Einführung in Software Engineering

Barbara Paech, Marcus Seiler

Institute of Computer Science
Im Neuenheimer Feld 326
69120 Heidelberg, Germany
http://se.ifi.uni-heidelberg.de
paech@informatik.uni-heidelberg.de

Kapitel 3: SWE im Großen – Kommunikation mit den KundInnen

Wdh (Folien 01) : Was heißt…

- Groß (Lines of Code, LOC)
  - Typische APP (Android)
    - 3.000-20.000
  - Typische Geschäftssoftware
    - z.B. HTTP Apache Server (120.000) - SAP Netweaver (240.000.000)

- Viele Beteiligte
  - NutzerInnen
    - APP Millionen
    - SAP Hunderttausende
  - Entwicklungsteam
    - APP 1-20 (ca. 5 core)
    - HTTP Apache ca. 100 (ca 20 core)

Gewinn!

Kosten!

Große vs. Kleine Software

- **Kleine SW:**
  Leichtgewichtige Entwicklung

Code

Direkte Kommuni-kation

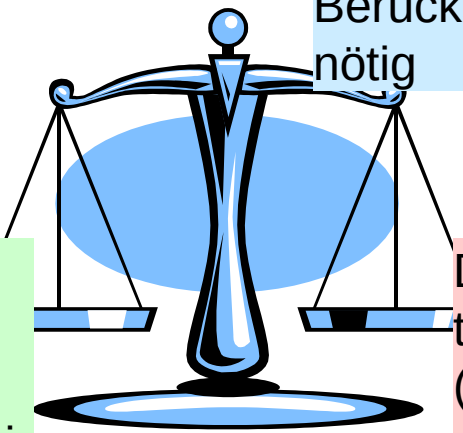
QS

Dokumen-tation (Anforde-rungen, Entwurf)

Verteilte Arbeit

- **Große SW:**
  Langfristiges, verteiltes Wissensmanagement

Ausgewogene Berücksichtigung nötig



Code

Direkte Kommuni-kation

QS

Dokumen-tation (Anforde-rungen, Entwurf)

Verteilte Arbeit

3. SWE im Großen (1. Teil)

- 3.1. Vorgehen
  - SCRUM Agiles Vorgehen im Großen
- 3.2. Einführung Kommunikation mit KundInnen
- 3.3. Nutzungsmodellierung
  - Einführung
  - Aufgaben, Rollen
  - Domänen- und Interaktionsdaten
  - Funktionen und UI-Struktur
  - Persona
  - GUI

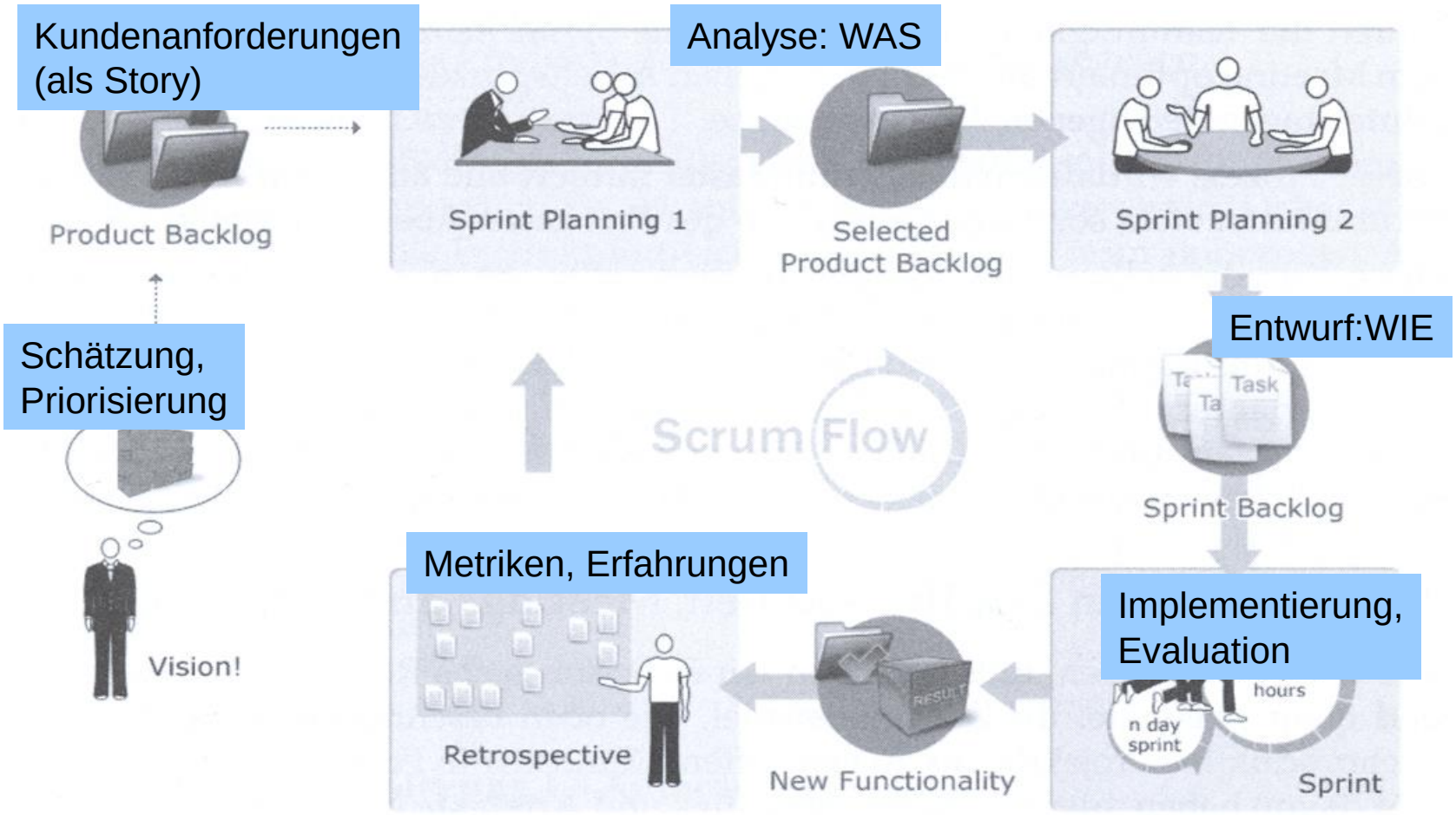
3.1.Scrum

[Gloger]

Weiteres agiles Vorgehen: Scrum

- Change-Management-Ansatz
- MitarbeiterInnen im Vordergrund
- Grundlegende Idee:
  - Software wird **schrittweise** erstellt (in sog. Sprints, auslieferungsfähige Zwischenergebnisse) siehe short increments bei XP
  - Tägliche Treffen: Daily Scrum
  - **Team** ist verantwortlich für Planung und Ergebnisse
- Prinzipien
  - Transparenz: Positives und Negatives
  - Beobachtung und Anpassung: z.B. Code-Reviews, Tests, Sprint-Review
  - Timeboxing: feste Länge für Tätigkeiten, ggf. Scope reduzieren
  - Dinge abschließen: 90% fertig hilft nicht
  - Maximieren von Geschäftswert
  - Teams scheitern nicht
  - Ergebnisorientierung

Scrum-Prozess



Scrum-Rollen

- **ProductOwner (ggf. aus KundIn, NutzerIn, Management)**
  - Repräsentiert KundIn
  - Erstellt Vision der Software
  - Erstellt User Stories
  - Verwaltet und priorisiert Anforderungen im Product Backlog
- **Team**
  - Verantwortlich für Planung und Ergebnisse
- **ScrumMaster (nicht Projektmanager!)**
  - Ermöglicht produktives Arbeiten des Teams
  - Vermittelt zwischen ProductOwner und Team
  - Entscheidet (im Notfall, d.h. wenn keine Einigung im Team)

Scrum-Artefakte

User Stories

- Besteht aus Karte, Konversation und Akzeptanzkriterien
- Karte: 1 Satz zur Anforderung „Als Benutzerrolle X will ich Y tun“
- Konversation: alle Fragen, die zum Verständnis nötig sind
- Akzeptanzkriterien: geben durch Detailanforderungen vor, wann Story implementiert ist

Product Backlog

- Liste aller bekannten User Stories
- Nur die User Stories des aktuellen Sprints sind fest
- User Stories werden priorisiert
- Vor jedem Sprint werden wichtige User Stories zerschnitten, um innerhalb eines Sprints umsetzbar zu sein

Zuordnung zu Kernfragen

	Hohe Qualität	Zufriedene NutzerInnen	Wartbarkeit (EntwicklerInnen)	Kosten/ Zeit
Timeboxing, Dinge abschließen				
Sprints (Beobachtung und Anpassung)				
Daily scrum (Transparenz)				
ProductOwner (Maximierung von Geschäftswert)				
ScrumMaster (Teams scheitern nicht)				
Product Backlog (Ergebnisorientierung)				

