LMU Ludwig-Maximilians-Universität München

Institut für Informatik LMU

Lehr- und

Heute:
- Restliche Folien zu Vorgehensmodellen
- Systemkonstruktion

Vorlesung im Wintersemester 2007 / 2008

Juristisches IT-Projektmanagement

Notwendige Vorbereitungen für komplexe IT-Projekte
Vertragsorientiertes Projektmanagement
Sanierung von IT-Projekten in der Krise

Dr. Frank Sarre
Lehrbeauftragter der LMU München

Terminplan (vorläufig)

Nr.	Datum	Thema
1	18.10.2007	Einführung und Grundbegriffe
2	24.10.2007	Systematische Projektdurchführung
3	31.10.2007	Systemkonstruktion
4	7.11.2007	**Fällt leider aus wegen dienstlicher Abwesenheit**
5	14.11.2007	Vertragstypen
6	21.11.2007	Projektmanagement
7	28.11.2007	Aktivitäten- und Fristenplan, Dokumentation, Quellcode
8	5.12.2007	Das Pflichtenheft
9	12.12.2007	Öffentliche Vergabe von IT-Leistungen
10	19.12.2007	Test und Abnahme von IT-Leistungen
	~~26.12.2007~~	**Weihnachtspause**
	~~2.1.2007~~	**Weihnachtspause**
11	9.1.2008	Mögliche Leistungsstörungen
12	16.1.2008	Gerichtlich verwertbare IT-Gutachten
13	23.1.2008	Sanierung von IT-Projekten
14	30.1.2008	Lessons Learned
15	6.2.2008	Gastvortrag: Claim Management

V-Modell ® XT

Im Internet zu finden unter www.vmodellxt.de (KBSt)

- **Nachfolgemodell** zum bekannten **V-Modell ´97**
- Nun überarbeitet durch TU München, TU Kaiserslautern, EADS, IABG und Siemens AG
- Für **öffentliche Auftraggeber** empfohlen

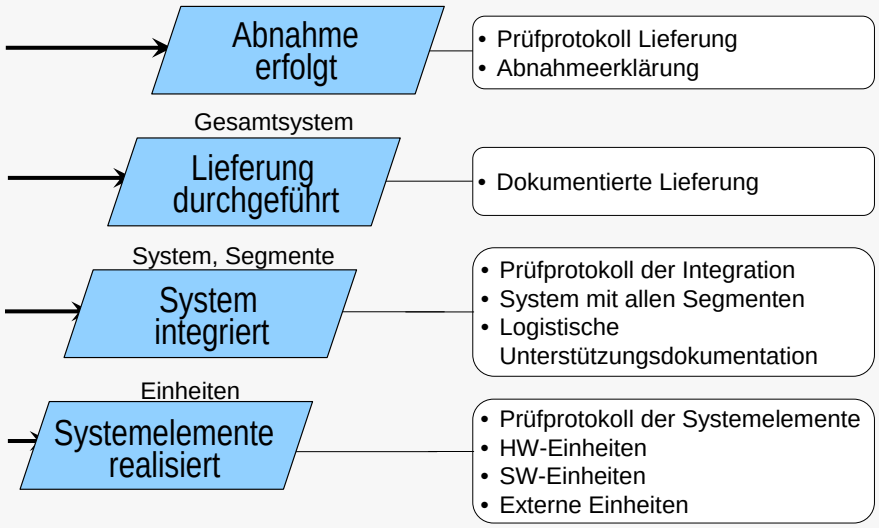
Das V-Modell ® XT enthält:

- Beschreibungen für alle **Projektergebnisse** mit allen **Abhängigkeiten** untereinander
- **Vorgehensweisen** für alle Ergebnisse in allen Projektabschnitten, auch detaillierte Beschreibung von **Aktivitäten**
- **Verantwortlichkeiten** / **Rollen** aller Beteiligten

