Fakultät Informatik/Mathematik
Lehrstuhl für Softwaretechnologie

HOCHSCHULE FÜR
TECHNIK UND WIRTSCHAFT
DRESDEN
UNIVERSITY OF APPLIED SCIENCES

Vorlesungsmitschrift

SOFTWARE ENGINEERING 2

Mitschrift von
Falk-Jonatan Strube
Vorlesung von
Prof. Dr. Anna Sabine Hauptmann

27. März 2018

INHALTSVERZEICHNIS

TEIL I.

VORLESUNG

EINFÜHRUNG

BELEGARBEIT

VO01.pdf
Folie 3

Im Protokoll (bei Bedarf) nicht vergessen:

- Teilnehmende (entschuldigt/unentschuldigt)
- Festsetzungsprotokoll (wer macht was bis wann?)

1. ENTWURF, ARCHITEKTUR, RISIKEN

Vorlesung
21.03.2017

1.1. RÜCKBLICK

VO01.pdf
Folie 7

VO01.pdf
Folie 8

1.2. PROZESS UND PRODUKT

VO01.pdf
Folie 10

1.3. STRUKTUR

VO01.pdf
Folie 11

1.4. GROB- UND FEINENTWURF

VO01.pdf
Folie 12

1.5. SYSTEM VS SOFTWARE-SYSTEM

VO01.pdf
Folie 13

VO01.pdf
Folie 21

1.6. KOMPLEXITÄT

VO01.pdf
Folie 14

1.7. RISIKO BEIM SOFTWARE-ENTWURF

VO01.pdf
Folie 18

1.8. FRAGEN BEI DER SOFTWARENTWICKLUNG

VO01.pdf
Folie 22

1.9. ZUSAMMENFASSUNG

VO01.pdf
Folie 23

2. MODULE, SCHNITTSTELLEN, ENTWURF, ARCHITEKTUR

Vorlesung
28.03.2017

VO02.pdf
Folie 3

2.1. KOMPONENTEN UND MODULE

VO02.pdf
Folie 4

VO02.pdf
Folie 5

2.2. SCHNITTSTELLEN

VO02.pdf
Folie 8

VO02.pdf
Folie 9

2.3. ENTWURF: PRINZIPIEN UND PERSPEKTIVEN

VO02.pdf
Folie 10

VO02.pdf
Folie 17

2.4. ARCHITEKTUR

VO02.pdf
Folie 11

2.4.1. DREI-SCHICHTEN-ARCHITEKTUR VERSUS PHYSISCHE VERTEILUNG

VO02.pdf
Folie 12

VO02.pdf
Folie 13

→ Die Drei-Schichten-Architektur sagt nichts über die physische Verteilung der Schichten aus.

2.4.2. MODEL-VIEW-CONTROL PARADIGMA

VO02.pdf
Folie 14

ENTKOPPLUNG VON VIEW UND MODEL

VO02.pdf
Folie 16

3. ENTWURFSPRINZIPIEN

3.1. SYSTEMSTRUKTUR

VO03.pdf
Folie 2

3.2. STRUKTUR IM INNEREN DES SYSTEMS

VO03.pdf
Folie 3

- Kopplung: Einfluss von „Außen“
- Kohäsion: Einfluss von „Innen“ (untereinander)

VO03.pdf
Folie 5

3.3. UNTERSTÜTZUNG VON LOSER KOPPLUNG / INNERE BINDUNG

VO03.pdf
Folie 6

Vergleich: Aufteilung in Klassen.

VO03.pdf
Folie 10

3.4. KOPPLUNG UND KOHÄSION

3.4.1. KOPPLUNG

VO03.pdf
Folie 8

3.4.2. KOHÄSION

VO03.pdf
Folie 9

3.5. BESCHREIBUNG EINER PROGRAMMIERSCHNITTSTELLE

(bspw. Klassenebene)
Jede öffentliche Operation/Funktion braucht:

- Signatur: Syntax – `____ name (...)`
- Wesen: Semantik – Was die Operation tut.
- Vorbedingungen
- Nachbedingungen
- Invariante (nicht veränderlich)
- Protokoll (synchron/asynchron)
- Kosten (Zeit, Rechenleistung, Dienste[→Geld], . . .)

