Software Entwicklung 2

Entwurf

Inhalt

- Ziele und Aufgaben des Entwurfs
- Funktionale Abstraktion
- Strukturdiagramme
- Strukturierter Entwurf
- Strukturierte Softwarearchitektur
- Modularer Entwurf
- Datenabstraktion
- Abstrakte Datenobjekte
- Abstrakte Datentypen
- Transformation von SA nach SD bzw. MD

Lernziele

- Ziele und Aufgaben des Entwurfs erläutern können
- Die Eigenschaften der funktionalen Abstraktion erläutern können
- Strukturdiagramme lesen und erzeugen können
- Die Unterschiede zwischen Funktionaler Abstraktion und Datenabstraktion erläutern können
- Die Unterschiede zwischen abstrakten Datenobjekten und abstrakten Datentypen erläutern können
- Die Eigenschaften abstrakter Datenobjekte und abstrakter Datentypen erläutern können
- Für eine gegebene Problemstellung eine Datenabstraktion realisieren können
- Funktionale Abstraktion und Datenabstraktion anwenden können
- Eine Transformation von SA nach SD durchführen können
- Eine Transformation von SA nach MD durchführen können

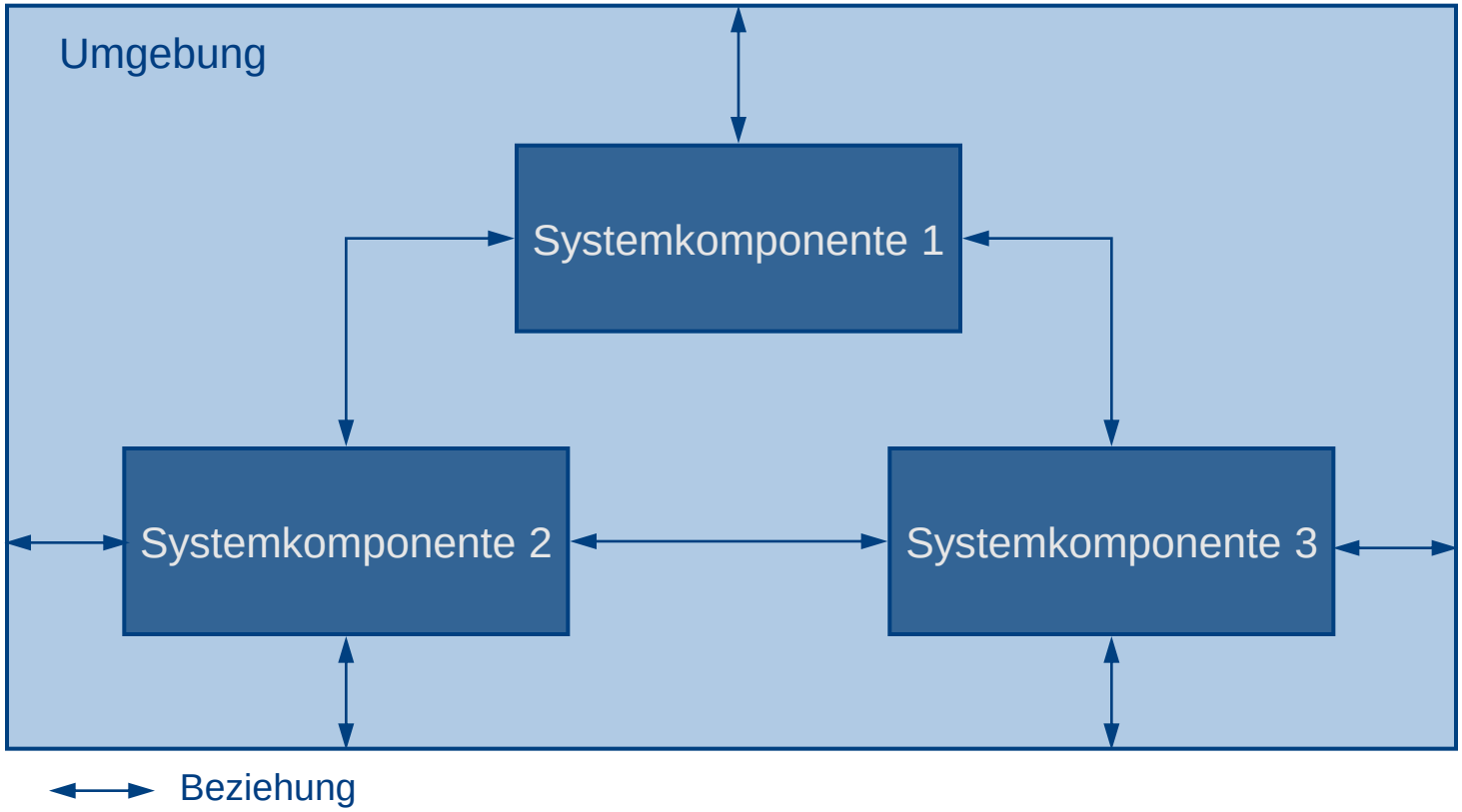
Ziele und Aufgaben des Entwurfs

- Aus den gegebenen Anforderungen an ein Software-Produkt eine software-technische Lösung im Sinne einer Software-Architektur entwickeln
- Auch »Programmieren im Großen« genannt
- Ausgangspunkt
  - Ergebnisse der Definitionsphase
- Vor den Entwurfsaktivitäten
  - Randbedingungen klären und festlegen
  - Umgebungsbedingungen klären und festlegen
  - Grundsatzentscheidungen fällen

Ziele und Aufgaben des Entwurfs

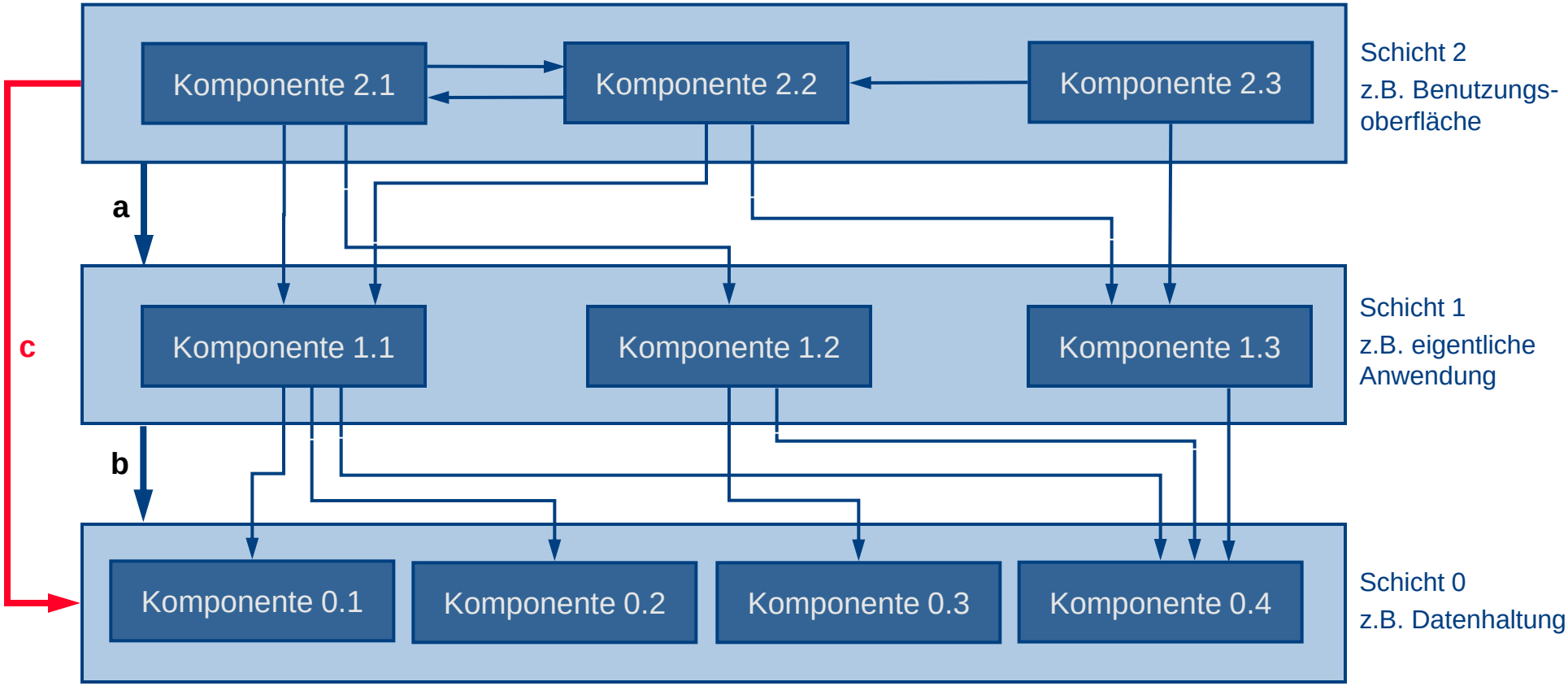
- Ziel
  - Für das zu entwerfende Produkt eine Softwarearchitektur erstellen, die die funktionalen und nichtfunktionalen Produktanforderungen sowie allgemeine und produktspezifische Qualitätsanforderungen erfüllt und die Schnittstellen zur Umgebung versorgt
- Software-Architektur
  - Beschreibt die Struktur des Softwaresystems durch Systemkomponenten und ihre Beziehungen untereinander, z.B.
    - Schichten mit linearer Ordnung
    - Schichten mit strikter Ordnung
    - Schichten mit baumartiger Ordnung