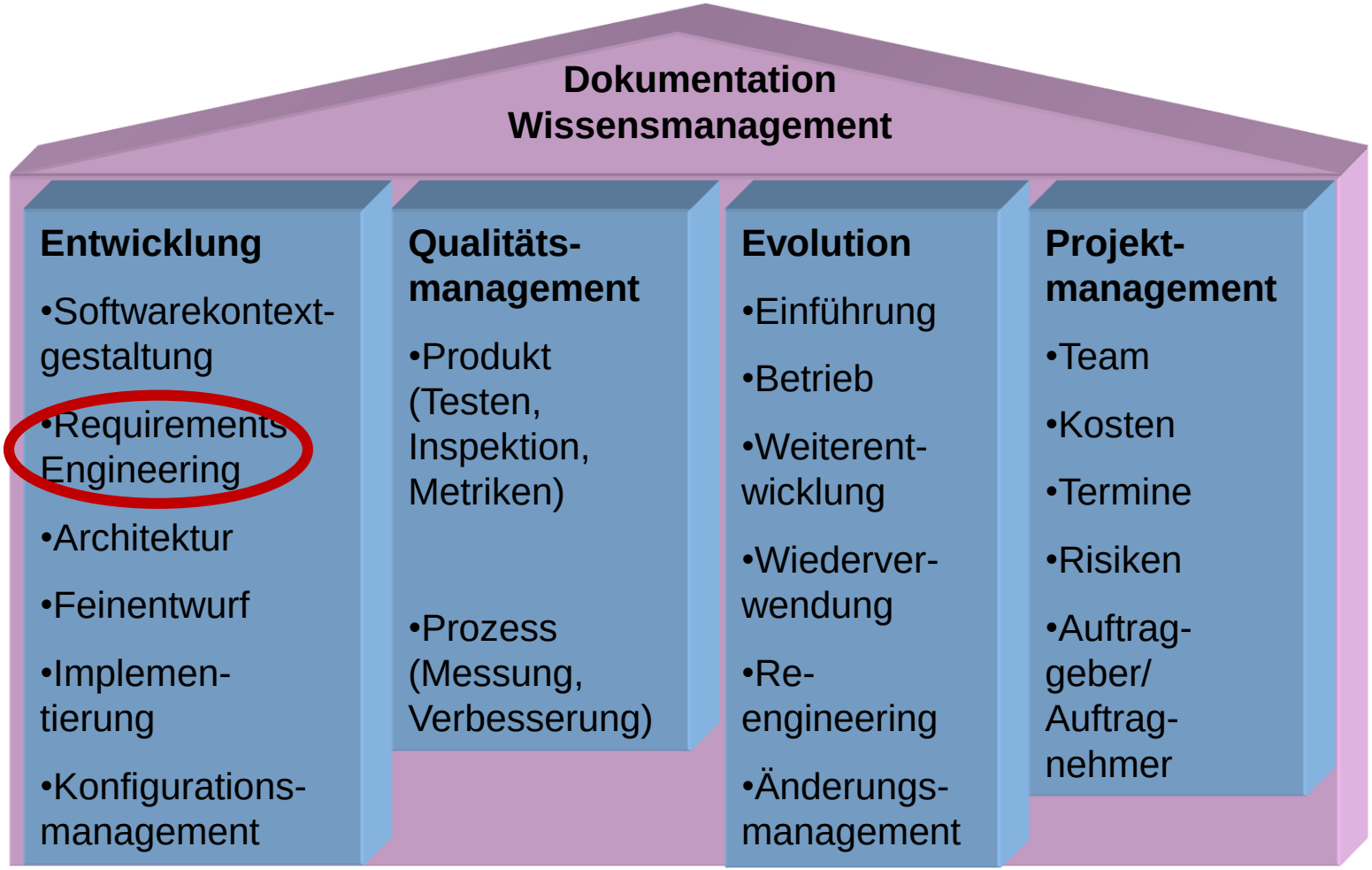
Unterstützung für Entwicklung großer SW

- Umgang mit Zeit: Prinzipien Timeboxing, Dinge abschließen
- KundInnen: eigene Rolle ProductOwner statt KundIn als Ansprechpartner für das Team
- Unterstützung des Teams: eigene Rolle ScrumMaster
- Überblick über große Menge von Anforderungen: Product-Backlog

- Aber immer noch wenig Unterstützung für Langlebigkeit, Wissensmanagement (übergreifende Anforderungs- und Entwurfsdokumentation)

3.2. Einführung Kommunikation mit KundInnen

Aufgabenbereiche des Software Engineering

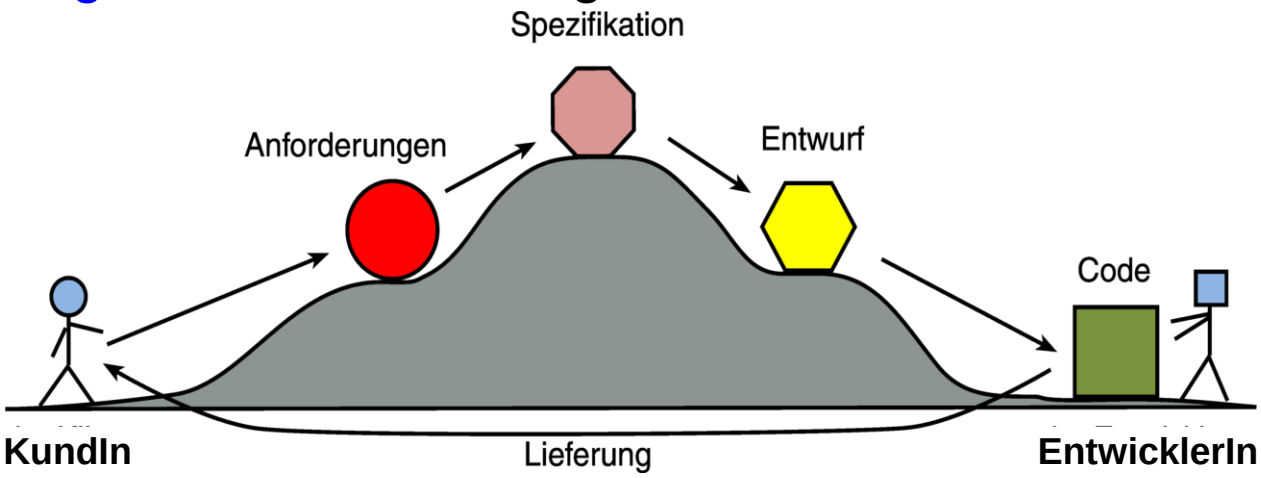


Wdh. Folien04: KundIn und Anforderungen

- KundIn erwartet, dass die Software über einen gewissen Zeitraum hinweg als willliger und billiger Diener zur Verfügung steht, also
  - bestimmte Leistungen erbringt,
  - ohne umgekehrt erhebliche Leistungen (in Form von Kosten, Aufwand, Mühe, Ärger) zu fordern.
    Die Software soll dienen, nicht umgekehrt.
- Werden die Erwartungen, die Anforderungen der/s KundIn, nicht vollständig und präzise erfasst, ist damit zu rechnen, dass das entwickelte Produkt die Anforderungen nicht (vollständig) erfüllt.
- Die vollständige und präzise Erfassung der Anforderungen ist die allerwichtigste technische Voraussetzung für eine erfolgreiche Software-Entwicklung.

Anforderungsaktivitäten im Überblick

- Die Anforderungen werden erhoben, in der Spezifikation formuliert, geprüft und anschließend in den Entwurf umgesetzt. Schließlich wird implementiert, auf verschiedenen Ebenen geprüft und korrigiert. Das Resultat geht zurück an die KundInnen.



- KundInnen bekommen nur dann, was sie haben wollen, wenn ihre Anforderungen sorgfältig erhoben und unterwegs nicht verfälscht wurden.

Der Nutzen der Spezifikation

- In der Praxis gibt es viele schlechte Spezifikationen; oft gibt es gar keine.
- Die Spezifikation ist aber notwendig für

1. die Abstimmung mit den KundInnen bzw. mit dem Marketing,
2. den Entwurf und die Implementierung,
3. das Benutzungshandbuch,
4. die Testvorbereitung,
5. die Abnahme,
6. die Wiederverwendung,
7. die Klärung späterer Einwände, Regressansprüche usw.,
8. eine spätere Re-Implementierung.

Nachteile bei fehlender Spezifikation (1)

1. Die Anforderungen bleiben ungeklärt, sie werden darum auch nicht erfüllt.
2. Den EntwicklerInnenn fehlt die Vorgabe, darum fragen sie „auf dem kurzen Dienstweg“ Bekannte, die bei KundIn arbeiten, oder sie legen mangels Kontakten die eigenen Erfahrungen und Erwartungen zu Grunde.
3. Die Basis für das Benutzungshandbuch fehlt, es wird darum phänomenologisch, d.h. experimentell, verfasst.
4. Ein gutes Benutzungshandbuch ist ein umformulierter Auszug aus der Spezifikation! Darum taugt es auch als Spezifikation.
5. Ein systematischer Test ist ohne Spezifikation unmöglich, denn es ist nicht definiert, welche Daten das System akzeptieren muss und welche Resultate es liefern soll.

Nachteile bei fehlender Spezifikation (2)

6. Wenn bei der Abnahme nicht entschieden werden kann, ob das System richtig arbeitet, wird die Korrektheit zur Glaubensfrage.
7. Oft zeigen sich echte oder vermeintliche Mängel der Software erst nach längerem Gebrauch. Ohne Spezifikation kann diese Unterscheidung aber nicht getroffen werden.
8. Wer eine Software(-Komponente) wiederverwenden will, muss wissen, was sie leistet. Das ist in der Spezifikation dokumentiert.
9. Wenn ein System ausgemustert und ersetzt wird, ist Aufwärtskompatibilität gefordert.

