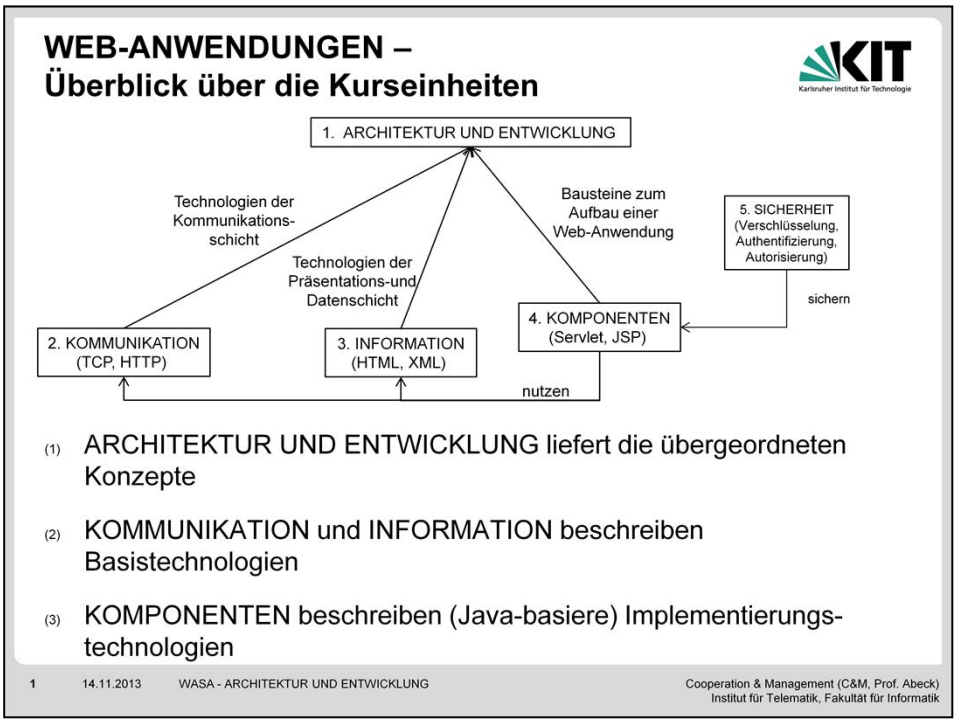
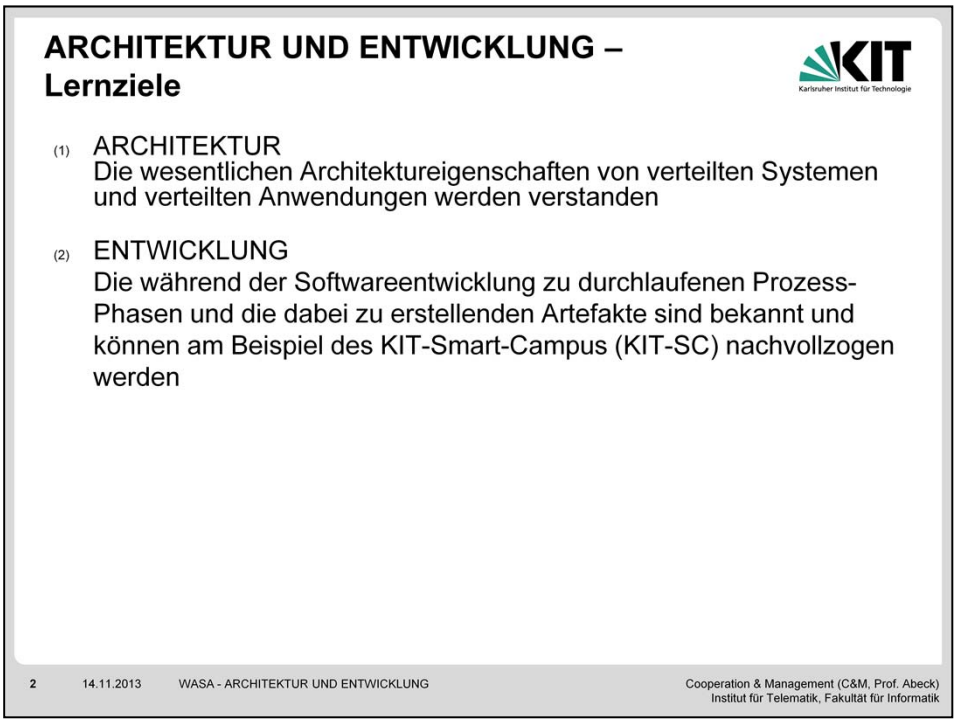


(1) Während die ARCHITEKTUR die Verteilungskonzepte und (statischen und dynamischen) Systemelemente einer Web-Anwendung fokussiert, behandelt die ENTWICKLUNG den systematischen Aufbauprozess eines solchen verteilten Systems.

(2) In diesen Kureinheiten zur KOMMUNIKATION und INFORMATION werden die wichtigsten Kommunikations- und Informationstechnologien zum Aufbau von Web-Anwendungen detailliert vorgestellt.

(3) KOMPONENTEN sind ein Bindeglied der übergeordneten Architekturkonzepte und der Basistechnologien, indem die softwaretechnischen Bausteine zur Entwicklung von verteilten (komponentenorientierten) Web-Anwendungen bereitgestellt werden.

(SICHERHEIT) Stellt eine zentrale Querschnittsfunktionalität dar, die eine Voraussetzung für den Einsatz von Web-Anwendungen in der Praxis darstellt.



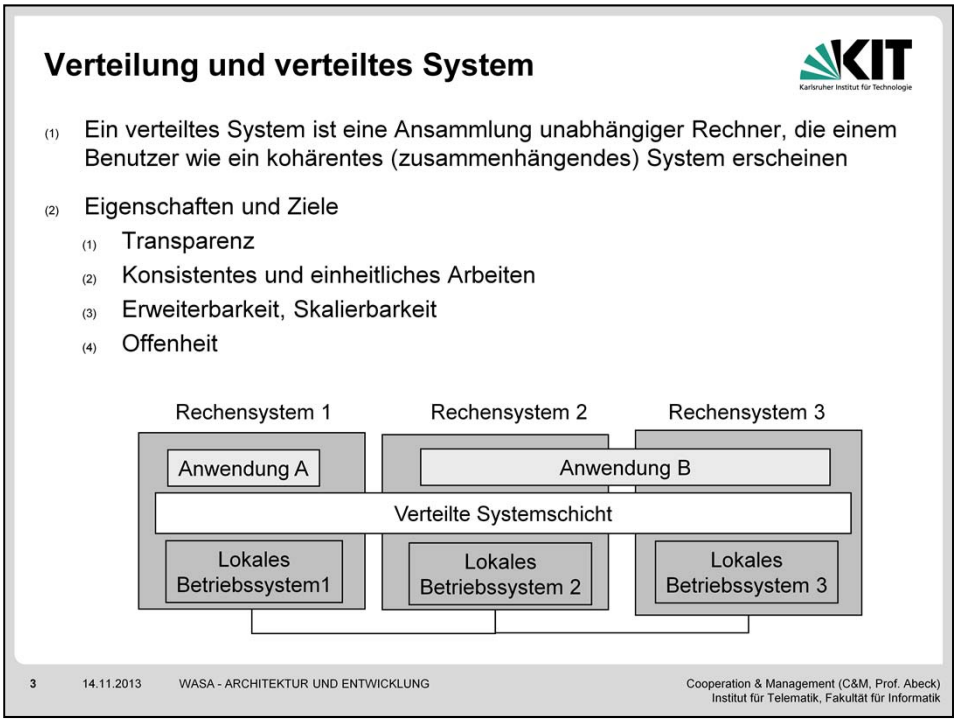
(1) Zunächst wird der Aspekt der Verteilung allgemein behandelt und anschließend werden verteilte (Web-) Anwendungen und deren Architektur vorgestellt.

(2) Der Schwerpunkt liegt auf den (aus konzeptioneller und architektureller Sicht besonders wichtigen) ersten zwei Phasen der Softwareentwicklung, der Analyse und dem Entwurf.

Hauptquellen:

[BD04] Bernd Bruegge, Allen H. Dutoit: Object-Oriented Software Engineering Using UML, Patterns and Java, Pearson Prentice Hall, 2004.

[TS08] Andrew S. Tanenbaum, Maarten van Steen: Verteilte Systeme – Prinzipien und Paradigmen, Prentice-Hall Pearson Studium, 2008.



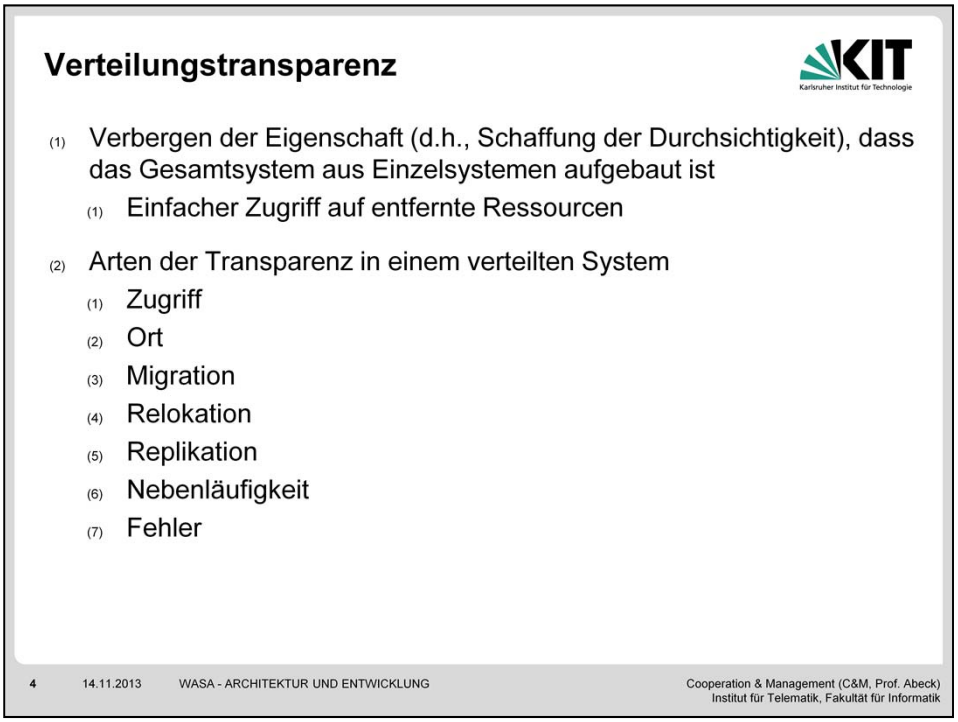
[TS08]
Sobald zwei oder mehr Rechensysteme über ein Kommunikationsnetz verbunden werden, entsteht ein verteiltes System.

(1) Wichtige Aspekte der Definition sind:
(i) Die einzelnen Rechner entsprechen Komponenten des Gesamtsystems, die autonom (-> unabhängig) sind.
(ii) Die Benutzer (Menschen oder Programme) eines verteilten Systems glauben, sie hätten es mit einem System zu tun (-> Kohärenz).
Anmerkung: Das Adjektiv "kohärent" hat gemäß Duden die Bedeutung von "zusammenhängend". Unter Kohäsion wird in der Physik der Zusammenhalt der Moleküle eines Körpers verstanden.
(iii) Da die Systeme keinen gemeinsamen Bus und Speicher besitzen, kommunizieren sie gemäß eines bestimmten Protokolls über das Kommunikationsnetz.

(2.1) Die Transparenz bezieht sich auf verschiedene Aspekte:
- Art der Systeme: Es ist für den Benutzer "durchsichtig" (verborgen, nicht relevant), welche Arten von Client- und Serversystemen mit welchen Betriebssystemen und welcher Hardware-Ausstattung eingesetzt werden, um eine gewisse Funktionalität zu erbringen.
- Kommunikation zwischen den Systemen: Hierzu gehört die Transparenz, welche Kommunikationsprotokolle die Systeme nutzen
- Aufbau des Gesamtsystems: Anzahl und Verteilung sind für den Benutzer "durchsichtig".
(2.2) Für alle Benutzer erscheint das System gleich, unabhängig davon, von welchem Ort und über welches Client-System darauf zugegriffen wird. Diese Eigenschaft resultiert aus der Kohärenz.
(2.3) Diese Eigenschaft resultiert aus den unabhängigen (lose gekoppelten, autonomen) Rechensystemen. Hierdurch ist möglich, dass Komponenten vorübergehend ausfallen können und durch andere Rechner ersetzt werden können, so dass der Ausfall für den Benutzer verborgen bleibt (-> Robustheit).

(Verteilte Systemschicht) Eine Softwareschicht, die logisch zwischen einer darüber liegenden Schicht von Anwendungen und Benutzern und einer darunterliegenden Schicht von Betriebssystemen angesiedelt ist. Entsprechend wird diese Schicht auch als "Middleware" bezeichnet.