- Systemkomponenten und ihre Beziehungen



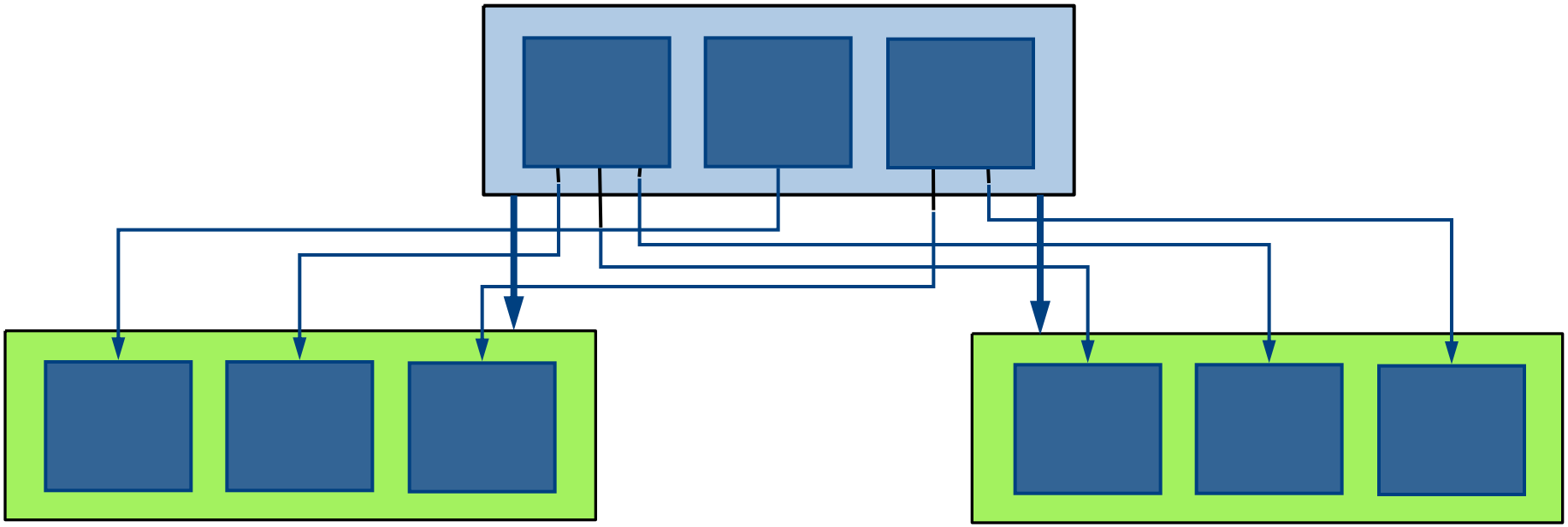
Ziele und Aufgaben des Entwurfs

- Beispiel für eine Drei-Schichten-Architektur



Ziele und Aufgaben des Entwurfs

- Beispiel für eine baumartige Architektur



Legende:

Komponente A benutzt Komponente B

Schicht A benutzt Schicht B

Ziele und Aufgaben des Entwurfs

- Schichtenarchitektur
  - Übersichtliche Strukturierung
  - Strenge Hierarchie zwischen den Schichten in Kombination mit liberalen Strukturierungsmöglichkeiten innerhalb der Schichten
  - Unterstützung von Wiederverwendbarkeit, Änderbarkeit, Wartbarkeit, Portabilität, Testbarkeit
  - Effizienzverlust, da alle Daten über verschiedene Schichten transportiert werden müssen (bei strikter Ordnung)
  - Eindeutig voneinander abgrenzbare Abstraktionsschichten lassen sich nicht immer definieren
  - Innerhalb einer Schicht kann Chaos herrschen

Ziele und Aufgaben des Entwurfs

- Aufgaben der Entwurfsphase
  - Ermitteln und Festlegen der Umgebungs- und Randbedingungen
  - Treffen grundlegender Entwurfsentscheidungen
  - Entwerfen einer Software-Architektur
    - Zerlegung des definierten Systems in Systemkomponenten
    - Strukturierung des Systems durch geeignete Anordnung der Systemkomponenten
    - Beschreibung der Beziehungen zwischen den Systemkomponenten
  - Spezifikation des Funktions- und Leistungsumfangs sowie des Verhaltens der Systemkomponenten
    - informal, semiformal oder formal
  - Festlegung der Schnittstellen, über die die Systemkomponenten miteinander kommunizieren

Ziele und Aufgaben des Entwurfs Ergebnisse

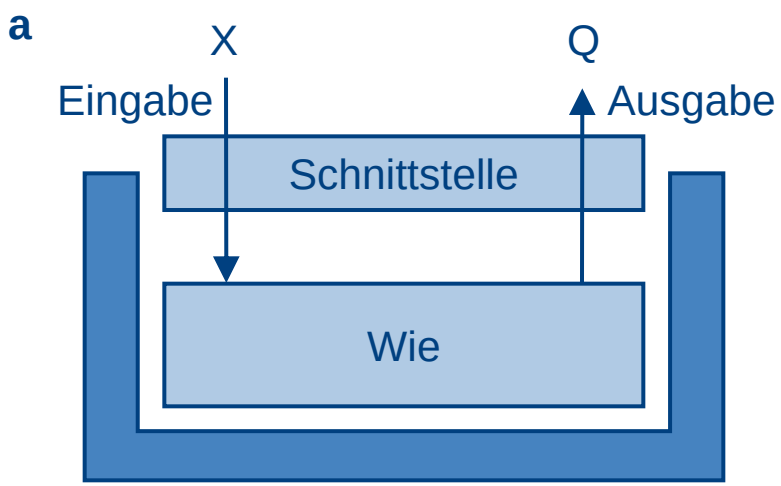
- Softwarearchitektur
  - Strukturierte oder hierarchische Anordnung der Systemkomponenten und ihrer Beziehungen untereinander
- Spezifikation jeder Systemkomponente
  - Festlegung von Schnittstelle, Funktions- und Leistungsumfang

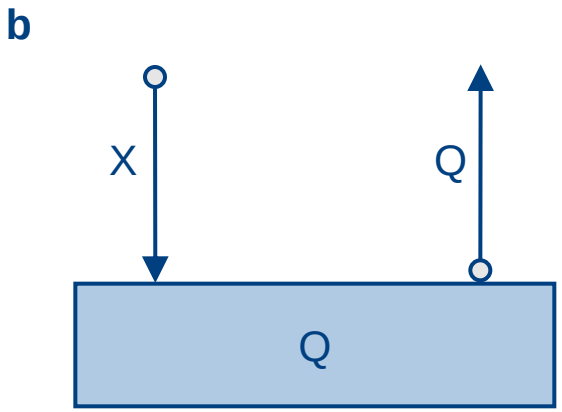
Funktionale Abstraktion

- Entwurf eines Software-Systems
  - Reduzierung der Problemkomplexität durch Wahl geeigneter Abstraktionen
- Funktionale Abstraktion
  - Stellt eine Leistung in Form einer abstrakten
    - Funktion
    - Operation oder
    - Prozedur zur Verfügung
  - Historisch erste Form der Abstraktion
  - Seit 1954 in Fortran, später in allen klassischen Programmiersprachen vorhanden
  - Oft operationale oder prozedurale Abstraktion genannt

Funktionale Abstraktion

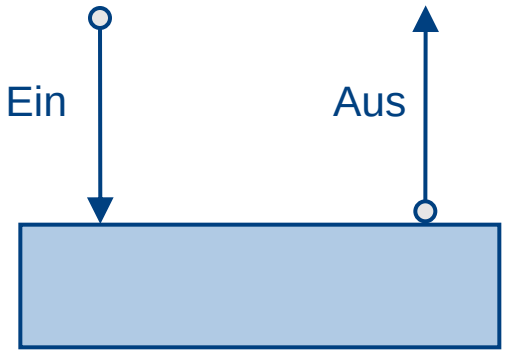
- Anwendung einer Standardfunktion oder einer selbst geschriebenen Funktion
  - Algorithmus interessiert nicht
  - Anwender nimmt an, dass z.B. die Operation Quadratwurzel ***Q*** richtig angewendet ein korrektes Ergebnis liefert, dass also ***Q(2)*** die Wurzel aus ***2*** hinreichend genau liefert
  - Darstellung als Modul bzw. als Strukturdiagramm