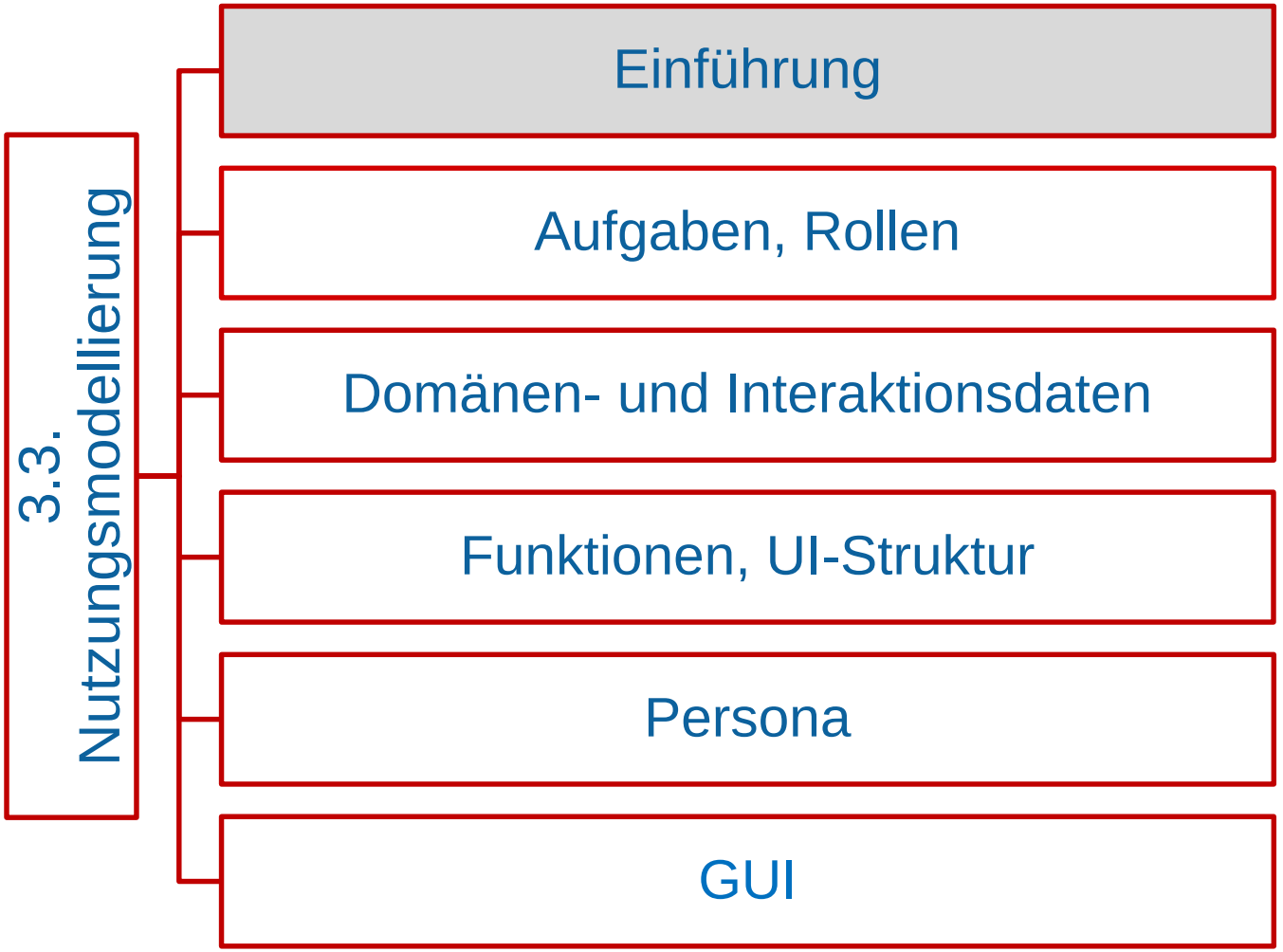
- „Spezifikation im Kopf“ gibt es nicht!

Schwierigkeiten des RE (1)

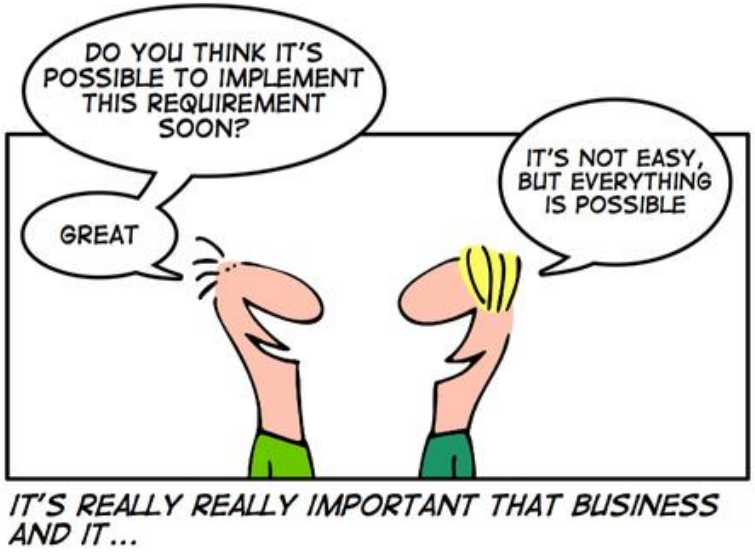
- NutzerInnen können sehr viele Anforderungen nicht nennen, weil sie sie nicht haben. Sie haben nur Wünsche und Ziele, die sich im Gespräch mit AnalytikerInnen auf Anforderungen abbilden lassen.
- EntwicklerInnen haben (bewusst oder unbewusst) eigene Interessen, die sie durchsetzen möchte (eine „hidden agenda“).
- KundIn hat Anforderungen, die er/sie nicht sagen will (also auch eine hidden agenda)
- NutzerInnen haben Anforderungen, die ihnen so selbstverständlich scheinen, dass sie sie nicht erwähnen.
- Am Ende des RE sollte eine Vision entstanden sein, in der
  - KundIn die Erfüllung der (reduzierten) Wünsche,
  - HerstellerIn ein realisierbares Produkt sieht.
- Beschrieben durch ein Dokument evtl. auch durch einen Prototypen.

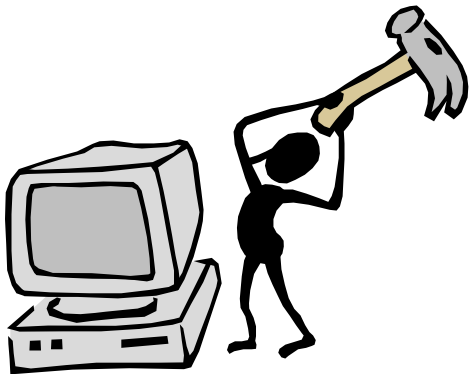
Schwierigkeiten des RE (2)

- Warum wird das RE in der Praxis oft vernachlässigt oder falsch fokussiert?
  - KundInnen wollen keine Veränderung, sondern eine Verbesserung. EntwicklerInnen, die das nicht begreifen, vernachlässigen die Ist-Analyse.
  - EntwicklerInnen sind oft der (grundfalschen) Überzeugung, bereits zu wissen, was gewünscht oder benötigt wird.
  - KundInnen legen dem RE Steine in den Weg:
    - Den EntwicklerInnen stehen die NutzerInnen nicht zur Verfügung.
    - Die Analyse-Aktivität wird als unproduktiv denunziert (da nichts lauffähiges entsteht).
    - KundInnen sabotieren den Prozess durch späte Änderungen.



Nutzungsgestaltung ist schwierig





Nutzungsbeschreibung ist schwierig

- Der erste Schritt der Nutzungsgestaltung ist das gemeinsame Verständnis.
- Aber Nutzungsbeschreibung ist auch schwierig!

- Siehe Beispiele in Trello (Spiel zu Sichten)
  - Insbesondere Granularität, Zusammenhang zwischen stories / features

- Siehe Beispiele in issue tracking systems (Hausaufgabe 4.4.4.3)
  - Insbesondere Vermischung mit Fehlermanagement

