Grundlagen Software Engineering

Software Engineering-Prinzipien

© Prof. Dr. Liggesmeyer, 1

Inhalt

- Prinzip der Abstraktion
- Prinzip der Strukturierung
- Prinzip der Hierarchisierung
- Prinzip der Modularisierung
- Geheimnisprinzip
- Prinzip der Lokalität
- Prinzip der Verbalisierung
- Abhängigkeiten zwischen den Prinzipien
- Allgemeine Methoden

© Prof. Dr. Liggesmeyer, 2

Prinzipien und Methoden

- Einführung
  - Die Software-Technik ist gekennzeichnet durch eine hohe Innovationsgeschwindigkeit
  - Im Bereich der Methoden und Werkzeuge werden ständig neue Methoden und neue Werkzeuge angekündigt
  - In dem rasanten Wandel fällt es schwer, das »Konstante«, das »Übergeordnete« zu erkennen
  - Im folgenden werden einige Prinzipien und Methoden vorgestellt, die eine gewisse »Allgemeingültigkeit« für die Software-Technik besitzen

© Prof. Dr. Liggesmeyer, 3

Prinzipien

- Prinzipien
  - sind Grundsätze, die man seinem Handeln zugrunde legt
  - sind allgemeingültig, abstrakt, allgemeinster Art
  - bilden eine theoretische Grundlage
  - werden aus Erfahrungen und Erkenntnissen hergeleitet und durch sie bestätigt

© Prof. Dr. Liggesmeyer, 4

Prinzipien

- Prinzipien
  - Prinzip der Abstraktion
  - Prinzip der Strukturierung
  - Prinzip der Hierarchisierung
  - Prinzip der Modularisierung
  - Geheimnisprinzip
  - Prinzip der Lokalität
  - Prinzip der Verbalisierung
  - Alle für den Software-Entwicklungsprozess relevant
    - Definition / Entwurf / Implementierung
  - Auch im Software-Management & -QS
  - Weitere Prinzipien decken meist nur spezielle Anwendungsbereiche ab

© Prof. Dr. Liggesmeyer, 5

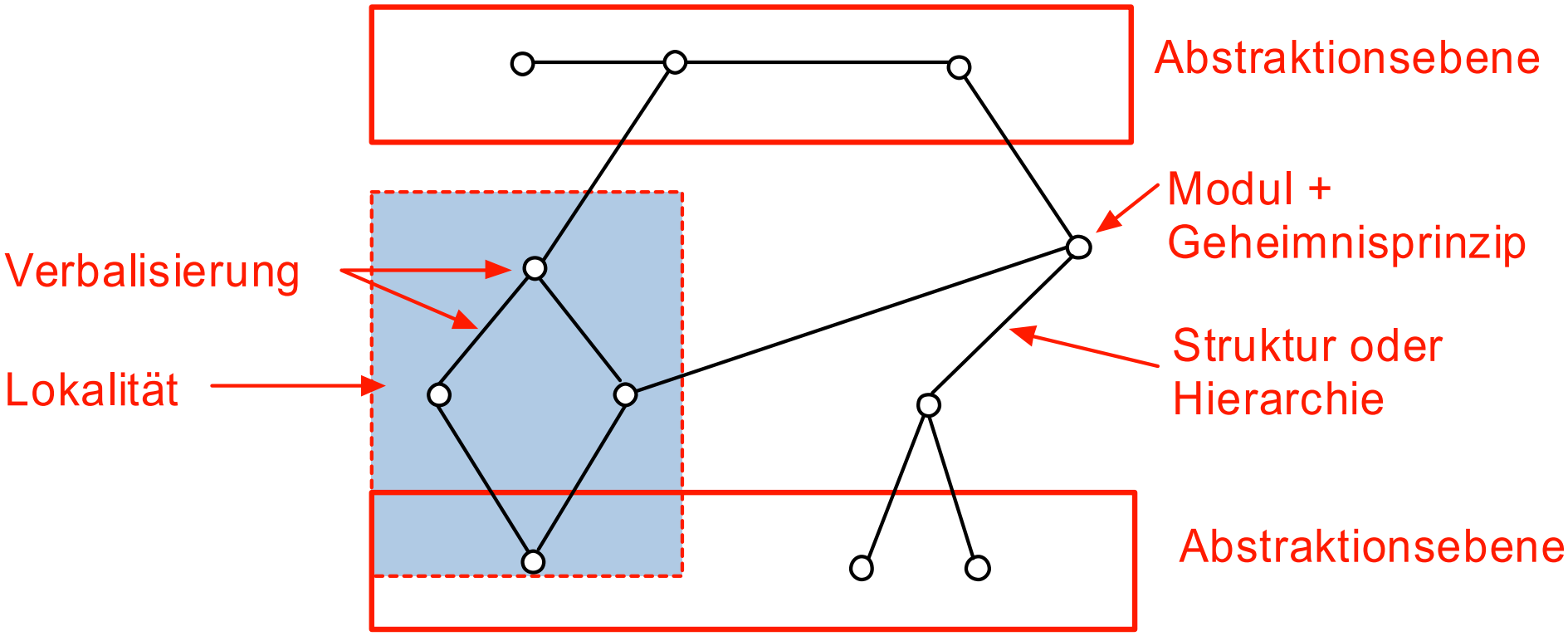
Prinzipien

- System
  - Ist ein Ausschnitt aus der realen oder gedanklichen Welt, bestehend aus Systemkomponenten bzw. Subsystemen, die untereinander in verschiedenen Beziehungen stehen
  - Systeme können in allgemeiner Form durch Graphen dargestellt werden
    - Systemkomponenten bzw. Subsysteme = (markierte) Knoten(punkte)
    - Beziehungen (Relationen) = verbindende (benannte) Linien (Kanten)

© Prof. Dr. Liggesmeyer, 6

Prinzipien

- Prinzipien und ihr Wirkungsbereich



Prinzip der Abstraktion

- Abstraktion
  - Verallgemeinerung, das Absehen vom Besonderen und Einzelnen, das Loslösen vom Dinglichen
  - Gegenteil von Konkretisierung
- Abstrahieren
  - Abgehen vom Konkreten, das Herausheben des Wesentlichen aus dem Zufälligen, das Beiseite lassen von Unwesentlichem, das Erkennen gleicher Merkmale
- Abstrakt
  - Abstrakt bedeutet also nicht gegenständlich, nicht konkret, nicht anschaulich, begrifflich verallgemeinert, theoretisch

© Prof. Dr. Liggesmeyer, 8

Prinzip der Abstraktion

- Modellbildung
  - Anstelle von Abstraktion
  - Durch das Abstrahieren vom Konkreten macht man ein Modell der realen Welt, d.h. das Modell repräsentiert die reale Welt durch sein charakteristisches Verhalten
- Abstraktion und Konkretisierung
  - Sind nicht absolut, sondern relativ
  - Es gibt mehr oder weniger Abstraktion und Konkretisierung
- Abstraktionsebenen
  - Abstufungen der Abstraktion

© Prof. Dr. Liggesmeyer, 9

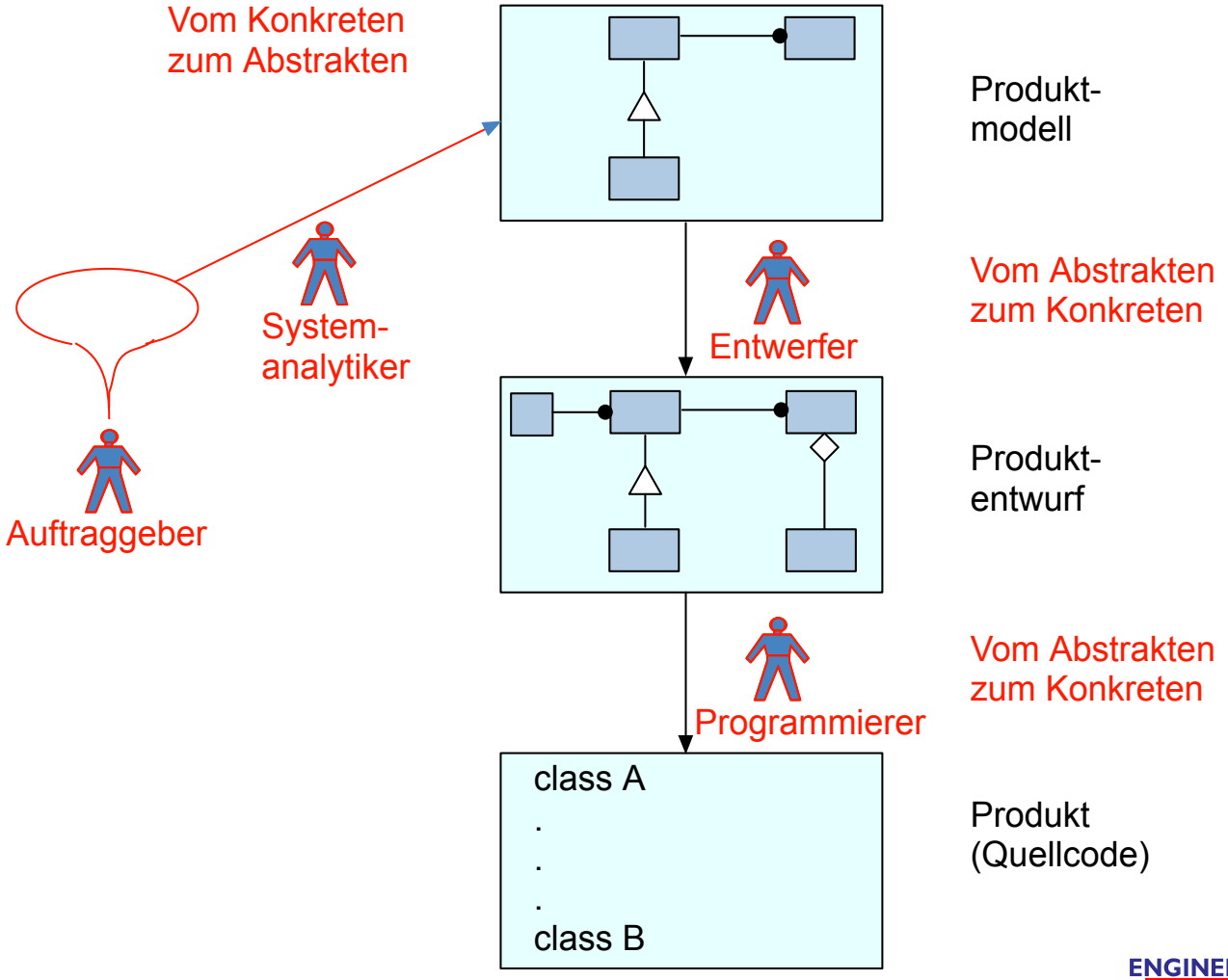
Prinzip der Abstraktion

- Schwierigkeiten
  - Es werden nicht die geeigneten Abstraktionen gefunden
  - Abstrahieren ist eine äußerst anspruchsvolle Tätigkeit
    - Es ist meist sehr schwierig, aus vielen konkreten Tatsachen das Wesentliche zu isolieren
  - Software-Entwicklung
    - Ständiges Wechselspiel zwischen »Abstrahieren« und »Konkretisieren«