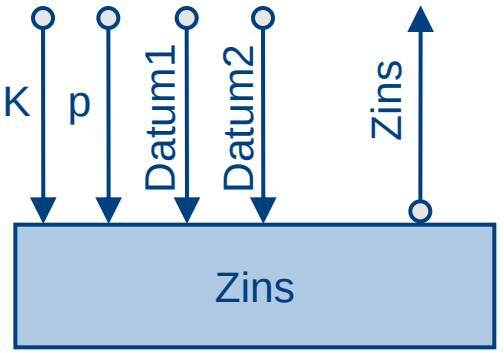




Funktionale Abstraktion

- Strukturdiagramm (*structure chart*)
  - Grafische Darstellung von funktionalen Abstraktionen und ihrem Zusammenwirken
  - Beispiel: Funktion für »Zinsberechnungen«



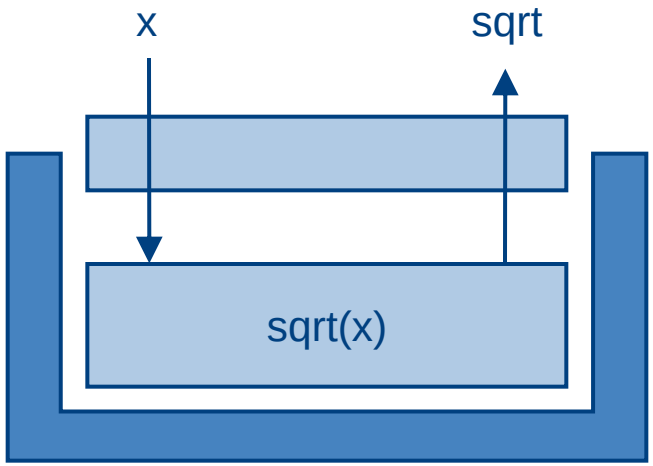


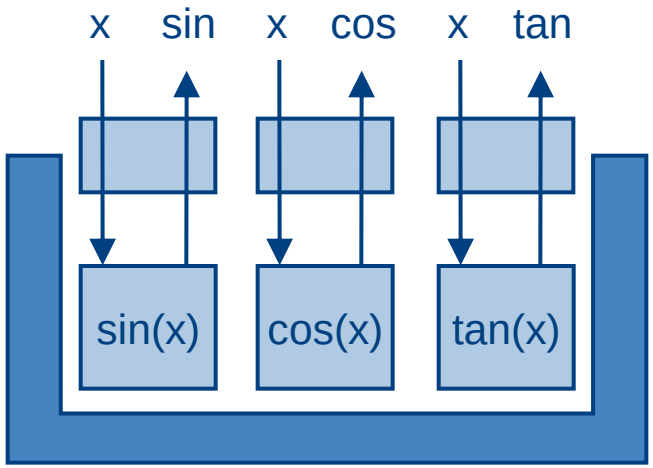
Funktionale Abstraktion

- Funktionales Modul / Funktionsmodul
  - Realisiert eine funktionale Abstraktion
  - Eigenschaften
    - Aktiv bzw. aktionsorientiert, es »tut« etwas
    - Besitzt ein Transformationsverhalten
      - Eingabedaten werden in Ausgabedaten transformiert
    - Identische Eingabedaten führen immer zu identischen Ausgabedaten
      - **Kein internes Gedächtnis**
  - Aufgaben
    - Steuerungs- und Koordinationsaufgaben
      - Beispiel: Hauptprogramm
    - Transformationsaufgaben unterschiedlicher Komplexität
      - Beispiel: Compiler
    - Auswertungsaufgaben, die man als Spezialfall von Transformationen ansehen kann
      - Beispiel: mathematische Routinen
    - Hilfsaufgaben

Funktionale Abstraktion

- Ausprägungen
  - Funktionales Modul = Funktionale Abstraktion
  - Funktionales Modul = mehrere, logisch, d.h. semantisch, zusammengehörende funktionale Abstraktionen
    - Beispiele
      - trigonometrische Funktionen
      - komplexe Arithmetik
      - Grafikfunktionen



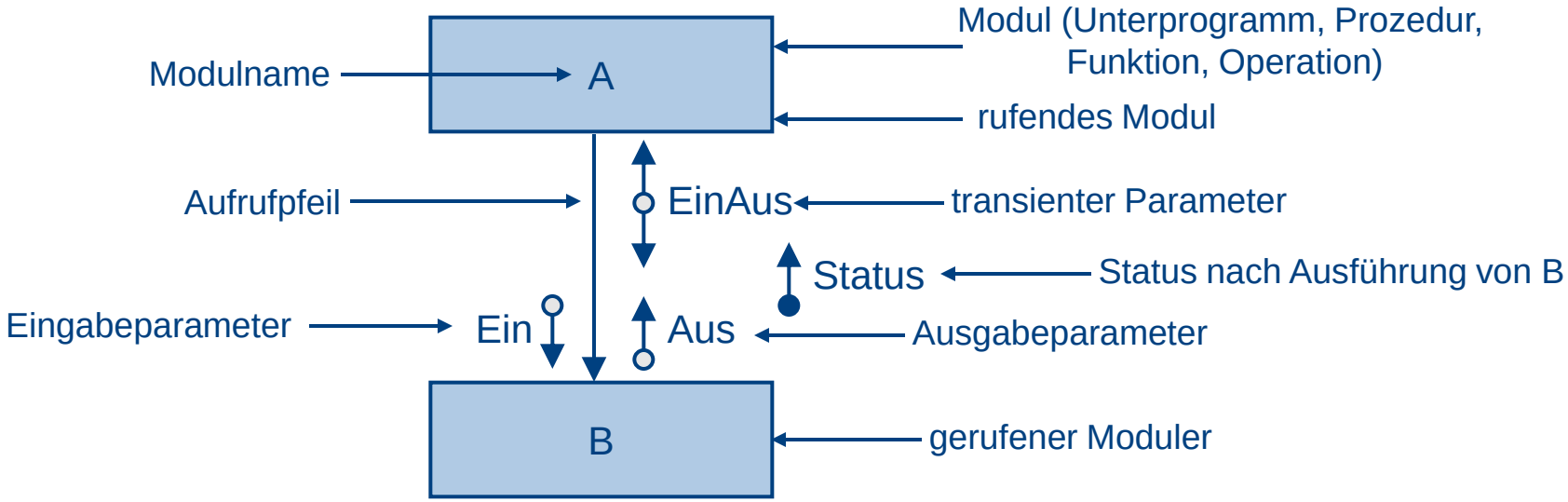


Funktionale Abstraktion

- Programmiersprachen
  - Durch Unterprogrammkonzepte realisiert
    - Prozeduren / Funktionen / Operatoren
  - C: Durch Funktionen implementiert
    - Prozeduren sind Sonderfälle von Funktionen
  - Java: Keine eigenständigen funktionalen Module
    - Realisierung durch Operationen, die zu Klassen gehören
- Softwarearchitektur
  - Funktionale Module und Datenabstraktionsmodule benötigt
  - Nur funktionale Module: Unübersichtliche und änderungsunfreundliche Architekturen

Strukturdiagramme

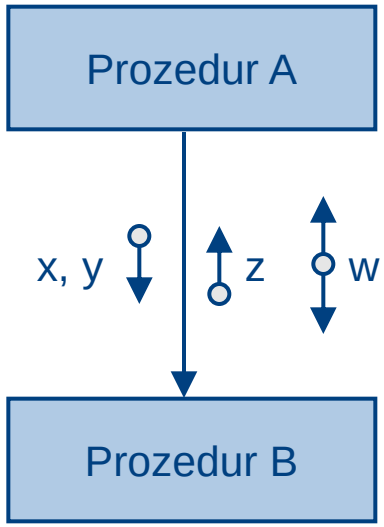
- Grafische Darstellung von funktionalen Modulen
- Verdeutlichung von Aufrufstruktur und Datenfluss zwischen Modulen