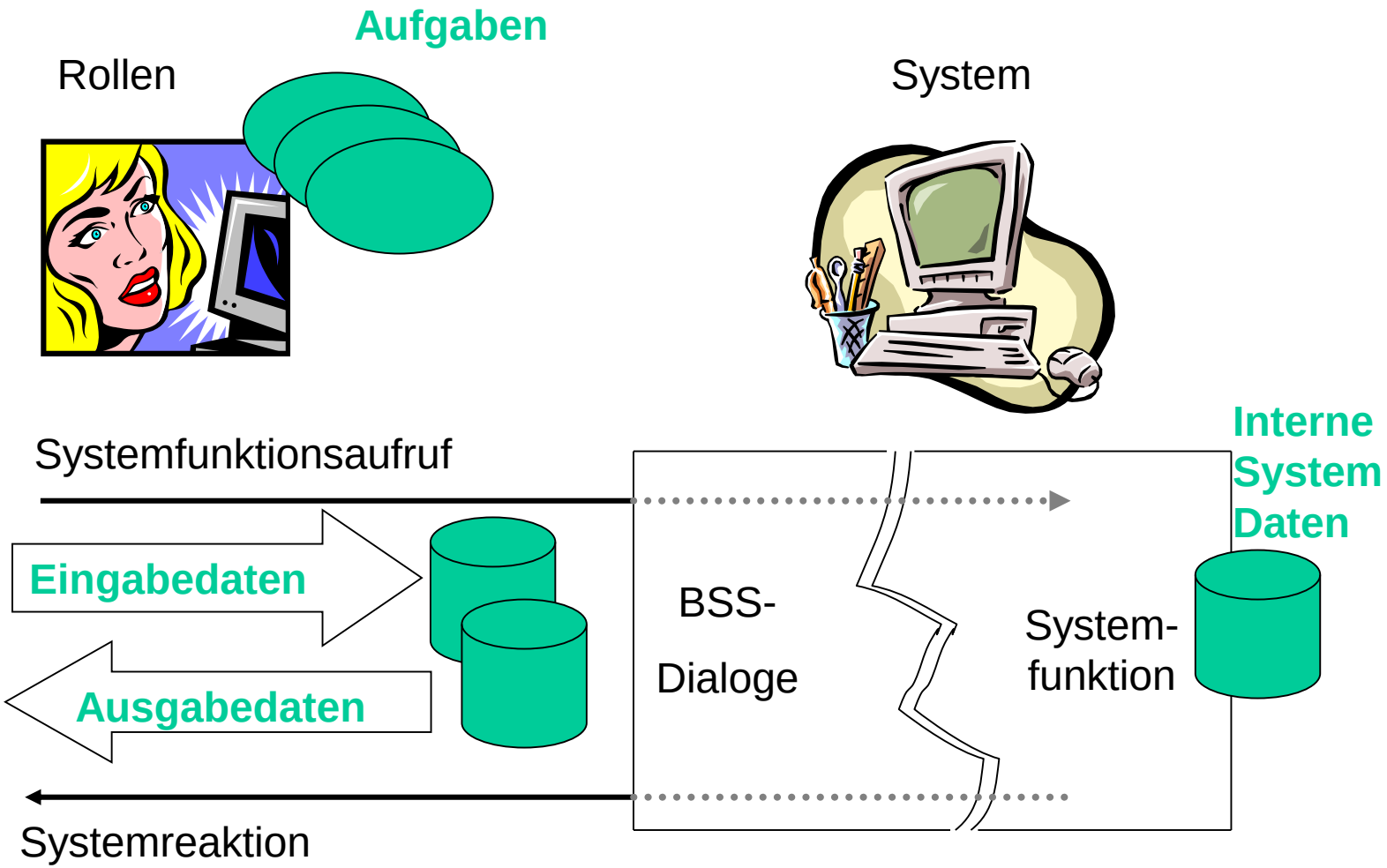
Fragen der Nutzungsbeschreibung

- Welche Beschreibung ist sowohl für NutzerInnen als auch für EntwicklerInnen verständlich?
  - Code und Klassendiagramme sind NICHT für NutzerInnen verständlich => Text oder domänennahe Notation
- Welche Granularität ist verständlich?
  - Aufgeschriebene Anforderungen sind oft zu abstrakt, so dass NutzerInnen sich nichts Konkretes darunter vorstellen können. => Es ist wichtig, den Kontext zu beschreiben und eine Vorstellung des Systems zu vermitteln.
  - Die genaue Beschreibung der Benutzungsschnittstelle ist oft zu detailliert und Anforderungen müssen zur Planung verteilt werden (Verteilung der Arbeit auf EntwicklerInnen über die Zeit) => Deshalb ist auch gröbere Beschreibung wichtig.

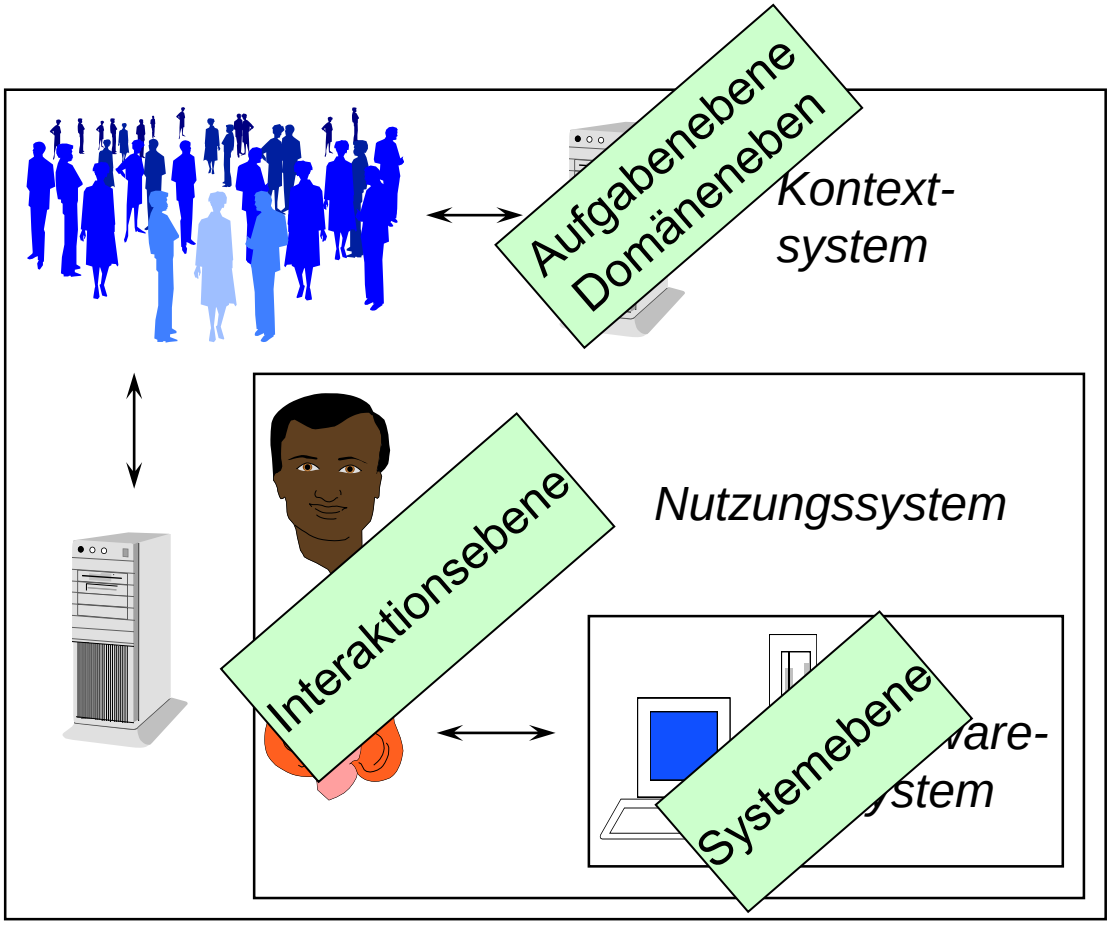
Verständnis der NutzerInnen

- Um die Sicht der NutzerInnen zu verstehen, muss man ihre „Welt“ verstehen. Dafür gibt es viele Beschreibungstechniken.
- Wir verwenden
  Aufgabenorientierte Anforderungsspezifikation
  - Beinhaltet unterschiedliche Techniken auf unterschiedlichen Abstraktionsebenen der Nutzungsmodellierung

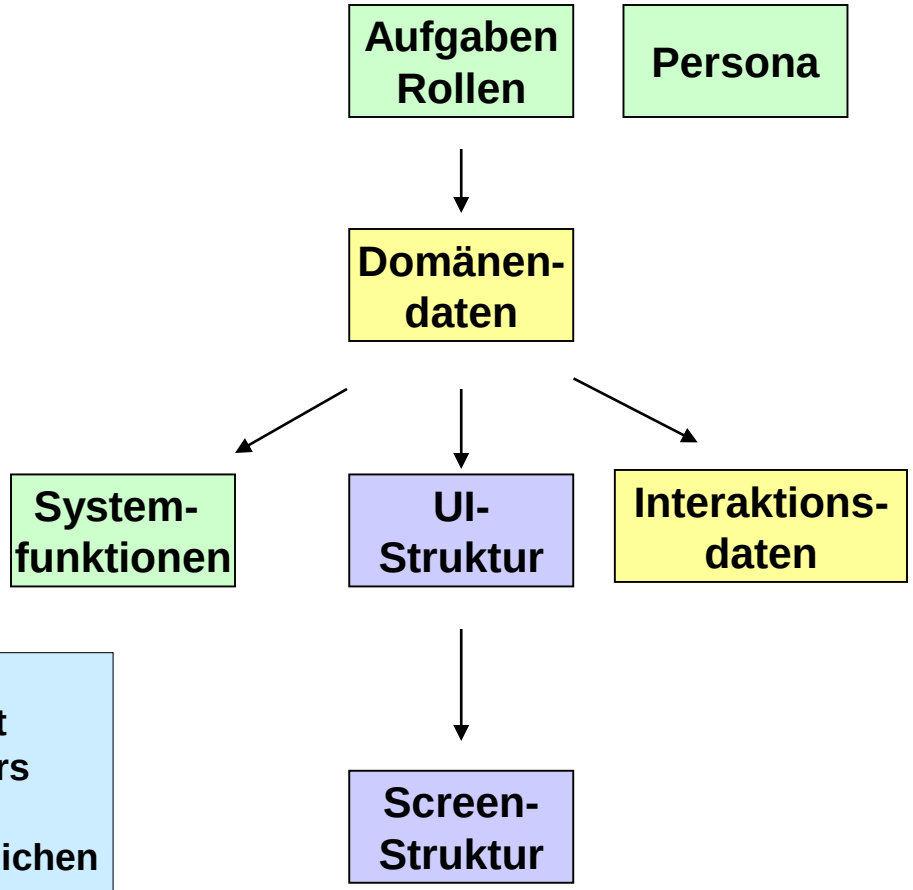
Aufgabenorientierte Anforderungsspezifikation



Gestaltungsbereiche der SW-Entwicklung



Beschreibungsebenen (vereinfacht)