© Prof. Dr. Liggesmeyer, 10

Prinzip der Abstraktion

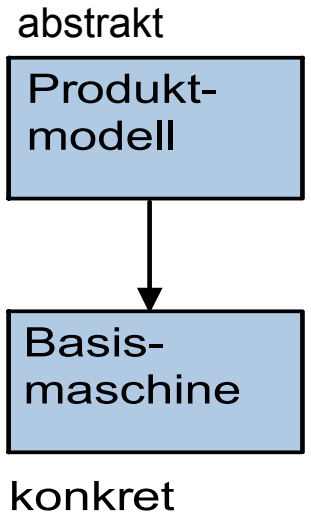
- Vom »Konkreten« zum »Abstrakten« und umgekehrt



© Prof. Dr. Liggesmeyer, 11

Prinzip der Abstraktion

- Vom Produktmodell zur Basismaschine
  - Kluft zwischen dem abstrakten Produktmodell und der konkreten Basismaschine muss durch die schrittweise Konkretisierung des Produktmodells überbrückt werden
  - Die Basismaschine repräsentiert in der Regel die vom Betriebssystem und der verwendeten Programmiersprache bzw. deren Laufzeitsystem bereitgestellte Funktionalität



Prinzip der Abstraktion

- Vergröberung vs. Abstraktion /Wendt 93/
  - Bei vergröberten Systemdarstellungen geht es darum, alle Details wegzulassen, die für ein intuitives Grobverständnis nicht gebraucht werden
  - Aussagen, die in der vergröberten Systemdarstellung gemacht werden, müssen nicht unbedingt auf das konkrete System exakt zutreffen
  - Beim Übergang von einer vergröberten Systemdarstellung zum konkreten System findet eine Präzisierung statt, d.h. dass bereits gemachte Aussagen teilweise wieder revidiert werden

© Prof. Dr. Liggesmeyer, 13

Prinzip der Abstraktion

- Demgegenüber muss beim Übergang von einer abstrahierten Systemdarstellung zum konkreten System eine Detaillierung vorgenommen werden, d.h. es werden weitere Aussagen hinzugebracht, ohne bereits gemachte Aussagen zu revidieren
- Alles was in einer abstrahierten Systemdarstellung ausgesagt wird, trifft ohne Abstriche auf das konkrete System zu
- Nach Ansicht von /Wendt 93/ sind komplexe Systeme ohne intensive Verwendung vergröberter Systemdarstellungen nicht beherrschbar

© Prof. Dr. Liggesmeyer, 14

Prinzip der Abstraktion

- Bezogen auf die Möglichkeiten des »Reengineering« bedeutet dies, dass aus vorhandenen Systembeschreibungen, insbesondere aus Quellcode, nicht automatisch Vergröberungen erzeugt werden können
- Die heutigen Konzepte und Methoden der Software-Entwicklung konzentrieren sich auf die Abstraktion

© Prof. Dr. Liggesmeyer, 15

Prinzip der Abstraktion

- Beispiel
  - Vergröberung:
    - Im OOA-Modell besteht zwischen zwei Klassen keine Beziehung
    - Im OOD-Modell wird bei der Präzisierung eine Beziehung ergänzt
  - Abstraktion:
    - Im OOA-Modell enthält eine Klasse das Attribut Geburtsdatum
    - Im OOD-Modell wird das abgeleitete Attribut Alter hinzugefügt

© Prof. Dr. Liggesmeyer, 16

Prinzip der Abstraktion

- Neue Konzepte
  - Sachverhalte stärker und mehrdimensional abstrahiert
- Dimensionen bzw. Sichten in der SWT
  - Daten
  - Funktionen
  - Dynamik
  - Benutzungsoberfläche

© Prof. Dr. Liggesmeyer, 17

Prinzip der Abstraktion

- Konzepte mit eindimensionaler Modellierung bzw. Abstraktion
  - Funktions-Baum
  - Kontroll-Strukturen
  - Datenflussdiagramm
  - Zustands-Automaten
  - *Data Dictionary*
  - Petri-Netz
  - Jackson-Diagramm
  - Interaktions-Strukturen
  - *Entity-Relationship*-Modell
  - Regeln

© Prof. Dr. Liggesmeyer, 18

Prinzip der Abstraktion

- Unterschiede im Abstraktionsgrad dieser Konzepte
  - Während z.B. ein Funktions-Baum und ein Datenflussdiagramm Sachverhalte nur »schwach« abstrahieren, ist ein *Entity-Relationship*-Modell eine »starke« Abstraktion

© Prof. Dr. Liggesmeyer, 19

Prinzip der Abstraktion

- Konzepte mit zweidimensionaler Modellierung
  - Strukturierte Analyse
    - Daten und Funktionen
  - *Realtime*-Erweiterung der strukturierten Analyse
    - Funktionen und Dynamik
  - Klassen-Diagramme
    - Daten und Funktionen
  - Die ersten beiden Konzepte führen zu einer »schwachen« Abstraktion, das letzte Konzept zu einer »starken« Abstraktion

© Prof. Dr. Liggesmeyer, 20

Prinzip der Abstraktion

- Die Ermittlung geeigneter Abstraktionen ist um so leichter
  - je weniger Dimensionen berücksichtigt werden müssen
  - je schwächer das zugrundeliegende Konzept abstrahiert
- Der Einsatz neuer Konzepte erfordert daher höhere Abstraktionsfähigkeiten von den Mitarbeitern

© Prof. Dr. Liggesmeyer, 21

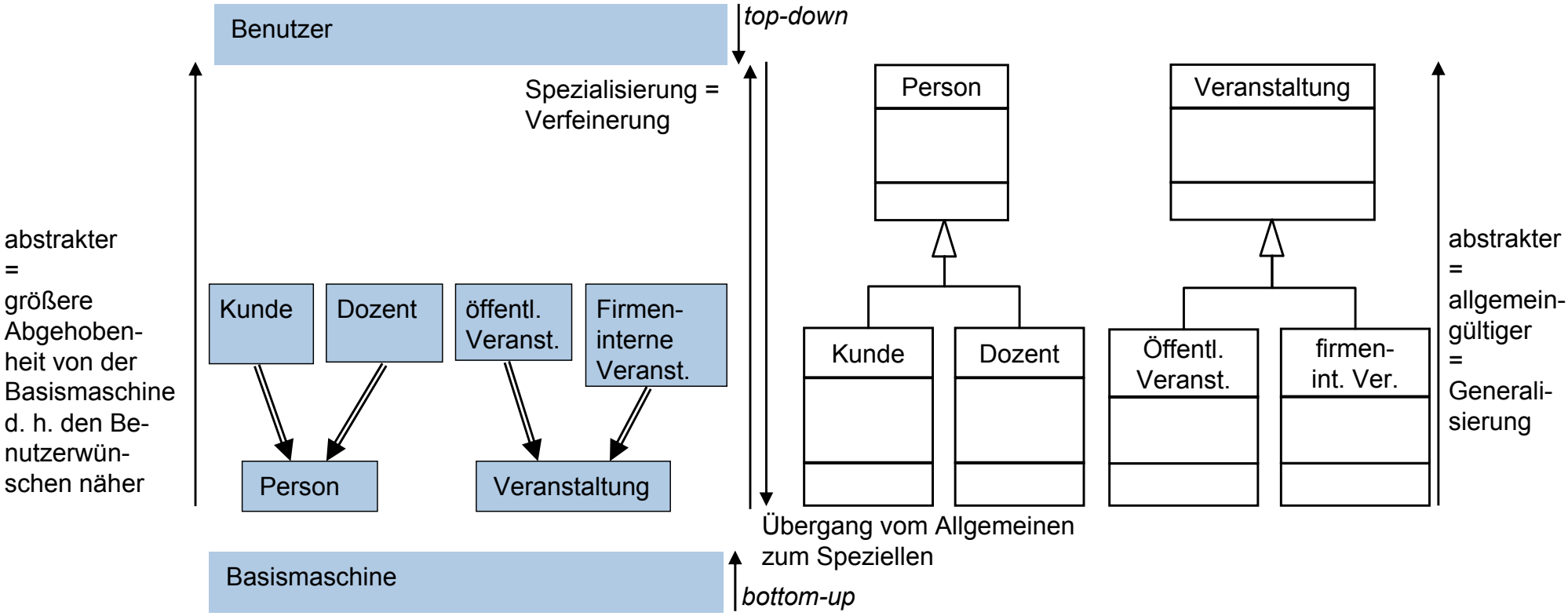
Prinzip der Abstraktion

- Analoge Entwicklung beim Software-Entwurf
  - funktionale Abstraktion
    - eindimensional, »schwache« Abstraktion
  - Datenabstraktion mit Datenobjekten
    - zweidimensional, »schwache« Abstraktion«
  - Datenabstraktion mit abstrakten Datentypen
    - zweidimensional, »starke« Abstraktion
  - Klassen
    - zweidimensional, »starke« Abstraktion

© Prof. Dr. Liggesmeyer, 22

Prinzip der Abstraktion

- Was »abstrakt« ist, kann unterschiedlich sein!