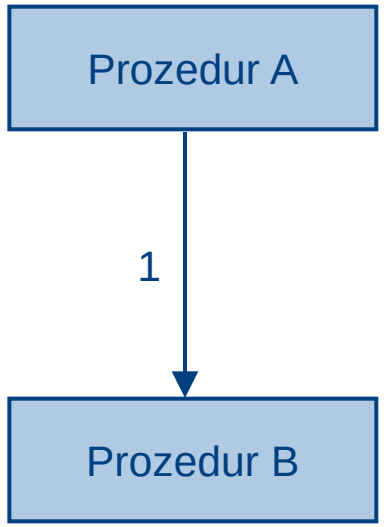
Strukturdiagramme

- Datenflüsse grafisch oder als Tabelle, bei großer Anzahl an Datenflüssen

a Datenflüsse grafisch



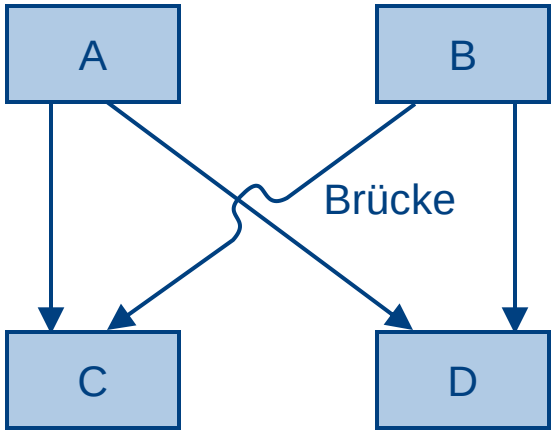
b Datenflüsse in Schnittstellentabellen

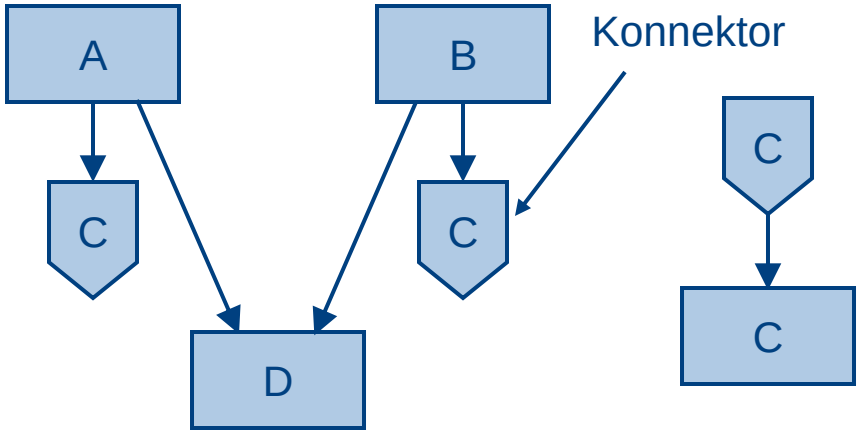


Nr	in	out	inout
1	x, y	z	w
...	...	...	...

Strukturdiagramme

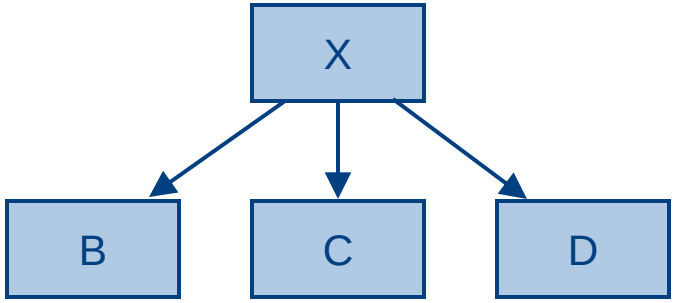
- Brücken und Konnektoren: Ein Modul wird nur einmal gezeichnet, auch wenn es von mehreren Stellen aufgerufen wird



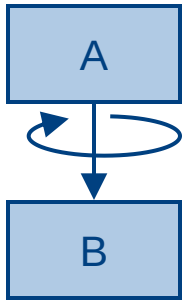


Strukturdiagramme

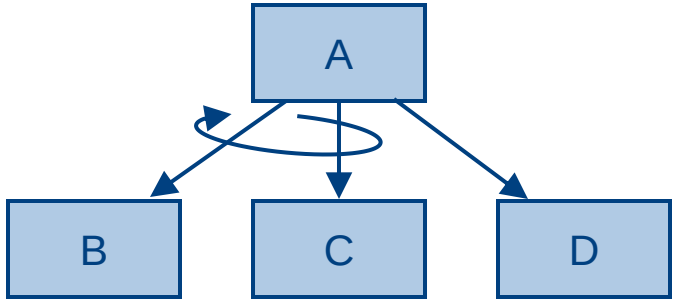
- Aufruf (keine bestimmte Reihenfolge)



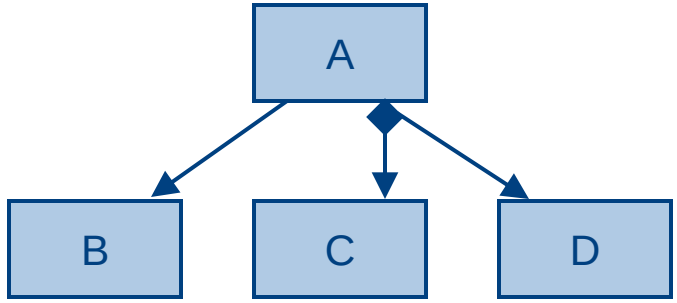
- Wiederholung von B



- Wiederholung von B und C



- Auswahl: C und/oder D werden in Abhängigkeit einer Bedingung aufgerufen



- Vorhandenes Modul (z.B. ein Betriebssystemdienst)

Strukturierter Entwurf
Zur Historie

- Edward Yourdon
  *30.4.1944
  Software engineering consultant
  New York
- Wegbereiter des
  strukturierten Entwurfs 1979
  (Buch: Structured Design,
  zusammen mit L. L. Constantine)
- Pionier bei der objektorientierten Analyse 1990
  (Buch: Object-Oriented Analysis, zusammen mit P. Coad)

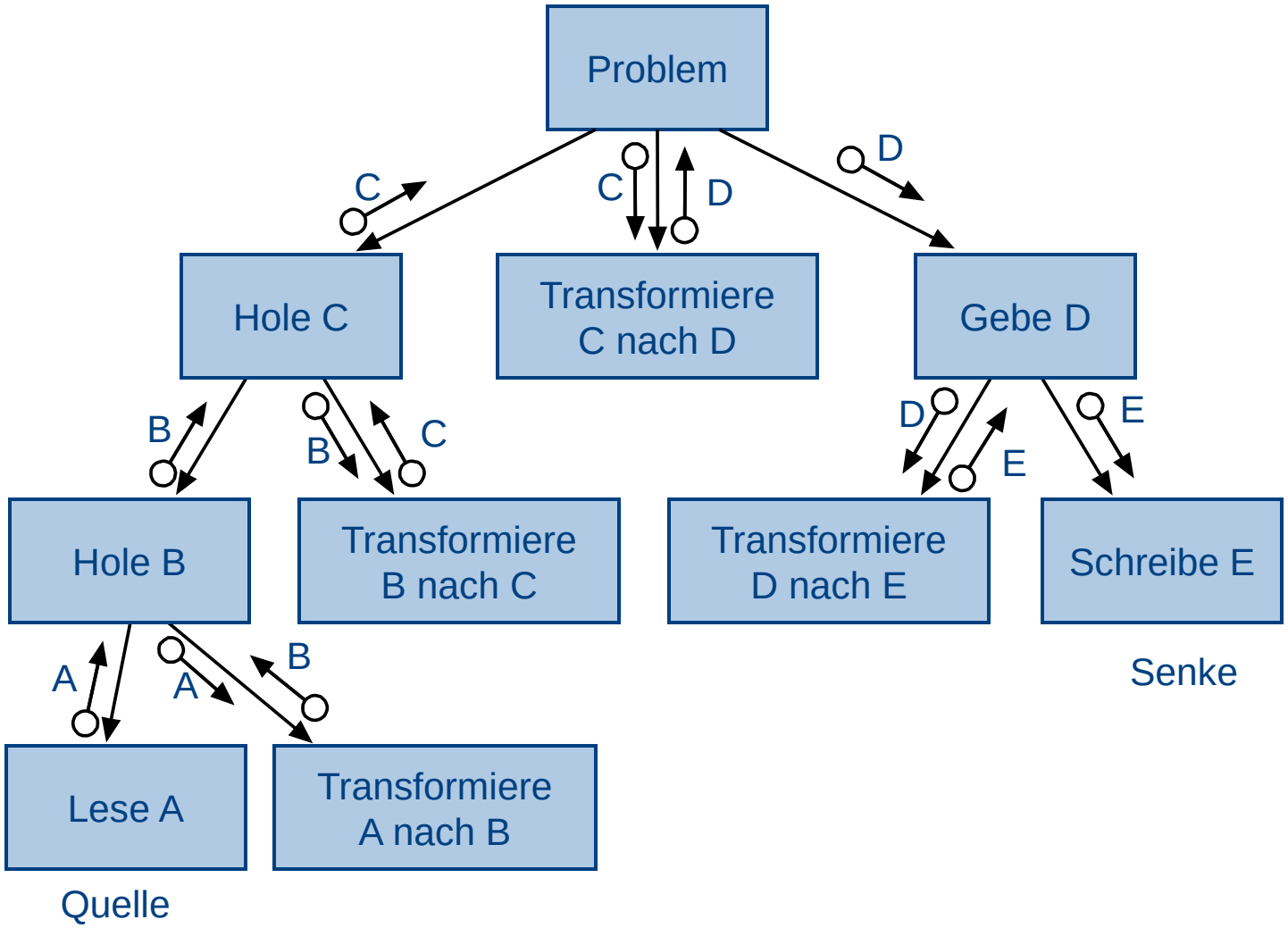
Strukturierter Entwurf

- Strukturierter Entwurf
  (composite / structured design, SD)
  - Entwurfsmethode von
    - /Stevens et al. 74/, /Stevens 81/
    - /Myers 75, 78/
    - /Yourdon, Constantin 79/
    - /Page-Jones 88/
- Ziel
  - Software-Architektur, die aus hierarchisch angeordneten funktionalen Modulen besteht
- Beschreibung durch
  - Strukturdiagramme und
  - Modulspezifikationen