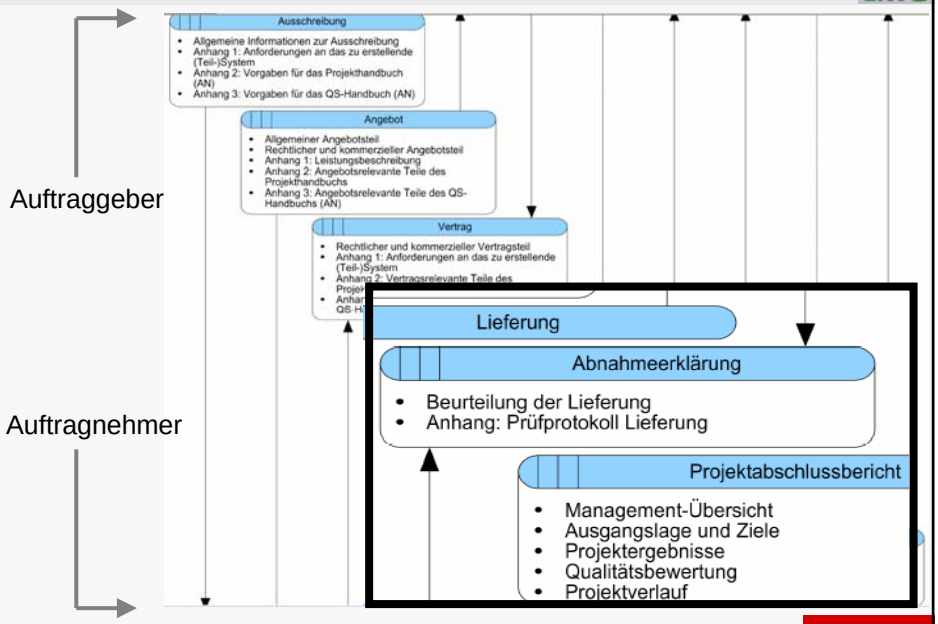
Kernpunkte der V-Modell ® XT Philosophie

- **Projektergebnisse** sind der Dreh- und Angelpunkt des Modells (hier „Produkte“ genannt)
- **Projektdurchführungsstrategien** und Entscheidungspunkte geben die Reihenfolge der Produktfertigstellung und somit die grundlegende Struktur des Projektverlaufs vor
- Die detaillierte **Projektplanung und -steuerung** wird auf der Basis der Bearbeitung und Fertigstellung von Produkten durchgeführt.
- Für jedes Produkt ist eindeutig eine **Rolle** verantwortlich und im Projekt dann eine der Rolle zugeordnete Person
- Die **Produktqualität ist überprüfbar** durch definierte Anforderungen an das Produkt und explizite Beschreibungen der Abhängigkeiten zu anderen Produkten

Entscheidungspunkte und Ergebnisse



Schnittstelle Auftraggeber / Auftragnehmer



Auswirkungen auf IT-Verträge

- Das V-Modell ® XT (2006) wird sich als Bestandteil der EVB-IT Vertragsmuster für **alle IT-Projekte der öffentlichen Hand** immer stärker durchsetzen.
- Da ca. 50% des gesamten IT-Projekt-Volumens in Deutschland von der öffentlichen Hand vergeben wird, ist anzunehmen, dass sich das Modell (oder Abwandlungen) **auch in der privaten Wirtschaft** etablieren wird.
- Das **konkrete Vorgehen** im Projekt sollte in jedem Fall durch Verfeinerung / **Tailoring** des V-Modells ® XT genau definiert werden. Tools helfen dabei.
- **IT-Vertrag** und **konkretisiertes Vorgehensmodell** sowie die geplante Art des **Projektmanagements**, **Qualitätsmanagements** und **Änderungsmanagements** sollten **eng verzahnt** werden.

Alternative Vorgehensmodelle

1. Das Spiralmodell
2. Das iterative Phasenmodell mit Prototypen
3. Evolutionäre Softwareentwicklung
4. Agile Modelle

Bewertung der Vorgehensmodelle

Anforderungen	Projektgröße / -dauer			Bekanntheit von Anforderungen		Änderungen an den Anforderungen			Zeit-rahmen
Ausprägungen	klein	komplex	lang	klar	unklar	keine	moderat	häufig	Hoher Zeitdruck
Wasserfall	+	-	-	+	-	+	o	-	-
V-Modell	-	+	o	+	-	+	+	-	-
Spiralmodell	o	+	+	+	+	+	+	o	-
Inkrementell	o	+	+	+	-	+	+	-	+
Evolutionär	o	+	+	o	+	o	o	+	+
Mit Prototyp	-	+	+	-	+	+	+	+	+

→ **Das zu verwendende Vorgehensmodell muss projektspezifisch ausgewählt werden!**