© Prof. Dr. Liggesmeyer, 23

Prinzip der Abstraktion

- Datenabstraktion mit abstrakten Datentypen
  - »abstrakter« = »größere Abgehobenheit von der Basismaschine«
  - »Abstrakter« = »Den Benutzerwünschen näher«
- Objektorientierter Entwurf
  - »abstrakter« = »allgemeingültiger«
  - »Abstrahieren« = »Generalisieren«
- 3 Abstraktionsebenen
  - Exemplar-Ebene
  - Typ-Ebene
  - Meta-Ebene, genauer gesagt die Meta-Typen-Ebene

© Prof. Dr. Liggesmeyer, 24

Prinzip der Abstraktion

- Exemplar-Ebene
  - Beschreibung menschlicher Handlungen, spezifischer Ereignisse, konkreter Sachverhalte in ihren Beziehungen und/oder zeitlichen Abläufen
  - Passive Elemente, auf die Bezug genommen wird, sind konkrete Gegenstände, Personen oder begriffliche Artefakte wie
    - »Haus Nr. 25«
    - »Guido Neumann«
    - »Konto 2324«

© Prof. Dr. Liggesmeyer, 25

Prinzip der Abstraktion

- Beziehungen verbinden diese miteinander z. B.
  - »Guido Neumann wohnt in Haus Nr. 25«
  - »Guido Neumann besitzt Konto 2324«
- Aktive Elemente sind einmalige, konkrete Handlungen, Aktivitäten, das Eintreten von Ereignissen wie
  - »Guido Neumann zieht in Haus Nr. 25 ein«
  - »Guido Neumann eröffnet Konto 2324«

© Prof. Dr. Liggesmeyer, 26

Prinzip der Abstraktion

- Typ-Ebene
  - Erlaubt generalisierende Aussagen über Elemente der Exemplar-Ebene
  - Passive Elemente sind Typen von Gegenständen, Personen oder begrifflichen Artefakten wie
    - »Haus«, »Kunde«, »Konto«
  - Klassifizierung gleichwertiger Beziehungen führt zu Beziehungstypen wie
    - »Kunde besitzt Konto«
  - Durch die standardisierte Beschreibung von Handlungen, Aktivitäten oder des Eintretens von Ereignissen entstehen aktive Elemente der Typ-Ebene wie
    - »Konto eröffnen«

© Prof. Dr. Liggesmeyer, 27

Prinzip der Abstraktion

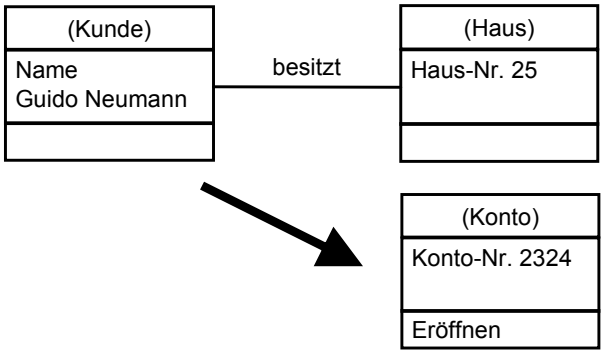
- Meta-Ebene
  - Entsteht durch die Zusammenfassung gleichartiger Elemente der Typ-Ebene
    - Passive Elemente wie »Haus« und »Kunde« werden unter dem Begriff »Klasse« subsumiert
    - »Attribute« fassen unstrukturierte passive Elemente zusammen
    - »Operationen« fassen aktive Elemente zusammen
  - Die Elemente der Meta-Ebene dienen hauptsächlich zur Begriffsbildung auf der Typ-Ebene
  - Außerdem ist die Meta-Ebene für die Werkzeugunterstützung relevant
    - Viele Werkzeuge arbeiten mit einem Meta-Modell

© Prof. Dr. Liggesmeyer, 28

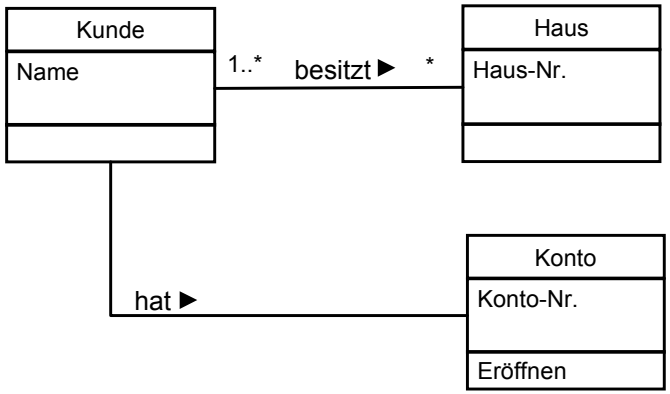
Prinzip der Abstraktion

- Beispiele für objektorientierte Abstraktionen

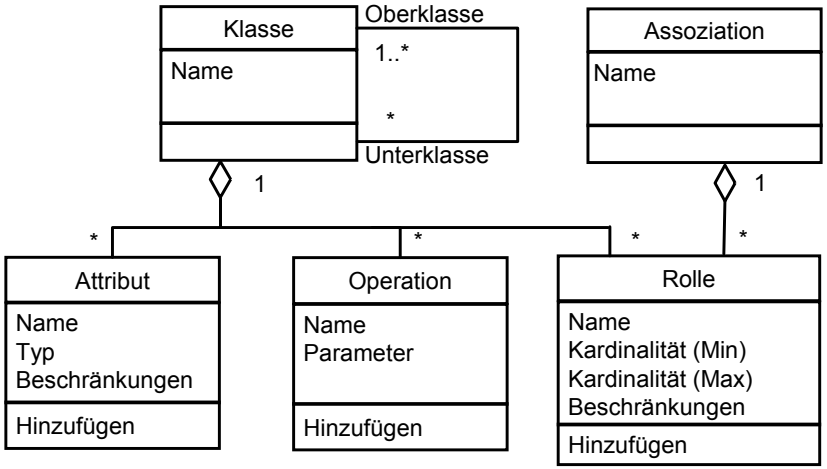
Exemplar-Ebene



Typ-Ebene



Meta-Ebene



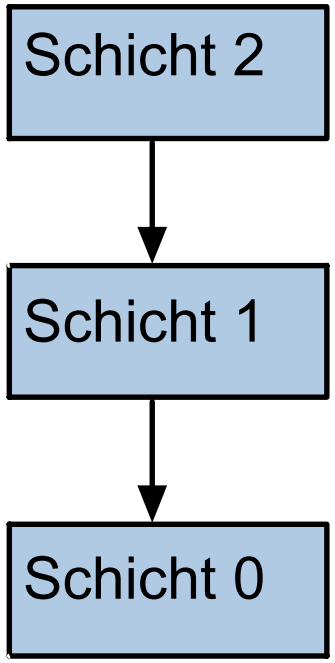
Prinzip der Abstraktion

- Neue Konzepte
  - Meist auf der Typ-Ebene anzusiedeln
  - Abstrahieren daher »stärker« als Konzepte auf der Exemplar-Ebene
- Zur Notation
  - Für »Abstraktionen« gibt es in der Regel keine eigenständige Notation
  - Das verwendete Konzept, z.B. Klassen, die damit verbundene(n) Notation(en), z.B. UML-Notation, und der Einsatz in der jeweiligen Entwicklungsphase geben indirekt das Abstraktionsniveau an

© Prof. Dr. Liggesmeyer, 30

Prinzip der Abstraktion

- Zur Darstellung von Abstraktions-ebenen bzw. -schichten wird oft pro Ebene ein Rechteck verwendet
- Die Rechtecke sind getrennt durch Pfeile übereinander angeordnet, wobei das am höchsten oben angeordnete Rechteck die »abstrakteste« Schicht darstellt
- Es ergibt sich eine lineare Ordnung



Prinzip der Abstraktion

- Vorteile der Abstraktion
  - + Erkennen, Ordnen, Klassifizieren, Gewichten von wesentlichen Merkmalen
  - + Erkennen allgemeiner Charakteristika
    - - Bildet Voraussetzung für Allgemeingültigkeit
  - + Trennen des Wesentlichen vom Unwesentlichen

© Prof. Dr. Liggesmeyer, 32

Prinzip der Strukturierung

- Struktur
  - Gibt die Anordnung der Teile eines Ganzen zueinander an
  - Ein Gefüge, das aus Teilen besteht, die wechselseitig voneinander abhängen
  - Ist ein (durch Relationen beschreibbares) Beziehungsgefüge und dessen Eigenschaften
  - Ist ein nach Regeln aus Elementen zu einer komplexen Ganzheit aufgebautes Ordnungsgefüge
  - Die reduzierte Darstellung des Systems, die den Charakter des Ganzen offenbart; losgelöst vom untergeordneten Detail beinhaltet sie die wesentlichen Merkmale des Ganzen

© Prof. Dr. Liggesmeyer, 33

Prinzip der Strukturierung

- Notation
  - Strukturen können in allgemeiner Form durch Graphen dargestellt werden
  - Die Semantik der Relation zwischen den Systemkomponenten bestimmt die Form der Struktur bzw. des Graphen
- Software-Technik
  - Strukturierung hat sowohl für das fertige Software-Produkt als auch für den Entwicklungs- und Qualitätssicherungsprozess eine große Bedeutung
  - Produkten und Prozessen soll eine geeignete Struktur aufgeprägt werden

© Prof. Dr. Liggesmeyer, 34

Prinzip der Strukturierung

Konzepte der Software-Technik und ihre Strukturen

Konzept	Semantik der Relation	Anzahl unterschiedlicher Systemkomponenten-Typen	Form der Struktur
Datenfluß-Diagramm	Daten fließen von A nach B	4 (DFD in SA)	gerichteter Graph
Entity-Relationship-Modell	Zwischen A und B	2	ungerichteter Graph
Programmablaufplan (PAP)	Auf A folgt zeitlich B	5	gerichteter Graph (irreflexiv, transitiv)
Zustandsautomat	Von Zustand A wird in Zustand B übergegangen	1	gerichteter Graph
Petri-Netz	Transition	2	gerichteter Graph
Netzplan (Kapitel II 2.3)	Auf A folgt zeitlich B	1	gerichteter, azyklischer Graph
Konfiguration (Kapitel II 6.3)	Auf A folgt zeitlich B	1	gerichteter, azyklischer Graph
Entwicklungs-Prozeß-Modell (Kapitel II 3.3)	Tätigkeit A erzeugt Teilprodukt B, Teilprodukt B wird verwendet von Tätigkeit C	2	gerichteter Graph

© Prof. Dr. Liggesmeyer, 35

Prinzip der Strukturierung

- Klassifikation nach 3 Zeitspannen, in der sie existieren
  - Statische Struktur
    - Dokumentationsstruktur
  - Dynamische Struktur
    - Laufzeitstruktur
  - Organisatorische Struktur
    - Entwicklungsstruktur

© Prof. Dr. Liggesmeyer, 36

Prinzip der Strukturierung

- Statische Struktur
  - Liegt vor, wenn eine Struktur ab einem Zeitpunkt vollständig vorliegt und erhalten bleibt
  - Dokumentiert einen Sachverhalt
- Dynamische Struktur
  - Wenn während der Laufzeit eines Systems eine Struktur entsteht
  - Beispiel
    - Durch rekursive Aufrufe entsteht während der Laufzeit eine dynamische Schachtelungsstruktur