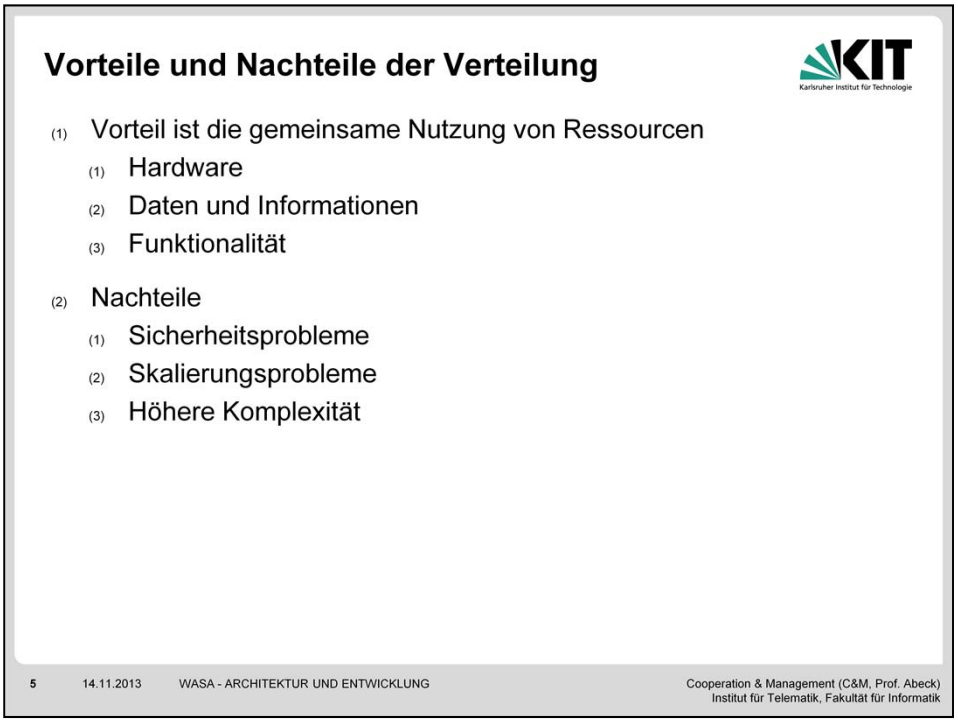
[TS08] Andrew S. Tanenbaum, Maarten van Steen: Verteilte Systeme – Prinzipien und Paradigmen, Prentice Hall Pearson Studium, 2008.



[TS08:21]
(1) Für den Benutzer erscheint das verteilte System dadurch wie ein einziges System.
(1.1) Beispiele für Ressourcen sind: (i) Rechner, (ii) Drucker, (ii) Speichergeräte, (iv) Dateien, (v) Webseiten, (vi) Kommunikationsnetze

(2) Das Konzept der Transparenz lässt sich auf verschiedene Aspekte eines verteilten Systems anwenden [:21].
(2.1) Verbirgt Unterschiede in der Datendarstellung und in der Art und Weise, wie auf eine Ressource zugegriffen wird.
Beispiel: Unterschiedliche Konventionen zur Benennung von Dateinamen in verschiedenen Betriebssystemen
(2.2) Verbirgt, wo sich die Ressource befindet. D.h., der Benutzer kann den Ort der Ressource nicht feststellen, was durch eine geeignete Namensgebung erreicht wird.
Beispiel: Namensgebung von Webseiten durch URLs (http://www.prenhall.com gibt nicht an, an welchem Ort der Web-Server des Verlags steht)
(2.3) Verbirgt, dass eine Ressource an einen anderen Ort verschoben werden kann. Hängt eng mit der Ortstransparenz zusammen.
Beispiel: Verschieben der Datei index.html der obigen Web-Site auf einen anderen Rechner.
(2.4) Verbirgt, dass eine Ressource an einen anderen Ort verschoben werden kann, während sie genutzt wird.
(2.5) Verbirgt, dass eine Ressource als Kopie an verschiedenen Stellen im Einsatz ist.
(2.6) Verbirgt, dass eine Ressource von mehreren konkurrierenden Benutzern gleichzeitig genutzt werden kann.
(2.7) Verbirgt den Ausfall und die Wiederherstellung einer Ressource.

[TS08] Andrew S. Tanenbaum, Maarten van Steen: Verteilte Systeme – Prinzipien und Paradigmen, Prentice-Hall Pearson Studium, 2008.



[TS08:21]
(1) Der Zugriff auf entfernte Ressourcen ist das Hauptziel von verteilten Systemen.
Ein Grund sind die niedrigeren Kosten, die durch die gemeinsame Nutzung von Ressourcen, wie z.B. Drucker oder Supercomputer, entstehen [:21].
(1.1) Bereitstellung von Infrastruktur-Ressourcen wie Prozessoren und Speicher, um hierauf Verarbeitungsprozesse (Funktionalität) ablaufen zu lassen und/oder Daten (Informationen) zu speichern.
(1.2) Beinhaltet die Speicherressourcen, um die Daten zu persistieren.
(1.3) Beinhaltet sowohl die Informationen, die Eingabe- bzw. Ergebnisparameter für der Funktionalität sind, als auch die Infrastruktur-Ressourcen (Prozessor und Speicher), um die Funktionalität zu erbringen.

(2.1) Beispiele sind das Abhören von Informationen, die Vorspiegelung einer falschen Identität (z.B. Angabe einer falschen Kreditkartennummer) oder die Erstellung eines Benutzerprofils (Privatheit).
(2.2) Die Skalierbarkeit eines Systems lässt sich in mindestens drei Dimensionen messen [:26]:
(i) Größe: Einfaches Hinzufügen von Benutzern und Ressourcen
(ii) Geographie: Benutzer und Ressourcen können weit auseinander liegen
(iii) Administration: Einfache Verwaltung trotz Verteilung über viele unabhängige administrative Organisationen
(2.3) Es werden verschiedene Arten von komplexen dezentralen Algorithmen (z.B. Uhrsynchronisation, Lastverteilung, Namensauflösung) benötigt, wodurch die Komplexität des Gesamtsystems steigt.

[TS08] Andrew S. Tanenbaum, Maarten van Steen: Verteilte Systeme – Prinzipien und Paradigmen, Prentice-Hall Pearson Studium, 2008.

[TS08:34]
(1) Werden insbesondere im Bereich des Hochleistungsrechnens verwendet.
Verteilte Rechnersysteme können homogen (Cluster) oder heterogen (Grid) sein.
(1.1) Die Hardware besteht aus ähnlichen Workstations oder PCs, auf denen das gleiche Betriebssystem läuft und die über ein schnelles Kommunikationsnetz verbunden sind.
(1.2) Jedes der im Grid zusammengeschlossenen Systeme kann in einer anderen administrativen Domäne liegen und sehr unterschiedliche Hardware, Software und Netztechnologie nutzen.

(2) Verteilte Unternehmensanwendungen, die auf verschiedenen Ebenen zu integrieren sind.
(2.1) Die unterste Integrationsebene besteht in der Bündelung von Anforderungen eines Clients an den Server zu einer größeren Anforderung, die als verteilte Transaktion auszuführen ist. Näheres zu diesen Systemen und Transaktionen findet sich auf einer nachfolgenden Seite.
(2.2) Aufgrund der Aufteilung von Anwendungen in eigenständige Komponenten mit voneinander unabhängigen Datenbanken besteht die Anforderung, dass Anwendungen direkt miteinander kommunizieren können und nicht nur über den transaktionsbasierten Anfrage-/Antwort-Mechanismus einer gemeinsam genutzten Datenbank.
Ermöglicht wird der direkte Informationsaustausch über eine gewisse Art der Middleware-Kommunikation:
(i) Remote Procedure Call (RPC): entfernter Aufruf einer lokalen Prozedur, in dem die Anfrage und Antwort jeweils als Nachricht verpackt werden; (ii) Remote Method Invocation (RMI); (iii) Message-Oriented Middleware (MOM)

(3) Verteilte pervasive (durchdringende, weit verbreitete) Systeme sind Teil der persönlichen Umgebung, die mobile und eingebettete Systeme beinhalten. Kennzeichen solcher Systeme sind (i) das Fehlen von menschlicher administrativer Steuerung und (ii) das Reagieren jedes Systems auf Veränderungen seiner Umgebung.
(3.1) Integration von PC, TV, Telefone, PDA, Spielkonsolen, aber auch Überwachungskameras, Beleuchtungssteuerung, Uhren in einem System.
(3.2) Verteiltes System mit Sensoren, die als vorzugsweise kabelloses Body-Area-Network (BAN) angeordnet sind.
(3.3) Zwischen 10 und mehr als 1000 vergleichsweise kleine Knoten, die alle mit einem Messgerät ausgestattet sind.
Sensornetze werden häufig als verteilte Datenbanken betrachtet. Aufgrund des geringen Leistungsbedarfs der häufig batteriebetriebenen Sensoren ist Effizienz eines der wichtigsten Entwurfskriterien.

BAN	Body Area Network
MOM	Message-Oriented Middleware
RMI	Remote Method Invocation
RPC	Remote Procedure Call

[TS08] Andrew S. Tanenbaum, Maarten van Steen: Verteilte Systeme – Prinzipien und Paradigmen, Prentice-Hall Pearson Studium, 2008.

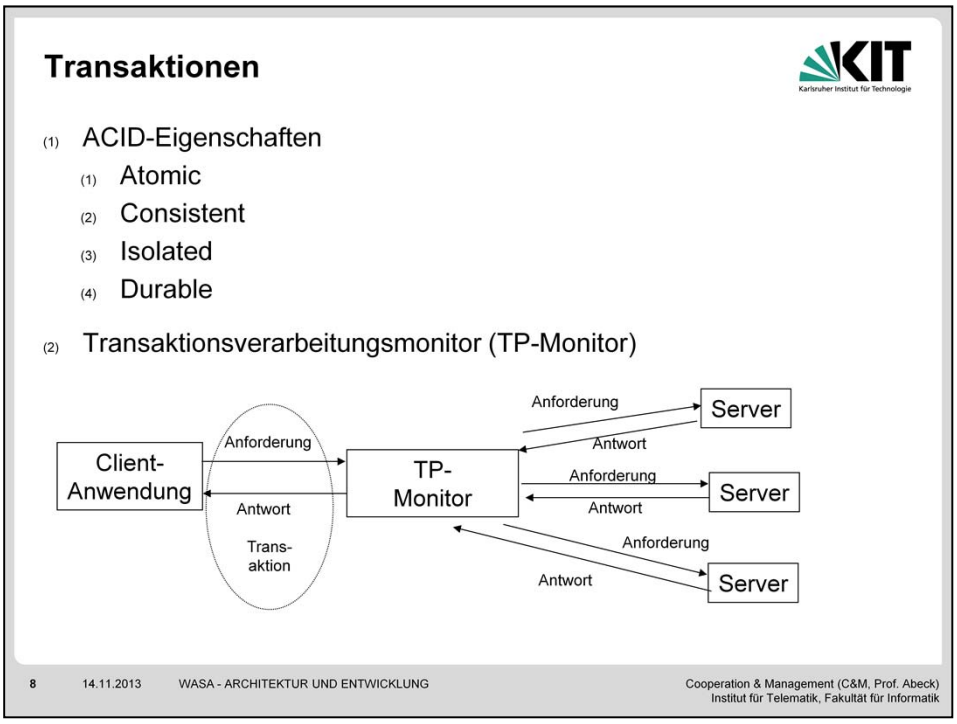
LZ ARCHITEKTUR – ÜA VERTEILUNG

(1) Es ist jeweils ein Vorteil und ein Nachteil von Verteilung zu nennen.

(2) Was wird allgemein und im Zusammenhang mit der Verteilung unter dem Begriff der Kohärenz verstanden?

(3) Was wird unter Verteilungstransparenz verstanden und welche Arten von Transparenz werden unterschieden?

(4) Wie heißt die Klasse von verteilten Systemen, die die Transaktionsverarbeitung und die Integration von Unternehmensanwendungen betreffen?



[TS08:38]
Eine Transaktion hat die Eigenschaft, dass entweder alle oder keine zur Transaktion gehörenden Operationen ausgeführt werden [:38].

(1) Diese Alles-oder-nichts-Eigenschaft wird durch folgende vier sog. ACID-Eigenschaften sichergestellt:
(1.1) Atomic: Eine Transaktion wird als eine für die Außenwelt unteilbare Aktion ausgeführt, bei der keine Zwischenzustände außen sichtbar sind.
(1.2) Consistent: Invarianten (z.B. Gesetz der Gelderhaltung bei der Überweisung eines Geldbetrags) gelten auch nach der Transaktion. Sie können für einen kurzen Moment während der Transaktion verletzt werden, wobei die Verletzung außerhalb der Transaktion nicht sichtbar ist.
(1.3) Isolated: Bei gleichzeitiger Ausführung von zwei oder mehr Transaktionen entspricht das Endergebnis dem sequentiellen Ausführen der Transaktionen in einer bestimmten vom System abhängigen Reihenfolge. Die Eigenschaft wird daher auch als "serialisierbar" bezeichnet.
(1.4) Durable: Nachdem eine Transaktion mit Commit bestätigt ist, wird das Ergebnis dauerhaft bzw. permanent existent.

(2) Diese die Transaktionsverarbeitung (engl. Transaction Processing, TP) unterstützenden TP-Monitore gehören zu den ersten Middleware-Systemen dieser Klasse von Systemen in Unternehmen (engl. enterprise).

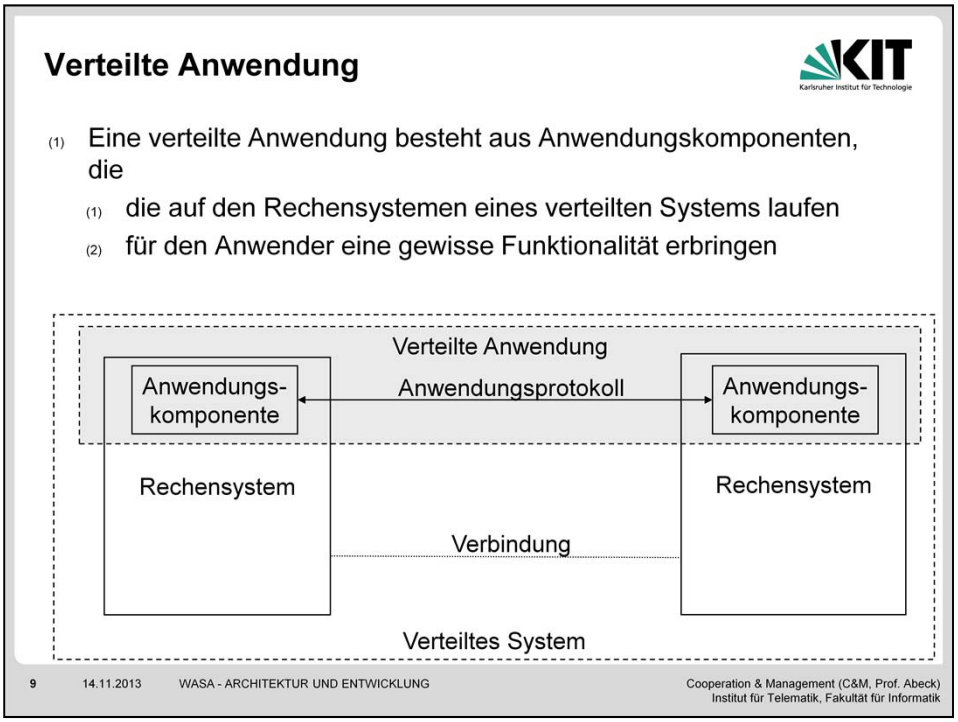
(Anforderung, Antwort) Der TP-Monitor erhält die Anforderung von der Client-Anwendung, deren Bearbeitung zu einer Folge von Anforderungen an beteiligte Server-Systeme führt.