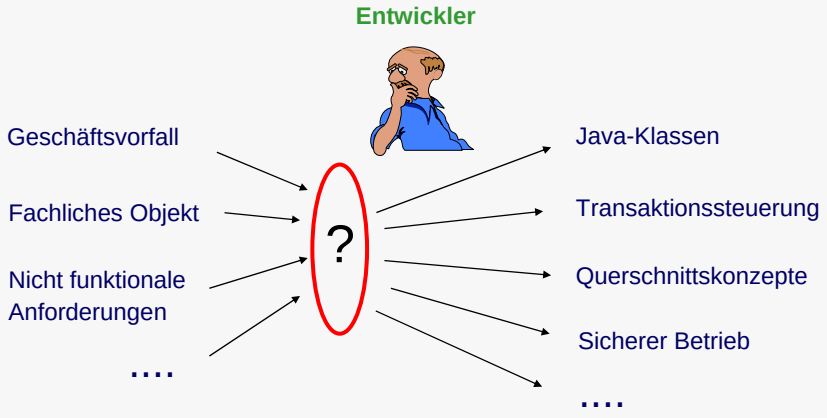
→ **In jedem Projekt muss ein ausgewähltes Vorgehensmodell auf die herrschenden Gegebenheiten angepasst werden!**

Vorlesung am 31.10.2007

Systemkonstruktion

Übergang vom Fachkonzept zum DV-Konzept

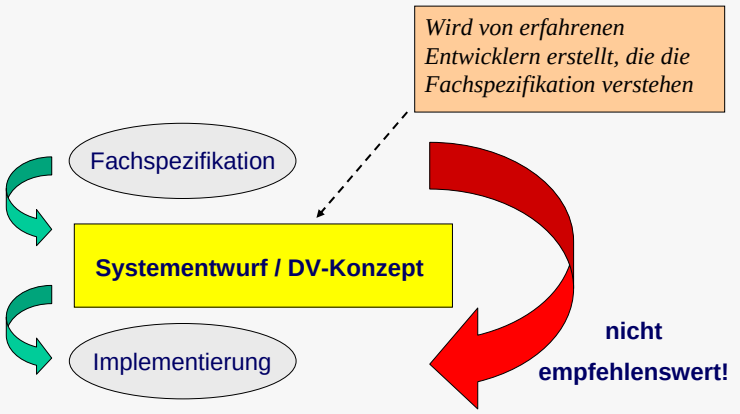
Aufgabenstellung



Bedeutung des DV-Konzepts

Das DV-Konzept dient als Beschreibung des Vorgehens, wie die Fachspezifikation später in der Realisierung umgesetzt werden soll.

Das DV-Konzept ist auch als „Bauplan“ des zukünftigen Systems zu verstehen.



Folgen einer fehlenden Systemkonstruktion

Beispiel

(tatsächlich passiert, Multi-Millionenprojekt im Mobilfunkbereich im Juli 1995):

- Grobe Probleme bei der Installation
- System stürzt oft ab
- Datenbank wird inkonsistent
- Multi-User-Betrieb kaum möglich
- Performance miserabel
- In der ersten Woche mehr als 200 gefundene Fehler

Typische Inhalte eines DV-Konzepts (1)

Architektur

- Darstellung der Systemstruktur (Komponenten, Zusammenspiel)
- Schichtenmodell
- Beschreibung einzelner Komponenten
- Fremdmodule
- Schnittstellenbeschreibungen, -techniken und –kontrakte
- Aspekte der Verteilung
- Prozessmodell

Typische Inhalte eines DV-Konzepts (2)

Programmierkonzepte

- Umsetzung von Anwendungsfällen und geforderten Funktionen
- Umsetzung nicht funktionaler Anforderungen
- Physisches Datenmodell (Datenbankdesign)
- Beschreibung betriebsrelevanter Techniken
- Batches (mit Steuerung)

Typische Inhalte eines DV-Konzepts (3)

Technische Fragestellungen

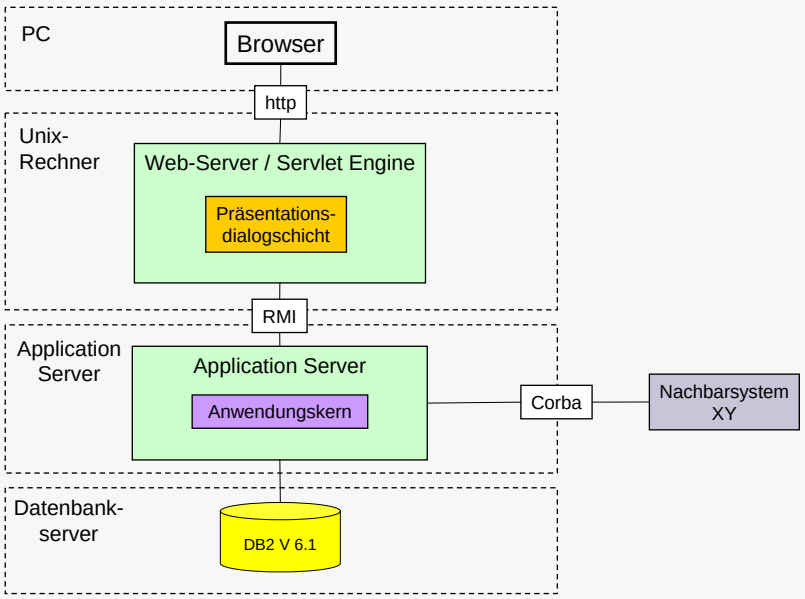
- Dialogkonzept / GUI-Programmierung
- Parallelverarbeitung
- Druckthematik
- Archivierung
- Historisierung
- Datensynchonisation
- Sicherheit (Echtheit, Verschlüsselung, …)
- Datenhaltung / Anbindung der Datenbank an den Anwendungskern
- Multi-User-Betrieb
- Austausch von Daten über Rechnergrenzen hinweg (Kommunikation)
- Monitoring des Systems (u.a. Performance)
- Fachliches Accounting
- Technisches Logging
- Workflow
- Fehlerbehandlung
- Transaktionskonzept
- Berechtigungskonzept

