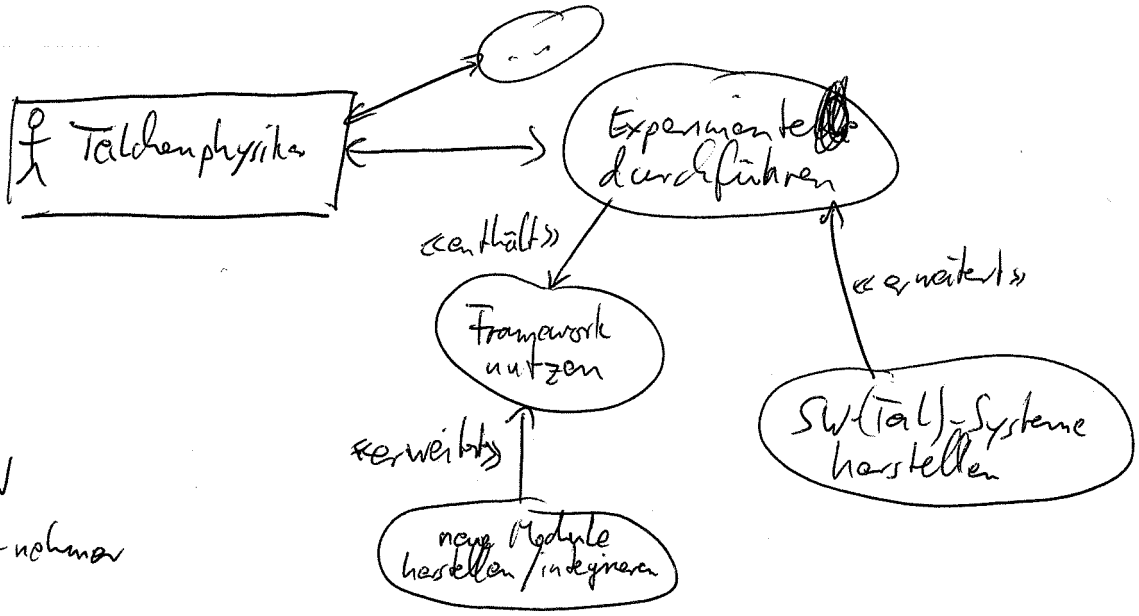
Abendvortrag

29.9.2010

Ist-Situation

"Use-case":



AG = AN

Arbeitgeber = -nehmer

Anforderungen

(Prof. Klaus Pohl)
siehe Bücher

- funktionale
(- nicht funktionale)
- Qualität
  - Benutzerfreundl.
  - Effizienz
  - ~~Zu~~verlässigkeit, Sicherheit ←
  - Flexibilität
  - Wiederverwendbarkeit

Portabilität
Kompatibilität
Erweiterbarkeit
Änderbarkeit

- Rahmenbedingungen
  - technische / technologische
  - organisatorisches
  - rechtlich

Modelle

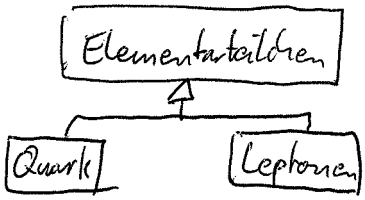
http://www.omg.org/UML/2.3

- Infrastructure PDF
- Superstructure PDF

Wiederverwendung vs. Wiederverwendbarkeit

OOAD ⇒ Probleme mit Objekten lösen
- abstraktion → bilde Klassen ("abstrakter Datentyp")
- methoden
- Zustände