→ Anforderungen (funktional [→ Signatur / Wesen], Qualitätsanforderungen [→ Vor-/Nachbed. / Invariant / Protokoll], Rahmenbedingungen [→ Kosten])

3.6. LANGLEBIGE STRUKTUR IM INNEREN SYSTEM

VO03.pdf
Folie 11

→ Open/Closed Prinzip

3.7. GRUNDLEGENDE ENTWURFSPRINZIPIEN

VO03.pdf
Folie 12

4. MUSTER

Vorlesung
11.04.2017/2

VO04.pdf
Folie 9

VO04.pdf
Folie 5

VO04.pdf
Folie 6

Kategorien von Mustern:

- Architektur-Unabhängig:
  - Architektur-Muster (im Grobentwurf)
  - Entwurfs-Muster (im Feinentwurf, näher an der Implementierung)
- Für bestimmte Implementierungen (abhängig von der Sprache):
  - Idiom

4.1. GRUNDLEGENDE ARCHITEKTUREN

4.1.1. SCHICHTEN / LAYER

VO04.pdf
Folie 14

4.1.2. DATENFLUSS UND FILTER / PIPES-AND-FILTERS

Daten fließen durch Filter-Komponenten (Beispiele: Compiler, Converter, . . .).

4.1.3. DEPOT-ARCHITEKTUR

In einem Depot sind Informationen, in Anwendungen/Teilsystemen werden diese dargestellt oder verarbeitet (Vergleich: objectiF/case4.0 . . . Klassendiagramm und Sequenzdiagramm greifen auf Objekte zurück [beim Löschen ist bspw. dann die Frage: Soll die Representation oder das Objekt im Depot gelöscht werden]).

4.1.4. CLIENT-SERVER-ARCHITEKTUR

Weiterentwicklung der Depot-Architektur: Das Depot verarbeitet nun auch Informationen (bekommt Funktionalität).

4.1.5. PEER-TO-PEER-ARCHITEKTUR

Spezielle Client-Server-Architektur, wo der Client und Server eine Komponente sind.

4.1.6. PROXY-PATTERN (VERMITTLER/BROKER)

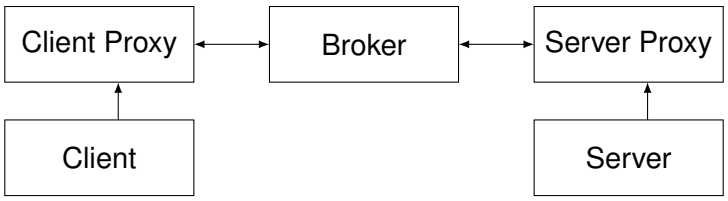
Vorlesung 18.04.2017

Kontext, Problem Netzwerk von Clients und Servern, die Informationen austauschen sollen.

Lösungsstrategie Eine spezielle Komponente übernimmt die VERMITTLUNG zwischen Clients und Server(n)

Konsequenzen

- Effizienz sinkt ↓
- Fehleranfälligkeit steigt ↑
  ⇒Fehlertoleranzmaßnahmen (redundant, Prüf-Funktionalitäten)
- Kommunikationsaufwand steigt ↑



4.1.7. SÄULENARCHITEKTUR

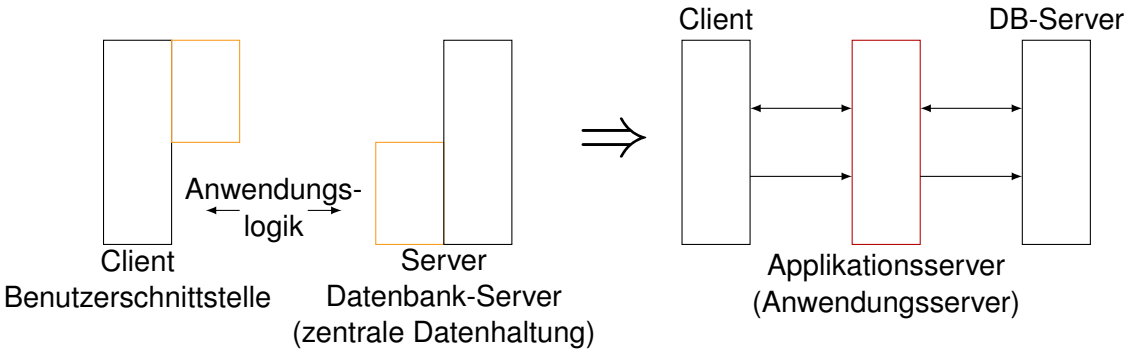
Client (Benutzerschnittstelle) ⟷ Anwendungslogik ⟷ Server (Datenbank-Server)

4.1.7.1. 3-SÄULEN-ARCHITEKTUR

Client ⟷ Applikationsserver (Anwendungsserver) ⟷ DB-Server

4.1.7.2. 4-SÄULEN-ARCHITEKTUR

Client ⟷ Formularerstellung ⟷ Applikationsserver (Anwendungsserver) ⟷ DB-Server



4.1.8. REFLECTION

Kontext, Problem REFLEXIVES SYSTEM, d.h. dynamische Veränderung von Struktur + Verhalten

Lösungsstrategie

- Metaebene → Informationen über spezielle Systemeigenschaften
- Basisebene → Anwendungslogik

⇝ Veränderung in der Metaebene beeinflussen die Basisebene

4.1.9. QUASAR

Vorlesung
25.04.2017

Referenzarchitektur
Wurde entwickelt bei der Firma „sd&m“ (1982-2001, seit 2001 bei „Capgemini“).