Typische Inhalte eines DV-Konzepts (4)

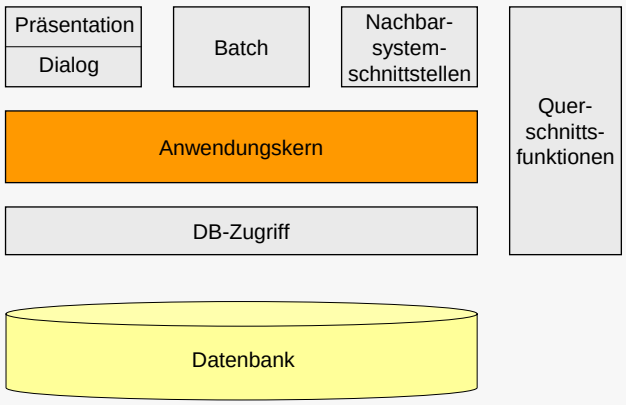
Weitere Themen (unsortiert):

- Erfahrungen aus Prototypen?
- Einstellmöglichkeiten des Systems?
- Platzanforderungen / Mengengerüste
- Datenmigration
- Notbetrieb
- Berücksichtigung von Richtlinien und Standards
- Vorgaben der Systeminfrastruktur?
- Infrastruktur von Testumgebung, Produktivumgebung, ...
- Einsatz von Entwurfsmustern
- „Release-Fähigkeit“
- Integrationsstrategie
- Anlaufplan zur Produktivsetzung
- Werkzeuge (Testdatengeneratoren, Testwerkzeuge, ...)
- Testbarkeit von Einzelkomponenten
- Wiederverwendbarkeit

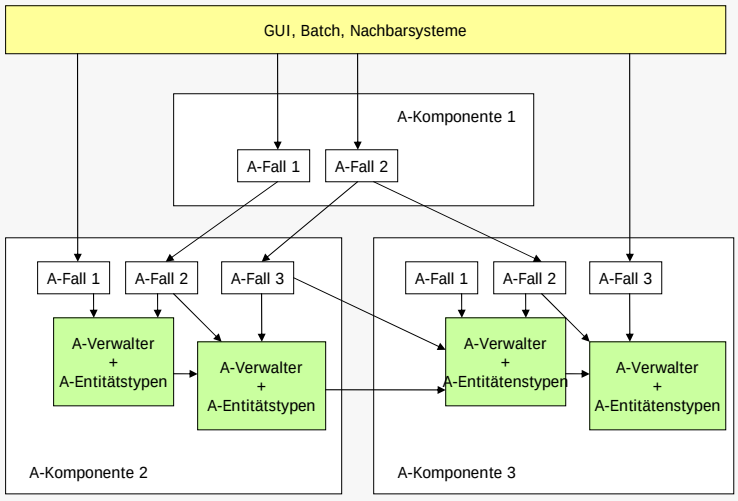
Architektur der technischen Infrastruktur



Übliche Dreischichtenarchitektur



Eine mögliche Anwendungskernarchitektur



Architektur des Anwendungskerns nach Quasar (Domäne „Betriebliche Informationssyteme“)

Anwendungsobjekte

Beispiel

```
public class Skilehrer extends Person implements ISkilehrer,
   Serializable
{
    private Id id;
    private Verfuegbarkeit verfuegbarkeit;
    //...
    public Skilehrer(Adresse adresse,
                     Datum geburtsdatum,
                     String name,
                     Sprache sprache,
                     Kurstyp kurstyp) { ... }

    public Id getId() { return id; }
    public Verfuegbarkeit getVerfuegbarkeit(){
       return verfuegbarkeit;}

    public boolean equals(Object x) { ... }
    …
}
```