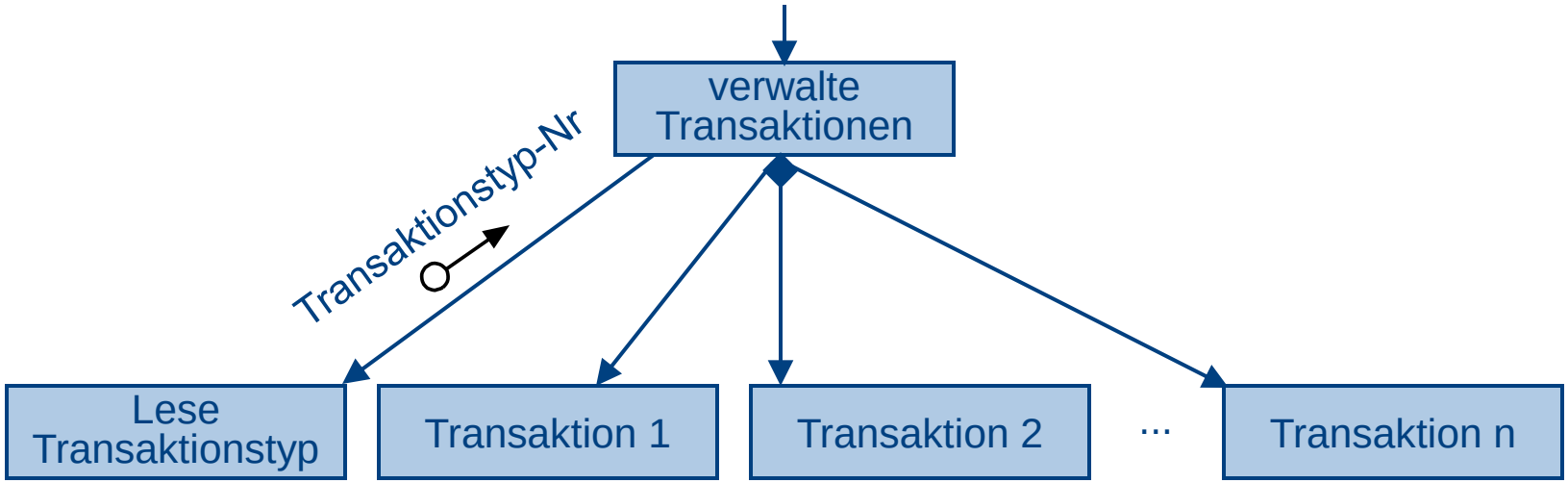
Strukturierte Software-Architektur

- Zwei Grundarchitekturen und Kombinationen daraus
  - Allgemeine Transformationsstruktur
  - Transaktionsstruktur
- Allgemeine Transformationsstruktur
  - Daten kommen von einer Quelle, werden transformiert und verschwinden in einer Senke
  - Merkmale
    - Module weitgehend kontextunabhängig
    - Zwischen den Kindern eines Vaterknotens bestehen keine Kommunikationsbeziehungen
    - Modul kann von mehreren Modulen aus aufgerufen werden

Strukturierte Software-Architektur
Allgemeine Transformationsstruktur



- Transaktion besitzt fünf Komponenten
  - Ein Ereignis innerhalb der Systemumgebung
  - Ein Stimulus an das System (*stimulus*)
  - Eine Aktivität des Systems (*activity*)
  - Eine Antwort vom System (*response*)
  - Ein Effekt auf die Systemumgebung (*effect*)



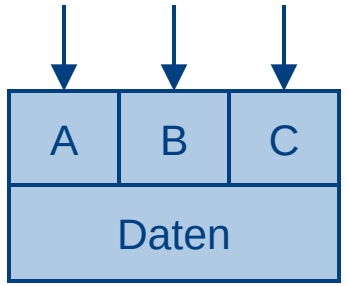
Strukturierte Software-Architektur
Beispiel: Transaktionsstruktur »Alarmanlage«

- Eine Person nähert sich dem überwachten Objekt (Ereignis)
- Ein Passiv-Infrarotmelder meldet die erfassten Wärmestrahlen an das System (Stimulus)
- Das System führt je nach aktuellem Systemzustand eine entsprechende Aktivität aus
  - Beispiel: Wechsel in den Zustand Voralarm und Einschalten der Beleuchtung (Antwort des Systems)

Strukturierte Software-Architektur

- Qualitätsziele
  - Starke Bindung (*strong cohesion*)
  - Lose Kopplung (*loose coupling*)
- Strukturierter Entwurf
  - Enthält von Konzept und Methode her nur funktionale Module
  - Erst Page-Jones führt auch abstrakte Datenobjekte ein, genannt Informational Cluster
  - Module, die mehrere funktionale Abstraktionen zur Verfügung stellen, sind nicht vorgesehen
- Bewertung
  - Nur noch in Sonderfällen sinnvoll

informational cluster



Modularer Entwurf
Zur Historie

- Prof. Dr. David Lorge Parnas
  *10.2.1941 in New York
  Communications Research Laboratory
  University Hamilton, Ontario, Canada
- Erfinder des Geheimnisprinzips (*information-hiding*) (1971)
- Wegbereiter des Modularen Entwurfs (1972)
  - Spezifikation von Moduln
  - Trennung von Spezifikation und Implementierung
  - modulare Software-Architektur
- Beiträge zur Disziplin Software-Technik (seit 1975)
- Beiträge zur Qualitätssicherung von Software

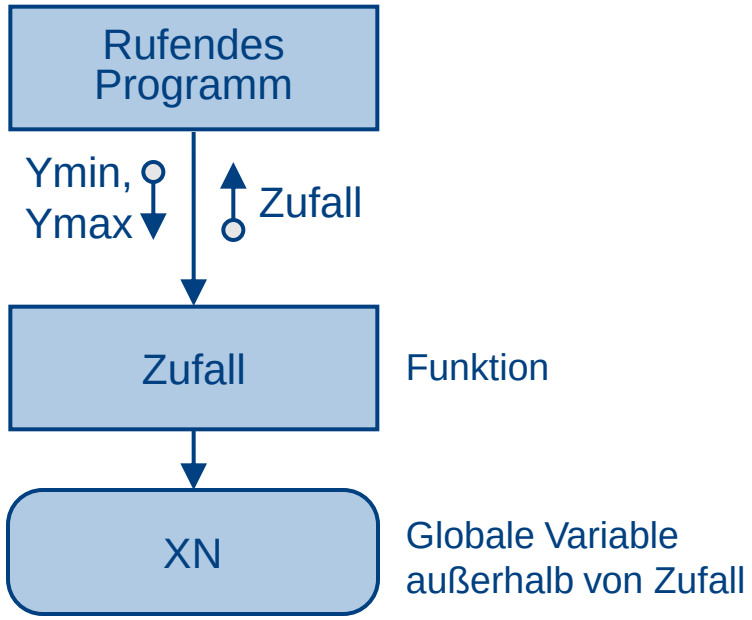
Modularer Entwurf

- Funktionale Abstraktion ist für viele Anwendungen nicht ausreichend
- Modularer Entwurf (modular design, MD)
  - Funktionale Abstraktion + »Datenabstraktion«
- Datenabstraktion
  - Manipulation komplexer Datenobjekte mit spezialisierten, problembezogenen Operationen
  - Abstrakte Datenobjekte
  - Abstrakte Datentypen (ADT)
  - Funktionale Abstraktion: Abstraktion vom Algorithmus
  - Datenabstraktion: Abstraktion von einer Datenstruktur und ihrer Zugriffsalgorithmen

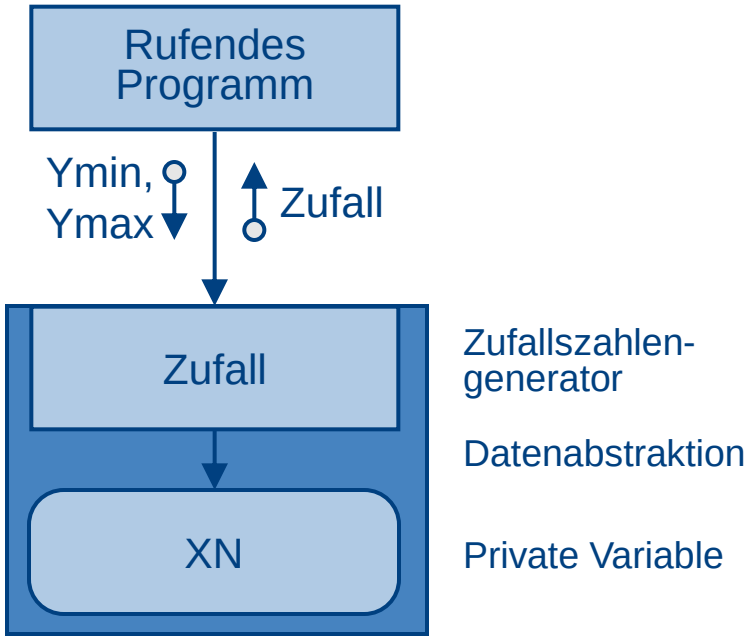
Datenabstraktion

- Beispiel: Zufallszahlengenerator
  - mit funktionaler Abstraktion (a)
  - mit Datenabstraktion (b)