(TP-Monitor) Dieser unterstützt ein transaktionsorientiertes Programmiermodell, durch das die Abbildung einer Client-Anforderung auf die Server-Anforderungen formuliert werden kann.

(Server) Die Integration von Anwendungen erfolgt bei diesem Ansatz auf der Server-Ebene.

ACID	Atomic, Consistent, Isolated, Durable
TP	Transaction Processing

[TS08] Andrew S. Tanenbaum, Maarten van Steen: Verteilte Systeme – Prinzipien und Paradigmen, Prentice-Hall Pearson Studium, 2008.



[Ha05]
(1.1) Es besteht ein enger Zusammenhang zwischen einer verteilten Anwendung und einem verteilten System.
(1.2) Eine verteilte Anwendung erbringt eine für eine bestimmte Gruppe von Anwendern nützliche Fachfunktionalität (oder Geschäftsfunktionalität).

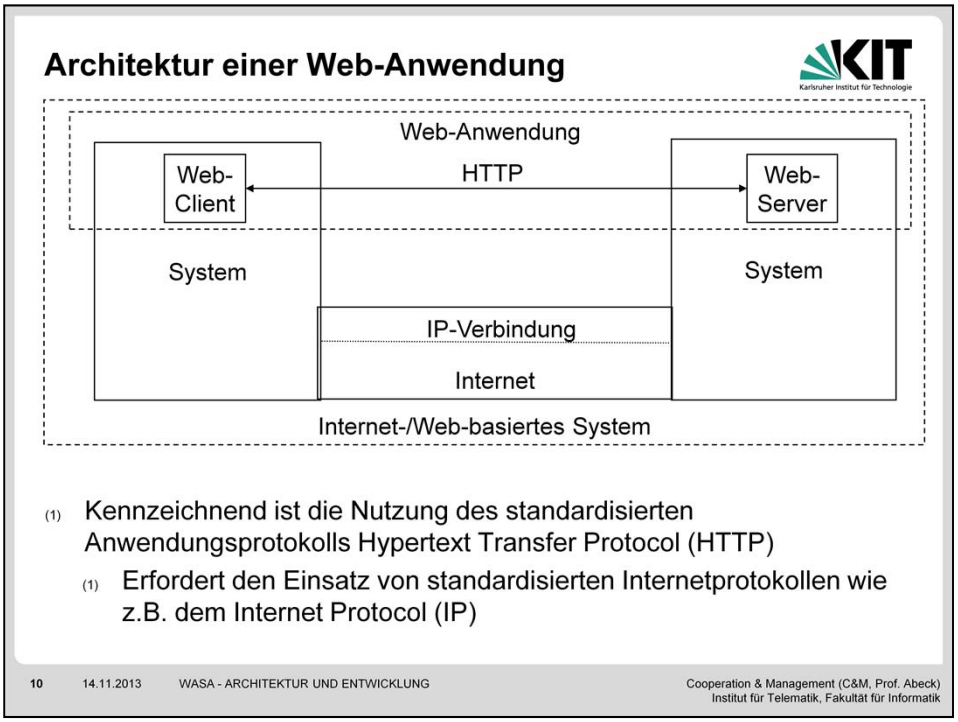
(Verteiltes System, Rechensystem) Ein verteiltes System setzt sich aus vernetzten Rechensystemen zusammen, die aus Hardware und Software bestehen und über den Austausch von Nachrichten kommunizieren.

(Anwendungskomponente) Die die verteilte Anwendung bildenden Anwendungskomponenten sind auf einem System laufende Softwarekomponenten.

(Anwendungsprotokoll) Die Kommunikation zwischen den Anwendungskomponenten erfolgt durch Ausnutzung der Kommunikationsmöglichkeiten des verteilten Systems.

In der Literatur wird eine verteilte Anwendung nicht immer als Teil des verteilten Systems angesehen, sondern als zwei Einheiten nebeneinander (bzw. übereinander) stehend betrachtet (z.B. [Ha05]) .

[Ha05] Ulrike Hammerschall: Verteilte Systeme und Anwendungen, Pearson Studium, 2005.



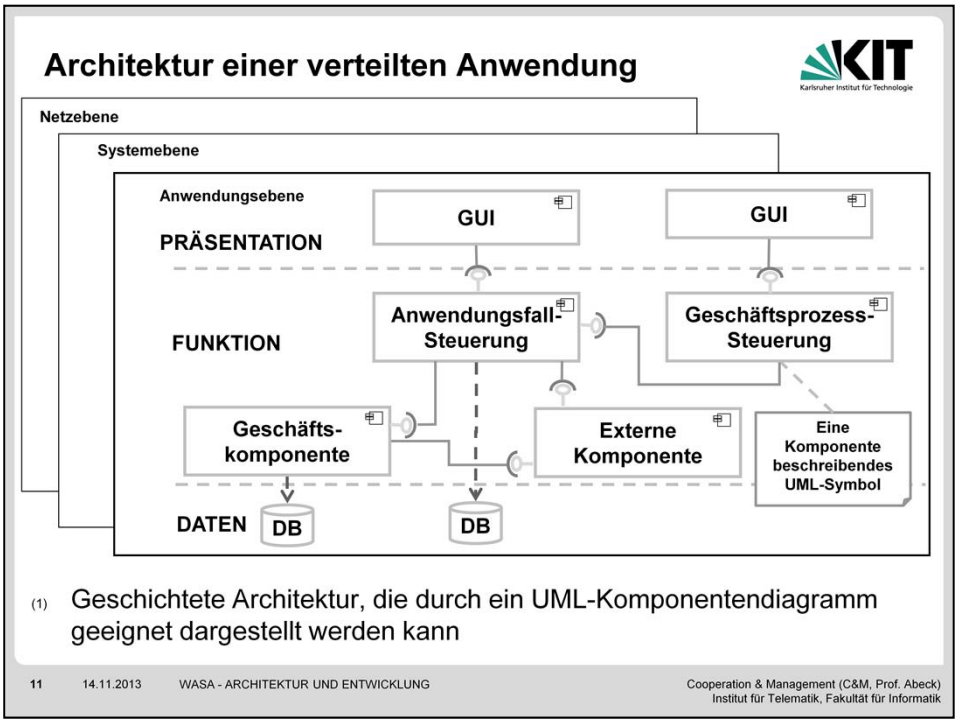
[KR05]

Web-Technologien werden in der Informatik dazu genutzt, verteilte Softwaresysteme zu entwickeln. Softwaresysteme, die über eine graphische Benutzeroberfläche (Graphical User Interface, GUI) räumlich verteilten Benutzern ihre Funktionalität anbieten, sind heute üblicherweise Web-basiert.

(1) Eine Web-Anwendung ist u.a. daran zu erkennen, dass es gewisse standardisierte Kommunikationsprotokolle zum Austausch der Informationen zwischen den das Web-basierte System bildenden Einzelsystemen und den darauf laufenden Web-Client und Web-Server verwendet.
Das Anwendungsprotokoll Hypertext Transfer Protocol (HTTP) kann als das Herz einer Web-Anwendung bezeichnet werden [KR05:89].

HTTP	Hypertext Transfer Protocol
GUI	Graphical User Interface

[KR05] James F. Kurose, Keith W. Ross: Computer Networking – A Top-down Approach Featuring the Internet, 3rd Edition, Addison Wesley, 2005.



[Ha05]
(Anwendungs-, System-, Netzebene) Eine verteilte Anwendung bildet die Anwendungsebene, die im Sinne des Schichtenprinzips auf der Systemebene aufsetzt. Die unterste Schicht stellt die Netzebene dar, durch die die räumliche Verteilung überbrückt wird.

Die Anwendungsebene kann ihrerseits aufgrund von drei logischen Aspekten, die in jeder verteilten Anwendung bestehen, in drei Schichten unterteilt werden [Ha05:25]:

(PRÄSENTATION) Realisierung der Benutzeroberfläche und der Präsentationssteuerung (z.B. Navigation innerhalb einer Web-Seiten)

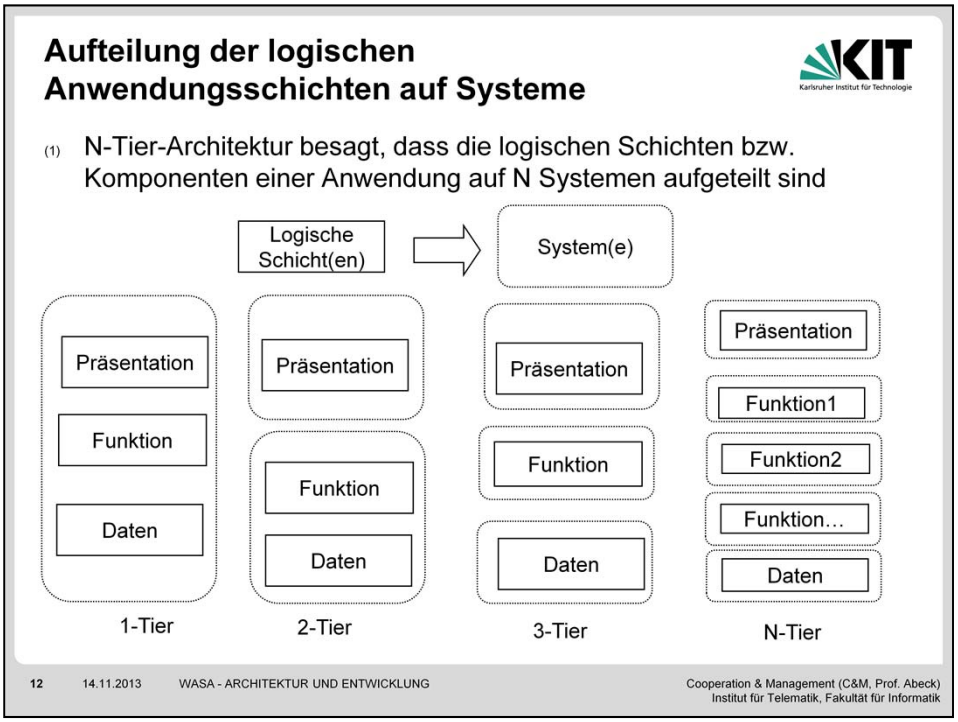
(FUNKTION) Enthält die eigentliche Geschäftslogik in Form von (dynamische Aspekte beinhaltende) Steuerungskomponenten (engl. control components) und (eher statische Aspekte beinhaltende) Geschäftskomponenten (engl. business components).

(DATEN) Persistenzhaltung der Daten in einer relationalen Datenbank oder in einem Dateisystem.

(1) Die Hauptbestandteile einer Softwarearchitektur sind die darin auftretenden Komponenten und deren Beziehungen in Form von bereitgestellter Schnittstelle (engl. provided, Ball-Symbol) und benötigter Schnittstelle (engl. required, Socket-Symbol). Zur Modellierung dieser Inhalte steht in der Unified Modeling Language (UML) das Komponentendiagramm zur Verfügung (siehe Kurseinheit UNIFIED MODELING LANGUAGE).

DB	Data Base
GUI	Graphical User Interface
UML	Unified Modeling Language

[Ha05] Ulrike Hammerschall: Verteilte Systeme und Anwendungen, Pearson Studium, 2005.



[AC+04]
Die Komponenten beschreiben die logische Architektur einer Anwendung und lassen sich auf ein oder mehrere Systeme verteilen. Je nach Anzahl n der an der Anwendung beteiligten Systeme spricht man von N-Tier-Architekturen (engl. tier, dt. Stufe) [AC+04:11, Ha05:25].

(1-Tier) Diese Architektur findet sich im Bereich der Großrechnersysteme, an denen (über einen Terminalserver und ein einfaches Terminal-Netz) 'dumme' Terminals angeschlossen sind.

(2-Tier) Entspricht der klassischen Client-Server-Architektur, wie diese auf der vorhergehenden Seite gezeigt wurde.

(3-Tier) Jede logische Schicht wird jeweils auf einem System ausgeführt.

(N-Tier) Mehr als 3 Tiers entstehen üblicherweise durch eine weitere Verteilung der mittleren Funktionsschicht (ggf. auch der Daten- oder Präsentationsschicht, wenn gewisse Verarbeitungs- oder Steuerungsaspekte auf eigene Systeme ausgelagert werden sollen).

[AC+04] Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju: Web Services: Concepts, Architectures, and Applications, Springer, 2004.
[Ha05] Ulrike Hammerschall: Verteilte Systeme und Anwendungen, Pearson Studium, 2005.

