Architektur, Analyse und Design

Wie man Software entwirft

Christian Rehn

TU Kaiserslautern

4. Mai 2012

Organisatorisches

- Folien enthalten nur wenig Text
  - Besser für Präsentation
  - Als Handout/Skript steht eine ausführlichere Version online
- Inhalte wichtig für Projekt und prüfungsrelevant!
- Die Saalübung heute ist eher vorlesungsartig
  - Geplant: ca. 60 min. „Vorlesung“ + ca. 30 min. Aufgaben
  - Trotzdem auch im ersten Teil Fragen und aktive Mitarbeit wichtig
  - Feedback erwünscht

Was ich hier erzähle

- SE2: Wie man Analyse- und Entwurfsmodelle *aufschreibt*
  - Notation: UML
- Hier: Wie man zu diesem Modellen kommt
  - Denkweise, Prinzipien, Daumenregeln, hilfreiches Wissen, ...

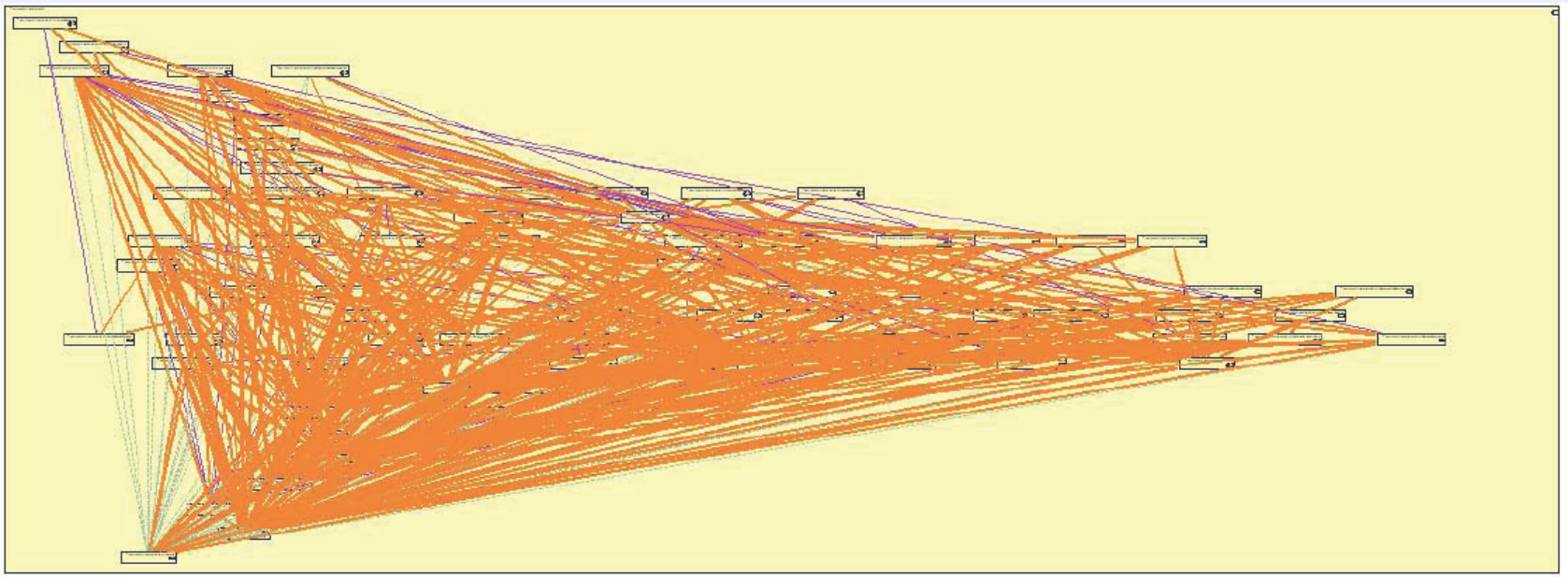
Überblick

Motivation und Begriffe

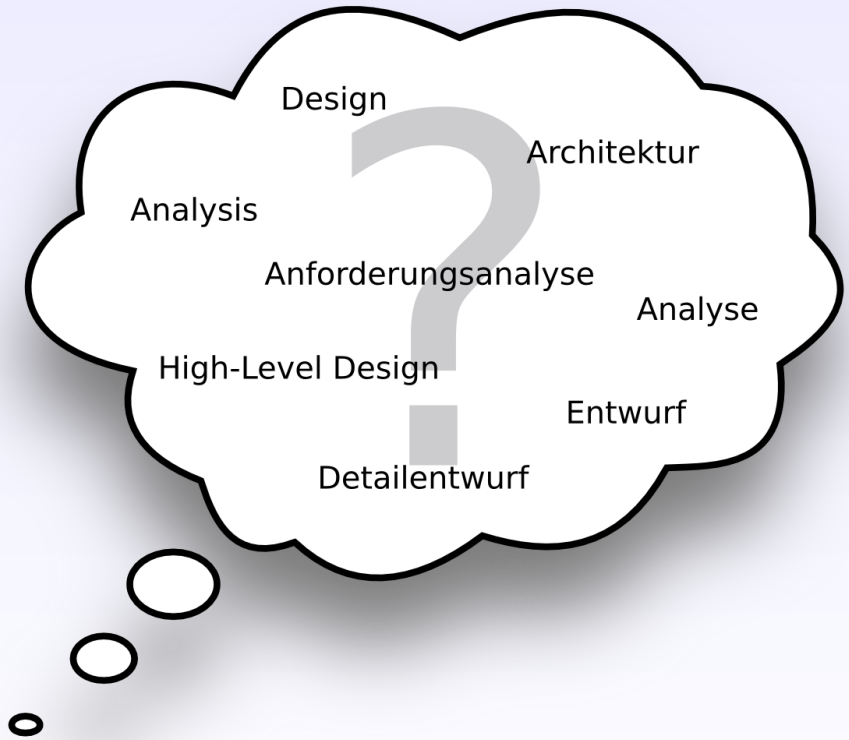
Warum überhaupt nachdenken? (1/2)

Warum sollten wir überhaupt nachdenken, bevor wir programmieren?

