Kapitel 10, 11, 12 gehören zusammen

Ein einfacher Softwareentwicklungsprozess von der Anforderungsanalyse über den Entwurf zur Programmierung und zum Test

Java-Beispielsammlung: Seiten 40 - 48

Sofort mit dem Programmieren zu beginnen, würde scheitern:
6 Dateien mit 378 Zeilen

Kapitel 10, 11, 12

Gehören zu den wichtigsten Gebieten der VL GdP:

Programmierung eingebettet in Kontext der SW-Entwicklung

11. Objektorientierte Softwarearchitekturen

Schwerpunkte

- Phase 'Entwurf' (Design)
- SW-Architektur-Beschreibungssprachen: UML
- Wie und wann findet man Klassen und Objekte?
- Beispiele:
  - Maus im Labyrinth
  - Einpass-Compiler
  - XCTL: Steuerung einer physikalischen Versuchsanlage

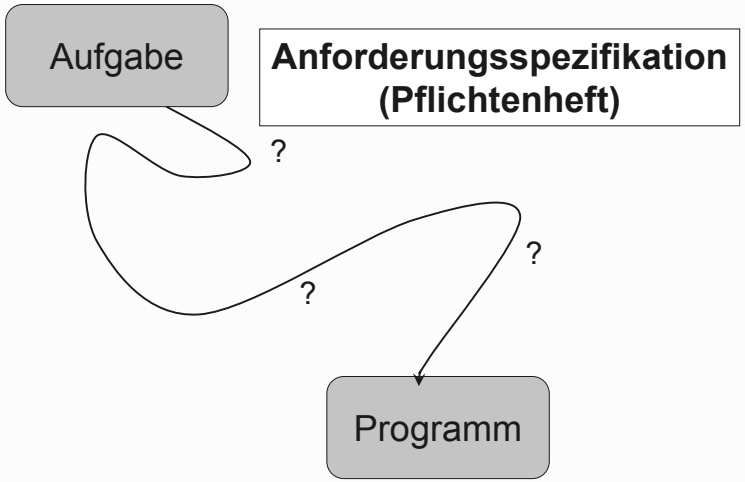
Aufgabenstellung: ‚Maus im Labyrinth'

Aufgabe:
Entwickeln Sie ein Programm, das die Bewegung einer Maus durch ein Labyrinth (Irrgarten) vom Eingang zum Ausgang simuliert.

Nicht sofort drauflos programmieren …

… sondern zunächst die Aufgabe präzisieren, sonst Lösung für das falsche Problem

Vom Problem zum Programm



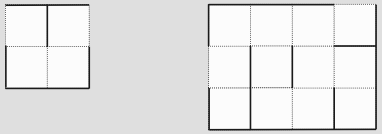
Aktuelle Stand: Anforderungsspezifikation (6 Seiten)

Entwickeln Sie ein Programm, das die Bewegung einer Maus durch ein Labyrinth (Irrgarten) vom Eingang zum Ausgang simuliert!

1. Das Labyrinth (Irrgarten)

Ein Labyrinth ist eine rechteckige Anordnung quadratischer Räume. Zwischen zwei benachbarten Räumen befindet sich entweder eine Wand oder eine Öffnung. Das Labyrinth wird durch eine zusammenhängende Wand umschlossen, die an einer oder zwei Stellen (ein Eingang und evtl. ein Ausgang) durchbrochen wird. Die Größe des Labyrinths (d. h. Länge und Breite) ist variabel.

Beispiele:



Anforderungsspezifikation (2)

2. Die Maus:

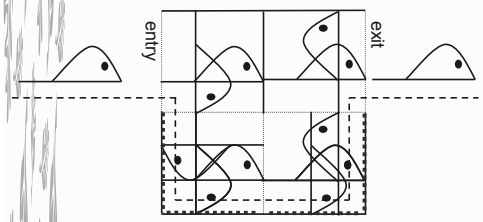
Die Maus hat keine Gesamtübersicht des Labyrinths.

a) Die Maus kann sich folgendermaßen bewegen: Linksdrehung, Rechtsdrehung (jeweils um 90 Grad), Schritt vorwärts in den benachbarten Raum.

b) Die Maus befindet sich entweder in einem Raum innerhalb des Labyrinths oder direkt vor dem Eingang oder am Ausgang. Außerdem hat sie eine bestimmte Blickrichtung. Sie kann entscheiden, ob sie sich innerhalb des Labyrinths befindet.

c) Die Maus kann nur in die Blickrichtung sehen. Sie kann entscheiden, ob sich in dieser Richtung eine Wand vor ihr befindet oder nicht.

Mouse movements: graphical and textual output




```
step forward; /* enter the maze */
WHILE(NOT outside the maze?)
   BEGIN /*do next step*/
         turn right;
         WHILE (facing a wall?) DO
                  turn left;
         ENDWHILE
         step forward;
   END
ENDWHILE
```

```
ENTER   step forward
WHILE   turn right
        step forward
WHILE   turn right
        turn left
        turn left
        step forward
WHILE   turn right
        turn left
        turn left
        step forward
WHILE   turn right
        step forward
```


Aktuelle Stand: Anforderungsspezifikation (6 Seiten)

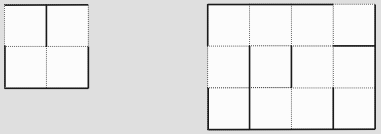
Entwickeln Sie ein Programm, das die Bewegung einer Maus durch ein Labyrinth (Irrgarten) vom Eingang zum Ausgang simuliert!