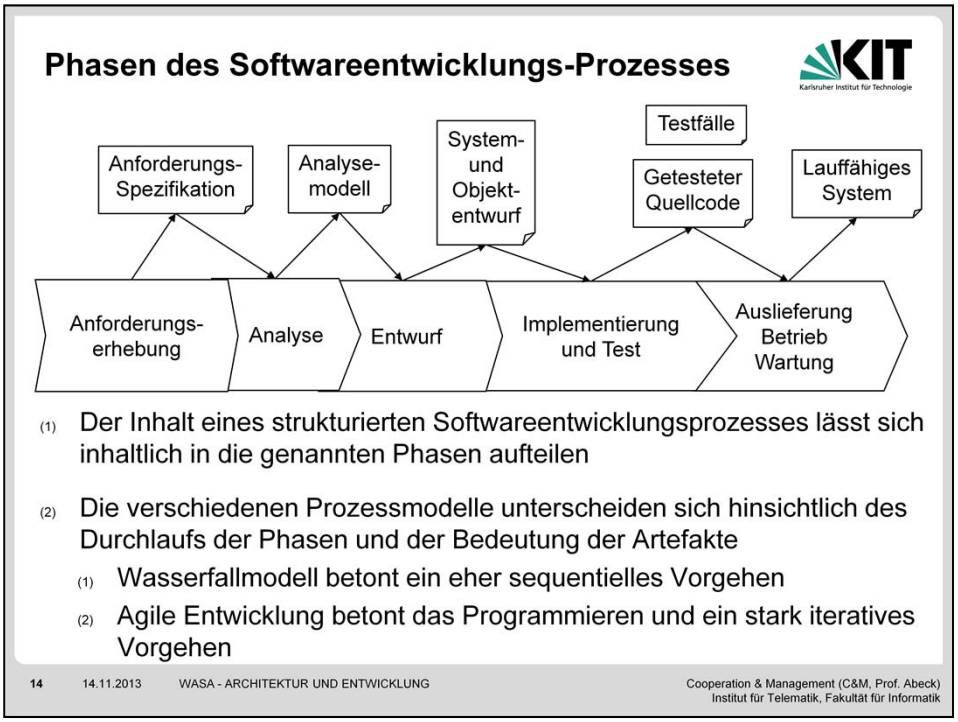
LZ ARCHITEKTUR – ÜA VERTEILTE ANWENDUNG

(1) Welche Transaktionseigenschaft stellt sicher, dass Invarianten nach Ausführung der Transaktion weiterhin gelten?

(2) Wie stehen ein verteiltes System und eine verteilte Anwendung zueinander?

(3) Durch welche Komponenten(typen) werden in der Architektur einer verteilten Anwendung dynamische Aspekte der Geschäftslogik beschrieben?

(4) Was ist ein(e) "Tier" und was bedeutet eine "3-Tier-Architektur"?



[BD04]
(Anforderungserhebung) Das Ziel dieser ersten Phase ist es, die Anforderungen an das System aus Gesprächen und Interviews mit dem Kunden zu entlocken (engl. to elicit, requirements elicitation) und zu dokumentieren [:121].

(Anforderungsspezifikation, Analysemodell) Während die Anforderungsspezifikation vom Benutzer und Kunden des Softwaresystems verstanden werden muss, ist das beim Analysemodell nicht zwingend der Fall. Daher müssen neue Erkenntnisse, die während der Analyse hinsichtlich der Anforderungen an das System gewonnen wurden und im Analysemodell dokumentiert wurden, zurück in die Anforderungsspezifikation übertragen werden, damit diese vom Benutzer verstanden und mit diesem diskutiert werden können [:175].
Beide Dokumente haben den selben Inhalt, der gemäß der Zielgruppen (Benutzer versus Entwickler) in unterschiedlichen Sprachen (informell versus semi-formal) beschrieben ist [:123].

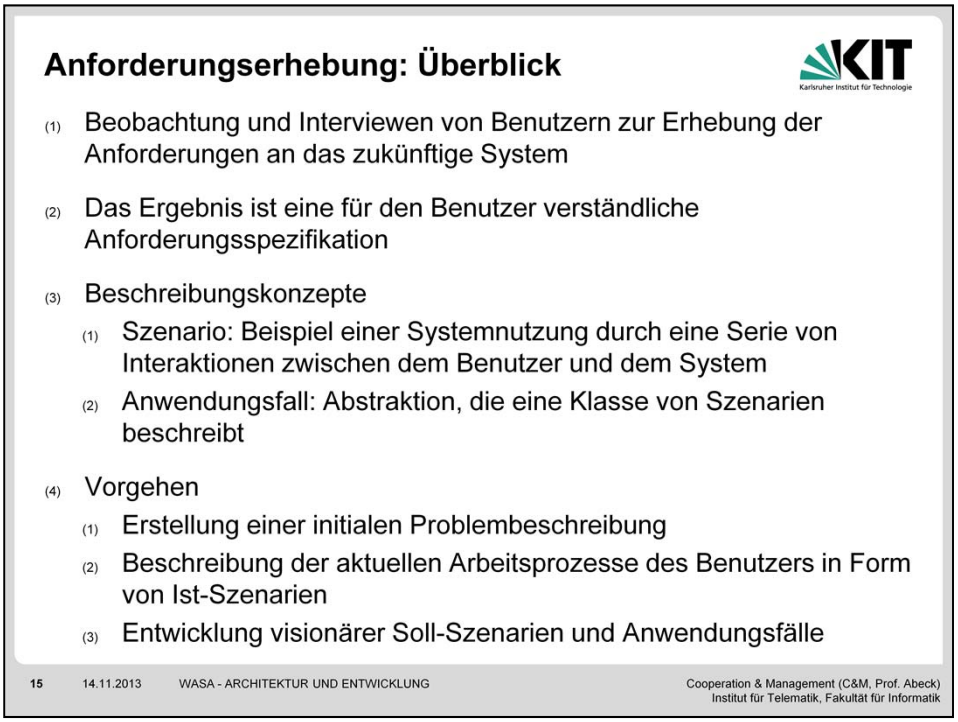
(System- und Objektentwurf) Durch diese in der Entwurfsphase erstellten Artefakte wird beschrieben, wie sich das Gesamtsystem geeignet in Teilsysteme zerlegen (engl. decompose) lässt [:227] und die Anwendungsobjekte werden identifiziert und auf die Systemkomponenten abgebildet [:303].

(1) Die Phasen können ggf. zusammengefasst bzw. weiter unterteilt sein. So ist häufig die Anforderungserhebung keine separate Phase, sondern wird als Teil einer Phase (Analysephase oder Requirements Engineering) gesehen.

(2) Eine Phase kann i.d.R. nicht vollständig abgeschlossen werden, bevor in die nächste Phase eingetreten wird. D.h., der Prozess ist iterativ in dem Sinne, dass ein Rücksprung in eine vorhergehende Phase grundsätzlich erfolgen kann.
(2.1) (2.2) Weitere Beispiele für Prozessmodelle sind: Rational Unified Process (RUP), eXtreme Programming (XP).

RUP	Rational Unified Process
XP	eXtreme Programming

[BD04] Bernd Bruegge, Allen H. Dutoit: Object-Oriented Software Engineering Using UML, Patterns and Java, Pearson Prentice Hall, 2004.



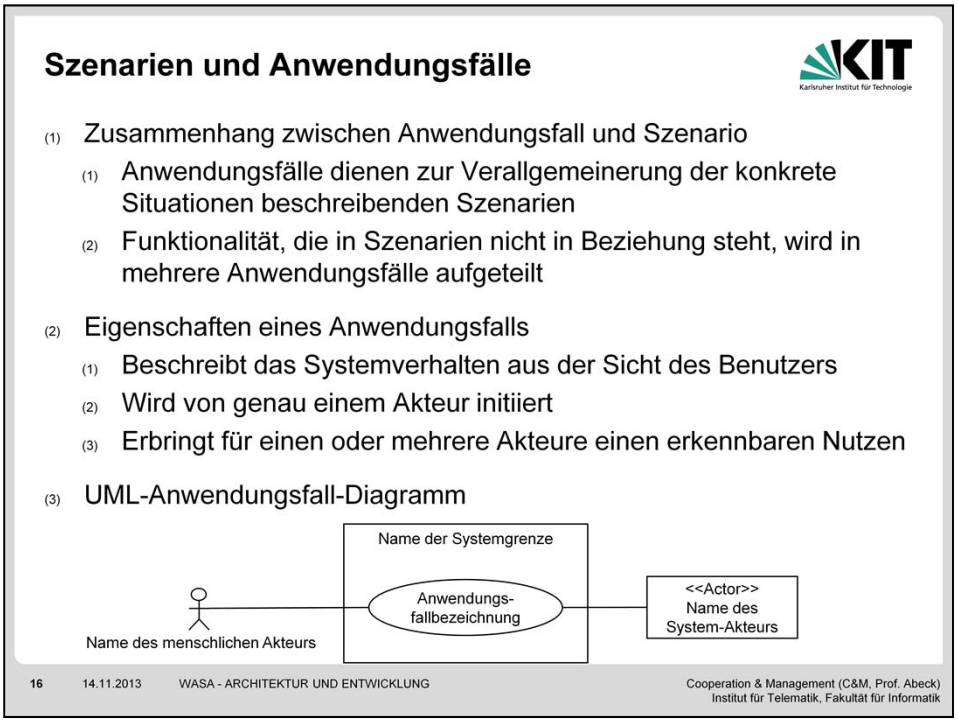
[BD04:121]
(1) Der Kunde und die Benutzer sind die Experten in ihrer Domäne und haben eine allgemeine Idee, was das System leisten soll.

(2) Die Anforderungsspezifikation erfolgt in natürlicher Sprache und ist so zu verfassen, das sie vom Benutzer verstanden wird.

(3) Szenarien und Anwendungsfälle werden in [:121] als Werkzeuge (engl. tools) bezeichnet.
Allgemein handelt es sich bei Szenarien und Anwendungsfällen um Artefakte, die ein Teil der Anforderungsbeschreibung sind.

(4.1) Diese beinhaltet die funktionalen und nicht-funktonalen Anforderungen. Nicht-funktionale Anforderungen betreffen die Qualität (z.B. Antwortzeit, Verfügbarkeit), in der das System die Funktionalität bereitzustellen hat.
Die nicht-funktionalen Anforderungen betreffen die Qualität (z.B. Antwortzeit, Verfügbarkeit), in der das System die Funktionalität bereitzustellen hat.
(4.2) Die Ist-Szenarien dienen dazu, die Arbeitsweise des Benutzers zu verstehen.
(4.3) In diesem Schritt erfolgt eine erste Ableitung der vom System geforderten Funktionalität.

[BD04] Bernd Bruegge, Allen H. Dutoit: Object-Oriented Software Engineering Using UML, Patterns and Java, Pearson Prentice Hall, 2004.



(1.1) Die Verallgemeinerung wird auch dadurch deutlich, dass anstelle konkreter Personen (Akteur-Instanzen) in Anwendungsfällen die Akteure auftreten.
(1.2) Bei Szenarien wird im Gegensatz zu Anwendungsfällen nicht darauf geachtet, dass die behandelte Funktionalität kohärent ist.

(2.1) Wird auch als externes Verhalten bezeichnet [BD04:44]. Ein Anwendungsfall verdeutlicht, dass eine Funktionalität bereitgestellt wird, aber nicht, wie das System diese erbringt [Ke06:201].
(2.2) Durch die Initiierung nehmen die beteiligten Akteure gewisse Funktionalität des Systems in Anspruch.
(2.2) (2.3) Anwendungsfälle sollten eine weitestgehend abgeschlossene und unabhängige Funktionalität beschreiben [BD04:159].

(3) Hinweis: Der Aufbau eines Anwendungsfall-Diagramm ist ausführlich in der Kurseinheit UNIFIED MODELING LANGUAGE beschrieben.

[BD04] Bernd Bruegge, Allen H. Dutoit: Object-Oriented Software Engineering Using UML, Patterns and Java, Pearson Prentice Hall, 2004.
[Ke06] Christoph Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006..

[BD04:126]
Die Aufteilung wurde gemäß dem (F)URPS+-Modell vorgenommen, das im Rahmen des Rational Unified Process entstand und dessen Name aus den Anfangsbuchstaben der aufgeführten Anforderungen zusammengesetzt ist (F steht für Functional und + für die weiteren Kategorien).

(1) Nutzbarkeit ist die Einfachheit, mit der ein Benutzer mit dem System umgehen kann. Nutzbarkeitsanforderungen beinhalten einzuhaltende Konventionen zur Benutzeroberfläche (z.B. Farbschemas, Logos -> Corporate Identity) oder die Verfügbarkeit von Benutzerdokumentation.

(2) Verlässlichkeit ist die Fähigkeit des Systems, die geforderten Funktionen zu erbringen. Anforderungen beinhalten eine akzeptierte Fehlerdurchschnittszeit (engl. mean time to failure) und die Fähigkeit, spezifische Fehler zu entdecken.
Verlässlichkeit bildet neben Robustheit und Ausfallsicherheit (engl. safety) die übergeordnete Eigenschaft der Abhängigkeit (engl. dependability).

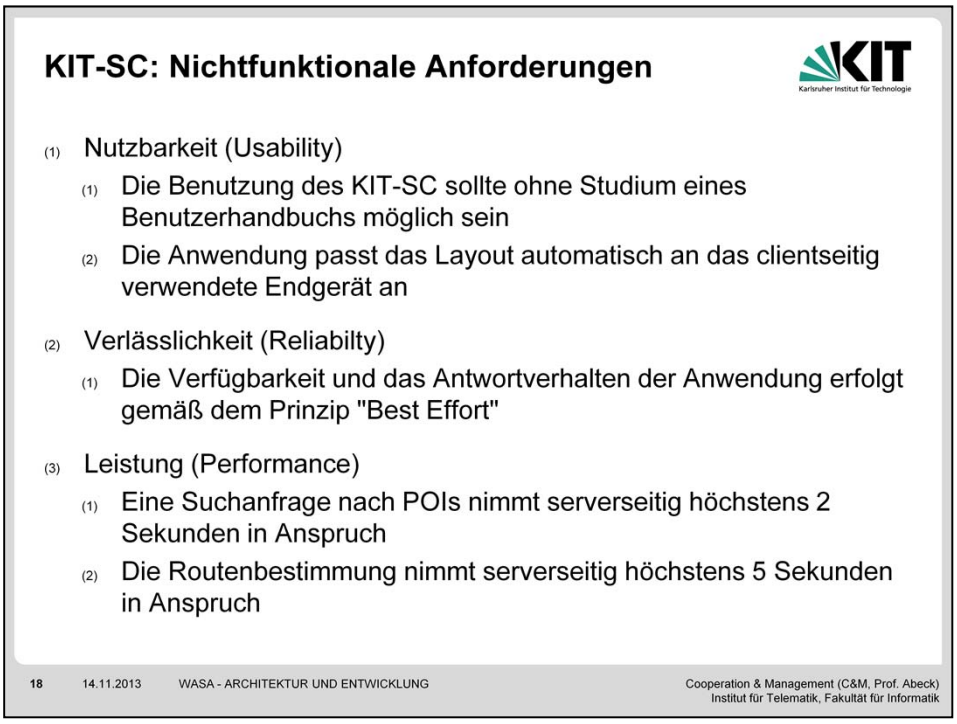
(3) Leistung umfasst verschiedene quantifizierbare Attribute des Systems:(i) Antwortzeit; (ii) Durchsatz; (iii) Verfügbarkeit; (iv) Akkuratheit

(4) Unterstützbarkeit betrifft die Einfachheit von Änderungen am System nach dessen Auslieferung und umfasst Anpassbarkeit, Wartbarkeit und Portabilität.