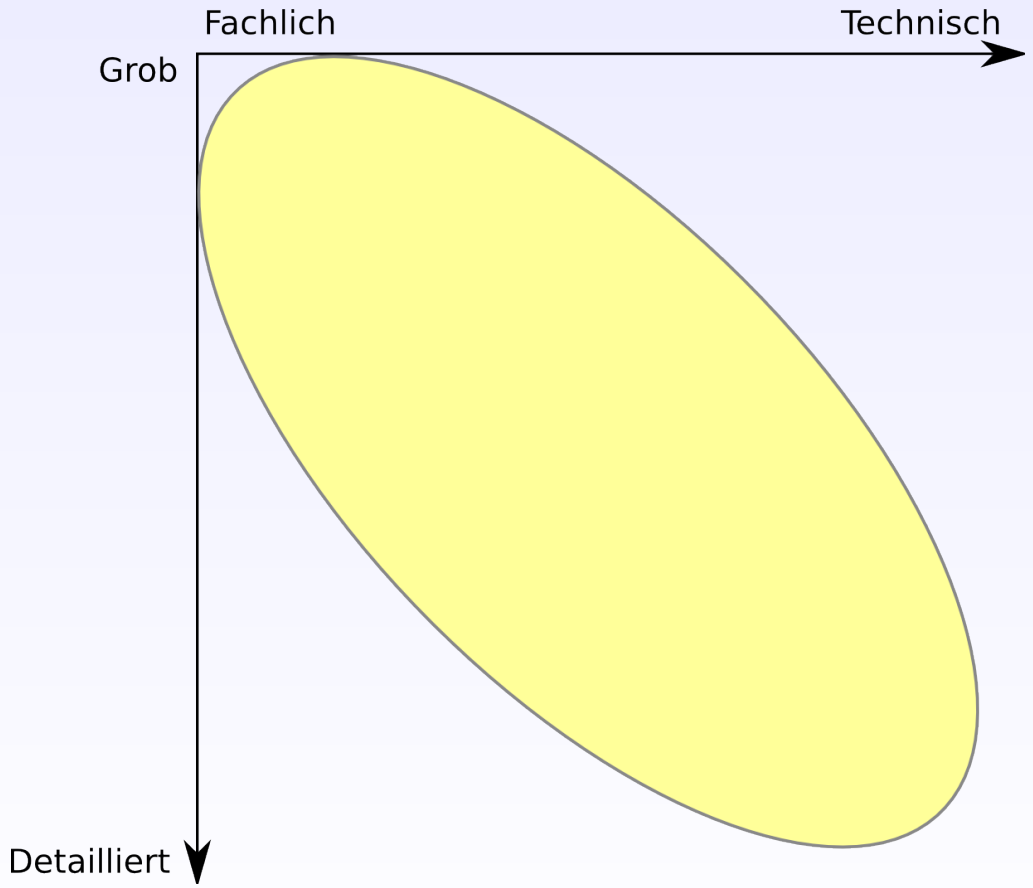
Warum überhaupt nachdenken? (2/2)