© Prof. Dr. Liggesmeyer, 37

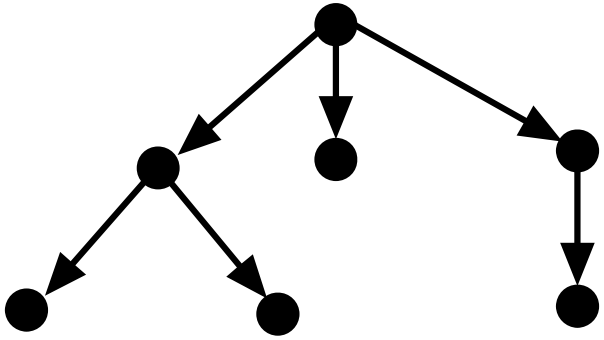
Prinzip der Strukturierung

- Organisatorische Struktur
- Entsteht während einer Software-Entwicklung und existiert oft bis zur Fertigstellung des Produkts, in manchen Fällen während des gesamten Lebenszyklus des Produkts
  - Kennzeichnend ist, dass sie nicht »auf einen Schlag« da ist, sondern mit Fortschreiten der Entwicklung »wächst«
  - Beispiel
    - Eine Konfigurationsstruktur Produkts entsteht während der Software-Entwicklung und setzt sich fort in der Wartungs- und Pflegephase
    - Die entstehende Struktur kann nicht von vornherein vollständig festgelegt werden

© Prof. Dr. Liggesmeyer, 38

Prinzip der Strukturierung

- Beispiel
  - Projektorganisation, die sich an der Subsystemstruktur orientiert
  - Die Relation lautet: v_i delegiert Aufgaben an v_j
  - Die Struktur kann nicht von vornherein festgelegt werden, da sich erst während der Entwicklung die Anzahl der Subsysteme ergibt, die wiederum die Anzahl der Teams determiniert



© Prof. Dr. Liggesmeyer, 39

Prinzip der Strukturierung

- Vorteile der Strukturierung
  - + Erhöhung der Verständlichkeit
  - + Verbesserung der Wartbarkeit
  - + Erleichterung der Einarbeitung in ein fremdes Software-Produkt
  - + Beherrschbarkeit der Komplexität eines Systems

© Prof. Dr. Liggesmeyer, 40

Prinzip der Hierarchisierung

- Hierarchie
  - Bezeichnet eine Rangordnung, eine Abstufung sowie eine Über- und Unterordnung
  - Ein System besitzt eine Hierarchie, wenn seine Elemente nach einer Rangordnung angeordnet sind
  - Elemente gleicher Rangordnung stehen auf derselben Stufe, sie bilden eine Ebene bzw. Schicht der Hierarchie
  - Viele in der Natur vorkommende komplexe Systeme verfügen über eine hierarchische Struktur

© Prof. Dr. Liggesmeyer, 41

Prinzip der Hierarchisierung

- Hierarchie vs. Struktur
- Prinzip der Hierarchisierung lässt sich dem Prinzip der Strukturierung unterordnen
  - Eine Hierarchie lässt sich nach denselben Kategorien klassifizieren wie eine Struktur
  - Die Semantik der Relation muss jedoch immer eine Rangfolge beinhalten
  - Hierarchische Systeme besitzen in der Software-Technik eine große Bedeutung
- Zur Notation
  - Analog zu Strukturen dargestellt
  - Kanten müssen immer gerichtet sein
  - Übergeordnete Elemente über den untergeordneten Elementen zeichnen

Prinzip der Hierarchisierung

Konzepte/Methoden und ihre Hierarchien

Konzept/Methode	Semantik der Relation	Anz unterschiedl. Systemkom-ponenten-Typen	Hierarchieform
Funktions-Baum	A besteht aus B, A ruft B auf	1	gerichteter Baum
Jackson-Diagramm	A besteht aus B	3	gerichteter Baum
Klassen-Diagramm	A vererbt an B	2	azyklisches Netz
Entscheidungsbaum	A wird vor B entschieden	1	gerichteter Baum
Strukturierte Analyse (SA)			
- Datenfluß-Diagramm	Daten fließen von A nach B	4	Gericht. Graph
- DFD-Hierarchie	Funktion A wird durch Diagramm B verfeinert	2	gerichteter Baum
Objektorientierte Analyse (OOA), Entwurf (OOD)			
- Assoziation	Zwischen A und B besteht eine Assoziation	2	Ungericht. Graph
- Aggregation	A besteht aus B, C ...	2	gerichtetes Netz
- Vererbung	A vererbt an B	2	azyklisches Netz
- Subsysteme	A faßt B, C ... zusammen	2	azyklisches Netz

Prinzip der Hierarchisierung

- Konzepte/Methoden und ihre Hierarchien

Konzept/Methode	Semantik der Relation	Anz unterschiedl. Systemkom-ponenten-Typen	Hierarchieform
Strukturierter Entwurf (SD)	A ruft B auf	1	azyklisches Netz
Modularer Entwurf (MD)			
- Aufruf	A ruft B auf	2 (3)	azyklisches Netz
- Import, Benutzbarkeit	A importiert B	2 (3)	azyklisches Netz
- Enthaltensein	A enthält B	2 (3)	gerichteter Baum
Aufbauorganisation			
- funktions-/ marktorientiert	A leitet B	1	gerichteter Baum
- Matrix	C untersteht A und B	1	azyklisches Netz

© Prof. Dr. Liggesmeyer, 44

Prinzip der Hierarchisierung

- Vorteile der Hierarchisierung
  - Wie bei der Strukturierung
  - Eine Hierarchie schränkt jedoch stärker ein als eine Struktur
    - + Beschränkung auf definierte, gerichtete Strukturen
    - + Verhinderung chaotischer Strukturen

Prinzip der Modularisierung

- Modularisierung in den Ingenieurdisziplinen
  - Ein Computersystem oder Fernsehgerät wird aus Modulen aufgebaut, wobei jedes Modul eine weitgehend abgeschlossene Bau- oder Funktionsgruppe darstellt
  - Im Fehlerfall wird das komplette Modul gegen ein fehlerfreies ausgetauscht
  - Das Modul muss eine festgelegte Schnittstelle zu den anderen Modulen des Gerätes besitzen
  - Alle Informationen müssen über diese Schnittstelle laufen

© Prof. Dr. Liggesmeyer, 46

Prinzip der Modularisierung

- Beispiel aus der Software-Technik
  - Dokumente – insbesondere Referenz- und Benutzerhandbücher – sollen modular aufgebaut sein
  - Wird z.B. eine Funktion in einem Software-Produkt geändert, dann soll es möglich sein, die zugehörige Funktionsbeschreibung im Referenzhandbuch gegen die neue auszutauschen
  - Solche partiellen Änderungen dürfen jedoch nicht dazu führen, dass das gesamte Referenzhandbuch gegen ein neues ersetzt werden muss
  - Beim partiellen Austauschen von Teilen müssen die Seitennumerierung und die Bezüge konsistent bleiben

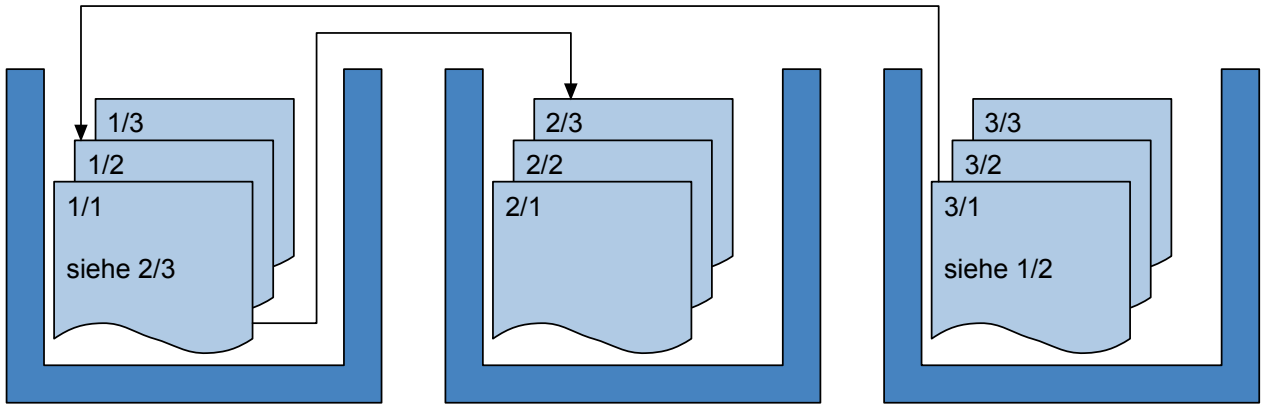
Prinzip der Modularisierung

- Eigenschaften von modularen Dokumenten
  - Kapitel- oder abschnittsweise Seitenzählung
    - Beispiel: Kapitel 2/Seite 5
  - Seitenzählung basiert auf dem Kapitelanfang
    - Bei Erweiterung eines Kapitels muss nicht das ganze Dokument neu numeriert werden
    - Alle Abbildungsnummern werden jeweils kapitelweise durchgezählt
  - Bezüge auf andere Kapitel möglichst gering
  - Geringer Kapitelumfang
  - Wünschenswert: Referenzverzeichnis am Ende jedes Kapitels, damit sichtbar welche Bezüge auf andere Kapitel vorkommen
  - Bei Referenzhandbüchern oft sogar seitenweise Modularität

© Prof. Dr. Liggesmeyer, 48

Prinzip der Modularisierung

- Dokumentenmodularität



- Beispiel
  - Der Kern von SA besteht aus einer Baumhierarchie von Datenflussdiagrammen
  - Jedes Datenflussdiagramm soll maximal eine DIN A4-Seite umfassen
  - Bezüge zwischen Datenflussdiagrammen werden über Einträge in das *Data Dictionary* hergestellt

ENGINEERING SOFTWARE DEPENDABILITY
© Prof. Dr. Liggesmeyer, 49

Prinzip der Modularisierung

- Modularisierung
  - Konzeption von Modulen beim Entwicklungsprozess
- Modul i.w.S.
  - Darstellung einer funktionalen Einheit oder einer zusammengehörenden Funktionsgruppe
  - Weitgehende Kontextunabhängigkeit, d.h. ein Modul ist in sich abgeschlossen
    - Von Modulumgebung weitgehend unabhängig entwickelbar, prüfbar, wartbar und verständlich
  - Definierte Schnittstelle für Externbezüge
    - Klar erkennbar und in einer Schnittstellenbeschreibung zusammengefasst
  - Im qualitativen und quantitativen Umfang handlich, überschaubar und verständlich