(5.1) Einschränkung z.B. bzgl. spezifischem Werkzeug, Programmiersprache, Hardware-Plattform [:127]
(5.2) Zwänge durch externe (Alt-)Systeme wie z.B. Austauschformate
(5.3) Zwänge hinsichtlich der Administration und dem Management des Systems in der operationellen Umgebung
(5.4) Zwänge hinsichtlich der Auslieferung, wie z.B. das zu verwendende Installationsmedium
(5.5) Lizenzierungs-, Regulierungs-, Zertifizierungsfragen

(F)URPS (Functionality) Usability Reliability Performance Supportability

[BD04] Bernd Bruegge, Allen H. Dutoit: Object-Oriented Software Engineering Using UML, Patterns and Java, Pearson Prentice Hall, 2004.



Die nicht-funktionalen Anforderungen zum KIT-SC wurden bereits in der Kurseinheit BEISPIEL KIT-Smart-Campus eingeführt.

(1) Die Nutzbarkeit ist eine wesentliche nicht-funktionale Anforderung, durch die die Akzeptanz des Benutzers bzgl. des Systems und damit die Benutzerzufriedenheit bestimmt wird.

(2.1) Nach bestem Bemühen (engl. best effort) heißt, dass keine Garantie hinsichtlich des jeweiligen Qualitätsaspekts (hier: Ausfall und Leistung des KIT-SC) gegeben werden können.

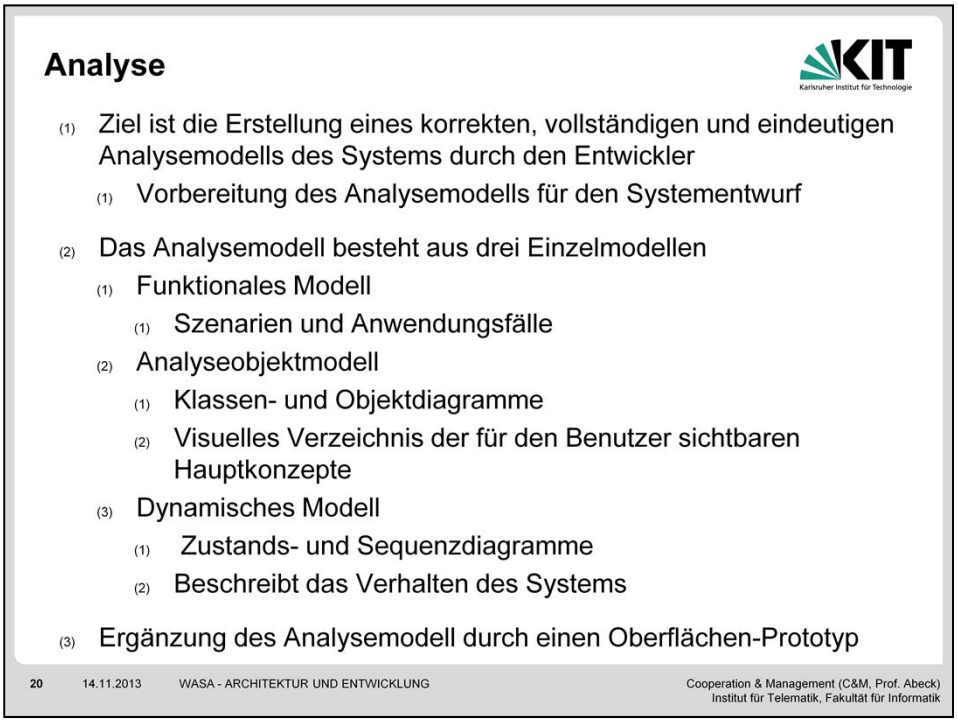
(3.1) (3.2) Diese Zusagen sind kein Widerspruch zu der "Best-Effort"-Aussage zum Antwortverhalten, da hierzu neben der Server-Seite auch das Netzverhalten und die Client-Seite beitragen.

LZ ENTWICKLUNG – ÜA ANFORDERUNGSERHEBUNG

(1) Was ist das Ergebnis der Anforderungserhebung und für welche Phase stellt dieses Ergebnis die Eingabe dar?

(2) Wie stehen Szenarien und Anwendungsfälle zueinander?

(3) Was wird durch die Abkürzung URPS+ beschrieben?



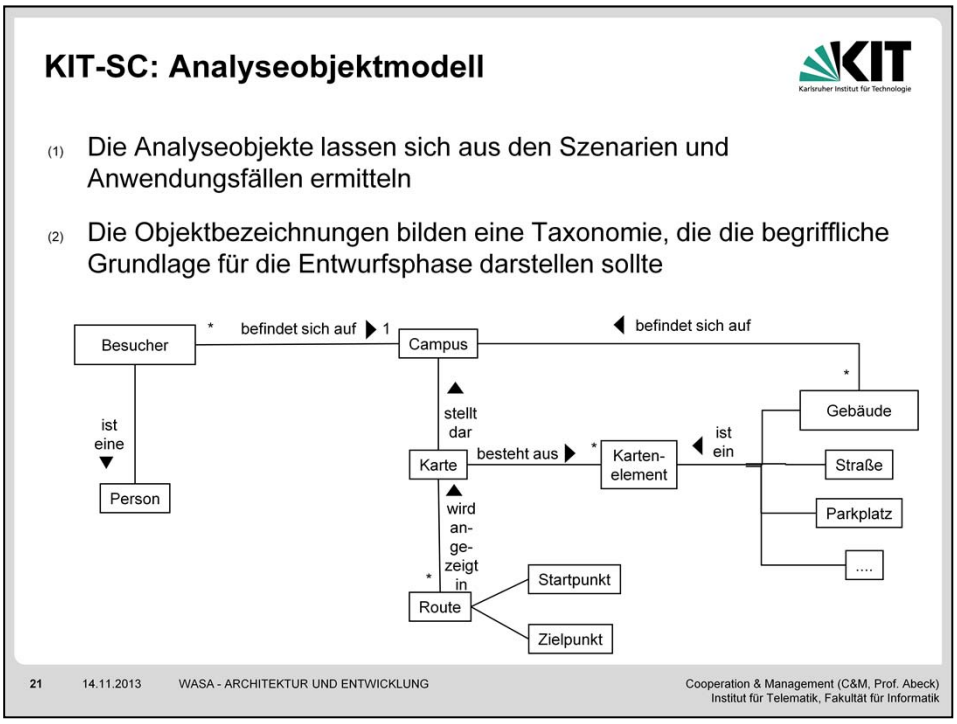
[BD04:173]
Der Zusammenhang zwischen der Analyse und der vorhergehenden Anforderungserhebung besteht darin, dass der Entwickler sich auf die Strukturierung und Formalisierung der erhobenen Anforderungen konzentriert.

(1) Der Entwickler validiert, korrigiert und präzisiert die Anforderungsspezifikation, indem er z.B. die Ausnahmefälle genauer untersucht.

(2) Das Analysemodell muss nicht mehr notwendigerweise von Benutzern und Kunden verstanden werden. Die Objekte und deren Dynamik werden aus den verfeinerten Anwendungsfällen abgeleitet.
(2.1) (2.1.1) Auf Szenarien und Anwendungsfälle wurde bereits ausführlich eingegangen.
(2.2) Das Analyseobjektmodell besteht gemäß Jacobson aus den drei Arten von Objekten (i) Entity, (ii) Boundary und (iii) Control [BD04:177]. Dieser als "Drei-Objekte-Typ" bezeichnete Ansatz beinhaltet einfache Heuristiken, durch die Modelle erstellt werden können, die hinsichtlich Änderungen elastisch (engl. resilient) sind.
(i) Entity-Objekt
- Stellt die persistente Information dar, die vom System behandelt wird
- Kandidaten: in der Anwendungsfall-Beschreibung auftretende Domänenbegriffe oder Dinge der realen Welt
(ii) Boundary-Objekt
- Beinhalten die Interaktionen zwischen Akteur und System
- Kandidaten: Oberflächenelemente und Nachrichten, über die der Benutzer mit dem System in Kontakt kommt
(iii) Control-Objekte
- Realisieren Anwendungsfälle
- Kandidaten: Jeder Anwendungsfall oder jeder Akteur im Anwendungsfall
(2.2.1) Eine Beschreibung der Diagramme wird in der Kurseinheit UNIFIED MODELING LANGUAGE gegeben.
(2.2.2) Zu beachten ist, dass das gesamte Analysemodell das System aus der Sicht des Benutzers darstellt.
Beispiele für Klassen, die keine Analyseklassen sind: Datenbank, SessionManager, Netzwerk
(2.3.1) Ein Sequenzdiagramm verknüpft Anwendungsfälle mit Objekten und ist nur bedingt für die Kommunikation mit dem Benutzer geeignet. Eigenschaften von Sequenzdiagrammen sind:
- Modellieren Interaktionen von Objekten
- Konzentrieren sich auf den Nachrichtenaustausch
- Haben nicht die Darstellung aller möglichen Abläufe zum Ziel

(3) Hierdurch lässt sich das Verhalten des Systems, das im dynamischen Modell beschrieben ist, für den Benutzer in angemessener Form darstellen.

[BD04] Bernd Bruegge, Allen H. Dutoit: Object-Oriented Software Engineering Using UML, Patterns and Java, Pearson Prentice Hall, 2004.



(1) Eine solche Bestimmung kann systematisch nach festgelegten Regeln und teilweise automatisiert erfolgen.

(2) Hierbei wird häufig eine Übersetzung von der Sprache des Benutzers (hier deutsch) in die englische Sprache vorgenommen.

(Besucher, Campus) Ein Besucher befindet sich auf genau einem Campus (Kardinalität 1) und auf dem Campus können sich beliebig viele Besucher befinden (*).

(Karte) Durch eine Karte (engl. map) wird der Campus und die sich darauf befindlichen Gegenstände wie Gebäude oder Straßen dargestellt.

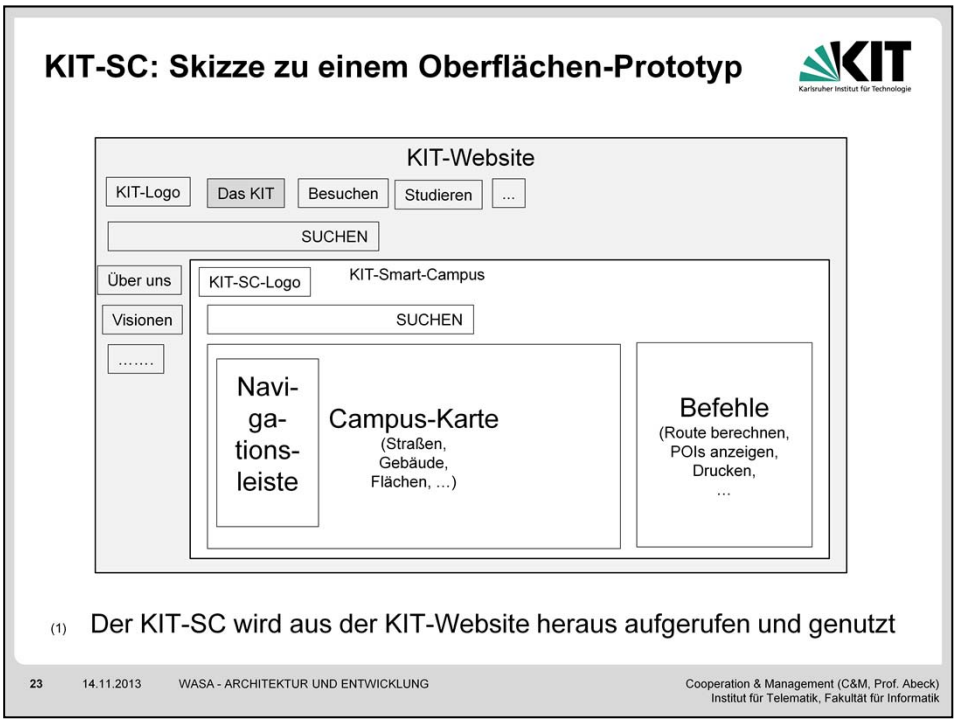
(Route) Besteht aus einem Startpunkt und einem Zielpunkt. Start- und/oder Zielpunkt können direkt durch Koordinaten oder durch eine Person oder durch bestimmte Kartenelemente, insbesondere Gebäude, angegeben werden.

Hinweise:
(1) Der Standardwert der Multiplizität ist 1, d.h. eine fehlende Angabe bedeutet, dass genau eine Ausprägung auftritt.
(2) Aus Gründen der Übersichtlichkeit wurden nicht alle Assoziationen im Diagramm berücksichtigt.