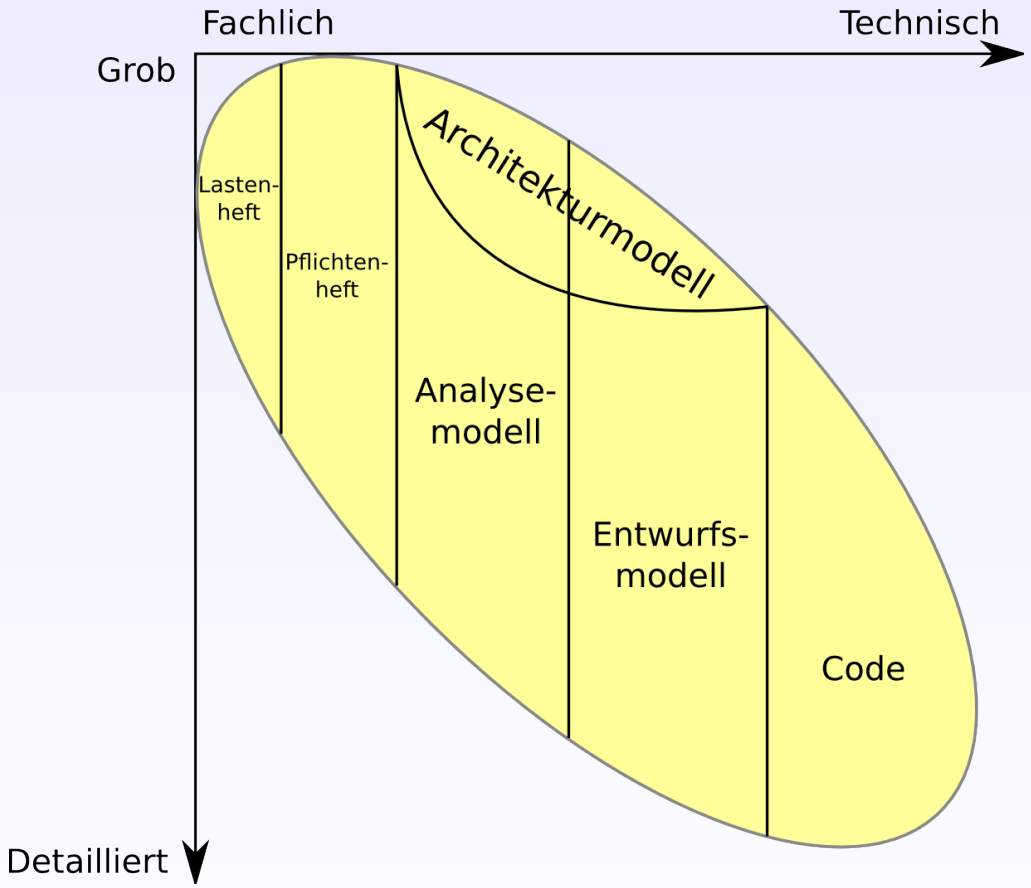
Begriffsverwirrung



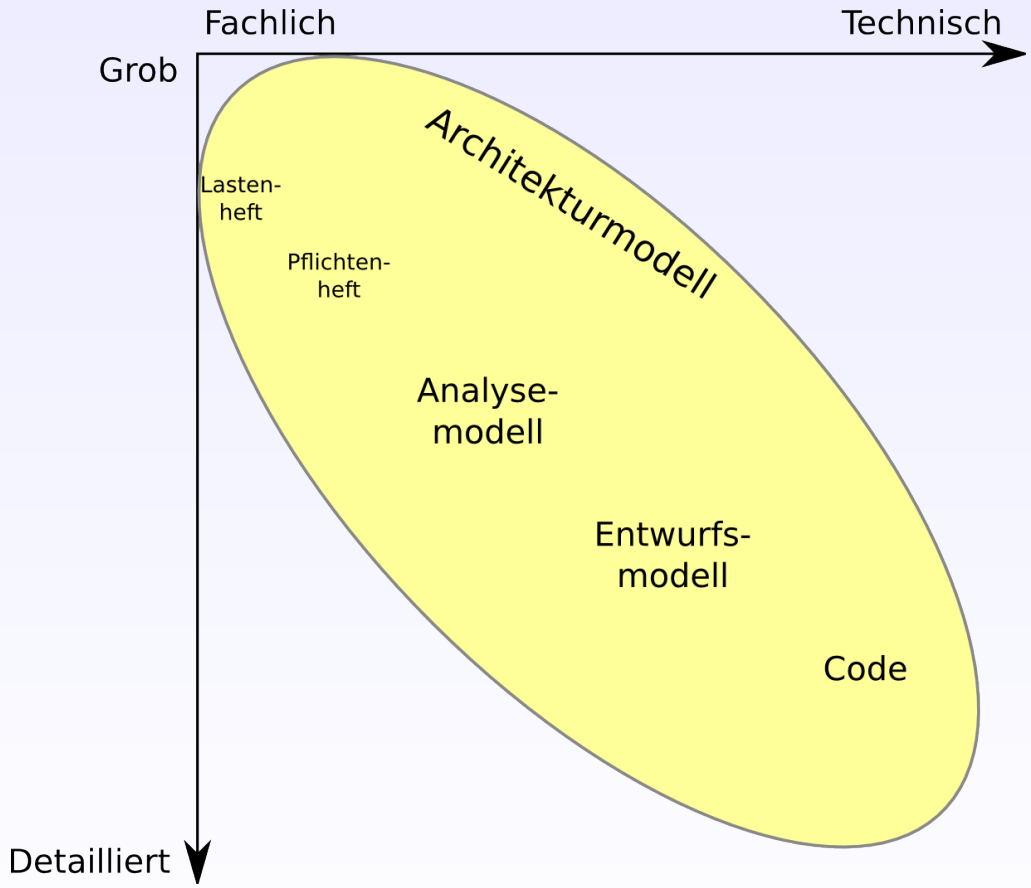
Das „Softwareentwicklungs-Ei“



Das „Softwareentwicklungs-Ei“



Das „Softwareentwicklungs-Ei“



OOA und OOD

natürliche vs. künstliche Klassen

Architektur

Architektur: Definition des SEI

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. [BCK03]

Architektur: Meine Definition

Die Softwarearchitektur beschreibt die groben Strukturen der Software und definiert wie man als Entwickler von der Software *denkt*.

Architektur

Architektur: Definition des SEI

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. [BCK03]

Architektur: Meine Definition

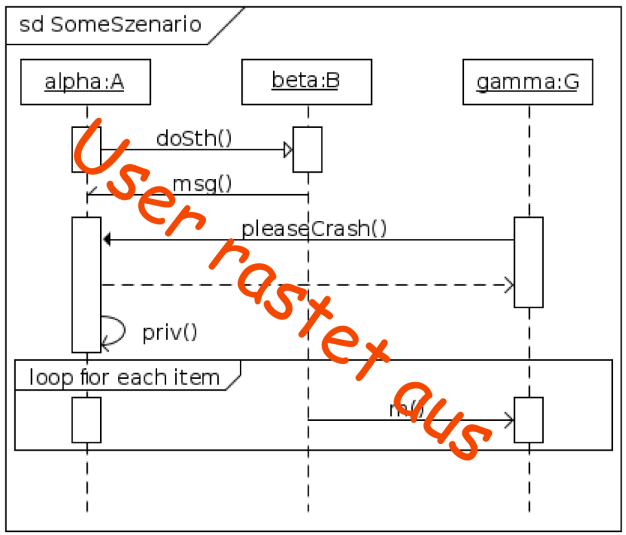
Die Softwarearchitektur beschreibt die groben Strukturen der Software und definiert wie man als Entwickler von der Software *denkt*.

Szenarien und Daumenregeln

Szenarien

Szenarien für ...

- Funktionalität
- Nicht-funktionale Anforderungen
- Softwareentwicklung und -wartung



Nutzen von Szenarien

- Identifizieren von Klassen, Methoden, etc.
- Auffinden von Problemen
- Vergleich unterschiedlicher Ansätze

The Tradeoff Game

Wie ich Softwareentwicklung sehe

Softwareentwicklung ist das ständige Ausbalancieren diverser Prinzipien („Daumenregeln“).

Meine obersten drei Daumenregeln

1. Brich die Regeln
2. Die Niemals-heißt-nie-nie-Regel
3. Die Antwort auf alle Fragen: it depends

Technical Debt

Grundlegende Daumenregeln

- Divide and Conquer
- Keep it Simple, Stupid (KISS)
- Abstraktionsprinzip
- ...

More is More Complex

[1]

[1] CC-BY Dezidor http://commons.wikimedia.org/wiki/File:Ravioly.jpg

Don't Repeat Yourself

DRY

~~sag nicht alles doppelt~~

~~wiederhol' dich doch nicht immer so~~

Modellprinzip

Objekte sollten vorrangig etwas *sein* und nur „nebenbei“ etwas *tun*.

Single Responsibility Principle (SRP)

Eine Modul sollte genau *eine* Aufgabe haben

Kapselung und Information Hiding

Objekte sind wie Abwasserleitungen

Encapsulate the Concept that Varies

Encapsulate the Concept that Varies

Überlege bei der Modellierung, was sich ändert oder wahrscheinlich ändern wird. Das sollte dann in einem separaten Modul gekapselt werden.

Speculative Design

Was wäre wenn. . .

Tell, don't Ask! oder: Do It Myself

Objekt: „Das kann ich alleine!“

Tell, don't Ask! – Gegenbeispiele

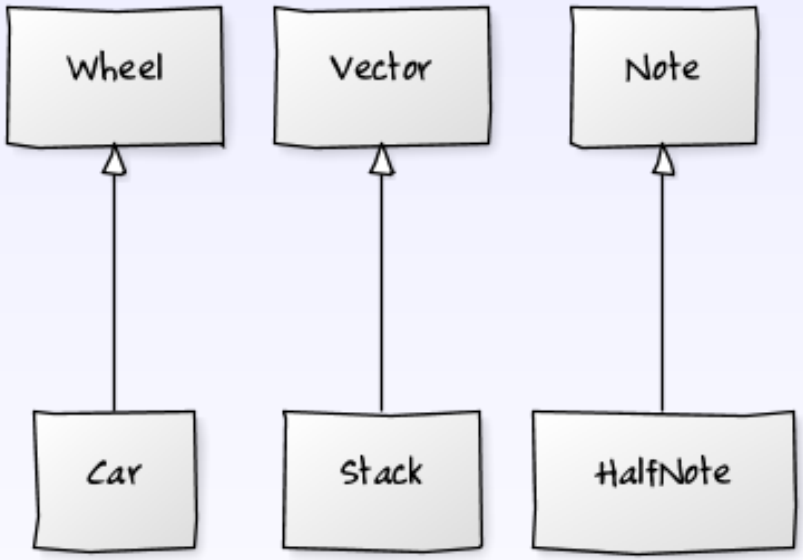
```
// ganz typisch :
myObject.setValue(myObject.getValue() + 1);

// oder auch:
if (myTime.getMinute() == 59)
{
    myTime.setMinute(0);
    myTime.setHour(myTime.getHour() + 1);
}
else
{
    myTime.setMinute(myTime.getMinute() + 1);
}
```