© Prof. Dr. Liggesmeyer, 50

Prinzip der Modularisierung

- Beispiel
  - Benutzerhandbuch
    - Benutzungsoberfläche und Druckausgaben in einem Kapitel zusammengefasst
    - Bei Änderung der Druckausgaben muss auch die Beschreibung der Benutzungsoberfläche ausgetauscht werden
  - Benutzungsoberfläche und Druckausgaben stehen in keinem so engen semantischen Zusammenhang, dass sie sinnvoll in einem Dokument-Modul zusammengefasst werden sollten

Prinzip der Modularisierung

- Modularität im Großen (Subsystemebene)
  - Hierarchische Zustandsautomaten
    - Harel-Automat
  - Hierarchische Petri-Netze
  - Hierarchische Netzpläne
  - Subsysteme in der objektorientierten Analyse und im objektorientierten Entwurf
  - Hierarchie von Datenflussdiagrammen in der strukturierten Analyse

© Prof. Dr. Liggesmeyer, 52

Prinzip der Modularisierung

- Modularität im Kleinen (Komponentenebene)
  - Klassen
  - Datenabstraktion
  - Prozeduren/Funktionen
  - Entscheidungstabellen-Verbunde
- Konzepte und Methoden
  - Ermöglichen eine Modularisierung, erzwingen sie in der Regel nicht
- Modularisierung eines Unternehmens
  - Auch bei der Gestaltung der Aufbauorganisation eines Unternehmens
  - »modulare« Aufbauorganisation

© Prof. Dr. Liggesmeyer, 53

Prinzip der Modularisierung

- Zusammenhang mit Abstraktion
  - Modularität im Großen ist eng verknüpft mit dem Prinzip der Abstraktion, da die Modularisierung gleichzeitig das Bilden von Abstraktionsebenen erfordert
- Zur Notation
  - Zur Kennzeichnung eines Moduls gibt es keine einheitliche Notation
  - Sie ergibt sich implizit durch die jeweils verwendete Notation für Systemkomponenten und Subsysteme

© Prof. Dr. Liggesmeyer, 54

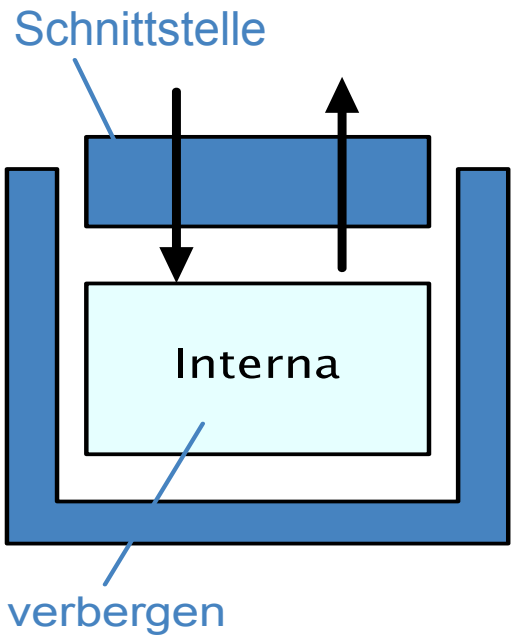
Prinzip der Modularisierung

- Bewertung
  - \+ Hohe Änderungsfreundlichkeit
    - \- Leichter Austausch und leichte Erweiterbarkeit
  - \+ Verbesserung der Wartbarkeit
    - \- Leichte Lokalisierung, Kontextunabhängigkeit
  - \+ Erleichterung der Standardisierung
  - \+ Erleichterung der Arbeitsorganisation und Arbeitsplanung
  - \+ Verbesserung der Überprüfbarkeit

© Prof. Dr. Liggesmeyer, 55

Geheimnisprinzip

- Geheimnisprinzip *(information hiding)*
  - »Verschärfung« des Prinzips der Modularisierung
  - Für den Anwender einer Systemkomponente sind die Interna der Systemkomponente verborgen
  - Sinnvoll nur in Verbindung mit der Modularisierung
    - Bei einem Modul ist nur die definierte Schnittstelle von außen sichtbar
  - Andere Sicht
    - Überflüssige Angaben, die zur Erledigung einer Aufgabe nicht benötigt werden, dürfen auch nicht sichtbar sein



Geheimnisprinzip

- Verkapselung
  - Im Zusammenhang mit abstrakten Datenobjekten, abstrakten Datentypen und Klassen entstanden die Begriffe
    - Verkapselung *(encapsulation)*
    - Einkapselung bzw. Datenkapsel
  - Zusammengehörende Attribute bzw. Daten und Operationen in einer Einheit zusammenfassen
  - Spezielle Form eines Moduls
  - Einhaltung des Geheimnisprinzips
    - Attribute und die Realisierung der Operationen außerhalb der Verkapselung nicht sichtbar
  - Das Geheimnisprinzip »verschärft« also eine Verkapselung

© Prof. Dr. Liggesmeyer, 57

Geheimnisprinzip

- Liberale Ausprägung
  - Alle Interna einer Systemkomponente sind für den Anwender physisch sichtbar
  - Aber: CASE-Systeme oder Compiler melden unerlaubte Zugriffe auf Interna
  - Der Anwender kann die Interna sehen und dadurch vielleicht dieses Wissen implizit bei der Anwendung dieser Systemkomponente mitverwenden
  - Dieses Wissen beeinflusst die Änderungsfreundlichkeit einer Systemkomponente
  - Sollen die Interna einer Systemkomponente geändert werden, ohne dass die Schnittstelle davon betroffen ist, dann darf der Anwender davon nichts merken

© Prof. Dr. Liggesmeyer, 58

Geheimnisprinzip

- Strenge Ausprägung
  - Außer der definierten Schnittstelle sind sämtliche Interna für den Anwender völlig unsichtbar
    - Beide Teile sind auch textuell völlig getrennt
  - Spezifikationen einer Systemkomponente muss exakt und vollständig das Verhalten beschreiben
    - Das kann sehr schwierig sein
  - In der Programmiersprache JAVA ist die physische Trennung von Spezifikation und Implementierung nicht mehr vorhanden

© Prof. Dr. Liggesmeyer, 59

Geheimnisprinzip

- Ausprägungen des Geheimnisprinzips in der OO-Welt
  - In einer Klasse kann auf die Attribute nur über die Operationen zugegriffen werden
  - Systemanalytiker konzipiert eine Klasse einschl. ihrer Attribute und sieht sie natürlich auch
  - Vererbungskonzept ermöglicht es, dass Unterklassen die Attribute der Oberklassen sehen
  - Geheimnisprinzip ist verletzt, wenn ein Objekt einer Unterklasse direkt auf die Attribute eines Objekts der Oberklasse zugreift
  - Abhängig von der Programmiersprache, kann das Geheimnisprinzip »durchlöchert« werden

© Prof. Dr. Liggesmeyer, 60

Geheimnisprinzip

- C++
  - Streng, durch das *private*-Konzept,
  - Liberal, durch das *protected*-Konzept
    - Innerhalb der Vererbungshierarchie kann auf die Attribute der Oberklassen direkt zugegriffen werden
  - Überhaupt nicht, durch das *public*-Konzept
    - Auf die Attribute kann frei zugegriffen werden
  - Beschränkt, durch das *friend*-Konzept
    - Für bestimmte Klassen oder Operationen ist der Attributzugriff möglich

© Prof. Dr. Liggesmeyer, 61

Geheimnisprinzip

- Konflikte bei ODBS
  - Daten aus objektorientierten Datenbanken, will man häufig mittels Prädikaten über Teile ihres Wertes selektieren
    - Beispiel: » ... where Umsatz > 5000«
  - ODBS erfordern eine differenzierte Anwendung des Geheimnisprinzips
  - Strikte Einhaltung für verändernde Zugriffe
  - Lesende Zugriffe sollten direkt möglich sein
  - Teil des Objektwertes streng kapseln und den eigentlichen Zustand repräsentieren
  - Ein anderer Teil kann zur Darstellung frei zugänglicher Eigenschaften verwendet werden

© Prof. Dr. Liggesmeyer, 62

Geheimnisprinzip

- Bewertung
  - + Die Anwendung einer Systemkomponente oder eines Subsystems wird zuverlässiger, da nur über die definierte Schnittstelle kommuniziert werden kann
  - + Der Anwender einer Systemkomponente oder eines Subsystems wird nicht mit unnötigen Informationen belastet
  - + Die Datenkonsistenz interner Daten kann besser sichergestellt werden, da direkte, unkontrollierbare Manipulationen nicht möglich sind

© Prof. Dr. Liggesmeyer, 63

Geheimnisprinzip

- Die Anwendungsschnittstelle muss vollständig und exakt beschrieben werden
- In einigen Fällen kann das Geheimnisprinzip auch »hinderlich« sein, z.B. beim deklarativen Zugriff auf gespeicherte Daten

- Generell sollte so viel wie möglich vom Geheimnisprinzip Gebrauch gemacht werden
- Eine »Aufweichung« des Geheimnisprinzips sollte auf Sonderfälle beschränkt bleiben
- Geheimnisprinzip immer im Zusammenhang mit dem Prinzip der Modularisierung verwenden

© Prof. Dr. Liggesmeyer, 64

Prinzip der Lokalität

- Lokalität
  - Zum Verstehen komplexer Probleme ist es notwendig, sich zu einem Zeitpunkt nur mit einer kleinen Anzahl von Eigenschaften zu beschäftigen
  - Diese sollen wesentlich für den gegenwärtigen Gesichtspunkt sein
  - Alle relevanten Informationen für wichtige Gesichtspunkte sollen lokal, d.h. an einem Platz, zur Verfügung stehen
  - Durch gute Lokalität kann das Zusammensuchen benötigter Informationen vermieden werden

© Prof. Dr. Liggesmeyer, 65

Prinzip der Lokalität

- Optimale Lokalität liegt vor
  - wenn zur Lösung eines Problems alle benötigten Informationen auf einer Seite zu finden sind
  - wenn andererseits nicht benötigte Informationen nicht vorhanden sind
- Viele Konzepte, Methoden und Programmiersprachen unterstützen von vornherein eine gute Lokalität, andere erschweren Lokalität

© Prof. Dr. Liggesmeyer, 66

Prinzip der Lokalität

- Beispiele
  - Entscheidungstabellen (ET) sind sehr kompakt und besitzen eine optimale Lokalität
    - Durch ET-Verbunde kann das Prinzip der Lokalität auch bei umfangreichen Tabellen eingehalten werden
  - Hierarchische Zustandsautomaten nach Harel erlauben pro Hierarchieebene eine lokale Sicht auf zusammengehörende Details
  - Hierarchische Petri-Netze liefern pro Hierarchieebene alle Informationen, die für bestimmte Gesichtspunkte nötig sind