1. Das Labyrinth (Irrgarten)

Ein Labyrinth ist eine rechteck[illegible] Räume. Zwischen [illegible]t sich entwede[illegible]s Labyrinth wird durch ei[illegible]schlossen, die an einer oder [illegible]gang und evtl. ein Ausgang) durchbrochen [illegible]. Die Größe des Labyrinths (d. h. Länge und Breite) ist variabel.

Jetzt mit dem Programmieren beginnen?

Beispiele:



Aufgabe

Anforderungsspezifikation (6 Seiten) : ...

Jetzt mit dem Programmieren beginnen?

Falls ja,

- Womit beginnen?
- Mit welchem Teil / welcher Komponente?
- Arbeitsteilung: je ProgrammiererIn eine Komponente

Probleme:

- Welche Komponenten existieren überhaupt?
- Welches Verhalten wird von ihnen verlangt?

Nächster Schritt: Softwarearchitektur bestimmen (Komponenten mit Interface)

Softwareentwicklung: Phasen and Ergebnisse

- Analyse & Definition
  → Anforderungsspezifikation
- Entwurf (Design)
  → Softwarearchitektur
- Implementation
  → Programm
- Test
  → Testprotokolle

Bisher in der VL/Praktikum/ÜA:
Klassen und Softwarearchitektur gegeben

Jetzt:
Klassen selbst finden und zwar vor der Programmierung

Das Problem

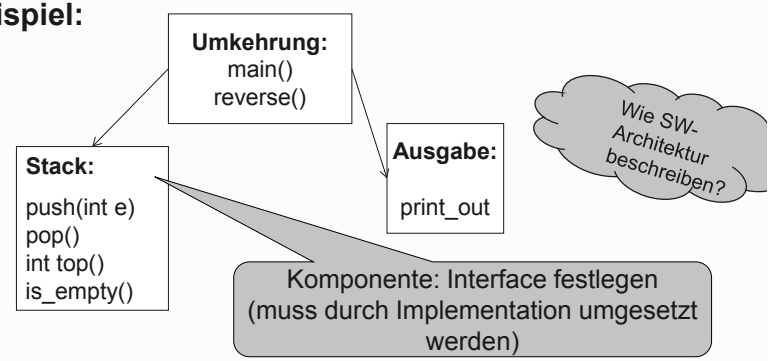
Das Finden einer geeigneten Softwarearchitektur kann schwieriger sein als die anschließende Implementation im Java-Programmcode.

Eine ungünstige Softwarearchitektur kann die Wartung der Software so behindern, dass eine Neuimplementation nötig wird.

Softwarearchitektur

SW-Architektur = Struktur der Software:
- Welche Komponenten existieren?
- Welche Relationen gibt es zwischen ihnen?
- Welches Interface besitzen die Komponenten?

Beispiel:



UML: Beschreibung von Softwarearchitekturen

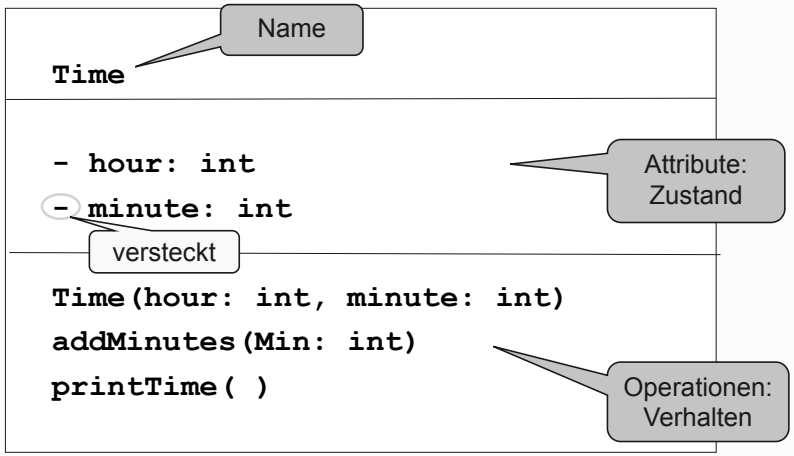
- Graphische Sprache zur Spezifikation von Softwarearchitekturen
- UML: Unified Modeling Language
  → Klassendiagramme für SW-Architekturen
- Jetzt: an Beispielen *einige* Ausdrucksmittel

Was für die Phase 'Implementation' die Programmiersprachen, sind für die Phase 'Entwurf´Architekturbeschreibungssprachen.