Ein Oberflächen-Prototyp wird auch als horizontaler Prototyp bezeichnet: Ausschließlich die (horizontale) Präsentationsschicht der Web-Anwendung wird mit Inhalt versehen, während die darunterliegende Geschäftslogik- und Datenschicht leer bleiben. Im Gegensatz dazu führt ein vertikaler Prototyp einen vertikalen Schnitt durch die Anwendungsarchitektur, indem eine Funktionalität (z.B. ein Anwendungsfall) von der Oberfläche über die Geschäftslogik bis zur Datenebene umgesetzt wird.

(1) Auf die genaue graphische Gestaltung und Aspekte der Nutzbarkeit (engl. usability) wird an dieser Stelle noch kein Schwerpunkt gelegt. Allerdings ist es sinnvoll, grundsätzliche Entscheidungen hinsichtlich der Strukturierung der Oberfläche und der Anordnung der Bedienelemente bereits in den ersten Skizzen vorzusehen, damit der Kunden eine Vorstellung des zukünftigen Systems erhält.
(1.1) Neben den statischen Aspekten der Oberfläche kann der Prototyp auch dazu genutzt werden, dynamische Aspekte aufzuzeigen und mit dem Kunden zu diskutieren. Hierbei besteht ein enger Zusammenhang mit den Sequenzdiagrammen, die aufgrund ihrer inhärenten Komplexität ggf. schwieriger nachzuvollziehen sind als die Darstellung der Abläufe am Oberflächen-Prototyp.

(2) Je nach Projekt kann es sinnvoll sein, mehr oder weniger Aufwand bei der Erstellung des Prototyps zu betreiben.
(2.1) Solche Skizzen entstehen meist im Gespräch zwischen dem Entwickler und dem Kunden.
(2.2) Der Einsatz eines graphischen Editors erhöht den Aufwand, bietet aber gegenüber handschriftlichen Skizzen den Vorteil der besseren Änderbarkeit. Außerdem kann die Darstellung der eingesetzten Symbole so gewählt werden, dass sich der Kunde das Aussehen der zukünftigen Oberfläche bereits gut vorstellen kann.
(2.3) Es werden erste Software-Artefakte entwickelt, was einen erhöhten Aufwand bedeutet. Das Ziel muss sein, dass diese Artefakte mit entsprechenden Anpassungen in der Implementierungsphase genutzt werden können.



Die Skizze vermittelt einen ersten Eindruck über die Oberflächen- Elemente und eine denkbare Anordnung (die sich im Verlauf der Entwicklung auch noch ändern kann).

(1) Wie die Skizze zeigt, soll der KIT-SC ein Teil der KIT-Website sein, deren Bestandteile grau hinterlegt sind. Daher soll bei der Gestaltung der KIT-SC-Oberfläche auf das Coporate Identity (CI) des KIT geachtet werden.

Die KIT-SC-Oberfläche besteht aus den folgenden Teilen:
(KIT-Smart-Campus) Titelzeile mit Logo und einem Suchfenster
(Campus-Karte) Fenster, in dem der aktuelle Ausschnitt des Campus angezeigt wird. Teil der Karte ist eine Navigationsleiste, die ein Verschieben und Größenskalierung der Karte ermöglicht.
(Befehle) Fenster, in dem die von der KIT-SC-Anwendung bereitgestellte Funktionalität in Form von durch den Benutzer auswählbaren Befehlen angeboten wird.

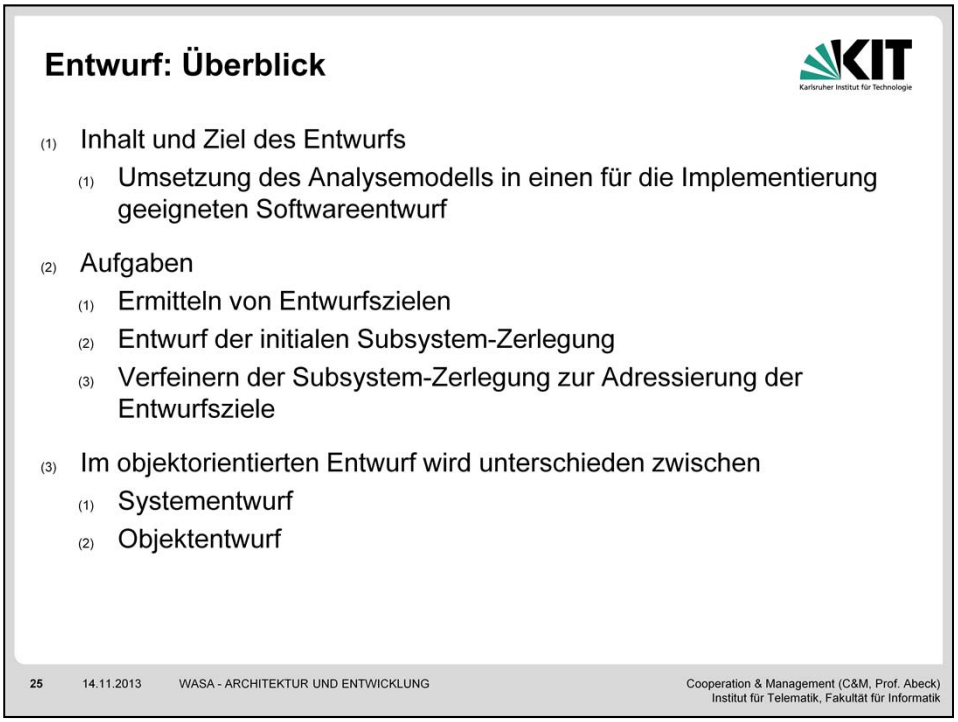
CI Corporate Identity

LZ ENTWICKLUNG – ÜA ANALYSE

(1) Was wird aus der Anforderungserhebungsphase in das Analysemodell übernommen?

(2) Woraus werden Analyseobjekte abgeleitet und welchen Zweck erfüllen diese?

(3) Wozu dient ein Oberflächen-Prototyp und was ist am Beispiel des KIT-SC daran ersichtlich?



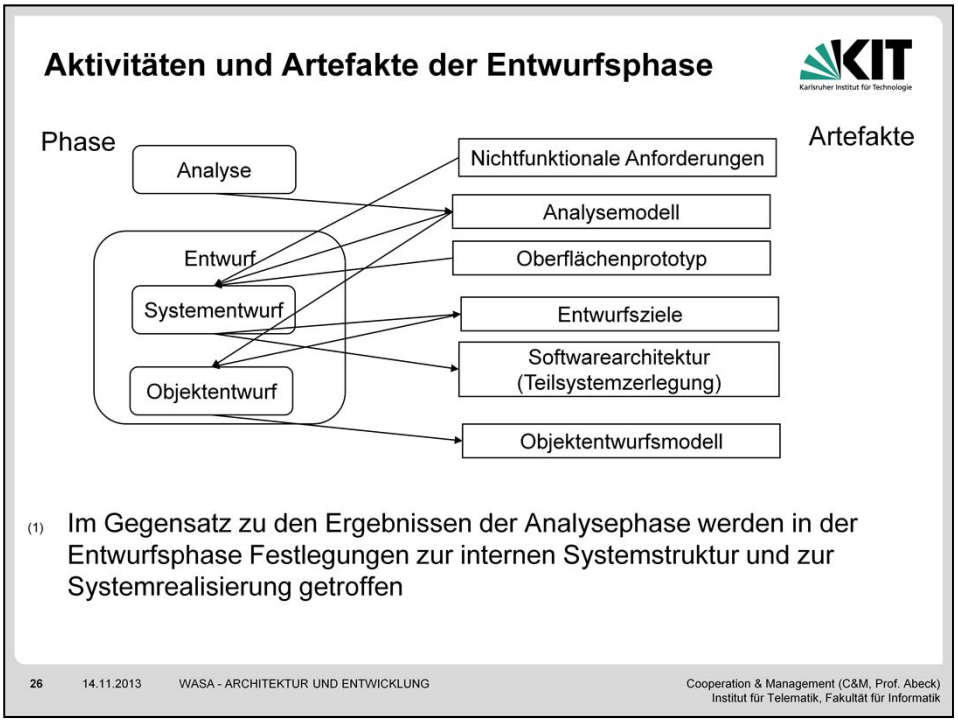
[BD04:223]
Der Entwurf eines Softwaresystems ist nicht algorithmisch, da insbesondere nicht alle Entwurfsfragen (engl. issues) vorausgesehen werden können [:223]

(1.1) Ein Softwareentwurf umfasst die folgenden Aspekte [So01:68]:
- Struktur der zu implementierenden Software (-> Architektur)
- Systemdaten
- Schnittstellen zwischen den Systemkomponenten
- Verwendete Algorithmen

Jede der Aufgaben steht im Zusammenhang mit dem übergeordneten Problem der Zerlegung des Systems.
(2.1) Die Entwickler identifizieren und priorisieren die gewünschten Qualitäten des zu optimierenden Systems.
(2.2) Die Entwickler zerlegen das System in kleinere Teile auf der Grundlage von Anwendungsfällen und Analysemodellen. Ausgangspunkt sind Standard-Architekturstile.
(2.3) Iterative Verbesserung der initialen Zerlegung, bis alle Ziele zufriedenstellend erfüllt sind.

(3.1) Der Systementwurf beschreibt das System in Form seiner Architektur, wozu u.a. zählen [:223+303]:
- die Zerlegung in Teilsysteme
- der globale Steuerfluss
- das Datenmanagement (engl. persistency management)
- Festlegung der Hardware/Software-Plattform
(3.2) Der Objektentwurf schließt die Lücke zwischen den in der Analysephase identifizierten Anwendungsobjekten und den vom Systementwurf bereitgestellten (schlüsselfertigen) Komponenten (engl. off-the-shelf components, shelf: Lager).

[BD04] Bernd Bruegge, Allen H. Dutoit: Object-Oriented Software Engineering Using UML, Patterns and Java, Pearson Prentice Hall, 2004.
[So01] Ian Sommerville: Software Engineering, Addison-Wesley Verlag, Pearson Studium, 2001.



[BD04:226]
(Nichtfunktionale Anforderungen) Beispiele sind eine maximale Antwortzeit, minimaler Durchsatz, Zuverlässigkeit (engl. reliability), zu unterstützende Betriebssystem-Plattform.

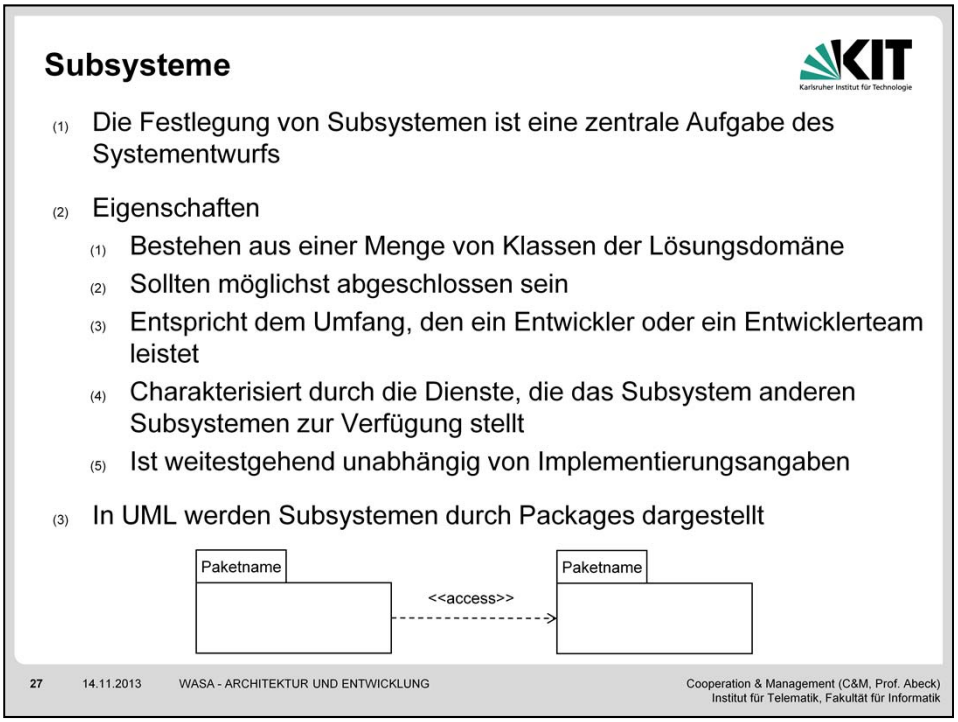
(Analysemodell) Beschreibt das System vollständig aus der Sicht des Nutzers (Akteur, engl. actor) [:226].

(Entwurfsziele) Werden von den nichtfunktionalen Anforderungen abgeleitet und beschreiben die Qualitäten des Systems, die vom Entwickler optimiert werden sollen [:227].

(Softwarearchitektur) Beschreibt die Verantwortlichkeiten und Abhängigkeiten zwischen Subsystemen, die Abbildung von Subsystemen auf die Hardware und wichtige strategische Entscheidungen, die den Steuerfluss, die Zugangskontrolle und die Datenspeicherung betreffen [:227].

(Objektentwurfsmodell) Schließen der Lücke zwischen den Anwendungsobjekten und den Fertigkomponenten (engl. off-the-shelf), indem zusätzliche Lösungsobjekte identifiziert und bestehende Objekte verfeinert werden [:303].

[BD04] Bernd Bruegge, Allen H. Dutoit: Object-Oriented Software Engineering Using UML, Patterns and Java, Pearson Prentice Hall, 2004.



[BD04:228]
(1) Durch die Identifikation geeigneter Subsysteme wird das Ziel verfolgt, die Komplexität des Gesamtsystems zu zerlegen und damit zu reduzieren.

(2.1) Ausgangspunkt sind die Klassen, die in der Analysephase identifiziert wurden.
(2.2) Diese Eigenschaft führt zu der Anforderung nach hoher Kohäsion und geringer Kopplung.
(2.3) Die Subsysteme sollten auch deshalb weitestgehend unabhängig voneinander sein, um den Kommunikations-Overhead zwischen den Entwicklern bzw. Entwicklerteams minimal zu halten.
(2.4) Ein Dienst ist in diesem Zusammenhang eine Menge an Operationen, die einen gemeinsamen Zweck teilen. Diese Menge an Operationen bildet die Subsystem-Schnittstelle, aus der im Objektentwurf die API (engl. Application Programming Interface) durch Verfeinerung und Erweiterung entwickelt wird [:230].
(2.5) Das betrifft insbesondere die Schnittstelle, die nicht auf interne Datenstrukturen, wie z.B. verkettete Listen, Arrays oder Hashtabellen verweisen sollte.