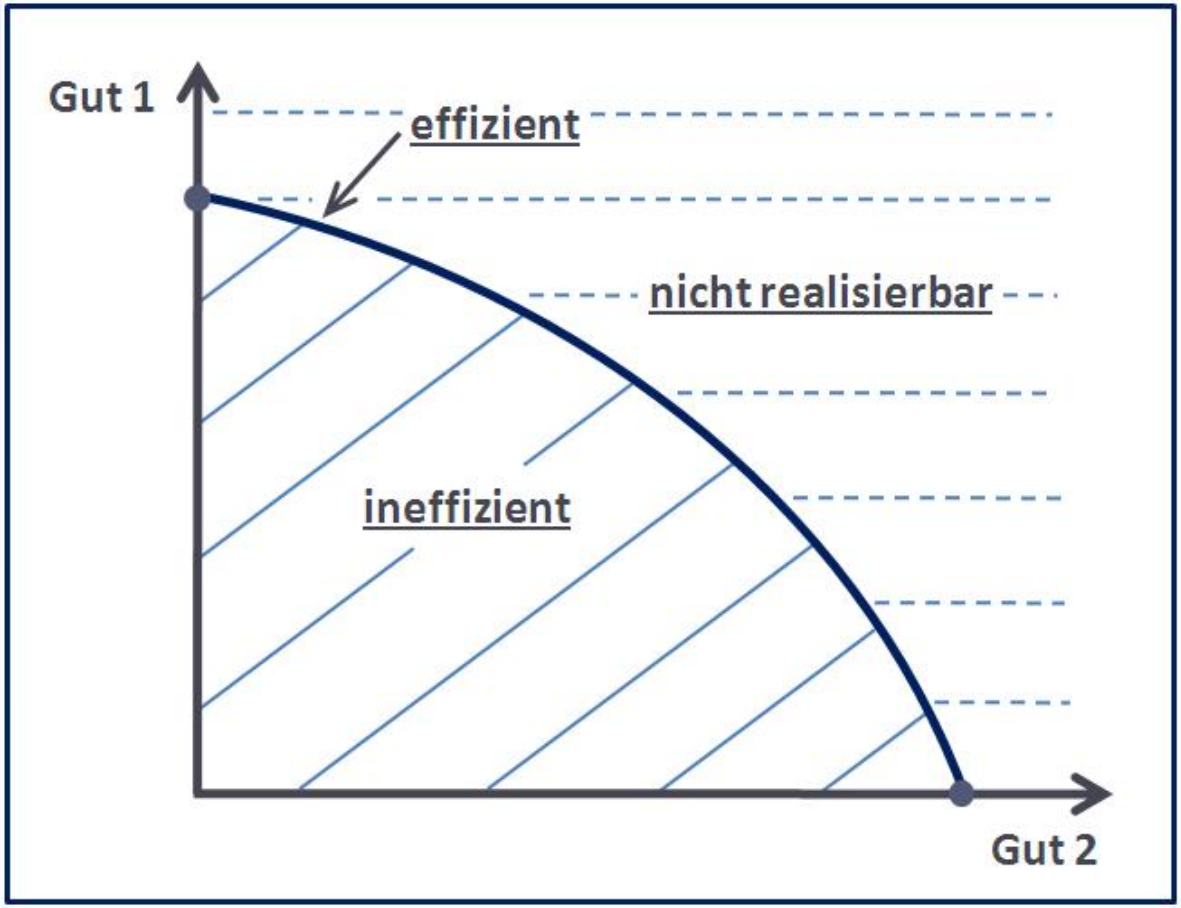
Delegation vor Vererbung

Vererbung wird oft überverwendet

Vererbung – Negativbeispiele



Prinzipien nutzen

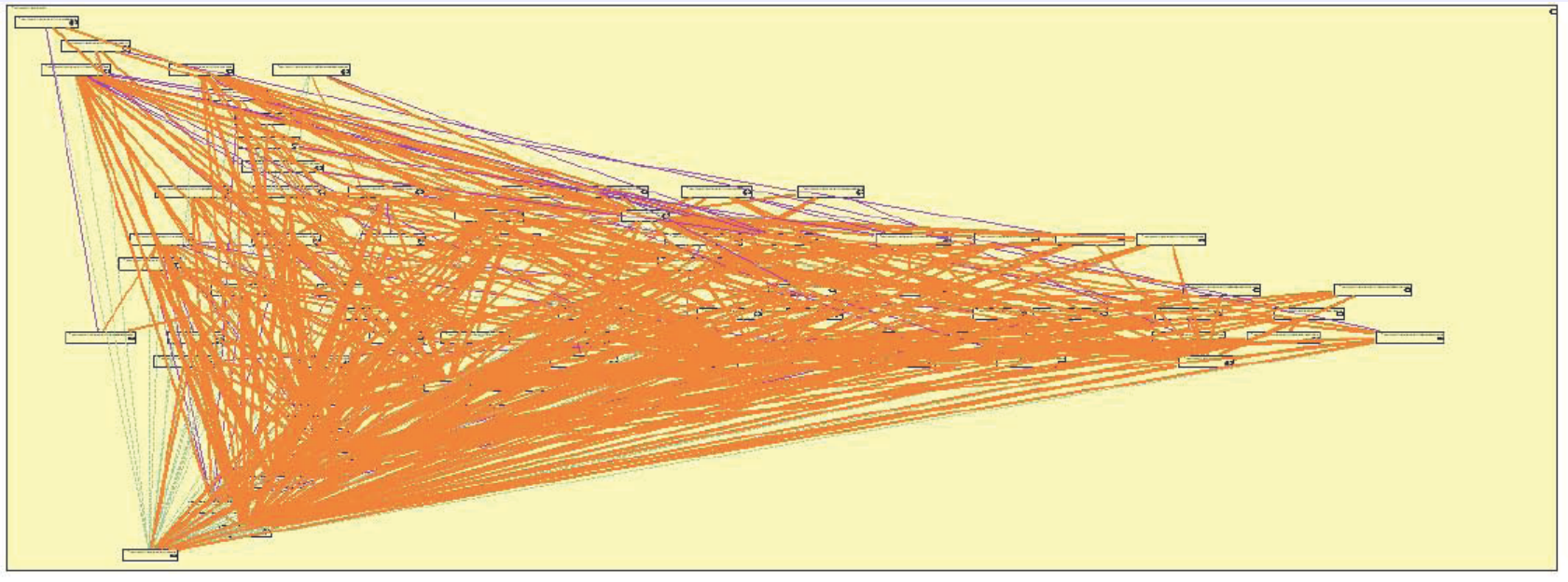


Abhängigkeiten und Schichten

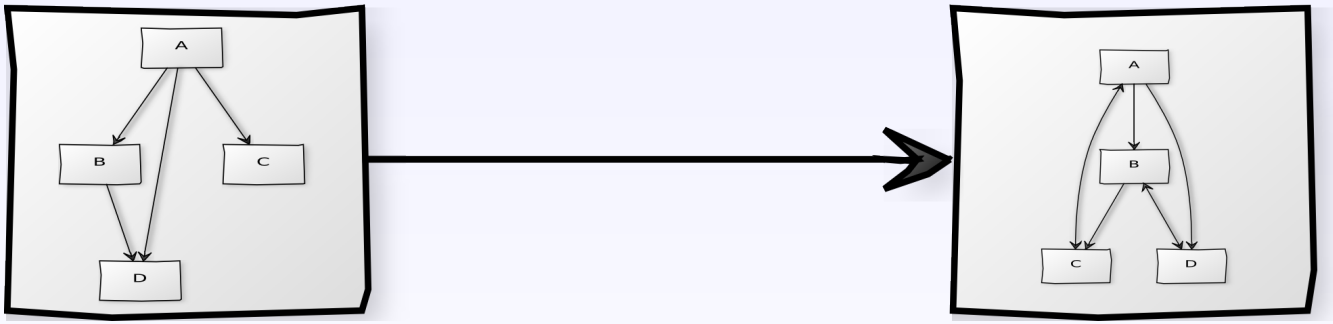
Ripple Effects

2

[2] CC-BY-SA Rainer Zenz http://commons.wikimedia.org/wiki/File:2006-01-14_Surface_waves-2.jpg



Bindung und Kopplung



Kopplungsarten

Kopplungsarten (von stark bis schwach)

1. Content coupling (Inhaltskopplung)
2. Common coupling (Bereichskopplung)
3. External coupling (Externdatenkopplung)
4. Control coupling (Kontrollkopplung)
5. Stamp coupling (Datenstrukturkopplung)
6. Data coupling (Datenkopplung)
7. Call coupling (Aufrufkopplung)

Kopplungsarten

Kopplungsarten (von stark bis schwach)

1. Content coupling (Inhaltskopplung)
2. Common coupling (Bereichskopplung)
3. External coupling (Externdatenkopplung)
4. Control coupling (Kontrollkopplung)
5. Stamp coupling (Datenstrukturkopplung)
6. Data coupling (Datenkopplung)
7. Call coupling (Aufrufkopplung)

Content Coupling (Inhaltskopplung)

Beispiel:

```
MyFancyOpenDialog dialog = new MyFancyOpenDialog();
dialog.setVisible(true);   // modal