Klassen:
- erste Wiederverwendung
- Vererbung ← zweite Wiederverwendung



Generalisierung (explizit nicht "Ableitung") IS-A-Beziehung

"fordere nicht mehr, liefere nicht weniger" Liskov-Substitution

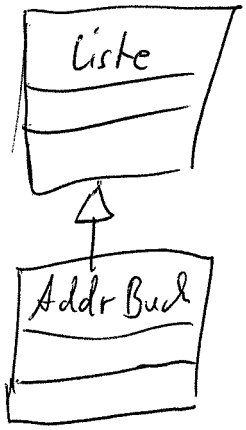
ungleich ∀ₒ
- Klassifizierung

1. Hta ist Lehrer
2. Lehrer ist Berufstätiger
3. Berufstätiger ist Mensch
4. Lehrer ist Besoldungsgruppe

(1 2 3) ⇒ Hta ist Mensch
(1 2 4) ⇒ Hta ist Besoldungsgruppe

1,4 obj → typ ▽
2,3 typ → typ 0

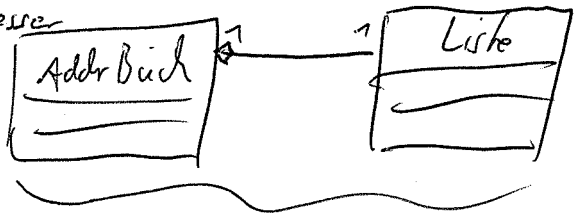
Beispiel:



get(int) ← instabile Basisklasse

get(name)

besser



Delegation
AddrBuch benutzt Liste

Klassendiagramm

1 – Konzeptionelle Sicht
2 – Spezifikations-Sicht ← Hier werden Abhängigkeiten gerichtet
3 – Implementationssicht

[Delegation nach oben muss geplant sein, nach unten kann man das nachrüsten]

[Generalisierung trennt Generelles von Speziellem]

Vererbung vs Generalisierung

Gütenweitergabe ≠
Genweitergabe ≠

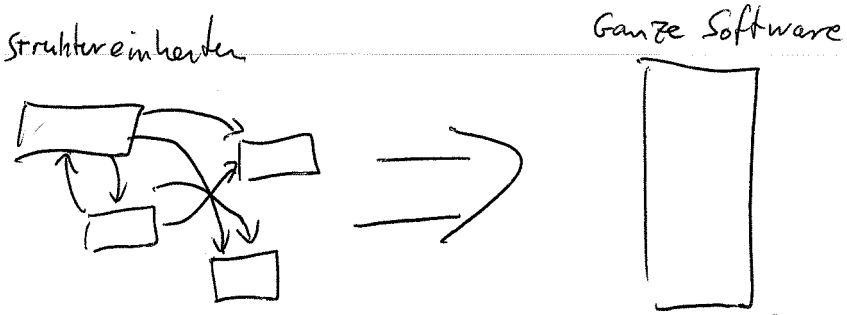
Einordnung von Typen

OOAD im Entwurf

Ziel: Baue Struktur, die man noch ändern kann*
im Laufe der Zeit mit Inhalt füllen

* Dokumentation!

Grobstruktur – Architektur
Feinstruktur – Feinentwurf



Komplexität in Abhängigkeiten verstecken ist schlecht!

besse: - enge Bindung (Zusammenhalt, Kohäsion)
- lose Kopplung

Define: Komponente

Define: Schnittstelle besser: Interface
Zwischen-Gesichte
Verbindungsstelle

„syntaktisches Merkmal" (Schnittstellen)

„funktionales" Merkmal (exportierte Dienste)

„nicht-funktionale Merkmale" (effizienz, ...)

Komponenten • exportieren Dienste
• importieren
• kapseln Implementierung !?!?

Komponieren von Komponenten → software

Schnittstellen definieren

speziell: programmschnittstelle : syntax-Aspekt
semantischen Aspekt

- synt. A: Rückgabewert, Argumente, in/out, ...
- sem. A.: was bewirkt? Performance, Sicherheit, Synchron/Asynch?
Organisatorische-/Rechtliche ....

[vgl. Anforderungen?]