© Prof. Dr. Liggesmeyer, 67

Prinzip der Lokalität

- Strukturierte Analyse (SA)
  - Erlaubt durch hierarchisch angeordnete Datenfluss-diagramme (DFD) eine lokale Zusammenfassung von Informationen
  - Aber: Zusammenhang zwischen den Datenflüssen benachbarter DFDs sowie der Aufbau des Speichers müssen jedoch aus dem *Data Dictionary* »herausgesucht« werden
- SA/RT *(structural analysis/real time analysis)*
  - Unterstützt eine lokale Anordnung der Prozesssteuerung durch CSpec-Seiten
  - Bezug zwischen Flussdiagrammen und CSpec-Seiten ist auf den Flussdiagrammen durch eine Balkennotation angegeben

© Prof. Dr. Liggesmeyer, 68

Prinzip der Lokalität

- Objektorientierte Analyse (OOA)
  - Durch Subsystembildung Zusammenfassung von Klassen zu einer Einheit
  - Dadurch wird das Lesen und Verstehen eines Klassen-Diagramms erleichtert
- Modularer Entwurf
  - Die wesentlichen Informationen zum Verständnis eines Moduls sind lokal angeordnet
  - Aber: Importbeziehungen oft nicht explizit
- Objektorientierte Welt
  - Verständnis einer Klasse erschwert, wenn die Klasse in einer Vererbungsbeziehung
  - Informationen entlang der Vererbungsbeziehungen zusammensuchen

© Prof. Dr. Liggesmeyer, 69

Prinzip der Lokalität

- Ada
  - Paketspezifikationen und Paketimplementation jeweils lokal zusammenhängend beschrieben
- C++
  - Schachtelung von Funktionen und Prozeduren nicht möglich
  - Dadurch sind auch lokale Hilfsfunktionen und -prozeduren global angeordnet und sichtbar
  - Überblick und Einarbeitung erschwert, da nicht benötigte Informationen sichtbar sind

Prinzip der Lokalität

- Pascal
  - Anfang eines Programms
    - Alle globalen Typ- und Datenvereinbarungen
  - Mitte: Alle Prozedurvereinbarungen
  - Ende
    - Anweisungen des Hauptprogramms
  - Verstehen eines Programms
    - Anweisungsteil am Ende des Listings und den Vereinbarungsteil am Anfang des Listings nebeneinanderlegen
- Modula-2
  - In der Schnittstellenbeschreibung von Modulen wird nur der Name von Funktionen angegeben, aber nicht die Parameterliste der Funktionen
  - Funktionsvereinbarung muss gesucht werden

© Prof. Dr. Liggesmeyer, 71

Prinzip der Lokalität

- These: Im Bereich der Implementierung
  - Eine Gruppe von ungefähr 30 Anweisungen ist die oberste Grenze, was beim ersten Lesen eines Listings einer Systemkomponente, eines Moduls oder einer Prozedur, Funktion bzw. Operation bewältigt werden kann
- Notation
  - Zur Kennzeichnung der Lokalität gibt es keine eigene Notation
  - Konzepte, Methoden und Programmiersprachen unterstützen meist implizit das Prinzip der Lokalität

© Prof. Dr. Liggesmeyer, 72

Prinzip der Lokalität

- Bewertung
  - \+ Ermöglicht die schnelle Einarbeitung
  - \+ Fördert die Verständlichkeit und Lesbarkeit
  - \+ Erleichtert die Wartung und Pflege
  - – Erschwert das Geheimnisprinzip

© Prof. Dr. Liggesmeyer, 73

Prinzip der Verbalisierung

- Verbalisierung
  - Gedanken und Vorstellungen in Worten ausdrücken und damit ins Bewusstsein zu bringen
  - Wird schon lange für die Programmierung propagiert
  - Es hat heute jedoch eine wesentlich umfassendere Bedeutung
  - Insbesondere in den frühen Phasen der Software-Entwicklung kommt ihr eine herausragende Bedeutung zu
  - Die in der Definitionsphase gewählten Begriffe, Klassifizierungen und Namen beeinflussen alle weiteren Phasen in ihrer Begrifflichkeit

© Prof. Dr. Liggesmeyer, 74

Prinzip der Verbalisierung

- Viele Konzepte und Methoden, die in der Definitionsphase eingesetzt werden, haben explizite Vorschriften und Regeln für die Namensgebung
- Die Überprüfung der Einhaltung ist Aufgabe der Qualitätssicherung

☐ Gute Verbalisierung durch

- aussagekräftige, mnemonische Namensgebung
- geeignete Kommentare
- selbstdokumentierende Konzepte, Methoden und Sprachen

© Prof. Dr. Liggesmeyer, 75

Prinzip der Verbalisierung

- Zitat /Wendt 93, S. 36/
  - »Da der Computer keine Anschauungssemantik kennt, kann er auch nicht dazu benutzt werden sicherzustellen, dass bei der Wahl von Namen für Module, ... zweckmäßige anschauungssemantische Bezüge hergestellt werden
  - Je komplexer die Systeme werden, um so undurchschaubarer wird der Namenswirrwarr, wenn den Entwicklern die Namenswahl freigestellt bleibt, d.h. wenn die Sicherstellung anschauungssemantischer Bezüge nicht bewusst als Engineering-Aufgabe wahrgenommen wird
  - Man bedenke, dass in der oben erwähnten Software in einer Million Zeilen C-Quellcode insgesamt rund 29 000 Namen vereinbart werden mussten«

© Prof. Dr. Liggesmeyer, 76

Beispiele für Verbalisierungsregeln

- OOA (objektorientierte Analyse)
  - Der Klassenname soll
    - ein Substantiv im Singular sein
    - so konkret wie möglich gewählt werden
    - dasselbe ausdrücken wie die Gesamtheit der Attribute und/oder Operationen
    - nicht die Rolle beschreiben, die diese Klasse in einer Beziehung zu einer anderen Klasse spielt
  - Ein Attributname soll
    - eindeutig & verständlich im Klassenkontext sein
    - den Namen der Klasse nicht wiederholen (Ausnahmen sind feststehende Begriffe)
    - bei strukturierten Attributen der Gesamtheit der Komponenten entsprechen

© Prof. Dr. Liggesmeyer, 77

Beispiele für Verbalisierungsregeln

- Der Name einer Operation ist so zu wählen, dass er
  - ein Verb enthält
  - dasselbe aussagt wie die Spezifikation der Operation
  - den Klassennamen nicht wiederholt (Ausnahme: feststehende Begriffe)
  - die funktionale Bindung der Operation bestätigt

Das Verb kann bei reinen Abfrage-Operationen durch ein »?« ersetzt werden

© Prof. Dr. Liggesmeyer, 78

Beispiele für Verbalisierungsregeln

- SA (strukturierte Analyse)
  - Ein Datenflussname
    - besteht aus einem Substantiv oder einem Adjektiv und einem Substantiv
    - enthält niemals ein Verb
    - ist so zu wählen, dass er nicht nur die Daten, die fließen, beschreibt, sondern etwas darüber aussagt, was über die Daten bekannt ist, z.B. geprüfte Kundennummer

  Seichte Namen wie Daten, Informationen sind zu vermeiden

Beispiele für Verbalisierungsregeln

- Ein Funktions- bzw. Prozessname
  - besteht aus einem einzigen starken Aktionsverb gefolgt von einem einzigen konkreten Objekt (z.B. erstelle Adreßaufkleber) oder einem konkreten Substantiv gefolgt von einem starken Aktionsverb (z.B. Adreßaufkleber erstellen)
  - repräsentiert die Aktion

  Seichte Namen wie verarbeite, bediene sind zu vermeiden

Beispiele für Verbalisierungsregeln

- Unternehmensmodellierung
  - Als Name eines Geschäftsprozesses sollte
    - die Gerundiumform eines Verbs, z.B. Verkaufen, oder
    - ein Substantiv gefolgt von einem Verb, z.B. Verkauf durchführen, oder
    - der Anfangs- und Endpunkt des Prozesses, z.B. Interessent bis Auftrag,

    gewählt werden

Beispiele für Verbalisierungsregeln

- Benutzungsoberfläche
  - Die Aufgaben müssen mit den Fachbegriffen beschrieben werden, die der Benutzer kennt
  - Die für seine fachliche Arbeitstätigkeit relevanten Aufgabenbereiche und die dafür im Software-System vorgesehenen Anwendungen muss der Benutzer ohne Schwierigkeiten identifizieren können

© Prof. Dr. Liggesmeyer, 82

Prinzip der Verbalisierung

- Wichtig ist
  - dass die Werkzeuge die Konzepte und Methoden unterstützen
  - dass die Compiler für die Programmiersprachen sowohl lange Namen als auch die geeignete Strukturierung von Namen erlauben
  - der Grad der Selbstdokumentation einer Programmiersprache, eines Konzepts oder einer Methode
    - Bei den Programmiersprachen sind besonders die Schlüsselwörter und die Möglichkeiten der Programmstrukturierung wichtig
    - Die Programmiersprache Ada gilt hierbei als vorbildlich, C++ als das Gegenteil

© Prof. Dr. Liggesmeyer, 83

Prinzip der Verbalisierung

- Beispiel
  - Darstellung einer Klasse mit
    - Klassennamen
    - Attributnamen
    - Operationsnamen
    - Wahlweise können die beiden letzten ausgeblendet werden
    - Einige Notationen: Nur Klassennamen
    - Einige Werkzeuge: Bei der Druckausgabe nur Klassennamen ausgegeben, obwohl Attribute und Operationen eingegeben wurden
    - Einige Werkzeuge: Schränken die Länge der Namen ein oder erzwingen eine bestimmte Syntax für die Namen

© Prof. Dr. Liggesmeyer, 84

Prinzip der Verbalisierung

- Bewertung
  - + Leichte Einarbeitung in fremde Modelle, Architekturen, Programme bzw. Wiedereinarbeitung in eigene Dokumente
  - + Erleichterung der Qualitätssicherung, der Wartung und Pflege
  - + Verbesserte Lesbarkeit der erstellten Dokumente