Trennung der Zuständigkeiten

Auch bekannt unter „Separation of concerns“

- **Jede Softwarekomponente bzw. jedes Softwaremodul sollte sich möglichst nur mit einer (technischen oder fachlichen) Aufgabe befassen**

→ Klarer Code
→ Verständliche Architektur
→ Bessere Wartbarkeit
→ Möglichkeit, wiederverwendbare Komponenten zu identifizieren
→ Kapselung von herstellerabhängigen APIs, um Austauschbarkeit von Produkten zu erreichen

Software-Kategorien nach Siedersleben

Auch bekannt unter: „Software-Blutgruppen“

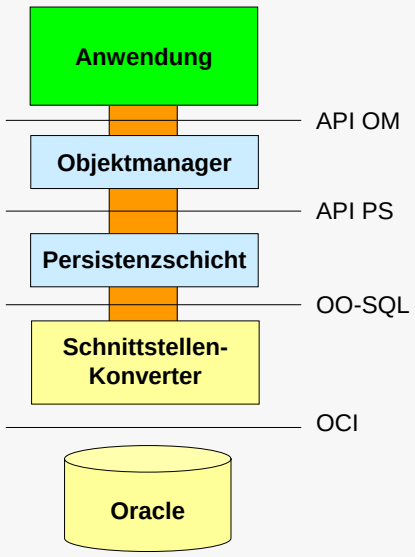
Software kann sein ...

0	bestimmt von gar nichts (Behälter, Strings) → *ideal wiederverwendbar, für sich alleine nutzlos*
A	bestimmt von der Anwendung (Kunde, Auftrag, Bestellung) → *das eigentliche Projektziel*
T	bestimmt von mindestens einem technischen API (z.B. Datenverwaltung) → *muss sein*
AT	bestimmt von der Anwendung und mindestens einem technischen API → *vermeiden; im Notfall sorgfältig abgrenzen*
R	Repräsentationssoftware (Transformation zwischen A und T; milde Art von AT)

Kombinationen
A + 0 = A
T + 0 = T
A + T = AT

Schnittstellen und Schichten

Beispiel



Schnittstellen aus Entwicklersicht

Warum sind Schnittstellen wichtig?

- Schnittstelle = Vertrag zwischen Nutzer und Anbieter
  → Der Anbieter ist austauschbar (ohne dass dies der Nutzer merkt)

Schnittstellen ...

- helfen, Abhängigkeiten zu reduzieren
- „verstecken“ Komplexität
- unterstützen die Entwicklung von Software im Team
- können problematisch werden, wenn sie häufigen Änderungen ausgesetzt sind

Fehlerbehandlung (1)

Was kann alles passieren?

- Fachliche Probleme
  - Konto nicht gedeckt
  - Es ist nicht die notwendige Berechtigung vorhanden
- Verletzte Vorbedingungen
  - Es wurde ein falscher Parameter übergeben
  - Die Buchung ist bereits storniert
- Technische Probleme
  - Netz temporär nicht verfügbar
  - Datenbank meldet unbekannte Fehlercodes zurück
  - Nachbarsysteme verhalten sich unerwartet
- „Hausgemachte“ Probleme
  - Programmierfehler
  - NullPointerException, ClassCastException, ….

Fehlerbehandlung (2)

Probleme mit Exceptions und Fehlern

- Welche Kategorien von Ausnahmen gibt es?
- Wie wird ein „Wildwuchs“ von Ausnahmen verhindert?
- Sind Ausnahmen von herkömmlichen Fehlern zu unterscheiden?
- Wer hat das Recht, Ausnahmen zu setzen?
- Wie werden „normale Return-Codes“ von Ausnahmen unterschieden?
- Wer hat das Recht bzw. die Pflicht, Ausnahmen zu fangen und zu behandeln?
- Was macht man in einer Ausnahmebehandlung mit einer unbekannten Ausnahme?
- Ab wann macht eine Fortführung des Programms keinen Sinn mehr?
- Wo liegen die Fehlermeldungstexte?

Fehlerbehandlung (3)

Anwendungsfehler ….

- haben nichts mit „Notfällen“ zu tun
- müssen vollständig spezifiziert werden
- werden üblicherweise über Return-Codes gemeldet
- werden unmittelbar vom Aufrufer (in der Anwendung) behandelt
- Der Rufende entscheidet letztlich, was ein echtes Problem ist und was nicht!