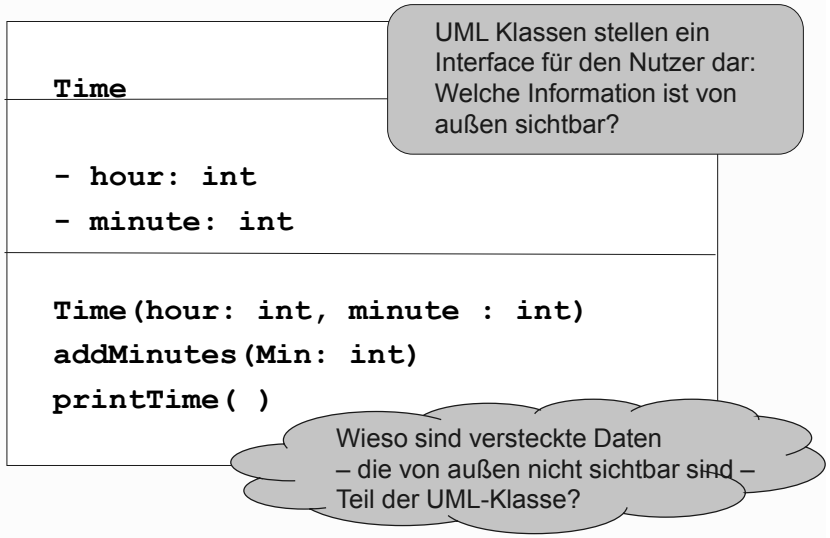
Diagrammarten in UML

- *Use Case - Diagramm:* grundlegende Programmfunktionen und ihre Nutzungsrechte durch Akteure (Teil der Anforderungsspezifikation)
- *Klassendiagramm:* Klassen und ihre statischen Beziehungen
- *Sequenzdiagramm:* Nachrichtenfluss, zeitliches Zusammenwirken von Objekten
- *Kollaborationsdiagramm:* wie Sequenzdiagramm
- *Package-Diagramm:* Modularisierung (Teilprojekte: Gruppen von Klassen und Packages)
- *Zustandsdiagramm:* dynamisches Verhalten von Objekten
- *Aktivitätsdiagramm:* Parallele Prozesse
- *Komponentendiagramm:* Übersetzungseinheiten, Hardwarestruktur …
- *Objektdiagramm:* Objekte und ihre Verbindungen (Momentaufnahme im laufenden System)

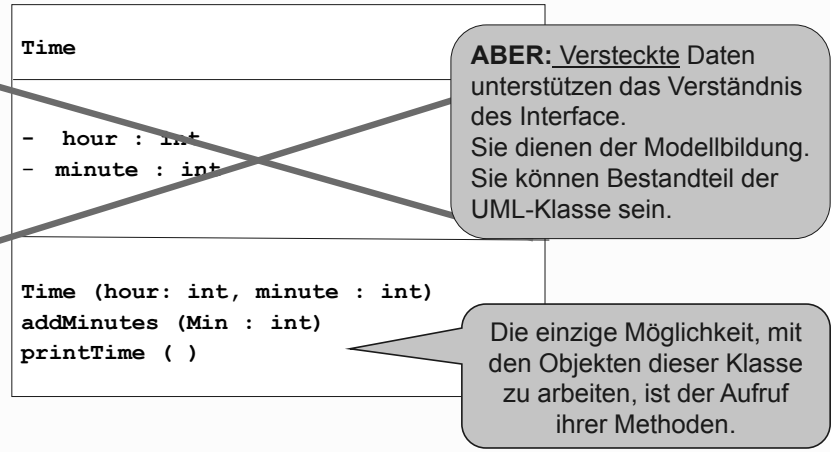
UML-Klassen: Struktur



UML-Klassen als Interface

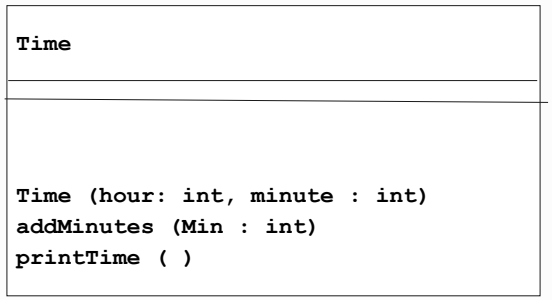


Das nutzbare Interface einer Klasse: nur die sichtbaren Elemente notwendig



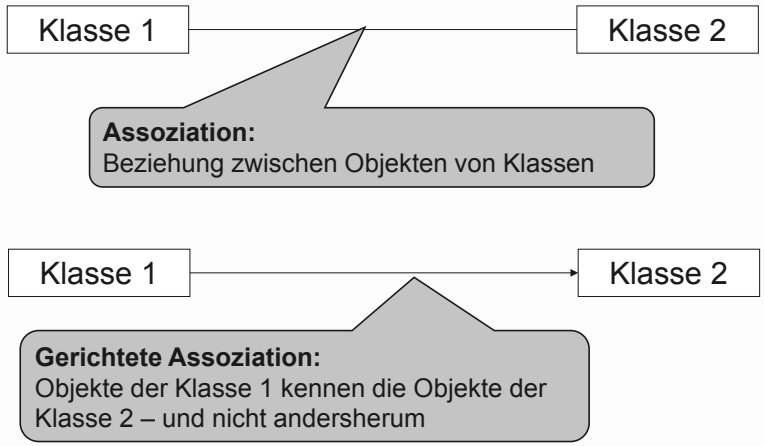
Das nutzbare Interface einer Klasse: nur die sichtbaren Elemente

Auch so möglich:

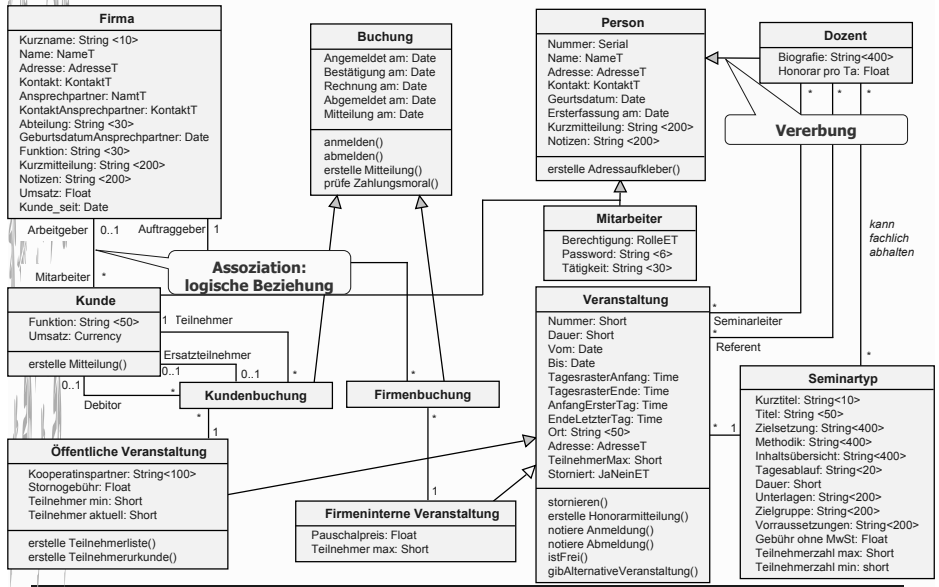


… aber mit versteckten Daten hour und minute besser verständlich.