© Prof. Dr. Liggesmeyer, 85

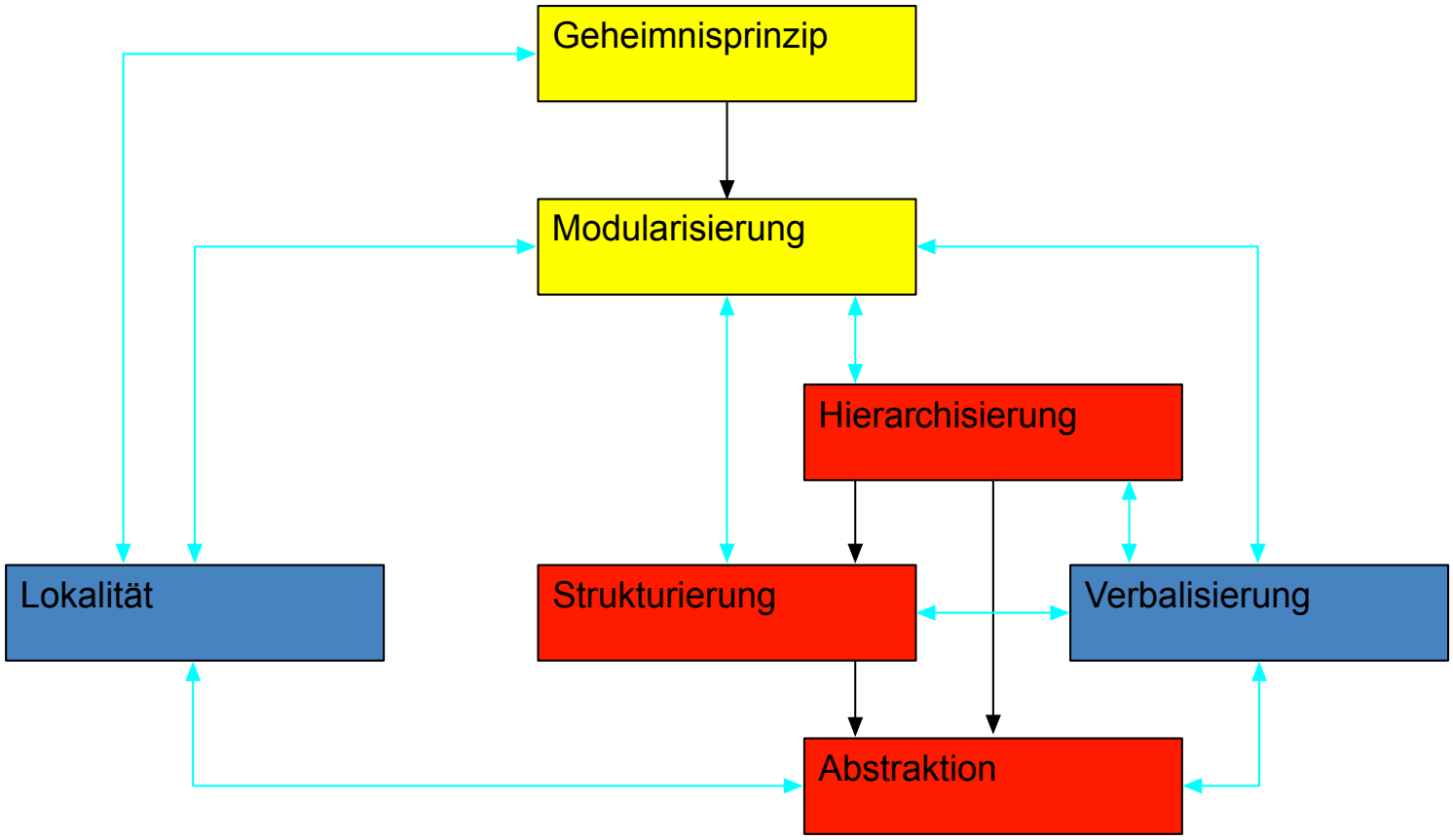
Abhängigkeiten zwischen den Prinzipien

- Prinzipien
  - Sind wechselseitig miteinander verwoben und setzen sich zum Teil gegenseitig voraus
  - Außerdem lässt sich keine kausale Kette herleiten, die besagt, in welcher Reihenfolge die Prinzipien anzuwenden sind

© Prof. Dr. Liggesmeyer, 86

Abhängigkeiten zwischen den Prinzipien

Abhängigkeiten zwischen den Prinzipien



Abhängigkeiten zwischen den Prinzipien

- Wechselbeziehungen
  - Bevor ein System strukturiert werden kann, muss der Abstraktionsprozess abgeschlossen sein
  - Eine Hierarchiebildung prägt einer Struktur eine Rangordnung auf
    - Daher sind Strukturierung und Abstraktion Voraussetzungen für die Hierarchisierung
  - Eine Modulbildung setzt entweder eine Strukturierung oder eine Hierarchisierung voraus
    - Es kann auch zunächst eine Modularisierung erfolgen, die dann zu einer impliziten oder expliziten Strukturierung oder Hierarchisierung führt

© Prof. Dr. Liggesmeyer, 88

Abhängigkeiten zwischen den Prinzipien

- Das Geheimnisprinzip ist eine »Verschärfung« des Modularisierungsprinzips
  - Daher ist die Modularisierung die Voraussetzung für das Geheimnisprinzip
- Das Lokalitätsprinzip steht in Wechselwirkung mit der Modularisierung und der Abstraktion
  - Die Abstraktion bestimmt die notwendigen Informationen
  - Die Modulbildung lässt diese lokal anordnen
  - Durch das Geheimnisprinzip kann dann gesteuert werden, dass nicht relevante Informationen auch nicht sichtbar sind

© Prof. Dr. Liggesmeyer, 89

Abhängigkeiten zwischen den Prinzipien

- Das Prinzip der Verbalisierung hat die meisten Wechselwirkungen zu anderen Prinzipien
  - Für Abstraktionsebenen, Strukturen, Hierarchien und Module müssen geeignete Namen gewählt werden
  - Bei der Namensgebung stellt man umgekehrt wieder fest, ob die Strukturen, Hierarchien, Abstraktionen und Module richtig gewählt wurden

© Prof. Dr. Liggesmeyer, 90

Allgemeine Methoden

- Methoden
  - Sind planmäßig angewandte, begründete Vorgehensweisen zur Erreichung von festgelegten Zielen
    - Im Allgemeinen im Rahmen festgelegter Prinzipien
  - 2 allgemeine, d.h. fachunabhängige, Methoden:
    - *Top-down*-Methode
    - *Bottom-up*-Methode
  - Beide orientieren sich vorwiegend an dem Begriffspaar »abstrakt / konkret«
  - Sie stehen in enger Beziehung zu dem Prinzip der Abstraktion

© Prof. Dr. Liggesmeyer, 91

Allgemeine Methoden

- *Top-down*-Methode
  - Vom Abstrakten zum Konkreten
  - Vom Allgemeinen zum Speziellen
- *Bottom-up*-Methode
  - Vom Konkreten zum Abstrakten
  - Vom Speziellen zum Allgemeinen
- Analog: Deduktion und Induktion in der Didaktik
  - Deduktive Vorgehensweise
    - Aus Allgemeinen wird der Einzelfall hergeleitet
  - Induktive Vorgehensweise
    - Vom Speziellen wird zum Allgemeinen hingeführt
    - Von mehreren Beispielen kann auf eine allgemeine Regel geschlossen werden

© Prof. Dr. Liggesmeyer, 92

Allgemeine Methoden

☐ Beispiele für *top-down*- & *bottom-up*-Methoden

Methode	Art	»top«	»bottom«
Software-Entwicklung			
■ Modellierung von Daten	*top-down*	abstrakte, strukturierte Daten	Datenelemente oder selbst definierendes Datum
☐ *Data-Dictionary* (I 2.8) ☐ Syntax-Diagramm (I 2.8) ☐ Jackson-Diagramm (I 2.9)			
■ Modellierung von Funktionen ☐ Funktionsbaum (I 2.6)	*top-down*	abstrakte, strukturierte Funktionen	Elementare Funktionen
■ Modellierung von Zuständen ☐ Hierarchische Zustandsutomaten (I 2.16.5)	*top-down*	Oberzustand	Unterzustand
■ Modellierung eines Petri-Netzes ☐ Hierarchische Petri-Netze (I 2.17.5)	*top-down*	Kanäle und Instanzen	Stellen und Transitionen
■ OOA-Modellierung (I 2.18)	*bottom-up*	Subsysteme	Klassen und ihre Beziehungen
■ SA-Modellierung (I 2.19)	*top-down*	Kontext-Diagramm	MiniSpecs

© Prof. Dr. Liggesmeyer, 93

Allgemeine Methoden

Beispiele für *top-down*- & *bottom-up*-Methoden

Methode	Art	»top«	»bottom«
■ SA/RT-Modellierung (I 2.20)	*top-down*	Kontext-Diagramm	Pspecs, Cspecs
■ Identifizieren von Klassen (I 2.18.6)	*top-down/ bottom-up*	Klassen	Attribute, Operationen
■ Modularer Entwurf (I 3.9)	*top-down/ bottom-up*	systemferne, benutzernahe Dienste (funktionale Module)	systemnahe Basisdienste (Datenabstraktions-module)
■ Objektorientierte Analyse/ Entwurf (I 3.10)	*top-down/ bottom-up*	allgemeingültige Klassen	spezielle/spezialisierte Klassen
■ Schrittweise Verfeinerung (I 4.3)	*top-down*	abstrakte Daten und Anweisungen	Daten und Anweisungen in der gewählten Programmiersprache
Software-Management			
■ Planung			
□ Netzpläne/Gantt-Diagramme (II 2.4)	*top-down/ bottom-up*	abstrakter Vorgang	Detaillierter Vorgang
Software-Qualitätssicherung			
■ Komponenten- und Integrationstest (III 5, 6)	*top-down/ bottom-up*	systemferne, benutzernahe Dienste	systemnahe Basisdienste

© Prof. Dr. Liggesmeyer, 94

Allgemeine Methoden

- »außen / innen«-Methoden
  - *outside-in*-Methode
    - Zunächst wird Umwelt eines Systems modelliert und davon ausgehend die Systeminterna
  - *inside-out*-Methode
    - Zunächst werden die Systeminterna und dann die Schnittstellen zur Umwelt eines Systems modelliert
  - Modellierung findet, zumindest beim 1. Vorgehensschritt, auf derselben Abstraktionsebene statt
  - Bei einer *top-down*- bzw. *bottom-up*- Methode werden hingegen bereits im 1. Vorgehensschritt verschiedene Abstraktionsebenen modelliert