Aufgaben ebene

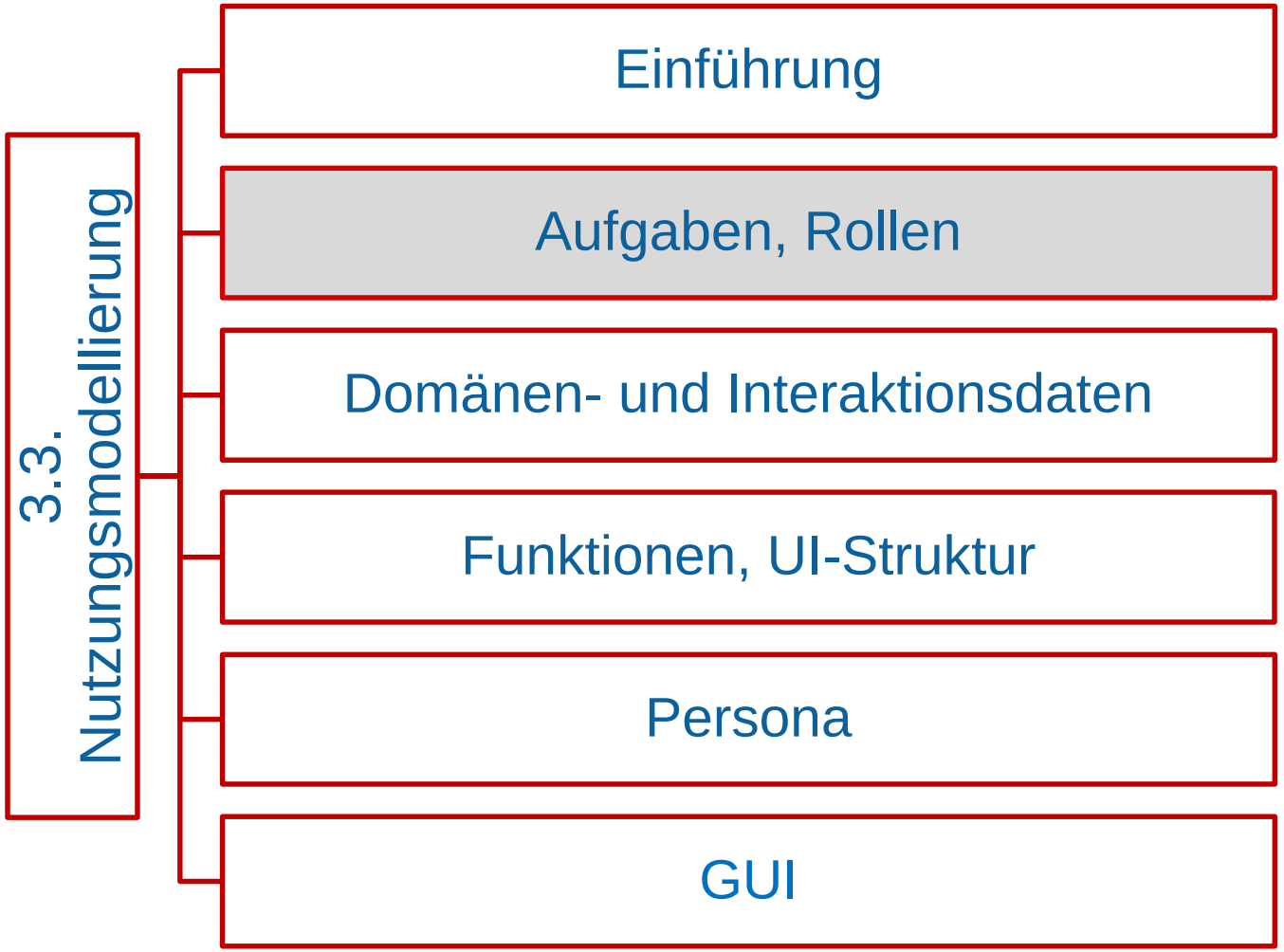
Domänen ebene

Interaktions ebene

Systemebene: GUI

Legende:
Grün: Aufgaben
Gelb: Daten
Lila: Struktur

Reihenfolge bedeutet inhaltliche Abhängigkeit
Manchmal zeitlich anders erstellt
Aber muss dann abgeglichen werden



Fragen der Aufgaben- und Domänenebene

- Zuerst Fokus auf Arbeitsplatzgestaltung (Verständnis des Kontexts der NutzerInnen)
  - Welche **Rollen** soll das System unterstützen?
  - Welche **Aufgaben** haben diese Rollen?
  - **Domänendaten:** Welche Informationen sind dabei wichtig?

Wie kann ich NutzerInnen beschreiben?

- Eine Nutzerrolle ist eine abstrakte Zusammenfassung von Bedürfnissen, Interessen, Erwartungen, Verhalten und Verantwortlichkeiten, die charakteristisch ist für eine Menge von zukünftigen Systembenutzern [nach Constantine/Lockwood99].
- Beispiele: Bäcker, Pflegekraft, Studienverantwortlicher

- Ein Nutzerprofil beschreibt das Wissen und die Fähigkeiten von typischen NutzerInnen.

- Kann erhoben werden durch
  - Befragung der NutzerInnen
  - Befragung von Surrogat-NutzerInnen (Marketing, Vertrieb, Hotline, TrainerIn)
  - Untersuchung von Dokumenten im Geschäftsprozess

Nutzerrolle und -profil

- **Rollenbeschreibung**
  - Aufgaben
  - Erfolgskriterien (d.h. wann hat die Rolle Ihre Arbeit gut gemacht)
  - Kommunikationspartner
  - Innovationsgrad (d.h. gab es die Rolle vorher schon)

- **Nutzerprofil**

  Wissen/Erfahrungen/Fähigkeiten
  - bzgl. Aufgaben
  - bzgl. Softwaresystem

Beispiel Rolle: FilmverwalterIn

- Für eine Software zur Filmverwaltung
- **Rollenbeschreibung FilmverwalterIn**
  - Aufgaben:
    - Filmverwaltung
  - Erfolgskriterium:
    - Vollständige Übersicht, schnelle Suche
  - KommunikationspartnerInnen:
    - FreundInnen
  - Innovationsgrad:
    - gering
- **Nutzerprofil**
  - Vorwissen Filmverwaltung:
    - Wahrscheinlich hoch
  - Vorwissen Software:
    - Hier kann es Neulinge und Erfahrene geben

Wie kann ich Aufgaben beschreiben?

- **Arbeitsaufgabe**
  - **Vorgabe zum Handeln, zur Erreichung eines bestimmten Ziels, unter bestimmten Voraussetzungen, mit bestimmten Vorgehensweisen**

- Eine Aufgabenbeschreibung ist eine abstrakte Zusammenfassung davon, warum die Aufgabe ausgeführt wird (Ziele, Ursachen, Priorität), wie sie ausgeführt wird und welche Abhängigkeiten das zur Umgebung hat (Vorbedingung, Eingabe, Ausgabe, Ressourcen, Gestaltungsmöglichkeiten)

- Kann erhoben werden durch
  - Befragung der NutzerInnen
  - Befragung von Surrogat-NutzerInnen (Marketing, Vertrieb, Hotline, TrainerIn)
  - Untersuchung von Dokumenten im Geschäftsprozess
  - Beobachtung

Wie kann ich Aufgaben beschreiben?

- **Aufgabeneinordnung**
  - Ziele
  - Eingriffsmöglichkeiten (d.h.welche Entscheidungsspielräume)
  - Ursachen
  - Priorität

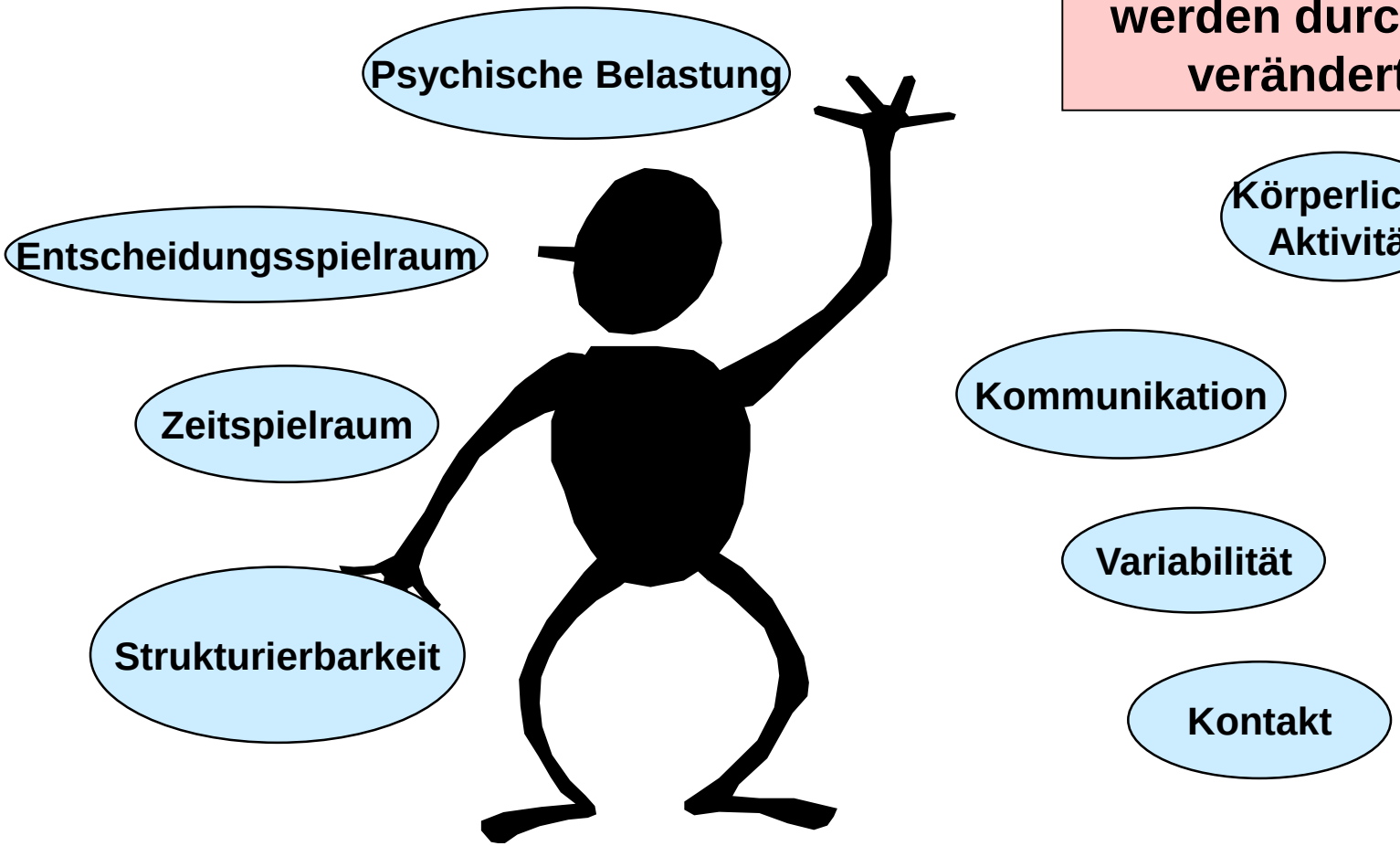
- **Aufgabendurchführung**
  - Durchführungsprofil (Häufigkeit, Kontinuität, Komplexität)
  - Vorbedingung
  - Info-In
  - Info-Out
  - Ressourcen (Arbeitsmittel, beteiligte Rollen)