Relationen zwischen Klassen: Assoziationen, Vererbung, ...



Beispiel für OO SW-Architekturen als Klassendiagramm: Seminarorganisation (kommerzielle Anwendung)



Wie findet man eine Softwarearchitektur?

Wie findet man Klassen?

Prinzipien ?

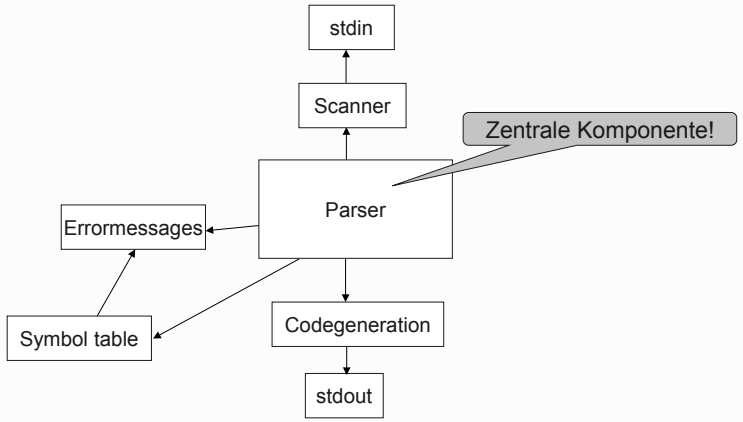
- Durch Zerlegung des Problems in Teilprobleme
  - Teilprobleme können durch Klassen realisiert werden

→ **Compiler**

Teilprobleme:
- Lexikalische Analyse (Scanner: Symbole erkennen)
- Syntaktische Analyse (Parser)
- Symboltabelle (Variablen mit Attributen abspeichern)
- Codegenerierung
- Fehlermitteilung

SW-Architektur eines Compilers: Zerlegung des Problems in Teilprobleme

- Einpass-Compiler
- Teilprobleme → Komponenten (Klassen)



Wie findet man Klassen?

- Durch Zerlegung des Problems in Teilprobleme
  - Teilprobleme können durch Klassen realisiert werden

  → **Compiler**

- Aus Objekten des Problembereichs
  - Anforderungsspezifikation untersuchen, um Objekte des Problembereichs zu finden

  → **Maus im Labyrinth**

Fallbeispiel: 'Maus im Labyrinth'

Wie findet man Klassen / Objekte ?

→ **aus Objekten des Problembereichs**

→ Anforderungsspezifikation untersuchen

Welche Objekte des Problembereichs sollten als Komponente (Klasse) des Systems implementiert werden?

Maus

Labyrinth (Maze)

Algorithmus zur Bewegung der Maus

Nutzeroberfläche / Ausgabe

Englisch: Maze = Irrgarten, Labyrinth = Labyrinth

Welche Beziehungen (Assoziationen)?

Welche Beziehungen?

Wer braucht wen?

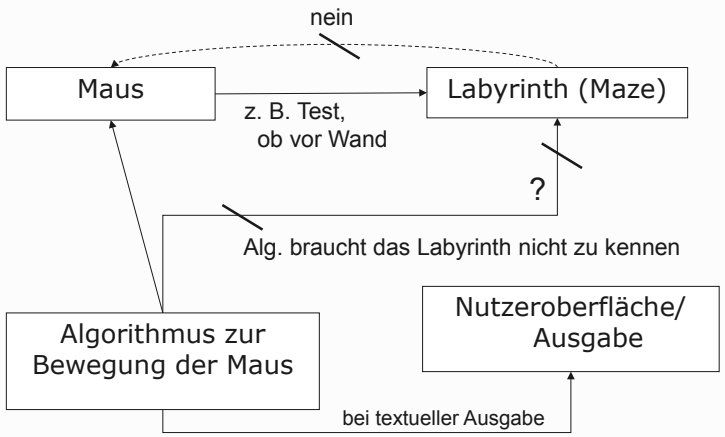
Maus

Labyrinth (Maze)

Algorithmus zur Bewegung der Maus

Nutzeroberfläche / Ausgabe

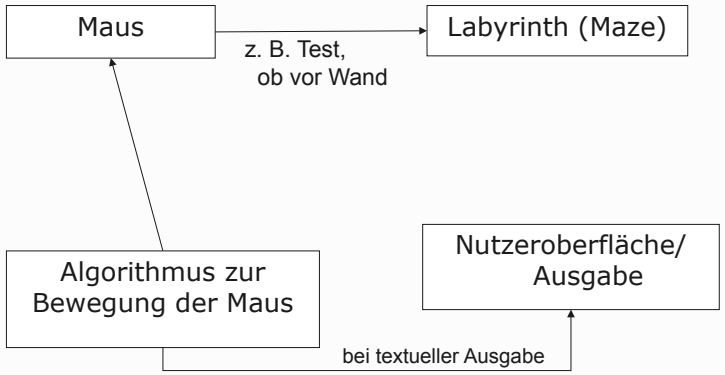
Welche Beziehungen (Assoziationen): Wer braucht wen? Wer kennt wen?



Wahrscheinlich: nur 3 gerichtete Beziehungen (von insg. 12 möglichen)

Welche Beziehungen (Assoziationen): Wer braucht wen? Wer kennt wen?