a



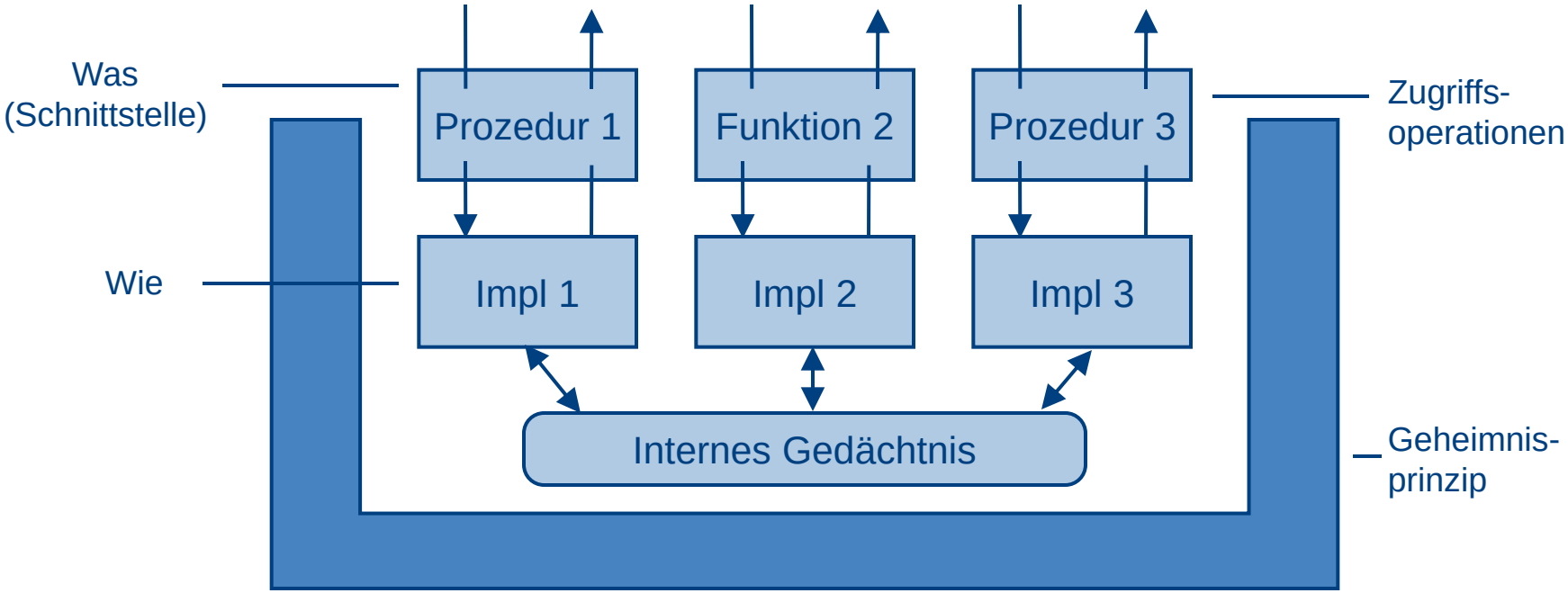
b



Abstrakte Datenobjekte

- Zusammenfassung von
  - Datenstrukturen und
  - Zugriffsoperationen, die darauf arbeiten
- Lebensdauer der in den Datenstrukturen gespeicherten Daten
  - Geht über das Aufruf-Ende einer Zugriffsoperation hinaus (Wesentlicher Unterschied zu funktionalen Abstraktionen)
- Charakteristika
  - Anwender kann nur über die Zugriffsoperationen auf die Daten der Datenstruktur zugreifen
  - Abstraktion besteht aus Anwendersicht darin, dass die Details der Datenstruktur verborgen sind
  - Manipulationen der Daten sind nur über die Zugriffsoperationen möglich
  - Direkter Zugriff auf Komponenten der Datenstruktur mit Basisoperationen ist nicht erlaubt

Abstrakte Datenobjekte
Aufbau eines Datenobjektmoduls



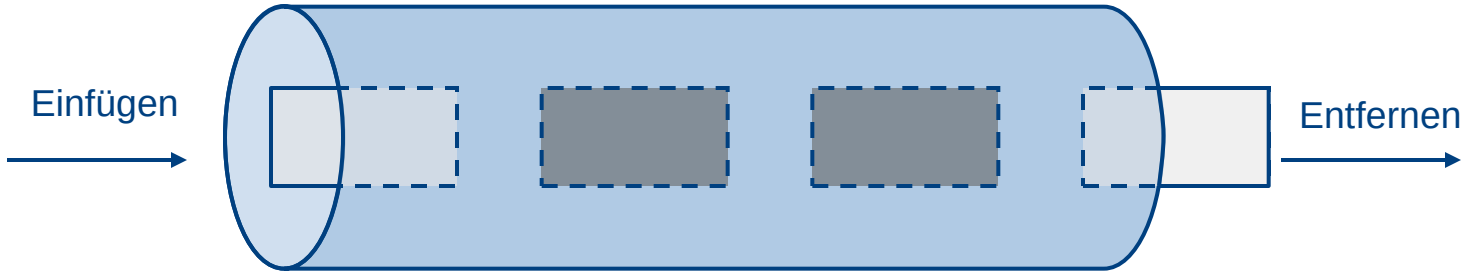
Abstrakte Datenobjekte
Daten-Objekt-Modul

- Hat passiven Charakter
- Verwaltet die Ablage von Daten (internes Gedächtnis), die über Zugriffsoperationen gelesen oder geschrieben werden
- Internes Gedächtnis ist entweder an die Laufzeit des entsprechenden Software-Systems gebunden oder wird in einer Datei aufbewahrt
- Ausführung einer Zugriffsoperation mit identischen Eingabedaten führt nur dann zu identischen Ausgabedaten, wenn das interne Gedächtnis denselben Zustand hat.
- Internes Gedächtnis ist außerhalb des Moduls nicht sichtbar (Geheimnisprinzip)
- Daten und Zugriffsoperationen werden im Modul zu einer Einheit zusammengefasst (Verkapselung)
- Beschreibt ein Objekt nicht durch dessen Struktur, sondern charakterisiert es ausschließlich durch die Definition der darauf ausführbaren Operationen
- Zugriffsoperationen sind selbst funktionale Abstraktionen
- Wird durch seine Beschreibung kreiert, d.h. mit der Beschreibung ist gleichzeitig die Deklaration genau eines Exemplars verbunden

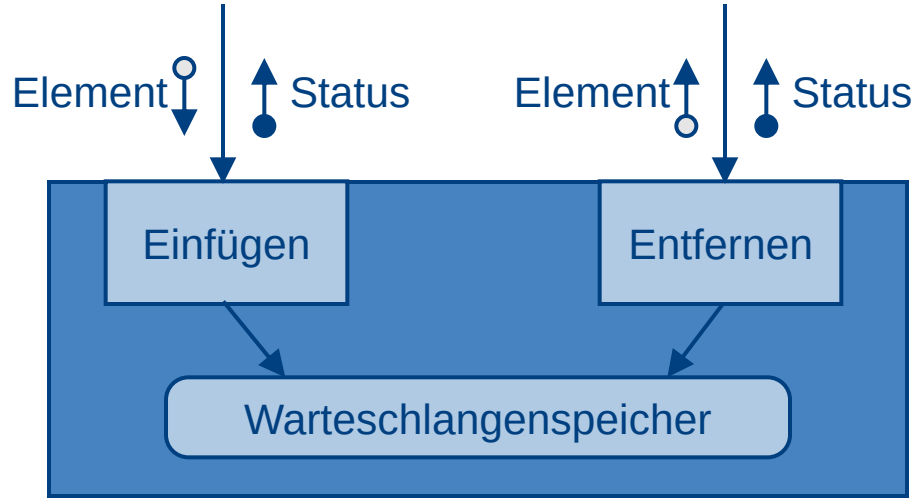
Abstrakte Datenobjekte
Vergleich: Objekt in der objektorientierten Softwareentwicklung

- Internes Gedächtnis entspricht den Attributen eines Objekts
- Zugriffsoperationen entsprechen den Objektoperationen

- Operation Einfügen
  - Hängt ein Element an die Warteschlange an
- Operation Entfernen
  - Entfernt das vorderste Element aus der Warteschlange
- Internes Gedächtnis
  - Bewahrt die Elemente auf, die sich momentan in der Warteschlange befinden

