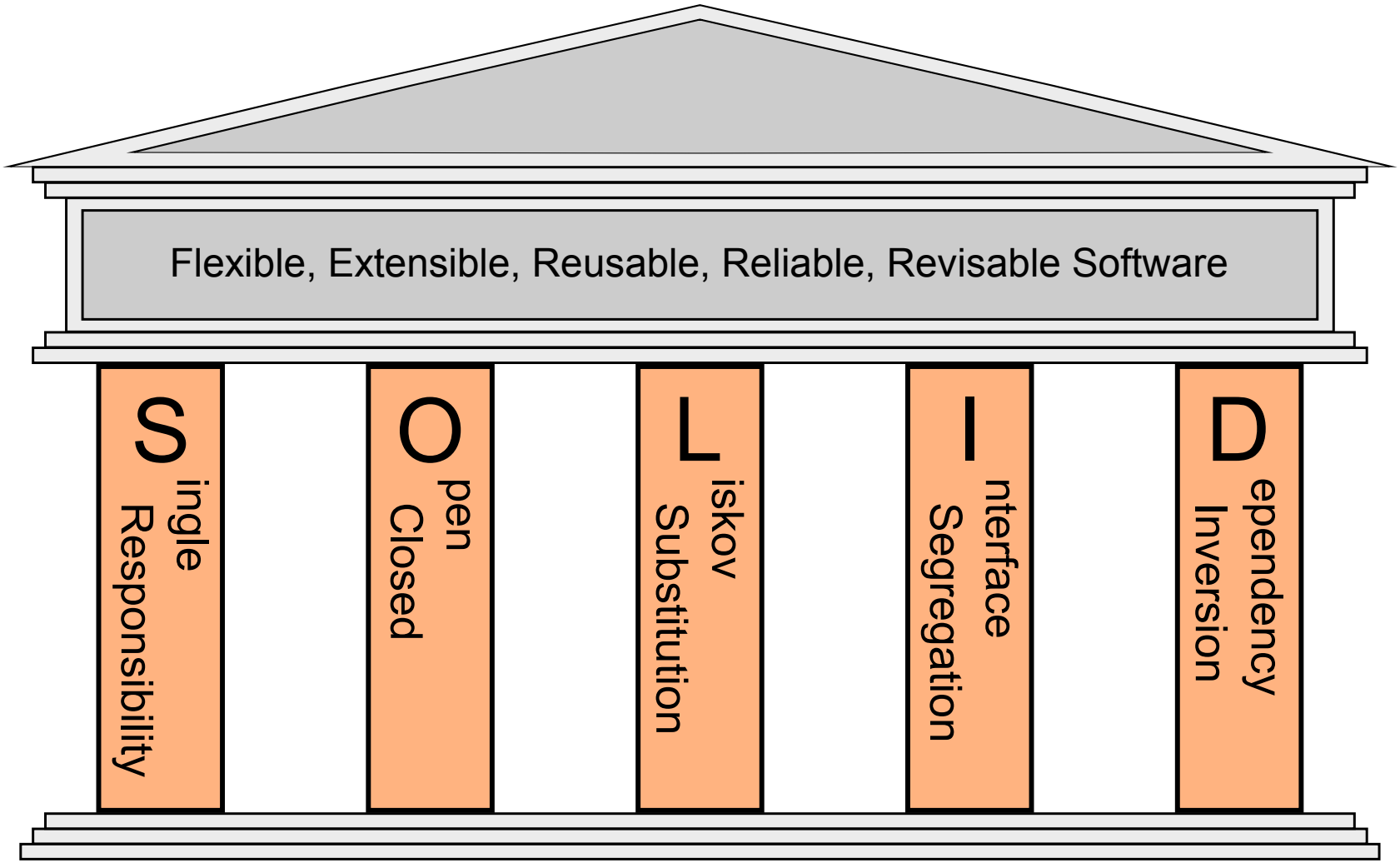
UNI FREIBURG

- unnötige Komplexität (*needless complexity*)
  - Der Entwurf enthält Infrastruktur, die keinen unmittelbaren Mehrwert (in der aktuellen Situation) bringt.
- unnötige Wiederholung (*needless repetition*)
  - Der Entwurf enthält Wiederholungen, die mit Hilfe einer Abstraktion zusammengefasst werden könnten.
- Intransparenz (*opacity*)
  - Der Entwurf ist schwer zu lesen und zu verstehen. Er drückt seine Absicht nicht klar aus.

☛ Das Auftreten der Symptome ist normal.
Entscheidend ist, etwas dagegen zu unternehmen.

Fünf Prinzipien guter Software-Architektur

Ein erster Überblick



Das Ergebnis jahrzehntelanger Erfahrung

Voraussetzung: objektorientierte Programmierung

UNI FREIBURG

- in der Praxis bewährt
  - Ergebnis jahrzehntelanger Erfahrung von Profis
  - nicht nur die Idee eines Autors
  - Die Prinzipien sind älter als ihr Name.
- hier objektorientiert vorgestellt
  - Objektorientierung erleichtert die Umsetzung.
  - Die Prinzipien sind darüber hinaus gültig.
- Prinzipien sind erst einmal abstrakt.
  - Anwendung auf die konkrete Situation erfordert Bewusstsein, Kenntnis und viel Erfahrung.

☛ Diese Prinzipien sind kein Heilsversprechen, aber hilfreich für qualitativ hochwertige Programme.

Prinzipien sind kein Selbstzweck

Voreilige Verwendung führt zu unnötig komplexem Code.

UNI FREIBURG

> *It is a mistake to unconditionally conform to a principle just because it is a principle.*
>
> – Robert C. Martin

- Fokus auf funktionierender, existierender Software
  - bestmögliche Lösung für bestehende Anforderungen
  - Nur existierende Software kann sich bewähren...
- Prinzipien erst anwenden, wenn notwendig
  - Voreilige Verwendung führt zu unnötig komplexem Code.
  - Kenntnis der Prinzipien und Vertrautheit mit ihrer Umsetzung sind zwingende Voraussetzung.

Robert C. Martin: Agile Software Development. Prentice Hall, Upper Saddle River 2003, S. 86

Zentrale Aspekte

UNI FREIBURG

- Software ist mehr als die Summe der Einzelteile.
- Architektur schlägt die Brücke vom sauberen Code einzelner Module zur eigentlichen Anwendung.
- Intuitive Lösungen widersprechen oft dem großen Ziel flexibler, erweiterbarer und wiederverwendbarer Software.
- Gute Architektur fokussiert auf die Funktionalität, nicht auf konkrete Umsetzungen.
- Für objektorientierte Architektur gibt es eine Reihe bewährter Prinzipien für „saubere Architektur“.