Finale Lösung



Wahrscheinlich: nur 3 gerichtete Beziehungen (von insg. 12 möglichen)

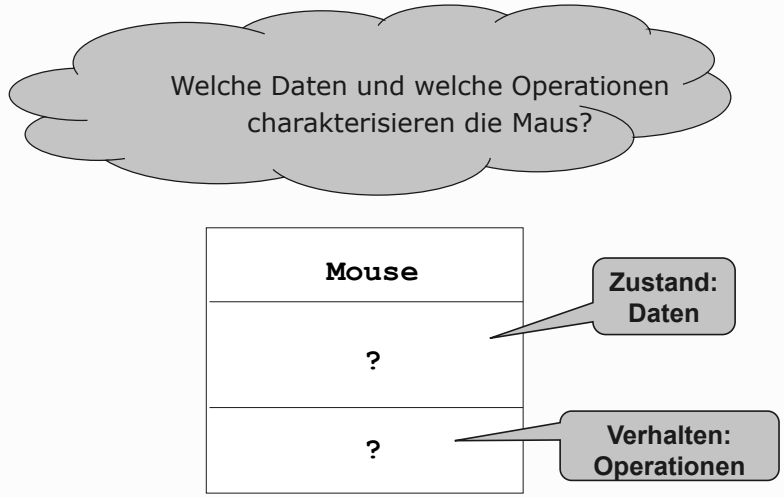
Softwarearchitektur: 'Maus im Labyrinth'

Aktueller Stand:

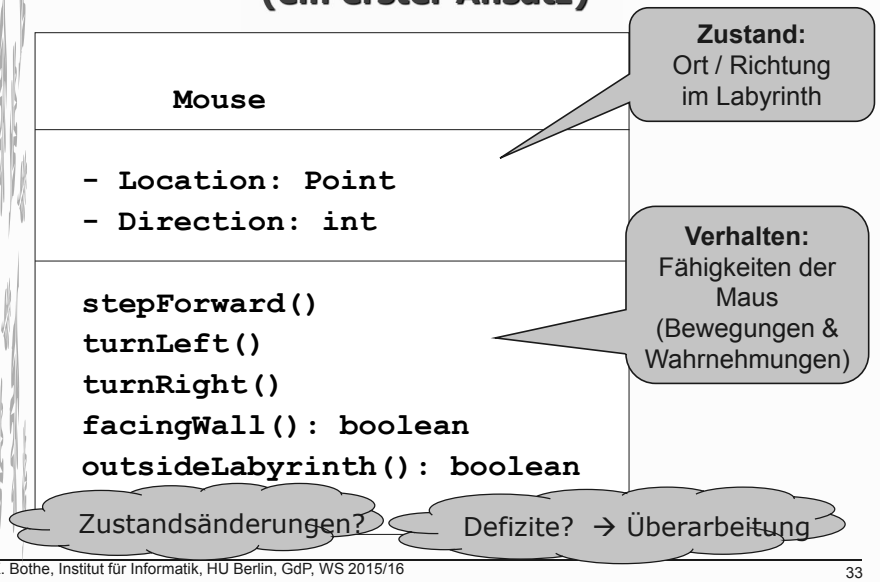
Softwarearchitektur:
- Welche Komponenten existieren? ✓
- Welche Relationen gibt es zwischen ihnen? ✓
- Welches Interface besitzen die Komponenten? ←

Klassendiagramme

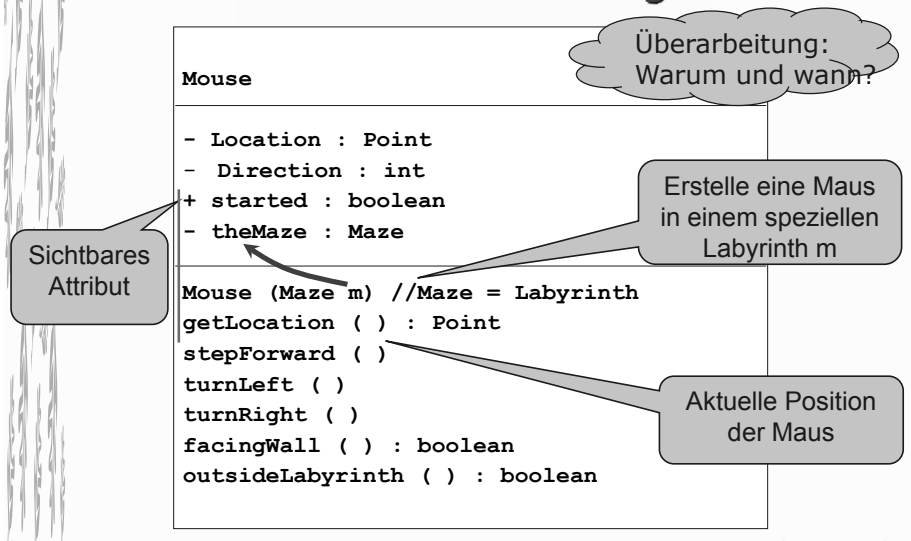
Klassendiagramm für eine Maus



Feinstruktur der Maus: Welche Daten, welche Operationen? (ein erster Ansatz)



Feinstuktur der Maus: erste Überarbeitung



Später – bei der Nutzung der Klasse/im Algorithmus – merkt man, dass noch Information fehlt.

Feinstruktur des Labyrinths

Klassendiagramm des Labyrinths: Daten und Operationen?

Maze (Labyrinth)
?
?

Interface des Labyrinths: Welche Informationen werden von einem Benutzer dieser Klasse benötigt? (Benutzer = Objekte der Klasse "Mouse")

Englisch: Maze = Irrgarten, Labyrinth = Labyrinth

Feinstruktur des Labyrinths?