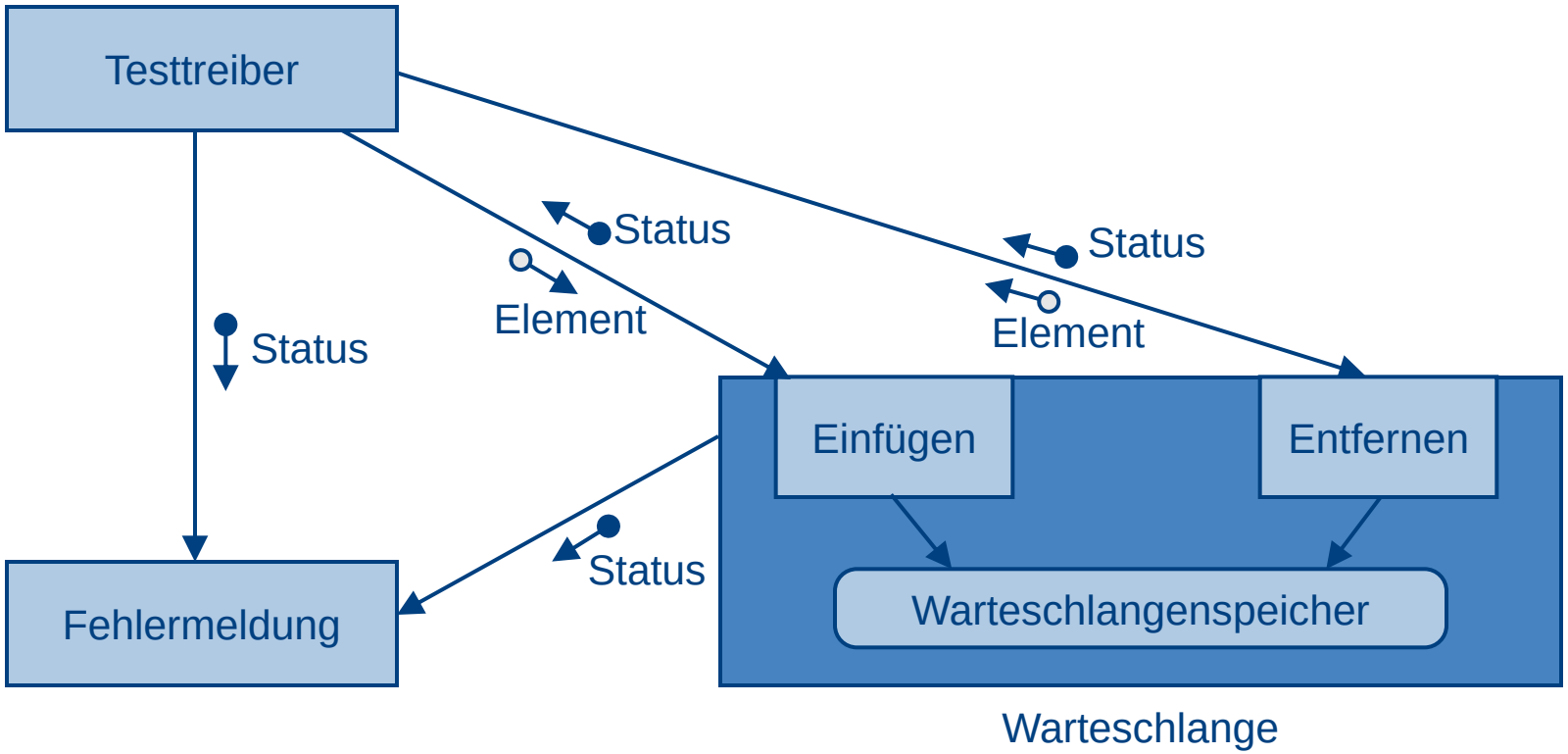


Abstrakte Datenobjekte
Warteschlange als Strukturdiagramm



Warteschlange

Abstrakte Datenobjekte
Strukturdiagramm der Anwendung Warteschlange



Abstrakte Datenobjekte
Modulspezifikation (Modulschnittstelle)

- Aufgabenbeschreibung
- Modultyp (Datenobjekt-Modul oder ADT-Modul)
- Leistungsmerkmale
  - Geschwindigkeit
  - Speicherplatz
  - Genauigkeit (bei numerischen Operationen)
- Voraussetzungen und Vorbedingungen für die Anwendung des Moduls
- Bedingungen, die nach der Anwendung des Moduls gelten
- Alle bereitgestellten Zugriffsoperationen

Abstrakte Datenobjekte
Modulspezifikation (Modulschnittstelle)

- Erforderliche Angaben für jede Zugriffsoperation
  - Eingabeparameter einschließlich Datentyp und Beziehungen untereinander
  - Ausgabeparameter einschließlich Datentyp und ihre Abhängigkeit von den Eingabeparametern und anderen Daten bzw. von internen Zuständen
  - Wirkung bzw. Effekte der Zugriffsoperation
  - Anwendungsvoraussetzungen und Vorbedingungen
  - Gültige Bedingungen nach der Anwendung
  - Verhalten bei inkorrekten Eingabewerten oder Fehlfunktion des zugrundeliegenden Basissystems

Abstrakte Datenobjekte
Modulspezifikation (Modulschnittstelle)

- Abhängigkeiten und Relationen zwischen den Zugriffsoperationen
  - Statische Relationen
  - Gegenseitige Ausschlussbedingungen
  - Zulässige Aufruffolgen
- Um die Abhängigkeiten zwischen den Zugriffsoperationen zu beschreiben, eignet sich die algebraische Spezifikation
- Gut geeignet sind auch Zustandsautomaten
  - Analog zur objektorientierten Welt können Objekt-Lebenszyklen auch bei abstrakten Datenobjekten beschrieben werden

Abstrakte Datenobjekte
Aufgaben abstrakter Datenobjekte

- Verwalten einer Sammlung (collection) von Einträgen
  - Beispiele: Warteschlange, Kunden, Dozenten
- Verwaltung einzelner komplexer Einträge
  - Beispiele: Hand bei der Robotersteuerung, multimediales Dokument
  - Die Verwaltung eines komplexen Einzeleintrags durch ein abstraktes Datenobjekt ist sinnvoll, um von der Realisierung zu abstrahieren
- Beide Aufgaben treten oft in Kombination auf

Abstrakte Datenobjekte
Vorteile eines Datenobjektmoduls

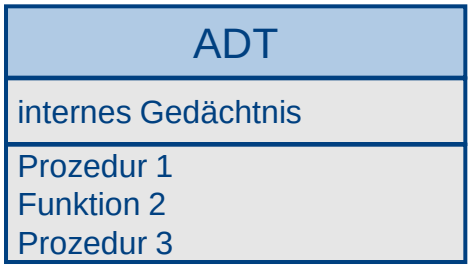
- Abstrakte Datenobjekte können allein durch Anwendung der definierten Zugriffsoperationen benutzt werden, ohne irgendwelche Kenntnisse der Implementation
- Durch die Datenabstraktion wird das »Was« vom »Wie« getrennt
- Die Zugriffsoperationen können ohne Rücksicht auf die Struktur des internen Gedächtnisses (z.B. Datei-, Satz- und Datenstrukturaufbau) von den Anforderungen der Anwendung her festgelegt werden
- Änderungen an einer Zugriffsoperation haben im Allgemeinen keine Auswirkungen auf andere Zugriffsoperationen
  - Ohne Datenabstraktion kann eine Änderung der Datenstruktur (einschl. Datei- und Satzstruktur) u.U. Auswirkungen auf alle zugreifenden Anweisungen haben
- Notwendige strukturelle Änderungen des internen Gedächtnisses haben im Allgemeinen keine Auswirkungen auf die Anwendung der Zugriffsoperation

Abstrakte Datenobjekte
Vorteile eines Datenobjektmoduls

- Je weniger Daten in der Parameterliste einer Zugriffsoperation übertragen werden
  - desto änderungsfreundlicher ist eine Datenabstraktion
  - desto mehr Zugriffsoperationen erhält man
- Der Anwender muss keine eigenen Programme schreiben, um die Datenstruktur zu manipulieren
- Die Semantik der Datenabstraktion ändert sich nicht, wenn nur die Implementierung modifiziert wird
- Eine Datenabstraktion kann durch mehrere Implementierungen realisiert werden

Abstrakte Datentypen (ADT)

- Trennung von Typdeklaration und Variablendeklaration
- Beliebig viele Exemplare können erzeugt werden
- Jedes Exemplar besitzt sein eigenes Gedächtnis
- Vergleich: Klasse – ADT
  - ADT entspricht weitgehend einer Klasse, aber ohne Vererbung



- Exemplare sind
  - im Verhalten identisch und
  - unterschieden sich durch den Typ des internen Gedächtnisses
- Ziel: Konstruktion typunabhängiger ADTs
- Weg: Typparametrisierung
  - Als formaler Parameter wird ein Typname angegeben
  - Bei der Erzeugung von Exemplaren wird auf der Parameterliste dann der für dieses Exemplar gewünschte Typ aufgeführt
- Beispiel: Für eine Anwendung werden drei verschiedene Warteschlangen benötigt
  - In einer Warteschlange sollen float-Zahlen verwaltet werden
  - In einer Warteschlange sind Einzelzeichen zu speichern
  - In einer Warteschlange sind Zeichenketten der Länge 24 aufzubewahren