if (dialog.getTextField().getText() != "")
{
    System.out.println("Die folgende Datei wurde ausgewählt: " + dialog.getTextField().getText());
}
```

Control Coupling (Kontrollkopplung)

Beispiel:

```
class MyFeedReader
{
    void deleteItem(FeedItem item, boolean recycle);
    {
        if (recycle)
            // throw in dust bin
        else
            // delete completely
    }
}
```

Die „guten“ Kopplungen

Stamp coupling (Datenstrukturkopplung)

Funktionsaufruf mit Übergabe einer komplexen Datenstruktur als Parameter

Data coupling (Datenkopplung)

Funktionsaufruf mit Übergabe von einfachen Parametern (`String`, `int`, etc.)

Call coupling (Aufrufkopplung)

Funktionsaufruf ohne Parameter

Kopplungsarten – Sonderformen

Sonderformen

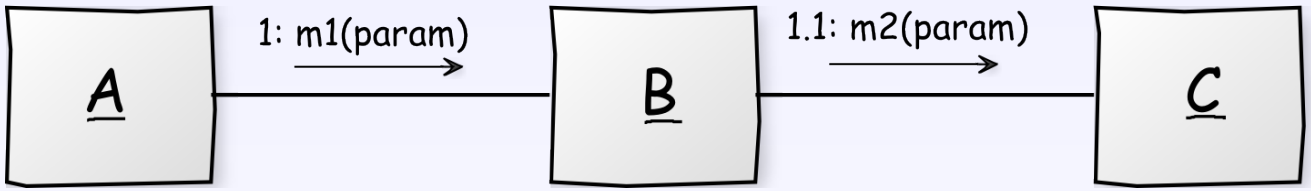
- Tramp coupling
- Hybrid coupling (Hybridkopplung)
- Temporal coupling (zeitliche Kopplung)
- Logical coupling (logische Kopplung)
- Cyclic depepndency (zyklische Abhängigkeit)

Kopplungsarten – Sonderformen

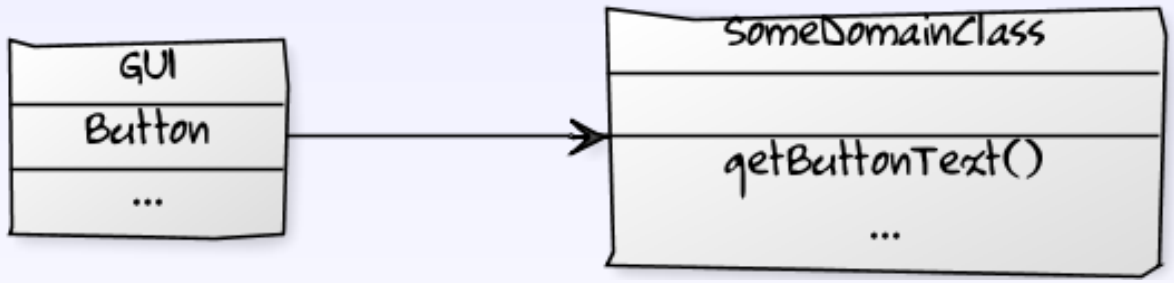
Sonderformen

- Tramp coupling
- Hybrid coupling (Hybridkopplung)
- Temporal coupling (zeitliche Kopplung)
- Logical coupling (logische Kopplung)
- Cyclic depepndency (zyklische Abhängigkeit)