Allgemeine Methoden

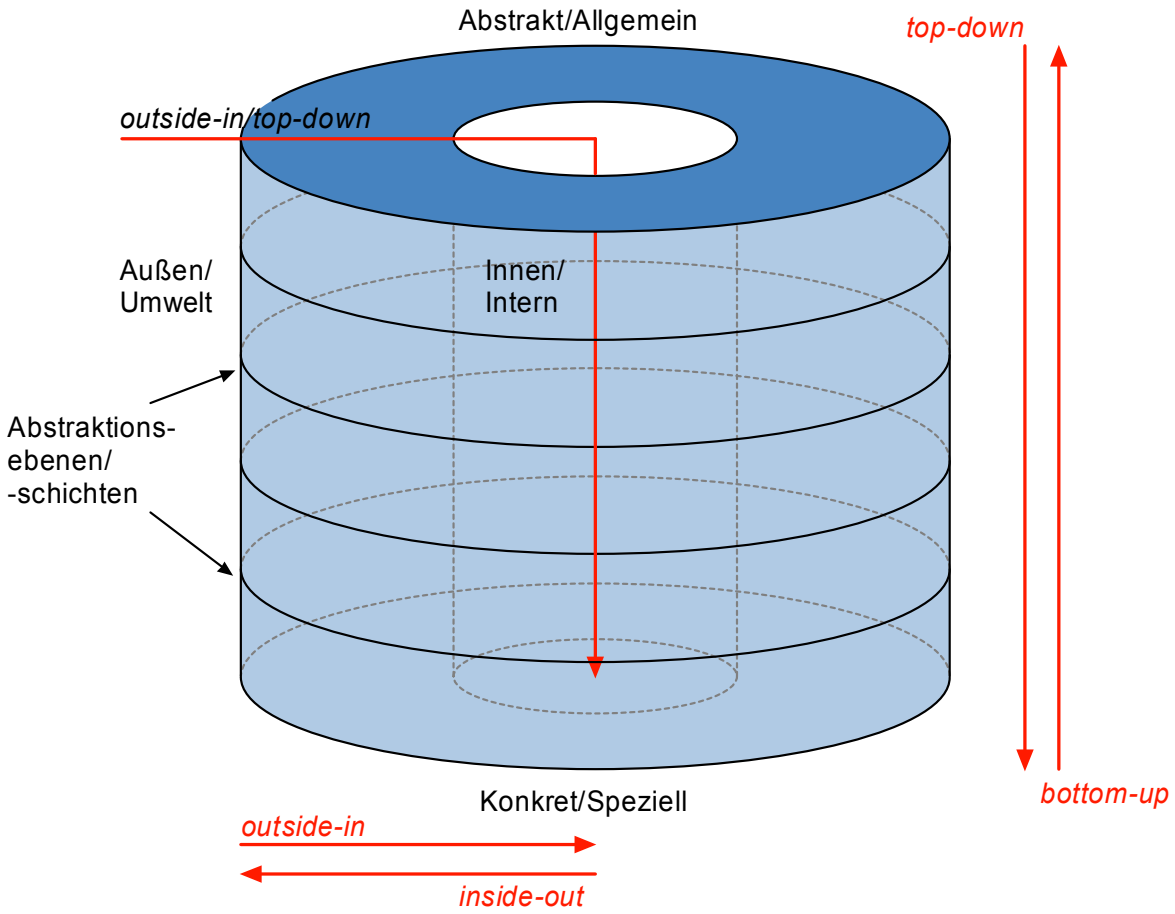
Beispiele für *outside-in*- & *inside-out*-Methoden

Methode	Art	»top«	»bottom«
Software-Entwicklung			
■ Modellierung von Datenflüssen			
☐ Datenfluß-Diagramml (I 2.7)	*outside-in*	Schnittstellen des Systems zur Umwelt (Informationsquellen/-senken)	Datenflüsse, Funktionen und Speicher im System
☐ SA-Modellierung (I 2.19)			
■ Modellierung eines Petri-Netzes (I 2.17.8)	*outside-in*	Schnittstellen des Systems zur Umwelt (Erzeugen und Löschen von Objekten)	Stellen und Transitionen
■ Modellierung zeitbasierter Vorgänge	*outside-in*	Benutzeraktionen / externe Ereignisse	Botschaften zwischen Klassen bzw. Objekten
☐ Interaktions-Diagramme (I 2.12)			
Software-Qualitätssicherung			
■ Komponententest			
☐ Funktions- und Strukturtest (III 5)	*outside-in*	Schnittstelle	Datenstrukturen und Algorithmen
☐ Integrationstest (III 6)	*outside-in* *inside-out*	höchste Schicht	niedrigste Schicht
Unternehmensmodellierung			
■ Objektorientierte Unternehmensmodellierung (V 2.2)	*outside-in*	Akteure, die mit dem Unternehmen kommunizieren	Unternehmen als System

© Prof. Dr. Liggesmeyer, 96

Allgemeine Methoden

Veranschaulichung der Methoden



© Prof. Dr. Liggesmeyer, 97

Allgemeine Methoden

- Praxis
  - Methoden werden nicht in »Rein-Form« angewandt
  - Oft wird abwechselnd »*top-down*« und »*bottom-up*« vorgegangen
    - Darauf achten, dass man sich in der Mitte trifft
  - Möglich ist auch, in der Mitte zu beginnen und dann von der Mitte *bottom-up* zum »*top*« hin und *top-down* zum »*bottom*« hin zu entwickeln *(middle-out)*
  - Analog kann dies auch bei *outside-in* und *inside-out* durchgeführt werden
  - Der abwechselnde Einsatz konträrer Methoden wird auch als »Jo-Jo-Methode« bezeichnet

© Prof. Dr. Liggesmeyer, 98

Allgemeine Methoden

- *ouside-in* / *top-down*-Methode
  - SA-Modellierung
  - Modellierung hierarchischer Petri-Netze
- Nachteil der SA-Modellierung
  - SA erlaubt es nicht, die Umwelt zu verfeinern, sondern nur die Interna
  - Schnittstellen können in jedem Datenflussdiagramm nur wiederholt, aber nicht weiter aufgeteilt werden
- Die Anwendung der *top-down*- bzw. *bottom-up*-Methode ist in der Software-Technik immer mit der Bildung von Abstraktionsebenen bzw. -schichten verbunden

© Prof. Dr. Liggesmeyer, 99

Allgemeine Methoden

- Bewertung
  - *top-down*-Methode
    - \+ Konzentration auf das Wesentliche möglich
      - Keine »Überschwemmung« mit Details
    - \+ Strukturelle Zusammenhänge werden leichter erkannt
    - – Es wird ein hohes Abstraktionsvermögen benötigt
    - – Entscheidungen werden u.U. vor sich hergeschoben, d.h. unbequeme Entscheidungen werden an tiefere Ebenen weitergereicht
    - – Der *»top«* ist oft nicht eindeutig zu bestimmen

© Prof. Dr. Liggesmeyer, 100

Allgemeine Methoden

- *bottom-up*-Methode
  - \+ Es ist eine konkrete Ausgangsbasis vorhanden
  - \+ Eine Begrenzung auf konkrete Teilgebiete ist möglich
  - \+ Wiederverwendbarkeit wird unterstützt
  - – Übergeordnete Strukturen werden durch Details überdeckt
  - – Es muss eine breite Basis gelegt werden, um das Ziel sicher zu erreichen

© Prof. Dr. Liggesmeyer, 101

Allgemeine Methoden

- *outside-in*-Methode
  - \+ Konzentration auf die Umwelt bzw. den Kontext und die Schnittstellen des Systems
  - \+ Beziehungen zur Außenwelt werden nicht übersehen
  - – Die Außenwelt lenkt von den internen Problemen ab
  - – Die Wiederverwendbarkeit wird erschwert

- *inside-out*-Methode
  - \+ Strukturüberlegungen stehen im Mittelpunkt
  - – Gefahr, dass Anforderungen der Umwelt übersehen oder zu spät erkannt werden

© Prof. Dr. Liggesmeyer, 102

Allgemeine Methoden

- Auswahl der richtigen Methode
  - Die Mehrzahl der Methoden der Software-Technik lassen sich entsprechend den aufgeführten allgemeinen Methoden klassifizieren
  - Die Anwendung dieser allgemeinen Methoden sollte nicht dogmatisch, sondern tendenziell gesehen werden
  - Neuentwicklung von Software-Produkten
    - *top-down*- und/oder die *outside-in*-Methoden bevorzugen
  - Wiederverwendbarkeit
    - Teilweise sinnvoll, *bottom-up* und/oder *inside-out* vorzugehen

© Prof. Dr. Liggesmeyer, 103

Allgemeine Methoden

- Eigenschaften einer guten Software-Methode
  - Das entwicklungsorientierte Vorgehen muss berücksichtigt werden
    - Die einzelnen Teile eines Software-Produktes entstehen nicht von heute auf morgen, sondern zeitlich hintereinander und/oder parallel
  - Beispiel: Pascal
    - Erlaubt keine entwicklungsorientierte Beschreibung
    - Vor dem ersten Compilerlauf muss das vollständige Programm vorhanden sein
    - Später geschriebene Prozeduren müssen jeweils zwischen bereits vorhandene Vereinbarungs- und Anweisungsteile eingeschoben werden

© Prof. Dr. Liggesmeyer, 104

Allgemeine Methoden

- Reihenfolge der Entwicklungsentscheidungen wird berücksichtigt
  - Der Software-Entwicklungsprozess erfordert ständig das Treffen von Entscheidungen
  - Mit jeder Entscheidung, die gefällt wird, wird der weitere Entwicklungsspielraum eingeengt
  - Eine gute Methodik zeichnet sich dadurch aus, dass sie Entscheidungen nur zu den Zeitpunkten verlangt, zu denen sie auch aus sachlicher Sicht sinnvoll überhaupt möglich sind
  - Entscheidungen, die erst spät getroffen werden können, sollen von der Methodik auch erst zu einem späten Zeitpunkt gefordert werden

© Prof. Dr. Liggesmeyer, 105

Allgemeine Methoden

- Beispiel
  - Oft wird bereits zu einem sehr frühen Entwurfszeitpunkt von der Methode her die Festlegung auf Wiederholungs- und Auswahlstrukturen verlangt
  - Solche Entscheidungen können aufgrund noch nicht vorhandener Informationen nicht begründet getroffen werden
  - Sie verlagern zu einem frühen Zeitpunkt den Schwerpunkt der Überlegungen auf Implementierungsdetails

© Prof. Dr. Liggesmeyer, 106

Allgemeine Methoden

- Anforderungen an eine »solide Methode« /Jackson 76/
  - Sie baut nicht auf die Intuition des Entwicklers
  - Sie beruht auf durchdachten Prinzipien
  - Sie ist lehrbar
  - Sie ist in der Anwendung unabhängig vom jeweiligen Anwender
  - Sie ist einfach und leicht zu verstehen

© Prof. Dr. Liggesmeyer, 107