Maze (Labyrinth)

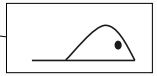
Zustand:

Daten zur Beschreibung des Labyrinths
→ hier nicht näher bekannt bzw. offen gelassen (Implementationsdetail)

Verhalten:

Informationen über das Labyrinth ermitteln
→ Operationen zur Analyse des Labyrinths

Feinstruktur des Labyrinths (ein erster Ansatz)



Welche Auskünfte muss das Labyrinth an andere Objekte (Mäuse) geben ?

```
Maze (Labyrinth)

outside(pos: Point): boolean
checkWall(direction: int, pos: Point): boolean
getStartLocation(): Point
```

Wo soll die Maus anfangs positioniert werden? (Wo ist der Eingang?)

Maus kennt eigene Position & Richtung:
- In Blickrichtung liegt eine Wand ?
- Position bereits außerhalb des Labyrinths ?

Feinstruktur des Labyrinths: erste Überarbeitung

```
Maze (Labyrinth)

- height: int
- width: int

getStartLocation(): Point
getStartDirection(): int
getSize(): Point
checkWall(direction: int, pos: Point) : boolean
checkWall(direction: int, col: int, row: int): boolean
outside(pos: Point): boolean
```

Die Höhe und Breite sind hilfreich bei der Zeichnung des Labyrinths

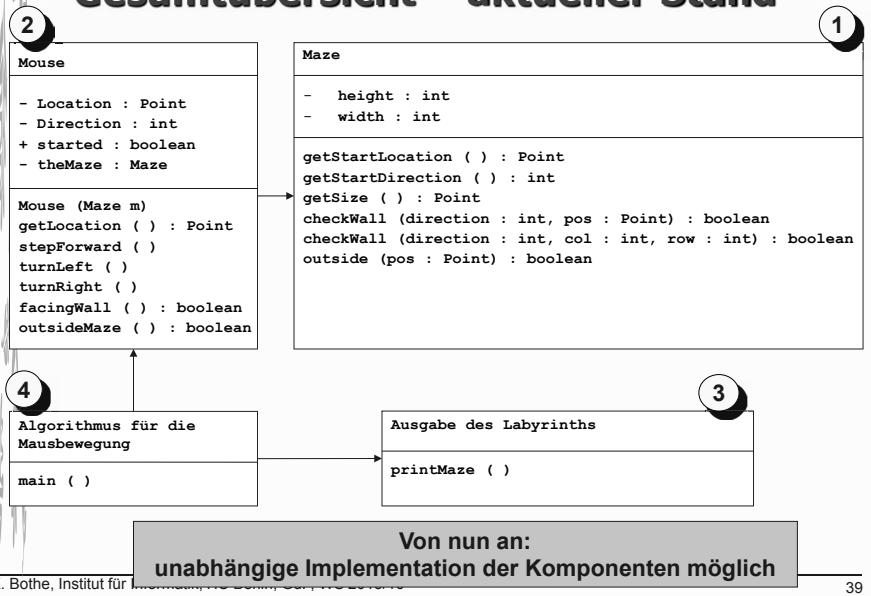
zu Beginn: Richtung

Höhe und Breite als Punkt

checkWall in zwei Varianten (Überladen)

Das Finden des Klassendiagramms ist ein iterativer Prozess: Beginn bei einer einfachen Lösung, dann ist schrittweises Erweitern und Modifizieren nötig.

Softwarearchitektur: Gesamtübersicht – aktueller Stand



Von nun an: unabhängige Implementation der Komponenten möglich

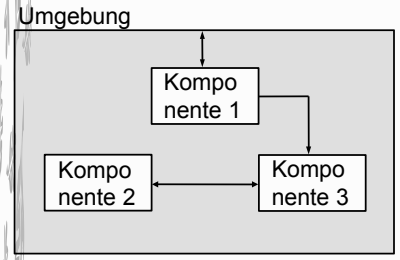
Probleme bei der Entwicklung einer Softwarearchitektur

- Softwarearchitektur:
  - Nicht eindeutig (viele gute und schlechte Lösungen)
- Gelingt nicht beim ersten Mal
- Längerer Prozess:
  - Softwarearchitektur schrittweise entwickelt
- Prinzip:
  - Beginn mit einer vorläufigen Architektur
  - Die Nutzbarkeit der Methoden stellt sich endgültig erst bei der Implementation (bei Nutzung im Algorithmus) heraus.
- "Study good examples of software systems" (Tewari, Friedman, LNCS 640)

SW-Architekturen: Verallgemeinerungen, Fallbeispiel

Software-Architekturen

Software-Architektur: Komponenten + Beziehungen



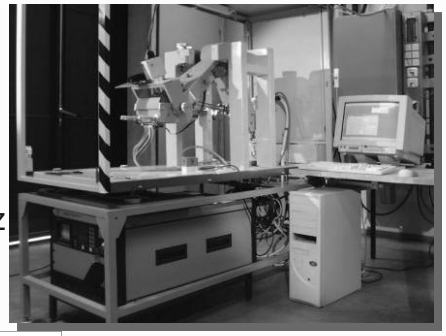
Bedeutung von SW-Architekturen:

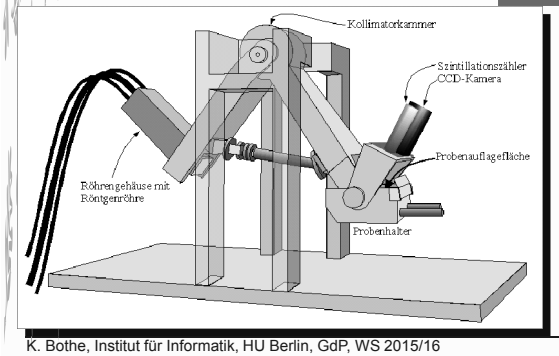
Beeinflusst viele SW-Qualitätsmerkmale:

- Verständlichkeit
- Nachnutzbarkeit (z. B. Komponenten in andere Umgebung einbauen)
- Wartbarkeit (Erweiterbarkeit, Modifizierbarkeit)
- Sicherheit
- Testbarkeit
- u.a.