Beispiel Aufgabenbeschreibung: Filmverwaltung

- **Ziel:**
  - VerwalterIn hat Überblick über die vorhandenen Filme
- **Eingriffsmöglichkeiten:**
  - Auswahl der Filme
- **Ursachen:**
  - Kein formaler Grund, eigener Wunsch
- **Priorität:**
  - Mittel
- **Durchführungsprofil:**
  - manchmal, Unterbrechung möglich, mittlere Komplexität
- **Vorbedingung:**
  - Filme vorhanden (ggf. noch nicht eingetragen)
- **Info-In:**
  - Film- und SchauspielerInnendaten
- **Info-Out:**
  - Überblick über Filme und SchauspielerInnen
- **Ressourcen:**
  - VerwalterIn,

Gute Aufgaben: Humane Arbeit

Achtung! Aufgaben werden durch IT verändert



Quelle: Eberhard Ulich: „Arbeitspsychologie“, Schäffer-Poeschel Verlag, Stuttgart, 1994

Verwendung der Aufgabeninformation

- **Ziel** => *Vorgabe für das Ergebnis*
- **Eingriffsmöglichkeiten** => *System darf diese Eingriffsmöglichkeiten nicht unnötig beschränken*
- **Ursachen** => *Zusammenhang zu anderen Aufgaben, Vorbedingungen*
- **Priorität** => *Bestimmt Unterstützung auf dem GUI, z.B. Funktionstaste bei hoher Priorität*
- **Durchführungsprofil** => *Haufigkeit und Komplexität bestimmt Unterstützung auf dem GUI (wie bei Prio), Unterbrechbarkeit muss vom System unterstützt werden*

- **Vorbedingung** => *muss vom System abgeprüft werden*
- **Info-In** => *Daten müssen im System sein oder eingegeben werden*
- **Info-Out** => *Daten müssen im System abgespeichert oder ausgegeben werden*
- **Ressourcen** => *Rollen geben Hinweise auf weitere NutzerInnen und ihr Zusammenspiel, Arbeitsmittel müssen im oder außerhalb des Systems bereitgestellt werden*

[Lauesen 2005]

IT-nahe Beschreibung von Teilaufgaben

- Wozu verwenden die NutzerInnen das System? Sie erledigen ihre Aufgaben (in ihrer Arbeit). Diese müssen von den EntwicklerInnen verstanden werden. Sie werden typischerweise durch Teilaufgaben (Aktivitäten) beschrieben.

- Aufgabenbeschreibung kann erweitert werden, um IT-Unterstützung zu identifizieren: Task & Support
  - http://www.itu.dk/people/slauesen/

- => Insgesamt: beschreibe wichtige zu unterstützende Aktivitäten
  - und ggf. Varianten: (a) und Probleme (p) und
  - Vorschläge für IT-Unterstützung (Systemverantwortlichkeiten)
- Tabellenartige Aufschreibung ist hilfreich, um Reihenfolgen zu vermeiden

[Lauesen 2005]

Template: Task & Support

Task: Name ***Eine Aufgabe umfasst eine größere Menge von Aktivitäten und erfordert die Bearbeitung unterschiedlicher Daten***	
Sub-Tasks and Variants	**Example Solutions**
1. Sub-Task: *Teilaufgaben entsprechen abgeschlossenen Teilzielen der Aufgabe. Sie benötigen typischerweise mehrere Systemfunktionen.*	*Lösungen können sich auf konkrete Systemfunktionen beziehen*
1a.Variant: *Beschreibt Sonderfall bei der Durchführung der Teilaufgabe*	*Hier sind Lösungsideen besonders wichtig*
1ap.Problem: *Beschreibt Problem bei der Variante oder der Teilaufgabe*	*Hier sind Lösungsideen besonders wichtig*

[Lauesen 2005]

Übung: Task & Support

Task: Bearbeite Anfrage in einer Hotline	
Sub-Tasks and Variants	**Example Solutions**
1. Nimm Anfrage über Post, Telefon oder Email an. Anfrage kann neu sein oder Bezug zu früherer Anfrage haben.	
2. Finde Eintrag zu der Anfrage (falls vorhanden)	
2a. Erzeuge neue Anfrage	
2ap.Problem: Es kann schwierig sein einen vorhandenen Eintrag zu finden, weil AnruferIn z.B. weder eigene ID noch AnfrageID weiß	

[Lauesen 2005]

Beispiel: Task & Support

Task: Bearbeite Anfrage in einer Hotline	
Sub-Tasks and Variants	**Example Solutions**
1. Nimm Anfrage über Post, Telefon oder Email an. Anfrage kann neu sein oder Bezug zu früherer Anfrage haben.	< keine Systemverantwortlichkeit>
2. Finde Eintrag zu der Anfrage (falls vorhanden)	Bei Anfrage durch Email werden die Daten automatisch aus der Email in eine Suchmaske übertragen
2a. Erzeuge neue Anfrage	
2ap.Problem: Es kann schwierig sein einen vorhandenen Eintrag zu finden, weil AnruferIn z.B. weder eigene ID noch AnfrageID weiß	Das System zeigt Einträge, die zum Namen der AnruferIn passen.

Übung Aufgabenbeschreibung

- Arbeitsblatt: Ergänzen Sie die User Task Beschreibung des Movie Managers

- Was ist bei der Beschreibung von Aufgaben zu beachten?

Zu verbessernde Beispiele

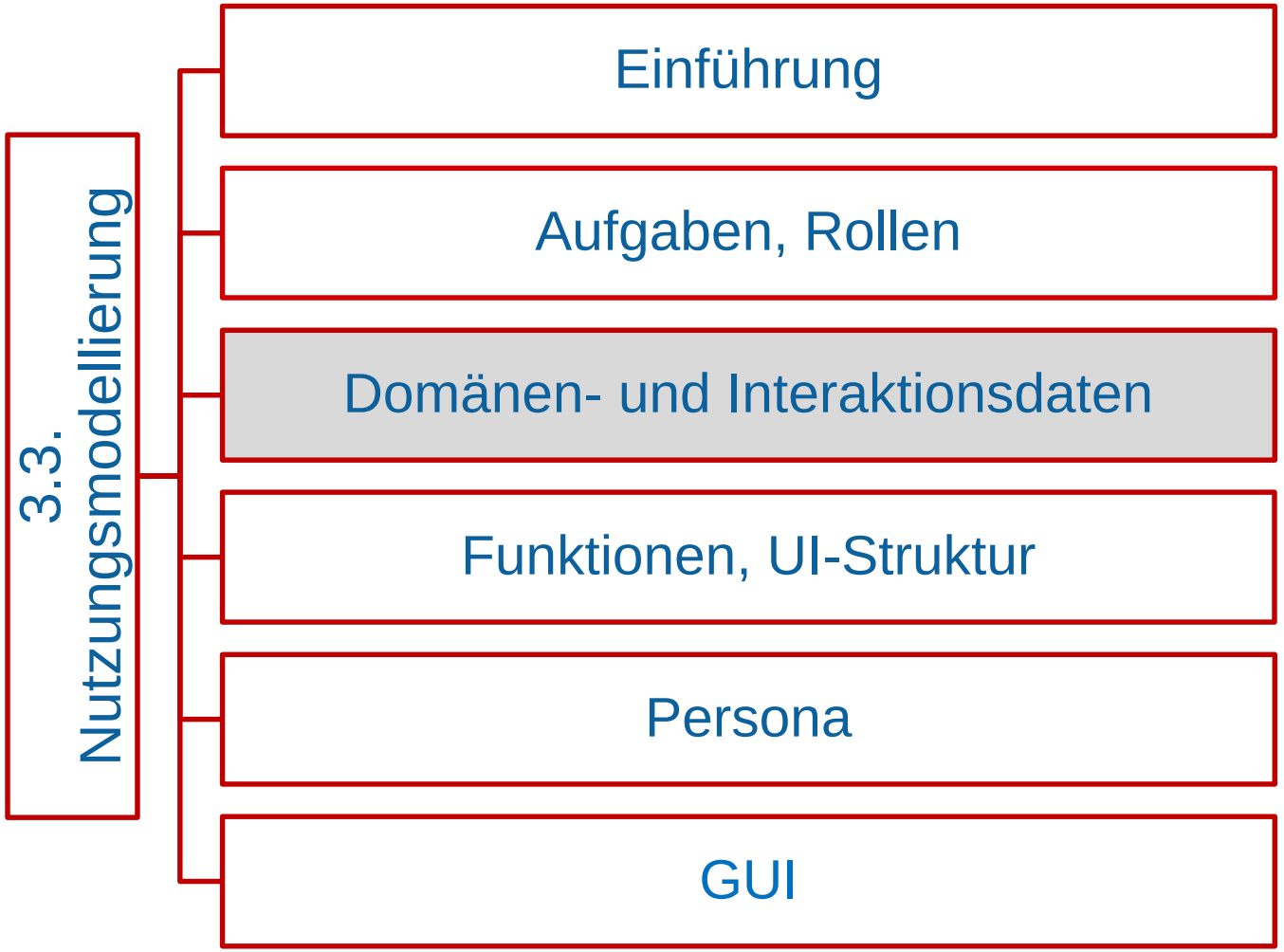
Name	Description	Example Solution
Show all episodes of a series at once (variant)	**Sub task ist keine task des user**	
Show all episodes of a season of a series (variant)	Maybe foldout menu series->seasons->episodes	