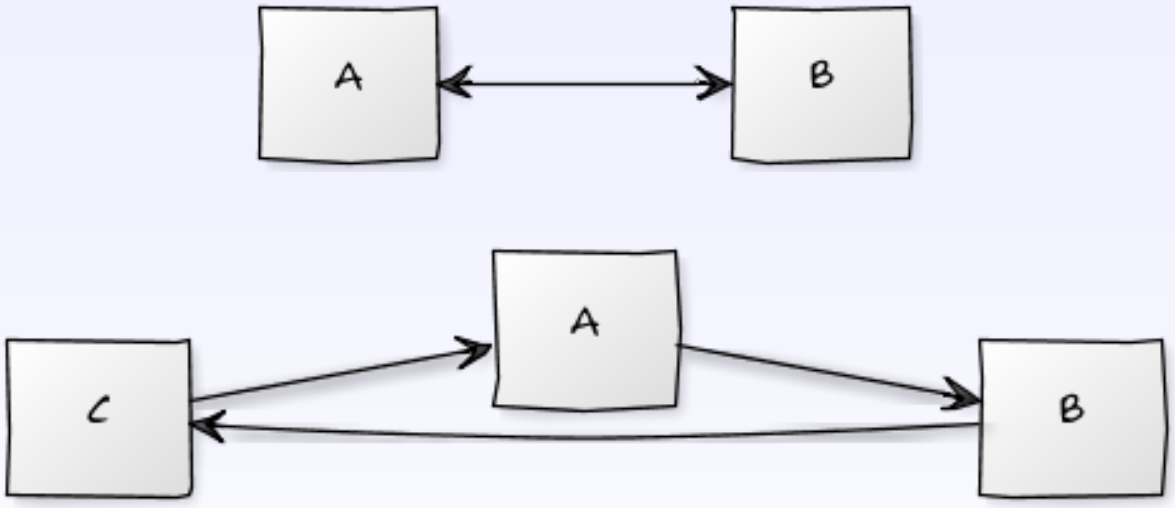
Tramp Couplung



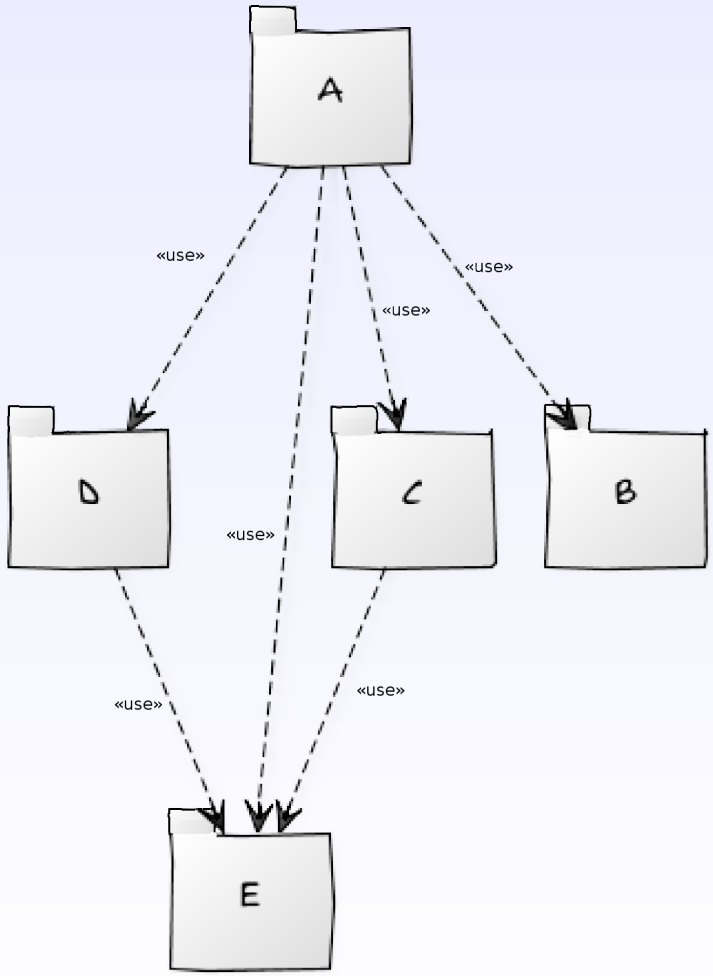
Logical Coupling (logische Kopplung)



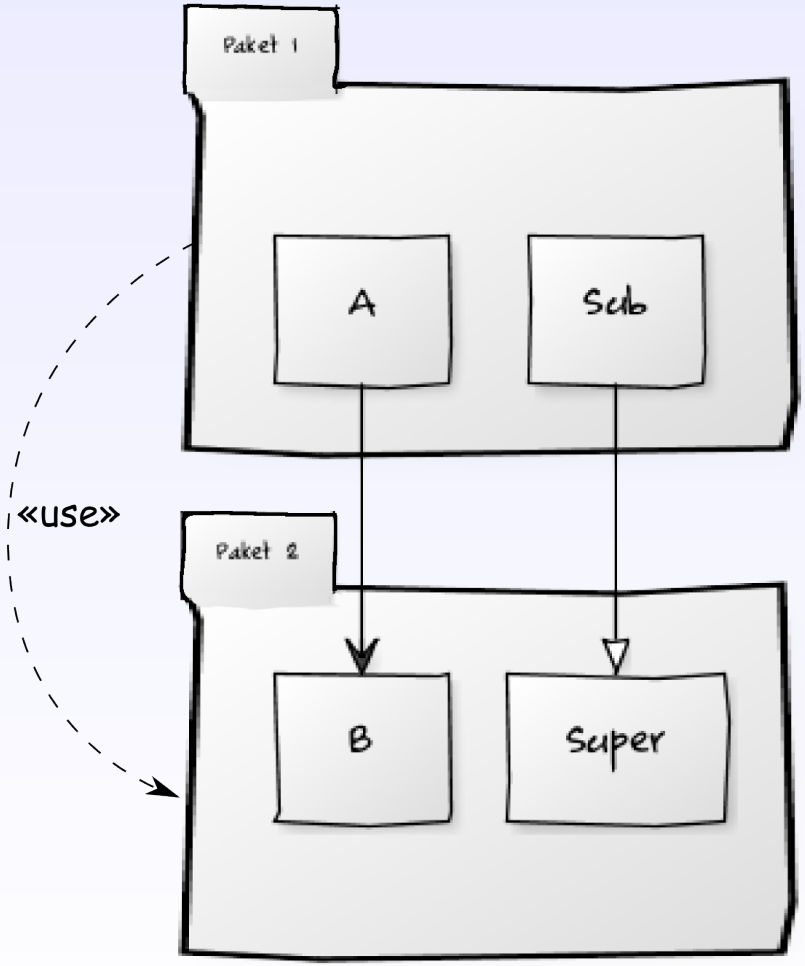
Cyclic Dependency (zyklische Abhängigkeit)