Kopplung (z.B. glob. Variablen) "voll"
Funktionskopplung "halb"
entkoppelt

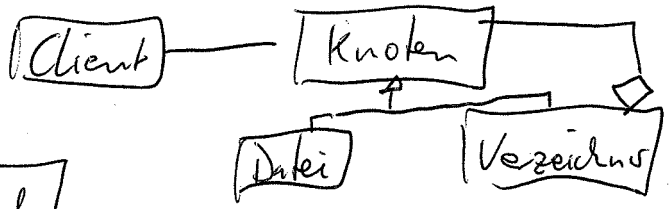
Preis: Rechenleistung, #Module steigt

Architekturen

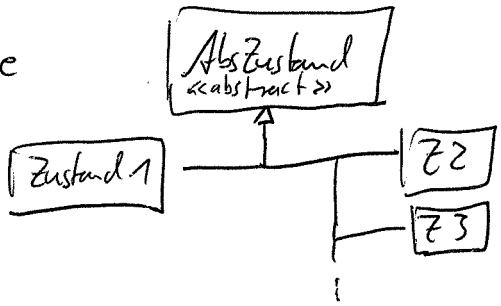
- Depot/Repository
- Pipes & Filter
- Model - Controler - Viewer ("trennung von Verantwortlichkeiten")
- Client - Server, P2P (vorallem nicht-funktionale Anforderungen im Vordergrund)
- Reflection (Meta - Basis - Info)
  Anfrage → Metainfo nachschlagen
  Antwort ← Basis info suchen

Entwurfsmuster

Composite



State



↳ alles Wiederverwendung

⇝ Was wird eigentlich Wiederverwendet?

- Quellcode ← Lowlevel — Highlevel → Entwurfsmuster
- Strukturen — Hierarchie — „Ideen" von Lösungen
- Architekturen

und letztendlich

- Frameworks

Analyse muster (noch eine Abstraktionsebene weiter)

- Messung

... to be contid..

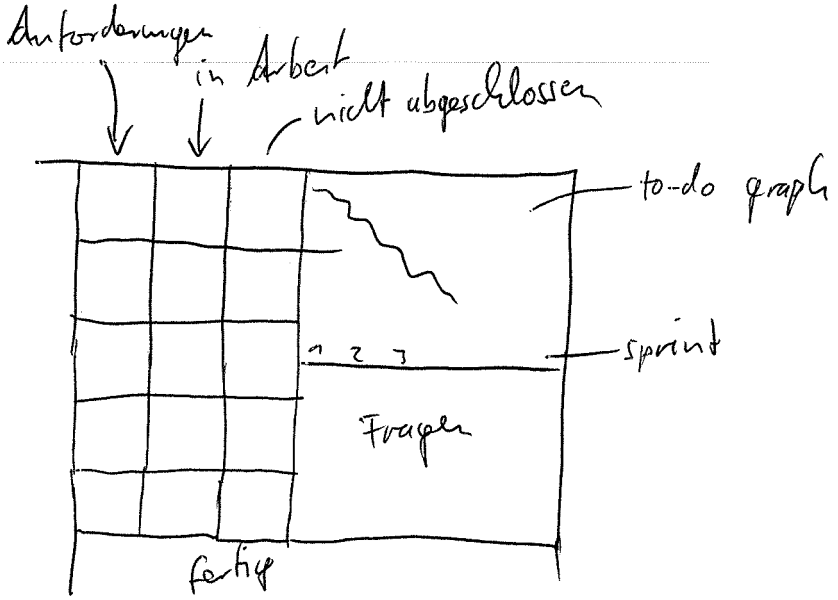
Risiken

- Architektur muss bekannt sein, können nicht geändert werden (Einarbeitungszeit, Folgekosten, ... ?)

Softwareentwicklung - Wiederverwendung von Methoden

- Code & Fix
- Linear
- nicht-Linear („Wasserfall" ↘)
  - ↳ iterativ-inkrementell
  - ↳ extreme programming } agile Softwareentw.
  - ↳ Scrum

Whiteboard:



QA:

pair-programming

Entwurf > 10%

Testing > 50%

Entwurf ist Königsdisziplin

Analyse → konvergent
Entwurf → Divergent
„neue Erfahrung"

- Testfälle in Analysephase "Test-Driven-Design"

- Anforderungsmanagement

  Weitblick → langsam

  Prefactoring → nur die Anforderung umsetzen, die Bekannt ist

  neue Anforderungen ⇝ neue Strukturen

  ggf. neu anfangen...

  vgl. "Anti-Patterns"