XCTL: Steuerung einer physikalischen Anlage

Messplatz



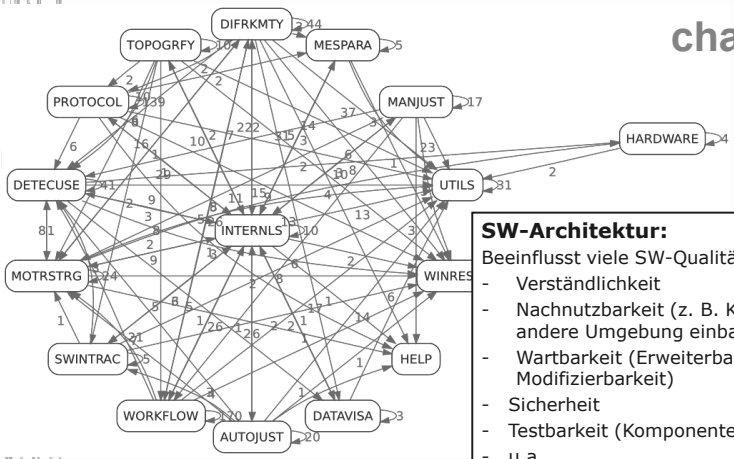


Röntgentopografie-Kamera

→ **Reales Beispiel für schlechte SW-Architektur**

Reale Architektur von XCTL (Inst. Physik)

chaotisch



SW-Architektur:

Beeinflusst viele SW-Qualitätsmerkmale:

- Verständlichkeit
- Nachnutzbarkeit (z. B. Komponenten in andere Umgebung einbauen)
- Wartbarkeit (Erweiterbarkeit, Modifizierbarkeit)
- Sicherheit
- Testbarkeit (Komponenten unab. testen?)
- u.a.

Software-Architektur: kann Gesamtprojekt gefährden

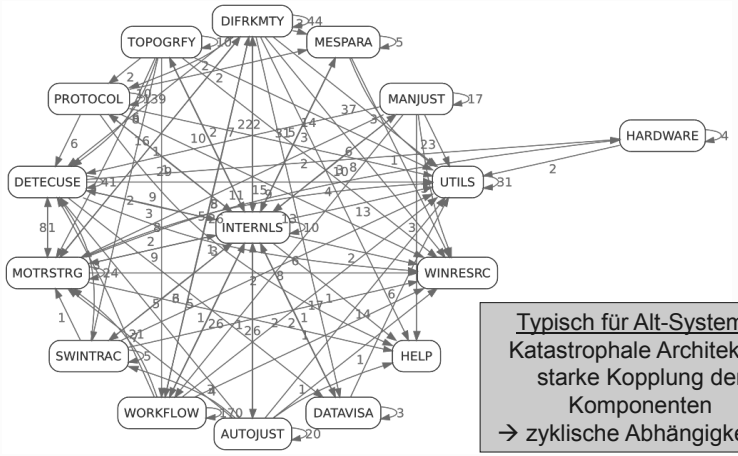
"The Importance of Software Architecture

- Software architecture forms the backbone for any successful software-intensive system. An architecture is the primary carrier of a software system's quality attributes such as performance or reliability. The right architecture - correctly designed to meet its quality attribute requirements, clearly documented, and conscientiously evaluated - is the linchpin for software project success.
  The wrong one is a recipe for guaranteed disaster."

(Webseite: Software Engineering Institute (SEI, Carnegie Mellon University))

backbone = Wirbelsäule, Rückgrad
carrier = Träger
conscientiously = gewissenhaft
linchpin = Dreh- und Angelpunkt

Reale Architektur von XCTL (Inst. Physik): Gefahr für das Gesamtprojekt

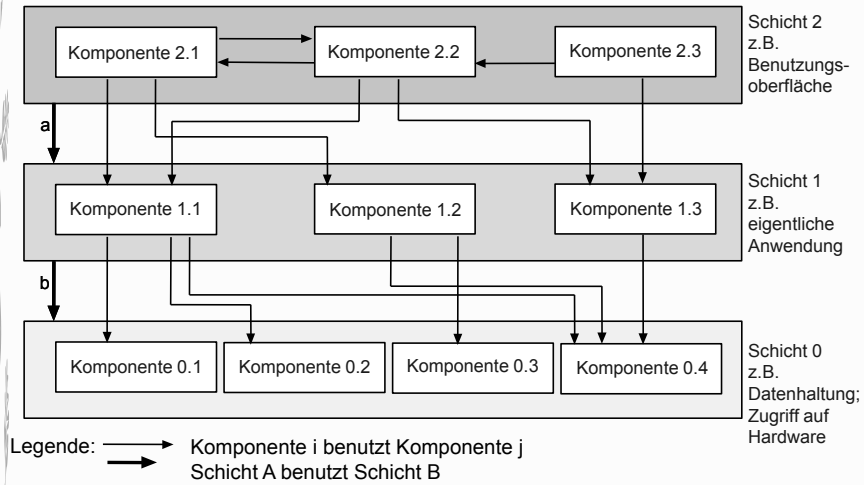


Typisch für Alt-Systeme:
Katastrophale Architektur, starke Kopplung der Komponenten
→ zyklische Abhängigkeiten

Schlussfolgerung: Wartbarkeit sehr stark eingeschränkt
Ein Problem: keine Schichtenarchitektur (GUI – Anwendung – Hardware)

Beispiel für eine Drei-Schichten-Architektur

Typische sinnvolle SW-Architektur



Was sind „gute" SW-Architekturen?