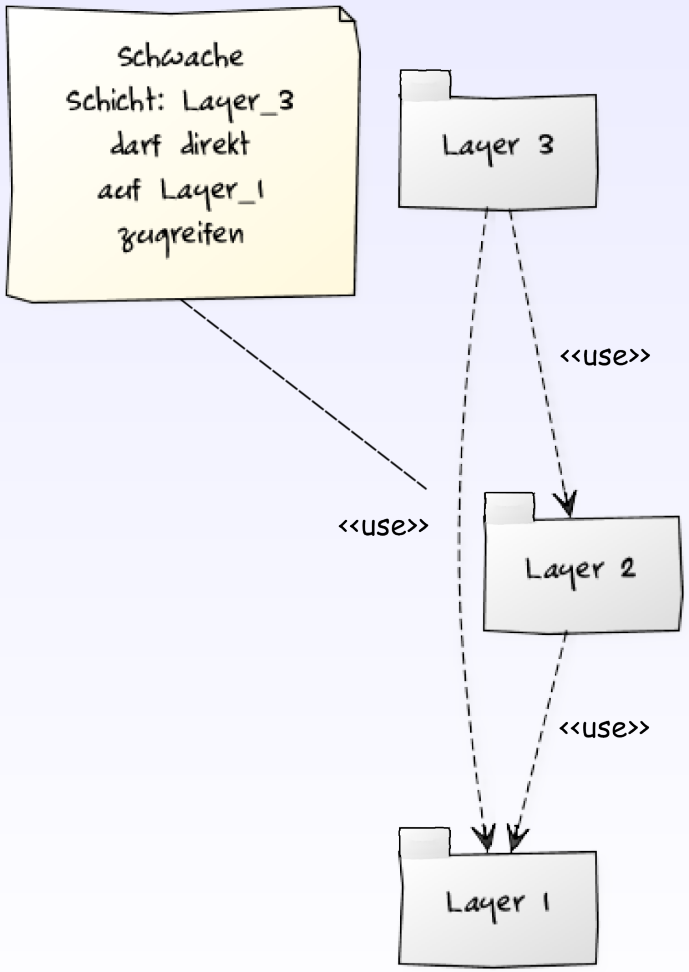
Kommunikation beschränken (1/2)



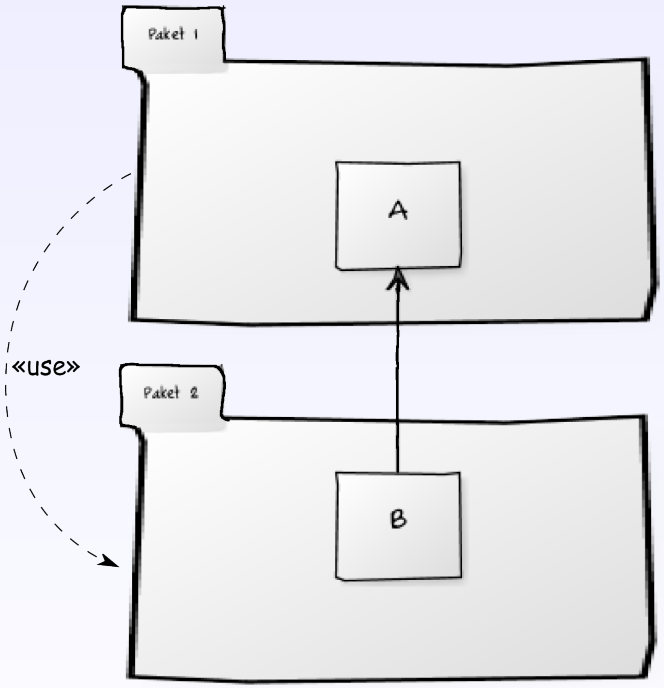
Kommunikation beschränken (2/2)

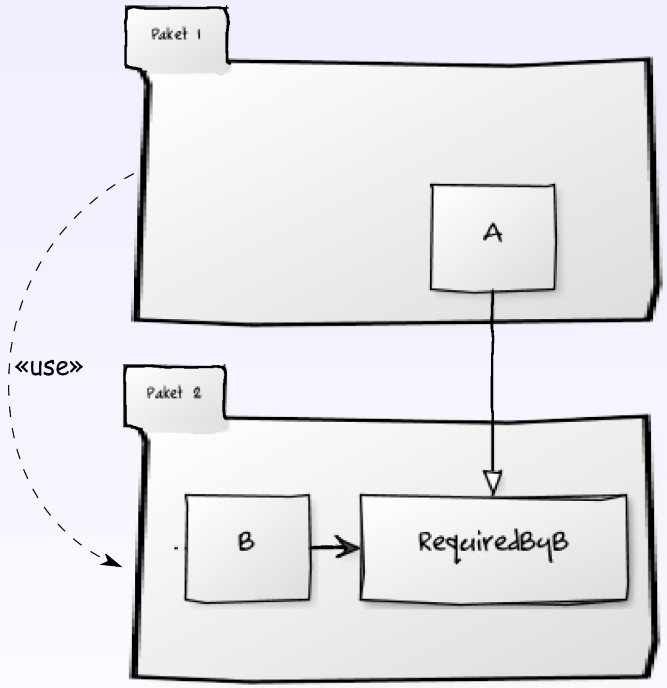


Schichten


Schwache
Schicht: Layer_3
darf direkt
auf Layer_1
zugreifen
Layer 3
<<use>>
<<use>>
Layer 2
<<use>>
Layer 1

Dependency Inversion





Patterns

Patterns

Definition Pattern

Ein „Muster“ oder „Pattern“ ist eine häufig anzutreffende, für gut befundene Lösung zu einem wiederkehrenden Problem.

Unterschiedliche Patterns

- Architekturpatterns (architectural patterns) bzw. „Architekturstile“ (architectural styles)
- Analysepatterns (analysis patterns)
- Entwurfsmuster (design patterns)
- Idiome (idioms)
- Anti-Patterns
- Code Smells
- ...