Zu fein. Sub task entsprechen einzelnen Systemfunktionen

List season	Show all season	Use System Function List Season
List episode	Show all episode	Use System Function List Episode
List series	Show all series	Use System Function List Series
Describe a series	Add and describe a series with typical data	Use System Function Modify Series
Describe a episode	Add and describe a episode with typical data	Use System Function Modify Episode
Describe a season	Add and describe a season with typical data	Use System Function Modify Series

Zu grob, Beschreibung zu kurz, Problem nicht klar, keine Lösungen

Manage Movie Series	add, describe, remove, combine	
Problems (problem)	Series contains Seasons, Seasons contain Episode at least 1, remove everythin in Series if deleted.	



Wie kann ich Domänendaten beschreiben?

- Domänendaten beschreiben Entitäten (Dinge und Konzepte), die im Kontext wichtig sind.
- Werden durch Entity-Relationship-Diagramme (ER-Diagramme) beschrieben => Domänendatendiagramm
- Oft auch Glossar ausreichend
- Erklären die in den Verhaltensbeschreibungen verwendeten Begriffe

Entity Relationship Diagramm

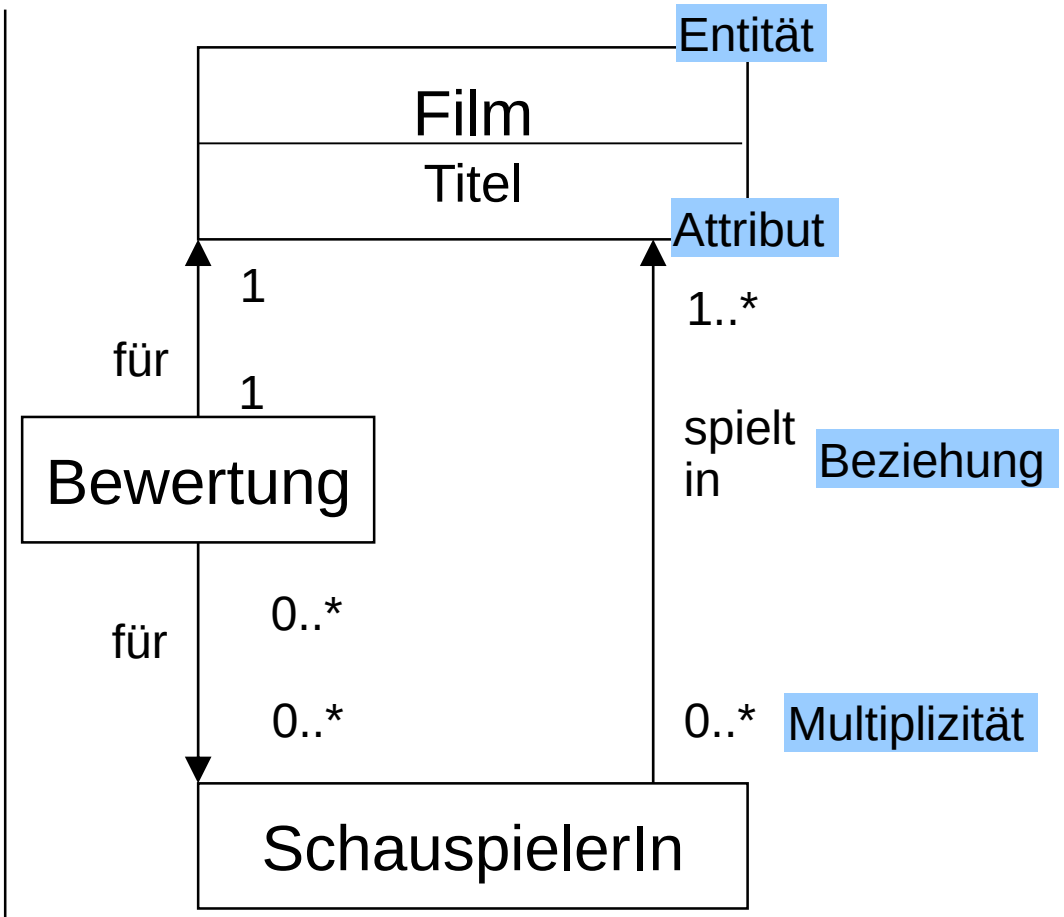
- ER-Diagramme (ERD) wurden ursprünglich nur für Datenbankentwicklung verwendet (Peter Chen, 1976)

- Hilft die grundlegenden Entitäten der realen Welt und deren Beziehungen zu verstehen

- Heutzutage kann man ERD als eine einfache Form der Klassendiagramme ansehen, die man verwendet, wenn man keine Operationen und nur einfache Beziehungen (d.h. keine Aggregation, Vererbung, etc) modellieren will.

Beispiel: Glossar und ERD

- Eine Film hat einen Titel.
- Eine SchauspielerIn spielt in mindestens einem Film mit.
- Eine Bewertung bezieht sich auf genau einen Film und die im Film vorkommenden SchauspielerInnen. Es gibt nur eine Bewertung pro Film.



Achtung: es ist nicht ausgedrückt, dass die Bewertung die SchauspielerInnen aus dem Film betrifft.

Unterscheidung Entität und Attribut

- Die zwei Grundregeln zur Unterscheidung von Entitäten und Attributen lauten:
  - Entitäten haben eine eigene Identität und mehrere Attribute. Diese Attribute werden für komplexere Berechnungen oder Überprüfungen gebraucht, die später als Teile von Systemfunktionen modelliert werden.
  - Attribute modellieren einen (evtl. veränderlichen) Wert, der unabhängig von der Entität nicht sinnvoll ist.
- Beispiel Datum
  - Ein **Attribut** Datum ist sinnvoll, wenn das Datum nur als zusammenhängender Wert abgespeichert, verändert und abgefragt wird.
  - Eine **Entität** Datum ist angemessen, wenn damit komplexere Systemfunktionen durchgeführt werden (z.B. Schaltjahrprüfung, Veränderungen einzelner Tage, Monate u.ä.).
- *Ist Bewertung als Entität sinnvoll?*
  - *Könnte auch Attribut von Film sein (wenn keine eigene Attribute).*

Identifikation von Assoziationen

- Assoziationen spiegeln Zusammenhänge aus dem Anwendungsbereich wider. Oft entsprechen diesen Zusammenhängen Adjektive oder Verben im Text. Z.B: „Eine SchauspielerIn spielt in mindestens einem Film mit.“