Ziel

- Trennung von Zuständigkeiten realisieren
- Programmierung gegen Schnittstellen realisieren
- Denken in Komponenten fördern

Trennung von Zuständigkeiten Ansatz bei Quasar: SW-Komponenten klassifizieren durch Definition von SW-KATEGORIEN

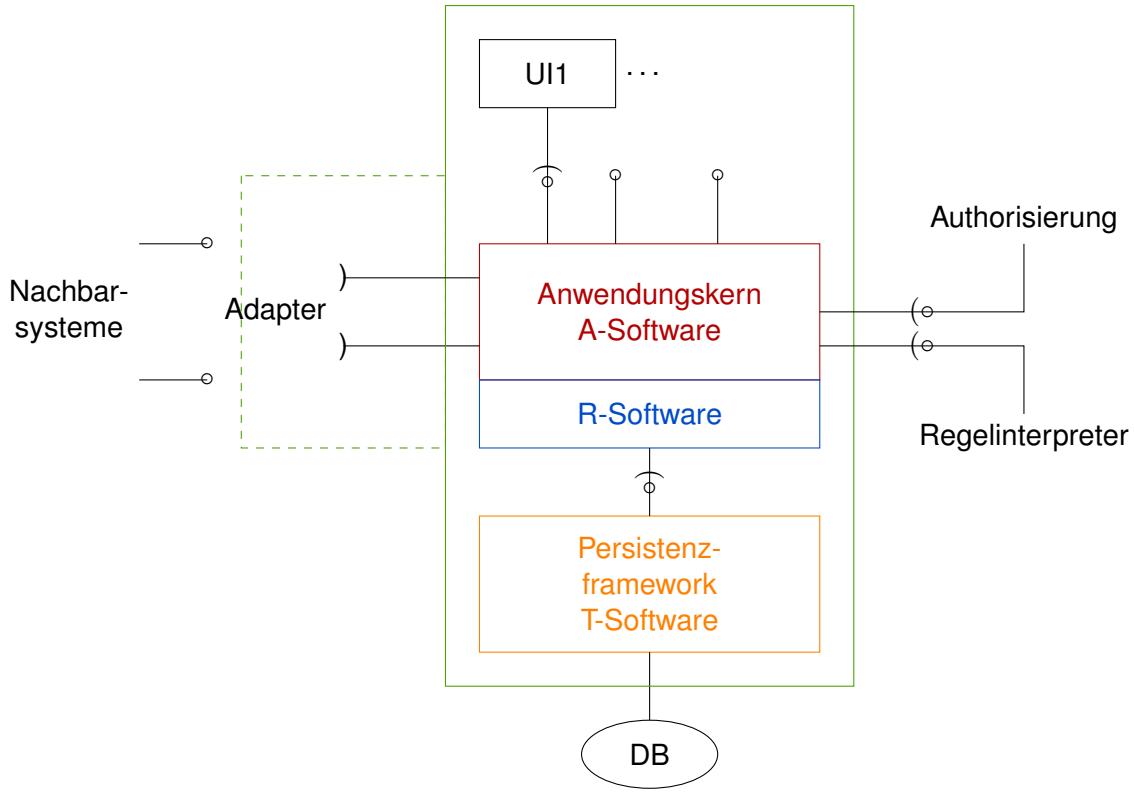
SW-Kategorien → Trennung von ANWENDUNG und TECHNIK

- Kategorie A-Software
  von fachlicher Anwendung bestimmt
- Kategorie T-Software
  technische Komponenten, von mindestens einer API bestimmt (DB-anschluss, Technologie der Oberflächenprogrammierung, . . .)

weitere:

- Kategorie AT-Software (möglichst vermeiden!)
  Mischung aus A- und T-Komponenten
- Kategorie R-Softwaree
  reserviert für Bausteine, die als Transformator zwischen A- und T-Software liegen

Einfache Darstellung:



Detaillierte Darstellung:

VO06Quasar.pdf
Folie 1

5. ENTWURFSMUSTER

Vorlesung
02.05.2017

5.1. OBJEKTORIENTIERTE PROGRAMMIERUNG

- Instanz (in der Realität) einer abstrakten Bauanleitung
- Kapselung
- Vererbung
- Schnittstelle wird benutzt (→ Kapselung)

→ Zusammenführung von Daten und Methoden.