Patterns nutzen

1. Problem haben
2. Sich an Pattern erinnern
3. Pattern nachschlagen
4. Prüfen, ob Pattern passt
5. ggf. Pattern anpassen
6. Pattern anwenden
7. Freuen

Fazit

- 3 Hilfsmittel um Software zu entwerfen
  - Szenarien, Prinzipien und Patterns
- Nützliches Wissen
  - RippleEffects, Bindung und Kopplung
  - Abhängigkeiten und Schichten

Anhang

Aufgabe 1 – Datum

Aufgabe 1

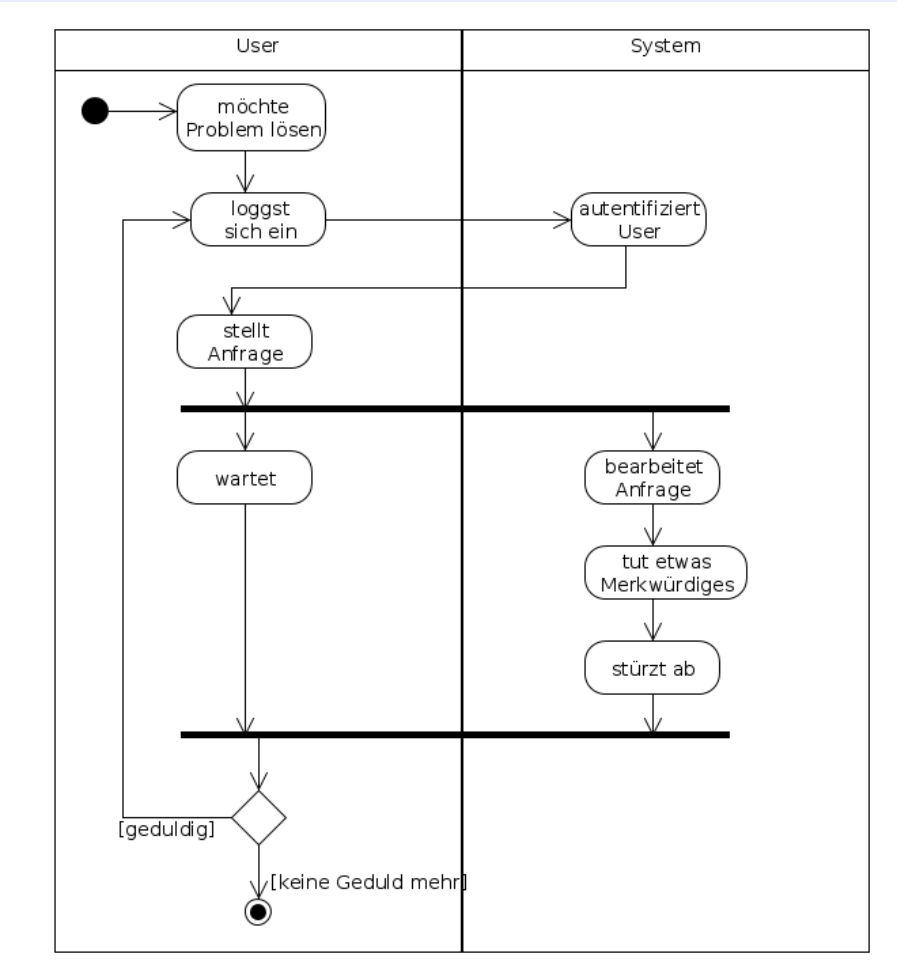
- Bibliothek für Datums- und Zeitwerte
- Für den „alltäglichen Gebrauch“ einsetzbar

Aufgabe 2 – Genealogie

Aufgabe 2

- Software für Genealogie (Ahnenforschung)
- Die Software soll Stammbäume, Verwandtschaftsbeziehungen, etc. graphisch darstellen, durchsuchbar machen, ...
- Für jede Information (Geburtsdatum, etc.) soll der User eine Quelle angeben können
- Außerdem ist zu beachten, dass manche Daten vage sein können

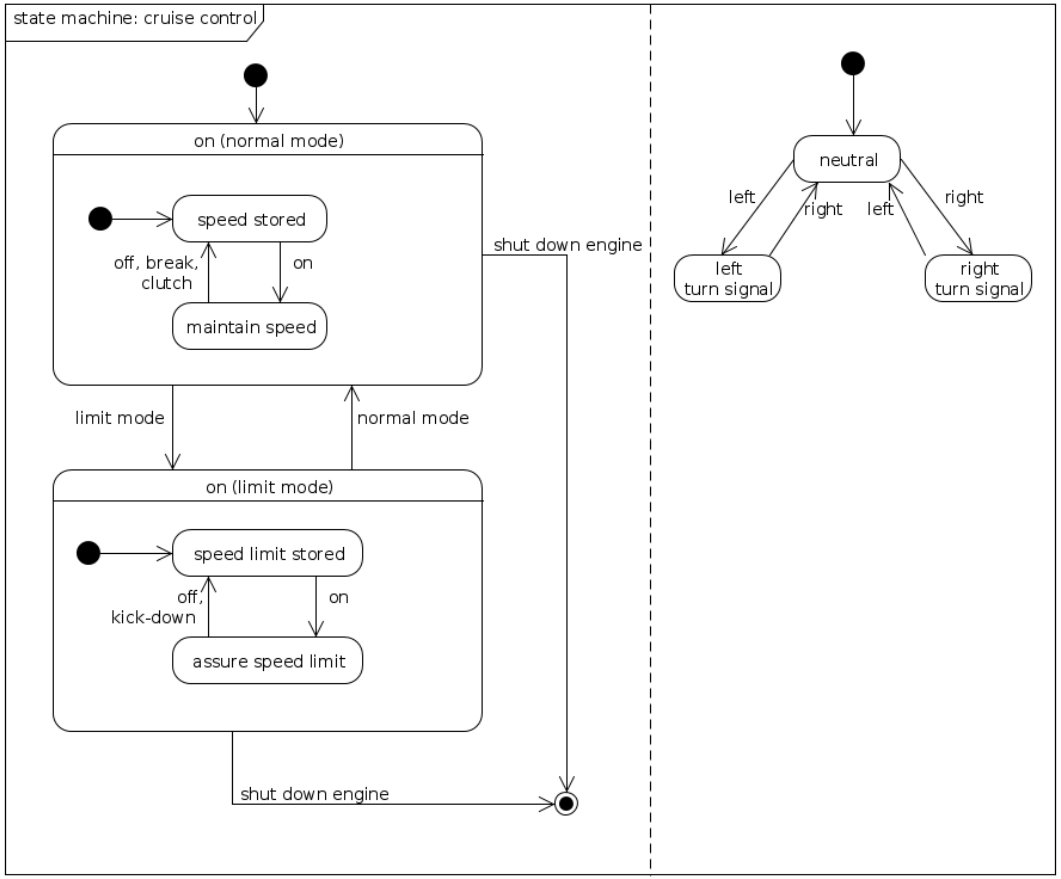
Aktivitätsdiagramme



Künstliche Modelle

Intuitive Modellierung ist nicht die einzige Möglichkeit

Zustandsdiagramme



Referenzen für Zitate


Len Bass, Paul Clemens, and Rick Kazman.
Software Architecture in Practice.
SEI Series in Software Engineering. Addison-Wesley, 2 edition, 2003.

(eigentliche Bibliographie im Handout)

Lizenz