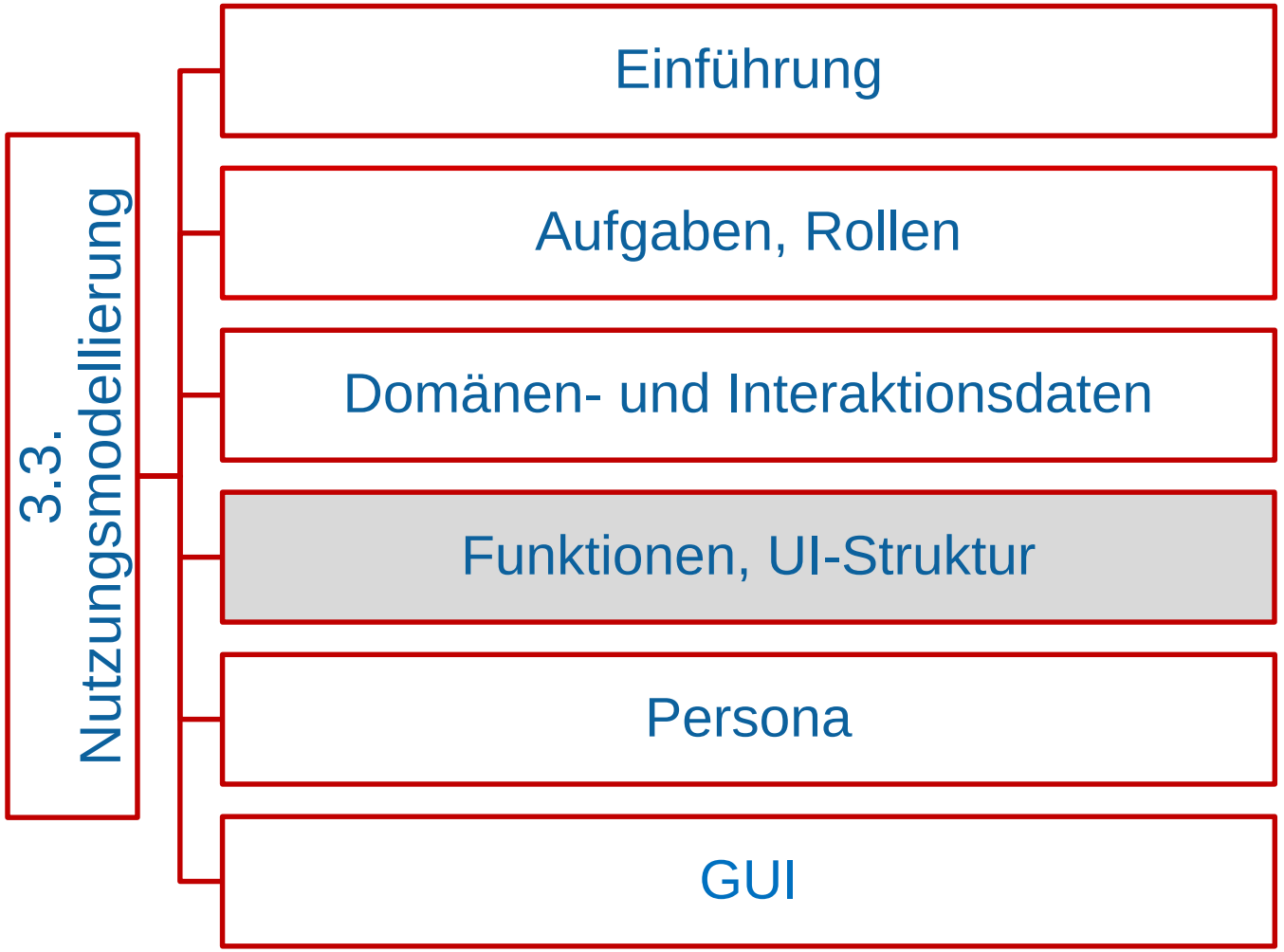
Domänen- vs. Interaktionsdaten

- Domänendaten entsprechen den bei der Beschreibung von Aufgaben verwendeten Begriffen. Sie beschreiben die Daten unabhängig vom Softwaresystem.

- Interaktionsdaten entsprechen den bei der Beschreibung von Systemfunktionen verwendeten Begriffen. Sie beschreiben Daten, die auf der Benutzungsschnittstelle ein- oder ausgegeben werden.
  - Detaillieren vor allem die Attribute der Domänendaten
- Meistens sind Domänendaten Teil der Interaktionsdaten.
  - Es kann aber auch Daten für die Aufgaben geben, die bei der Interaktion keine Rolle spielen (falls nicht vom System unterstützt), und umgekehrt (falls Aufgabenbeschreibung noch nicht so detailliert war).

Zusammenfassung Aufgaben- und Domänenebene

- Für die Nutzungsgestaltung ist es wichtig, den Kontext der NutzerInnen zu verstehen.
- Aufgaben, Domänendaten und Rollen charakterisieren diesen Kontext.
- Hier wird sichergestellt, dass **die richtigen Aufgaben** unterstützt werden. Dabei wird darauf geachtet, technische Überlegungen nicht unnötig früh einzuführen.



Fragen der Interaktionsebene

- Fokus auf der Gestaltung der Grenze/Schnittstelle zwischen Mensch und Maschine bei der aufgabengerechten Umsetzung der Systemverantwortlichkeiten
  - Welche Funktionen bietet das System an?
    - **Funktionsbeschreibung, Interaktionsdaten**
  - In welchen Zusammenhängen können NutzerInnen welche Funktionen aufrufen? Welche Daten sehen sie dabei? Wie gliedert sich die Gesamtfunktionalität in Teilbereiche?
    - **User Interface Struktur (UI-Struktur)**
      - ***Achtung: Nicht GUI-Struktur, d.h. konkretes Layout noch nicht wichtig***

Wie beschreibe ich eine Systemfunktion?

Ähnlich zur Aufgabenbeschreibung, aber hier „Aufgaben“ des Systems!

Name	Kurzbezeichnung der Funktion
Eingangsdaten	Welche Daten verarbeitet die Funktion?
Ausgangsdaten	Welche Daten erzeugt oder verändert die Funktion?
Beschreibung	Wie soll das Ergebnis **normalerweise** berechnet werden?
Ausnahmefälle	Welche Ausnahmen gibt es? Was soll dann passieren?
Regeln	Komplexe funktionale oder kausale Zusammenhänge bei der Berechnung
Qualitätsanforderungen	Welche übergreifenden Eigenschaften sind wichtig?
Vorbedingungen	Zustand von System und Umgebung aus Sicht des Aktors *bevor* die Funktion ausgeführt wird
Nachbedingung	Zustand des Systems aus Sicht des Aktors *nachdem* die Funktion erfolgreich beendet ist

Beispiel Systemfunktionsbeschreibung

Name	Sortiere Filme
Eingangsdaten	Liste der Filme, Sortierkriterium
Ausgangsdaten	Sortierte Liste der Filme
Beschreibung	Funktion sortiert Liste nach Kriterium
Ausnahmefälle	keine
Regeln	Berechnungsregeln entsprechend zu den möglichen Sortierkriterien
Qualitätsanforderungen	Sortierung großer Mengen in kurzer Zeit
Vorbedingungen	Liste der Filme ist nicht leer
Nachbedingung	Menge der Filme nicht verändert

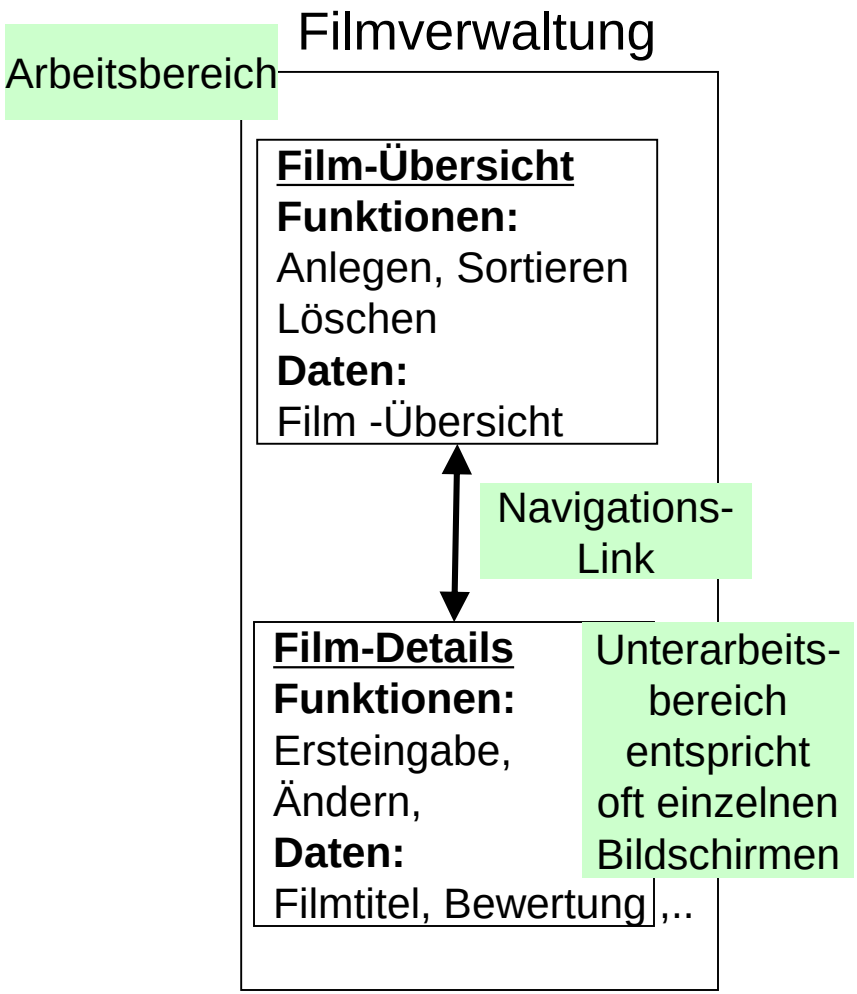
Strukturierung der Interaktion

- Wenn es viele Funktionen gibt, müssen die Daten und Funktionen in verschiedene zusammenhängende Bereiche gebündelt werden => UI-Struktur

- UI-Struktur besteht aus Arbeitsbereichen und Navigationslinks
- **Arbeitsbereich (auch Workspace, Virtual Window)** gruppiert zusammengehörige Funktionen und Daten
- UI-Struktur abstrahiert vom konkreten Layout und Bildschirmaufteilung, stellt logische Sicht der NutzerInnen auf die Interaktionsstruktur dar

- Entsteht parallel zu den UC

Beispiel UI-Struktur



Achtung:
Navigationsbeziehung
muss nicht immer
in beide Richtung gehen.
Insbesondere gilt
das nicht, wenn
ein Arbeitsbereich
sichtbar bleibt,
nachdem der andere
geöffnet wurde

Arbeitsbereich

- Ein Arbeitsbereich umfasst Daten und Funktionen, die die NutzerInnen im Zusammenhang nutzen wollen.
- Logische Gruppierung!

- Im konkreten GUI muss ein Arbeitsbereich wg. Größe des Bildschirms evtl. auf mehrere Sichten verteilt werden (siehe GUI-Ebene).
- Im konkreten GUI können auch mehrere Arbeitsbereiche gleichzeitig geöffnet sein.

- Ein Arbeitsbereich kann unterstrukturiert werden.

Übung UI-Struktur

- Erstellen Sie eine UI-Struktur für die Möglichkeiten in Powerpoint

3. Literatur (1.Teil)

- Constantine L, Lookwood L (1999) *Software For Use*, ACM Press
- Gloger B (2008) *Scrum – Produkte zuverlässig und schnelle entwickeln*, Hanser Verlag
- Lauesen S (2005) *User Interface Design - A software engineering perspective*, Addison-Wesley
- Ludewig J, Lichter H (2010) *Software Engineering,* dpunkt
- Wirdemann R (2009) *Scrum mit User Stories*, Hanser Verlag