SW-Architektur bewerten

Was sind „gute" SW-Architekturen?

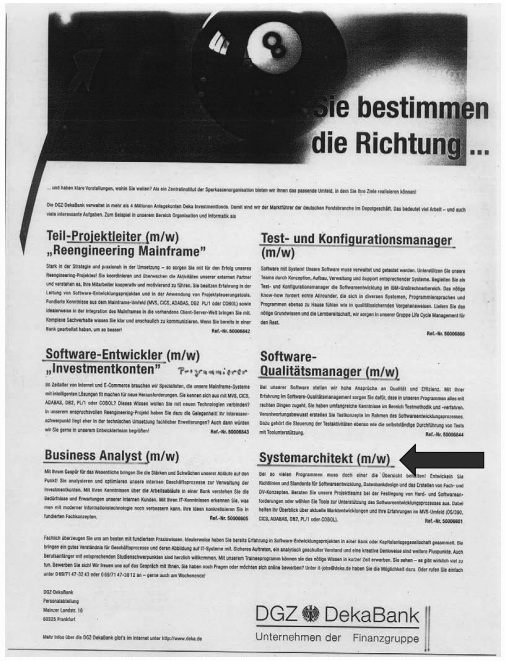
schwache Kopplung: Beziehungen zwischen den Komponenten gering

starke Kohäsion (Bindung): Komponenten dienen der Lösung einer gemeinsamen Aufgabe

Software-Architektur: schwer zu korrigieren …

"Eine einmal eingerichtete Softwarearchitektur ist später nur mit hohem Aufwand abänderbar. Die Entscheidung über ihr Design ist somit eine der kritischsten und wichtigsten Punkte im Entwicklungsprozess einer Software". (Wikipedia / Balzert).

Software-Architektur: schwer zu korrigieren …
… aber lohnenswert bei intensiverer beabsichtigter Weiterentwicklung



Stellenanzeigen: Softwarearchitekturen betont

Spezialist

Software-Entwickler (m/w)

Dipl.-Informatiker/Dipl.-Ing.
Kennziffer SZ/401, mehrere Positionen zu besetzen

Ihre Aufgabe
- Architekturentwurf, Systemdesign und Softwareentwicklung für Mikroprozessor-Systeme im Bereich der Prozeßautomatisierung
- Inbetriebnahme und Funktionstest bis zur Serienreife

Ihr Profil
- abgeschlossenes Studium der Fachrichtung Informatik, Softwaretechnik, Elektrotechnik o.ä.
- mehrjährige Berufserfahrung in der Softwareentwicklung für Steuerungstechnik
- Erfahrung in der Programmierung mit C, C++ und Assembler sowie im Software-Engineering
- Freude am selbständigen, eigenverantwortlichen Arbeiten in Projekten
- gute Englischkenntnisse

Wir sind ein auf Bankensoftware spezialisiertes Softwarehaus im Großraum München mit namhaften Kunden im In- und Ausland. Unser Produkt **OBS Online Banken System** unterstützt alle Bereiche des internationalen Bankgeschäfts, besonders im Handels- und Wertpapierbereich.

Zur Verstärkung unseres jungen Teams suchen wir

APPLIKATIONSENTWICKLER/IN

Ihre Aufgaben:

- Analyse der fachlichen Anforderungen
- Design der Anwendungen
- Programmierung
- Test und Qualitätssicherung
- Implementation beim Kunden

vgl. Phasen des SE

Ihr Profil:

- Cobolkenntnisse oder vergleichbare Kenntnisse
- Kenntnisse eines der folgenden Systeme: UNIX, Open VMS, OS/400
- SQL und Datenbankkenntnisse, z. B. Oracle
- Kaufmännische Grundkenntnisse

„Handwerkszeug“

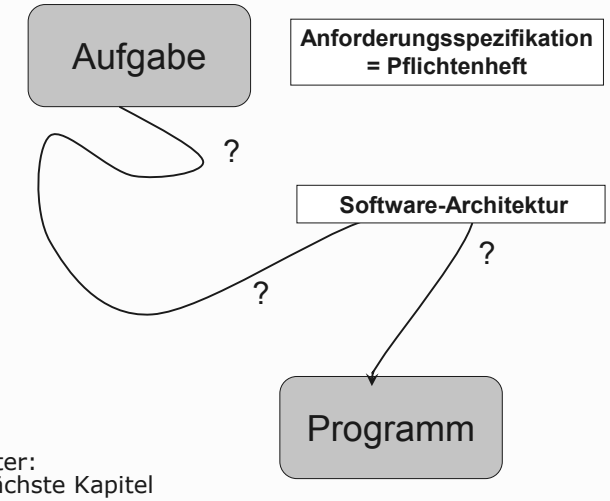
Wir bieten:

- Innovatives Tätigkeitsumfeld mit viel Freiraum für eigenverantwortliches Arbeiten
- Attraktive leistungsorientierte Vergütung
- Möglichkeit zu Dienstreisen in Europa und Asien

Bitte senden Sie Ihre Bewerbungsunterlagen an Frau Alexandra Zimmermann.

DIE SOFTWARE
Peter Fitzon GmbH
Weißenfelder Straße 1–3, 85599 Parsdorf
Tel. 0 89/9 04 60 71

Vom Problem zum Programm: Maus im Labyrinth



→ weiter:
nächste Kapitel