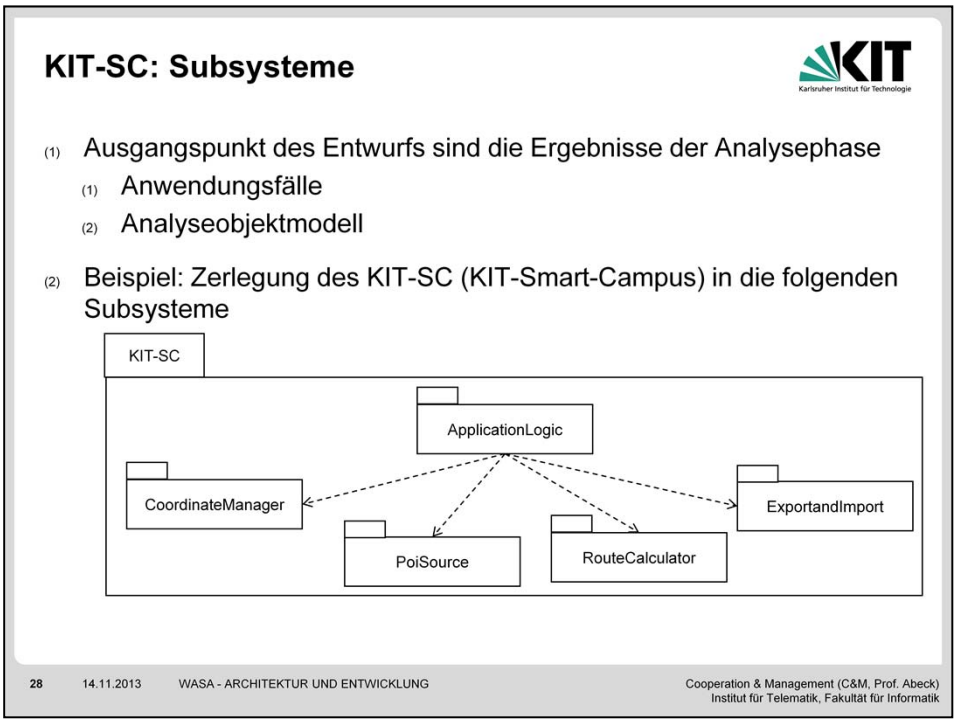
(3) Auch gewisse Programmiersprachen stellen Sprachkonstrukte zur Subsystem-Bildung zur Verfügung (z.B. Packages in Java oder Module in Modula).
Unabhängig von der Unterstützung durch die Modellierungs- oder Programmiersprache muss die Subsystem-Zerlegung sauber dokumentiert werden [:229].

(Paketname) Pakete ermöglichen in UML eine logische Gruppierung von Elementen und die Definition von Namensräumen [Ke06:175].
In Paketen können wiederum Pakete enthalten sein, so dass eine hierarchische Gruppierung möglich ist. Durch Pakete werden insbesondere Klassen gruppiert.

Hinweis: Die Subsysteme (bzw. Pakete) stellen Kandidaten für die Verteilung (engl. deployment) von Teilen des Softwaresystems auf reale Systeme (d.h. Knoten) dar. Da der Entwurf aber noch unabhängig von der konkreten Abbildung auf die Hardware sein soll, werden Paketdiagramme und noch nicht Verteilungsdiagramme verwendet.

API Application Programming Interface

[BD04] Bernd Bruegge, Allen H. Dutoit: Object-Oriented Software Engineering Using UML, Patterns and Java, Pearson Prentice Hall, 2004.
[Ke06] Christoph Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006.



(1) Diese Ergebnisse sind in der vorhergehenden Phase der Anforderungserhebung zu entwickeln.

(2) Die Teilsystem-Zerlegung ergibt sich aus dem Herauslösen der enthaltenen (Teil-) Anwendungsfälle, die in einer <<include>>-Beziehung zum Gesamt- Anwendungsfall stehen.
Da die Artefakte zum Entwurf und zur Implementierung des KIT-SC-Systems durchgängig englischsprachig gehalten werden, erfolgt an dieser Stelle eine Übersetzung der in der Analysephase ggf. verwendeten deutschsprachigen Begriffe.

(KIT-SC) Das Paketdiagramm orientiert sich an [C&M-PSE1011-E-T1:2.1].

(CoordinateManager) Bietet eine Schnittstelle zum Umgang mit Koordinaten an, insbesondere zur Konvertierung von "String" zu "WorldPosition" und von "WorldPosition" zu "String" an. [:2.1.5].
(PoiSource) Bietet eine Schnittstelle zum Beziehen von POIs (PointOfInterest) aus verschiedenen Datenquellen und über unterschiedliche Anfragen [:2.1.6].
(RouteCalculator) Ermöglicht die Berechnung einer Route.
(ExportandImport) Vereinigt die Import- und Exportfunktionen von Ansichten der graphischen Benutzeroberfläche.

[C&M-PSE1011-E-T1] Cooperation & Management: Praxis der Software-Entwicklung (PSE), WiSe10/11, Entwurfsdokument (E), Team1 (T1), Karlsruher Institut für Technologie (KIT).

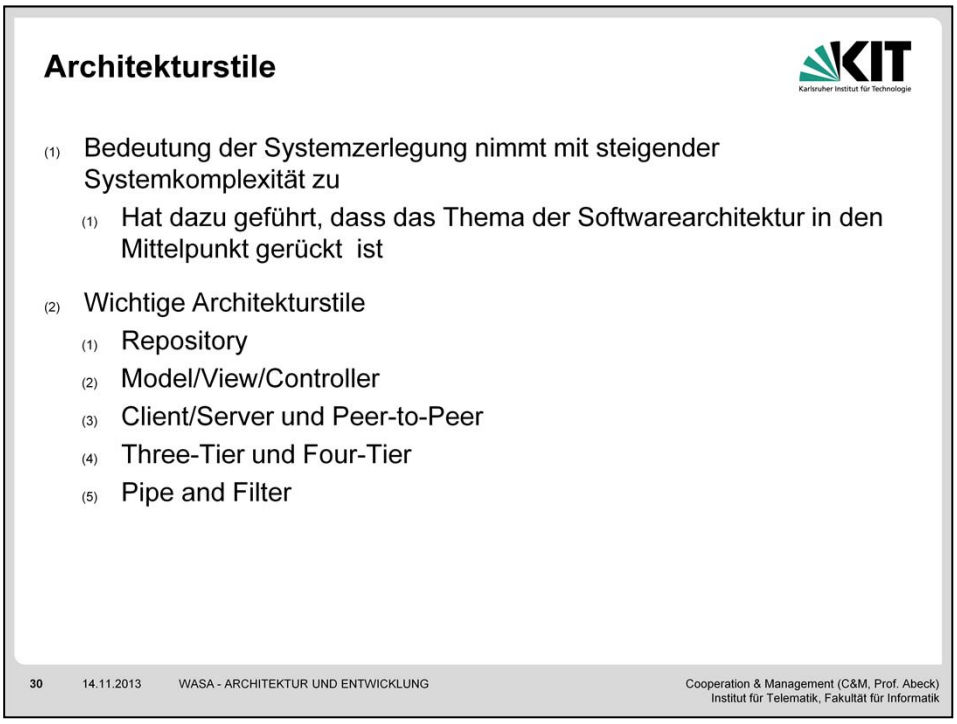
[BD04:230]
Kopplung und Kohäsion sind zwei wichtige Entwurfsziele und Eigenschaften von Subsystemen.

(1.1) Ist die Anzahl gering, so spricht man von loser Kopplung (engl. loose coupling), andernfalls von starker Kopplung (engl. strong coupling).
(1.2.1) Der daraus resultierende Vorteil besteht darin, dass ein Fehler (-> Robustheit) in einem Teilsystem oder eine Änderung (-> Wartbarkeit) an einem Teilsystem nur einen kleinen Einfluss auf die übrigen Teilsysteme hat.
(1.2.2) Ein Beispiel ist die Ergänzung eines Entwurfs, in dem Subsysteme direkt auf ein Subsystem "Datenbank" zugreifen, um ein zusätzliches Subsystem "Speicherung". Die Idee ist, dass die Zugriffsschnittstelle von "Speicherung" von den Datenbankdetails abstrahiert und sich dadurch weniger häufig ändert als die Schnittstelle von "Datenbank".
Es muss immer ein Kompromiss zwischen loser Kopplung und dem Aufwand gefunden werden, der durch die Einführung von zusätzlichen Subsystemen entsteht.

(2.1) Die Kohäsion ist dann hoch (high), wenn die im Subsystem enthaltenen Objekte eng verbunden sind und ähnliche Aufgaben erledigen. Stehen die Objekte in keinem engen Bezug, ist die Kohäsion niedrig (engl. low).
(2.2) Die Kohäsion eines Subsystems lässt sich durch eine Zerlegung verringern. Das führt aber aufgrund der erhöhten Anzahl an Schnittstellen zu einer verstärkten Kopplung, weshalb Kopplung und Kohäsion konkurrierende Ziele darstellen.

(3.1) (3.2) (3.3) Wird auch als 7plusminus2-Konzept bezeichnet.
(3.2) Eine Schicht ist eine Gruppierung von Subsystemen, die in Beziehung stehende Dienste erbringen [:234].
(3.3) Oft reichen für einen guten Systementwurf sogar nur drei Schichten aus.

[BD04] Bernd Bruegge, Allen H. Dutoit: Object-Oriented Software Engineering Using UML, Patterns and Java, Pearson Prentice Hall, 2004.



[BD04:237]

(1) Die Veränderung oder Korrektur von schwachen Systemzerlegungen ist schwierig, sobald die Entwicklung gestartet wurde, da die meisten Subsystem-Schnittstellen geändert werden müssen.

(1.1) Diese adressiert gerade die angemessene Systemzerlegung sowie den damit zusammenhängenden globalen Steuerfluss oder zwischen den Subsystemen laufende Kommunikationsprotokolle.

(2) Diese Architekturstile stellen eine gute Basis für die Architektur unterschiedlichster Systeme dar.

(2.1) Zugriff auf eine einzige Datenstruktur, die Repository genannt wird. Durch die zentrale Lokation der Daten sind Nebenläufigkeit und Integrationsfragen zwischen Subsystemen einfacher zu behandeln.

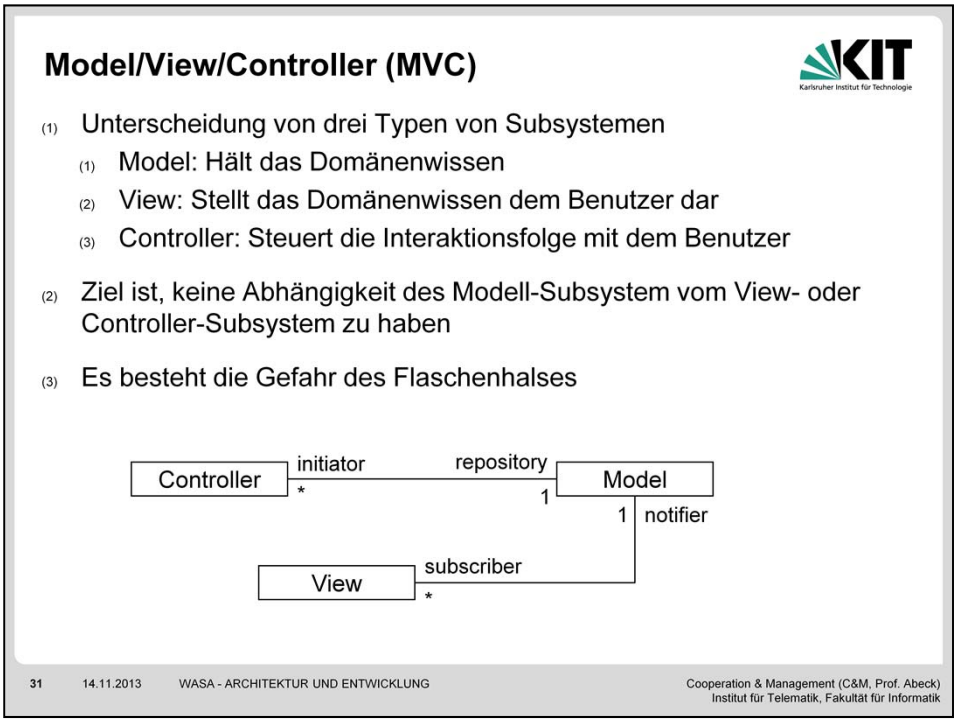
(2.2) Der nachfolgend noch genauer beschriebene MVC-Stil ist ein Spezialfall des Repository-Stils, bei dem das Modell die zentrale Datenstruktur implementiert.

(2.3) Peer-to-Peer ist eine Verallgemeinerung des Client/Server-Architekturstils; beide Stile werden nachfolgend noch genauer behandelt.

(2.4) Entspricht der weiter vorne in dieser Kurseinheit behandelten logischen 3-Schichtenarchitektur, wobei hier der Begriff des "Tier" synonym zur Schicht (engl. layer) verwendet wird.

(2.5) Eine Verkettung von Subsystemen (Filter) durch Verbindungen (engl. pipe). Das prominenteste Beispiel dieses Stils ist die Unix Shell.

[BD04] Bernd Bruegge, Allen H. Dutoit: Object-Oriented Software Engineering Using UML, Patterns and Java, Pearson Prentice Hall, 2004.