VO06.pdf
Folie 3

5.1.1. OBJEKT

VO06.pdf
Folie 5

VO06.pdf
Folie 6

VO06.pdf
Folie 7

5.1.2. VERERBUNG

VO06.pdf
Folie 8

VO06.pdf
Folie 9

→ Polymorphie (mit früher/SPÄTER Bindung)

5.1.2.1. SUBSTITUTIONSPRINZIP

VO06.pdf
Folie 10

5.1.2.2. CHANCEN UND GEFAHREN DER VERERBUNG

VO06.pdf
Folie 11

5.2. ENTWURFSMUSTER

VO06.pdf
Folie 12

VO06.pdf
Folie 14

VO06.pdf
Folie 15

5.2.1. ERZEUGERMUSTER

VO06.pdf
Folie 16

Vorlesung
09.05.2017

Problem Komplexe Objekte erstellen.

Kontext Erzeugung und Darstellung eines Objektes getrennt (Methode der Erzeugung interessiert Client nicht): Konvertierung von Zeichenketten, Erzeugung von Fensteranordnungen [bspw. in Eclipse].

Lösungsstrategie Ein „Direktor“ delegiert das Erbauen an einen „Erbauer“, der das Produkt dann ausgibt.
$\rightarrow$ einfache Erweiterung und gute Kapselung

5.2.2. SINGLETON

Problem absichern, dass nur ein Objekt instanziiert wird

Kontext Zugriff auf zentrale HW-Ressourcen (z.B. eine Instanz einer Drucker-Warteschlange)

Lösungsstrategie Anzahl von Instanzen zählen (wenn 0, dann instanziieren – sonst nicht).

- Variable, die den Wert aufnimmt
- Kontrollierte Instanziierung
  - im Standardkonstruktor prüfen

```
Standardkonstruktor privat
GibInstanz(){
  // prüfen
  // Aufruf des Standardkonstruktors, abhängig vom Prü
     fergebnis
}
```

Singleton
- instanz: singleton
- singleton(); + GibInstanz(): singleton;

5.2.3. PROTOTYP

Problem Ein Sachverhalt → viele Ausprägungen.

Kontext Erzeugung eines neuen Dokuments in Word → Dokument wird nach Prototyp (Vorlage) erzeugt → Vorlage zur Serienfertigung.

Lösung Nur eine Klasse → eine Instanz gespeichert → beliebig viele Kopien davon können erstellt werden.

Prototyp
+ clone();

5.2.4. FABRIKMETHODE

Problem verschieden Produkte mit ähnlichem Aufbau

Lösung (abstrakte) Klasse, die einen Grundaufbau für alle Produkte enthält.

TEIL II.

PRAKTIKUM

1. EINFÜHRUNG MVC

INHALTE

- Wiederholung objectiF
  Persektive: Entwurf → technische Sicht
  Klassendiagramm → statische Struktur
- Darstellung des VERHALTENS (Dynamik, Progress)
  → Sequenzdiagramm
- Konkretes Beispiel: MVC

NOTIZEN

- `Controller` und `View` hängen immer zusammen, da die Art, wie ich „kontrolliere“ abhängig ist von der Art, wie mir zu Kontrollierendes angezeigt wird.
- Das Klassendiagramm(KD) zeigt mögliche Botschaftswege.
  Das Sequenzdiagramm(SD) zeigt die zeitliche Reihenfolge konkreter Botschaften (d.h. Methodenaufrufe).
- Modellelemente:
  - Instanz (Objekt) als Lebenslinie
    → rot: zugeordnete Klasse), schwarz: ohne zugeordnete Klasse
    → verdickt: wenn Instanz an Botschaft beteiligt ist
  - Botschaft
  - Systemgrenze

2. NOTENVERWALTUNG

1. Methodik:

   Modell (objectiF) $\xrightarrow{\text{Forward Engineering}}$ $\xleftarrow{\text{Reverse Engineering}}$ Implementation (MS VisualStudio .NET)

   $\Rightarrow$ Roundtrip-Verfahren

2. Architektur:
   3-Schichten-Architektur
   - Präsentation
   - Logik
   - Datenhaltung

   Im objectiF: Schichten sind Pakete $\rightarrow$ Paketdiagramm
   Im VisualStudio: Schichten sind Projekte $\rightarrow$ Projektmappe
   Vereinbarung: Verbindung der Schichten zwischen objectiF und VisualStudio außer bei der Präsentation

Vorlesung
13.04.2017

- Dialog bauen $\rightarrow$ MS Visual Studie .NET
- KD: DBService $\rightarrow$ objectiF
- Paket Typen (Student, Fachnote) $\rightarrow$ objectiF
- Synchronisation:
  Modell – Implementation objectiF $\rightarrow$ MS Visual Studio