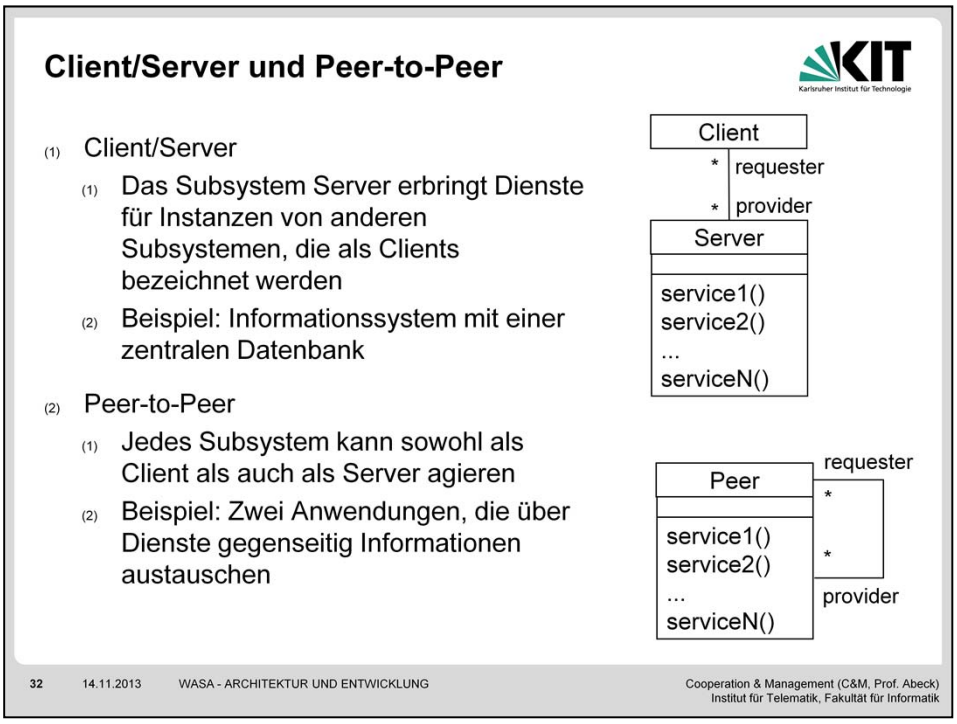


[BD04:239]
(1.1) Das Domänenwissen wird in der zentralen Datenstruktur (entsprechend eines Repository, dt. Ablage) gehalten.
(1.2) Der View (dt. Sicht) wird durch ein Subscribe/Notify-Protokoll (dt. Abbonnieren-Benachrichtigen) vom Modell benachrichtigt, sobald das Modell geändert wurde.
(1.3) Der Controller (dt. Steuerung) sammelt die Eingaben vom Benutzer und sendet Nachrichten an das Modell.

(2) Der Grund hierfür ist, dass sich die View und der Controller sehr viel häufiger ändern als das Modell und durch die Unabhängigkeit solche Änderungen keinerlei Einfluss auf das Modell haben.

(3) Der Flaschenhals ist ein grundsätzliches Problem des Repository-Architekturstils, weil das Modell eine zentrale Datenstruktur darstellt.

[BD04] Bernd Bruegge, Allen H. Dutoit: Object-Oriented Software Engineering Using UML, Patterns and Java, Pearson Prentice Hall, 2004.

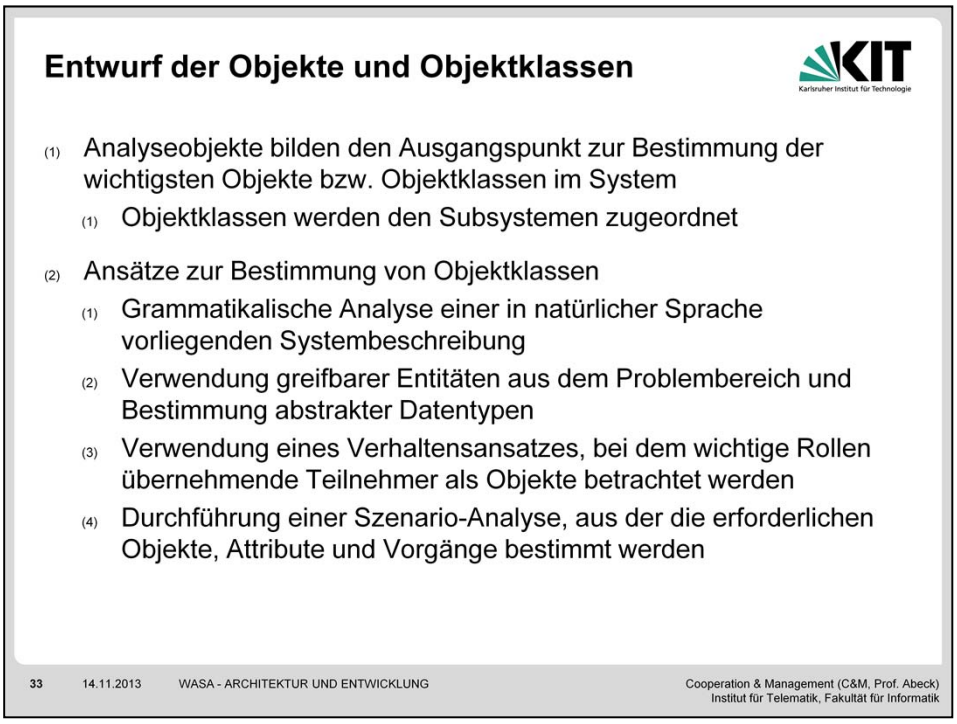


[BD04:242]
(1) Der Client-Server-Architekturstil ist eine Spezialisierung des Repository-Architekturstils.
(1.1) Die Anfrage eines Dienstes erfolgt meist durch einen RPC-Mechanismus (Java RMI, CORBA, HTTP).
(1.2) Der Client/Server-Architekturstil ist für verteilte Systeme geeignet, die große Datenmengen verwalten.

(Client-Server-Assoziation) Server (-> provider) bedienen beliebig viele (*) dem Server im Voraus nicht bekannte Clients (-> requester) und ein Client kann auf zahlreiche (*) verschiedene Server zugreifen, wie die im Web bestehende Situation verdeutlicht.

(2) Der Peer-to-Peer-Architekturstil (engl. peer, dt. Ebenbürtiger, engl. peer-to-peer, dt. gleichberechtigt) ist eine Verallgemeinerung des Client-Server-Architekturstils.
(2.1) In dem Sinne, dass jedes Subsystem Services anfragen (engl. request) und erbringen (engl. provide) kann.
(2.2) Aufgrund des komplexeren Steuerflusses sind Peer-to-Peer-Systeme schwieriger zu entwerfen als Client/Server-Systeme [:243].

[BD04] Bernd Bruegge, Allen H. Dutoit: Object-Oriented Software Engineering Using UML, Patterns and Java, Pearson Prentice Hall, 2004.

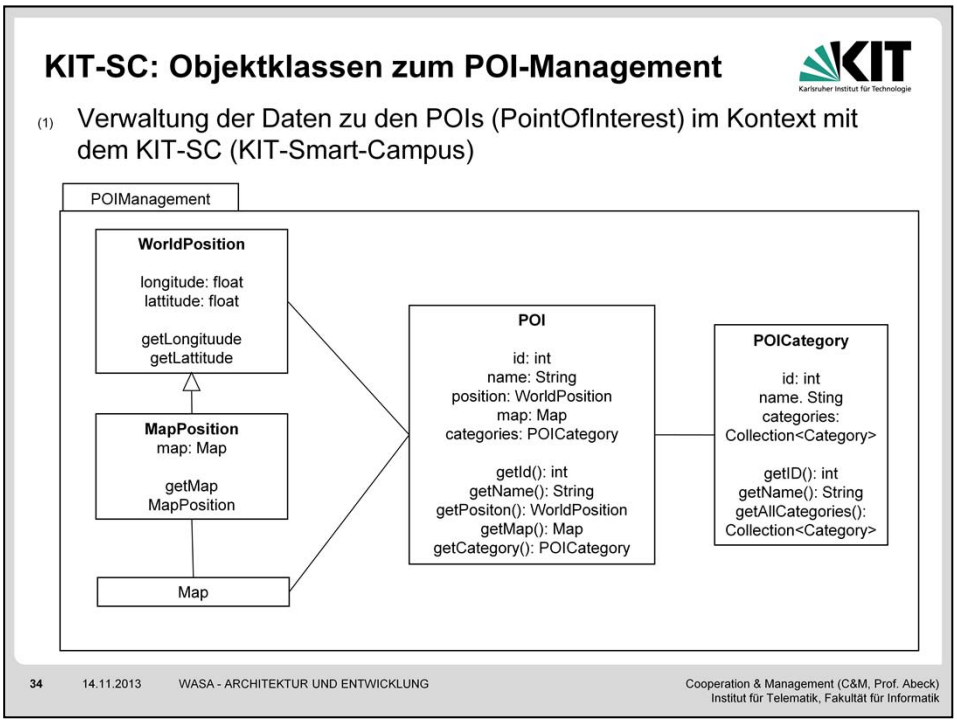


[So01:281]
Das hier verfolgte Vorgehen orientiert an den Beschreibungen von Sommerville [So01:281] und Brügge [BD04:228].

(1) Es bestehen zahlreiche Wissensquellen für die Entdeckung von Objekten und Objektklassen. Grundsätzlich erfolgt eine Verfeinerung der in der Analyse aufgestellten informellen Beschreibungen [So01:282].
(1.1) Es sollte erreicht werden, dass die in einem Subsystem enthaltenen Objektklassen eine möglichst hohe Kohäsion besitzen und Objektklassen verschiedener Subsysteme eine möglichst geringe Kohäsion aufweisen [BD04:232].

(2) Die nachfolgenden Ansätze schließen sich nicht gegensätzlich aus, sondern können synergetisch dazu beitragen, die Objekte zu bestimmen, da viele verschiedene Wissensquellen genutzt werden sollten [:282].
(2.1) Hierbei werden Objekte und Attribute zu Substantiven und Vorgänge und Dienste werden zu Verben.
(2.2) Beispiele sind Rollen wie die des Managers, Ereignisse wie Anfragen, Standorte wie Büros.
Die abstrakten Datentypen bestimmen die aus der Lösungsdomäne abzuleitenden Speicherstrukturen, die zur Unterstützung dieser Objekte notwendig sein könnten.
(2.3) Hierbei muss der Entwickler zunächst das gesamte Systemverhalten verstehen. Hieraus wird abgeleitet, wer das Verhalten auslöst und wer sich daran beteiligt (-> Teilnehmer).
(2.4) Dieser Ansatz kann durch die CRC-Analysemethode, bei der die Analysten und Entwickler die Rollen von Objekten aufgreifen, sehr effektiv unterstützt werden.

[BD04] Bernd Bruegge, Allen H. Dutoit: Object-Oriented Software Engineering Using UML, Patterns and Java, Pearson Prentice Hall, 2004.
[So01] Ian Sommerville: Software Engineering, Addison-Wesley Verlag, Pearson Studium, 2001.



[CM-PR-PSE-EKIT-SC]
(1) Die Objektklassen zum POI-Management behandeln die folgenden Aspekte zu den POIs (PointOfInterest):
(i) Die Informationen, die zu einem POI relevant sind, werden in der Klasse "POI" erfasst.
(ii) Zur Gruppierung von POIs aufgrund der Art des Standorts dient die Klasse "POICategory".
(iii) Die Einordnung in Karten und damit die Einbettung des POI-Managements in die Gesamtanwendung KIT-Smart-Campus erfolgt über die Klasse "Map" sowie die Klassen zur Positionsangabe "MapPosition" und "WorldPosition".

(POI) Hierdurch werden die zu einem POI relevanten Informationen festgelegt.
(getMap) Gibt die Karte zurück, auf der der POI liegt.
(getCategory) Gibt die Kategorie zurück, zu der der POI gehört.

(POICategory) Dient zur Darstellung von POI-Kategorien und verwaltet außerdem alle im System vorhandenen Kategorien [:36].

(Map) Dient zur Darstellung einer Karte und verwaltet außerdem alle im System vorhandenen Karten [:37].
(MapPosition) Dient zur Darstellung einer Position auf einer bestimmten Karte [:39].
(WorldPosition) Dient zur Darstellung einer Position auf dem KIT-Campus durch Angabe der geographischen Länge (engl. longitude) und der geographischen Breite (engl. latitude).

Beispiele für Klassen, die im KIT-SC auftreten, aber nicht unmittelbar das POI-Management betreffen, sind:
- Route: Weg, der durch seinen Start- und Endpunkt vom Typ "MapPosition" bestimmt ist [:35].
- MapSection: Kartenausschnitt, der durch zwei Positionsangaben vom Typ "WorldPosition" (linke obere Ecke und rechte untere Ecke) bestimmt wird [:40].
Hinweis: Ein Zusammenhang wird zwischen diesen beiden Klassen durch die zur Klasse "Route" gehörenden Methode "getBoundingBox()" hergestellt, die die Route umschließenden Kartenausschnitt wiedergibt [35].

[CM-PR-PSE-EKIT-SC] Cooperation & Management: Praxis der Software-Entwicklung, Karlsruher Institut für Technologie (KIT), Entwurfsdokument zum KIT-Smart-Campus, C&M (Prof. Abeck).

LZ ENTWICKLUNG – ÜA SYSTEM- UND OBJEKTENTWURF

(1) Was ist die übergeordnete Aufgabe des Entwurfs?

(2) Welche Artefakte fließen in den Entwurf ein?

(3) Was sagen die Heuristiken über Architekturschichten und Subsysteme aus?

(4) Warum ist die Kohäsion ein konkurrierendes Entwurfsziel zur Kopplung?

(5) Wodurch entsteht beim Model-View-Controller-Architekturstil die Gefahr eines Flaschenhalses?

(6) Wie hängen die beiden Architekturstile Client/Server und Peer-to-Peer zusammen?

(7) Was ist der Ausgangspunkt der Objektklassen-Bestimmung und was ist einer von verschiedenen kombinierbaren Ansätzen?