Abstrakte Datentypen
Architekturdarstellung und Exemplardarstellung

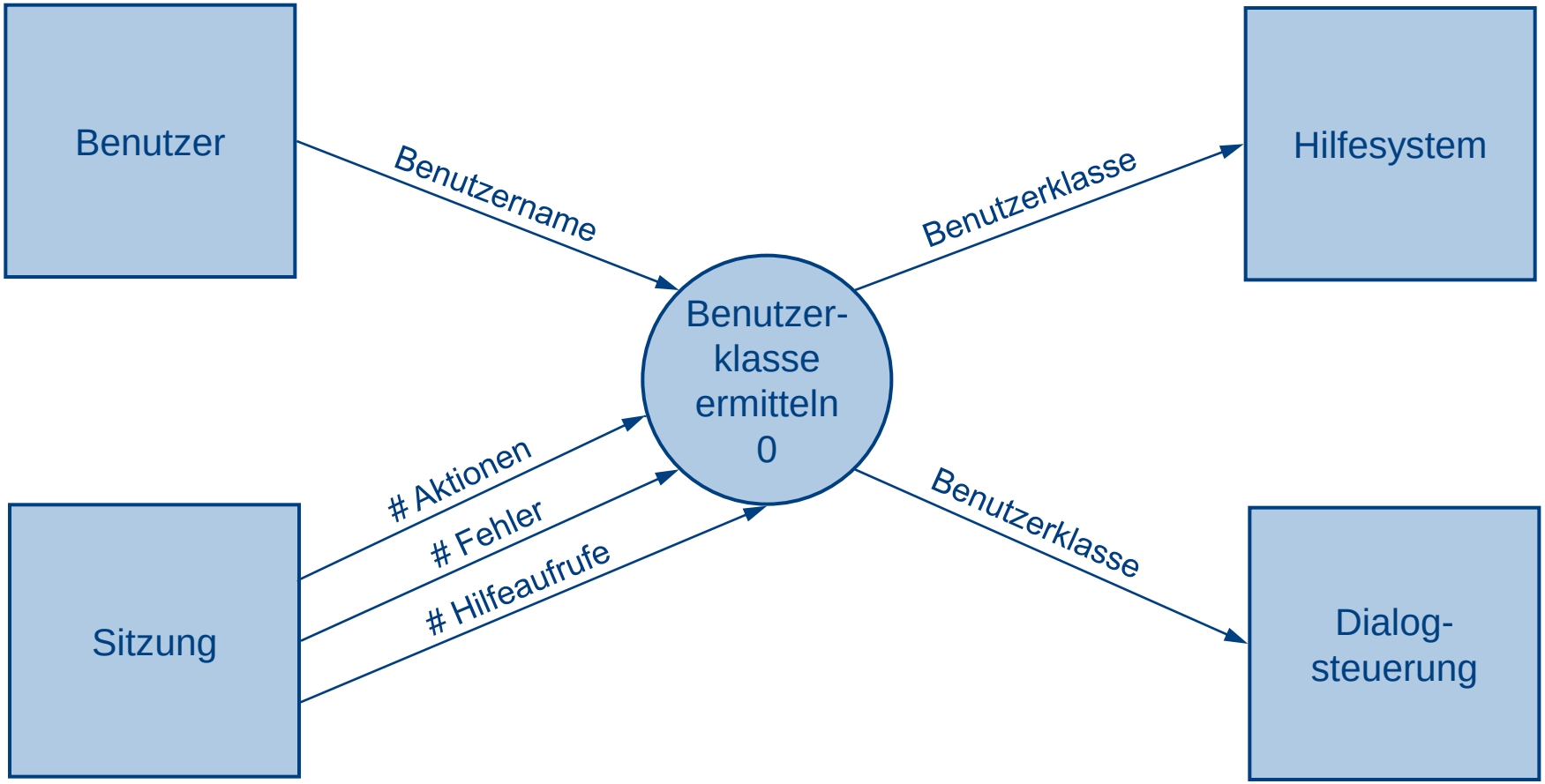
- Abstrakte Datenobjekte, die über Datenobjektmodule eingeführt wurden, erscheinen in der Software-Architektur
- Abstrakte Datenobjekte, die als Exemplare von ADTs erzeugt wurden, erscheinen nicht in der Architekturdarstellung
- Die Erzeugung eines abstrakten Datenobjekts wird im Rumpf eines Moduls durch eine einzige Zeile hingeschrieben und ist daher keine Entwurfskomponente

Transformation von SA nach SD

- Zwei Methoden
  - Transaktions-Analyse
    - Die Transaktionstypen eines Systems werden identifiziert
    - Jeder Transaktionstyp wird als Entwurfseinheit verwendet und für sich entworfen
  - Transformations-Analyse
    - Der Teil eines DFD, der durch die Transaktions-Analyse isoliert wurde, wird in ein Strukturdiagramm konvertiert
- Ableitung in drei Schritten
  - Aufbrechen des SA-Modells in handhabbare Einheiten durch die Transaktions-Analyse
  - Konvertieren jeder Einheit in ein gutes Strukturdiagramm durch die Transformations-Analyse
  - Zusammenfügen der getrennten Einheiten zu einem Gesamtsystem

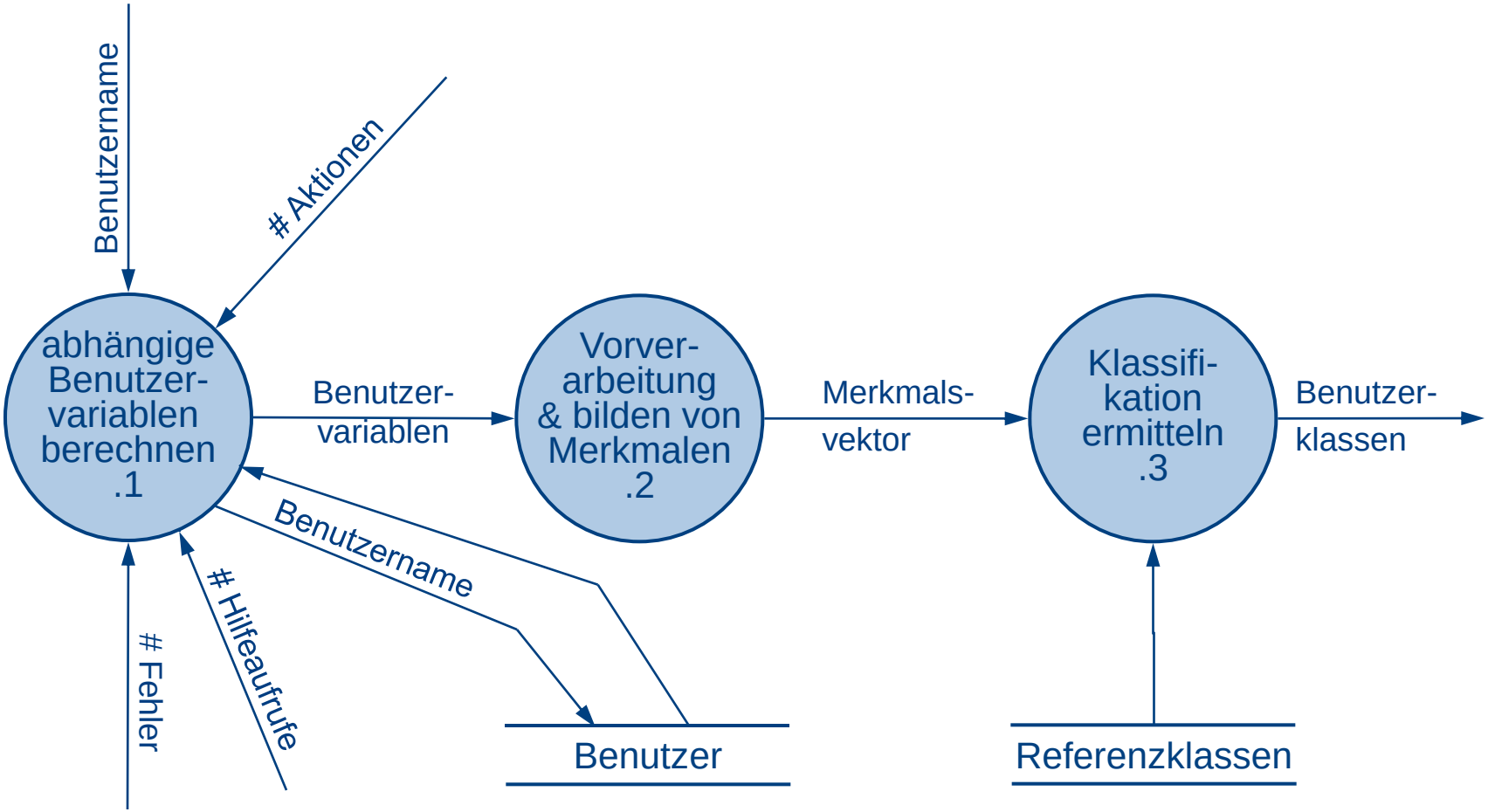
Transformation von SA nach SD

- Beispiel: Kontextdiagramm für ein Benutzermodell



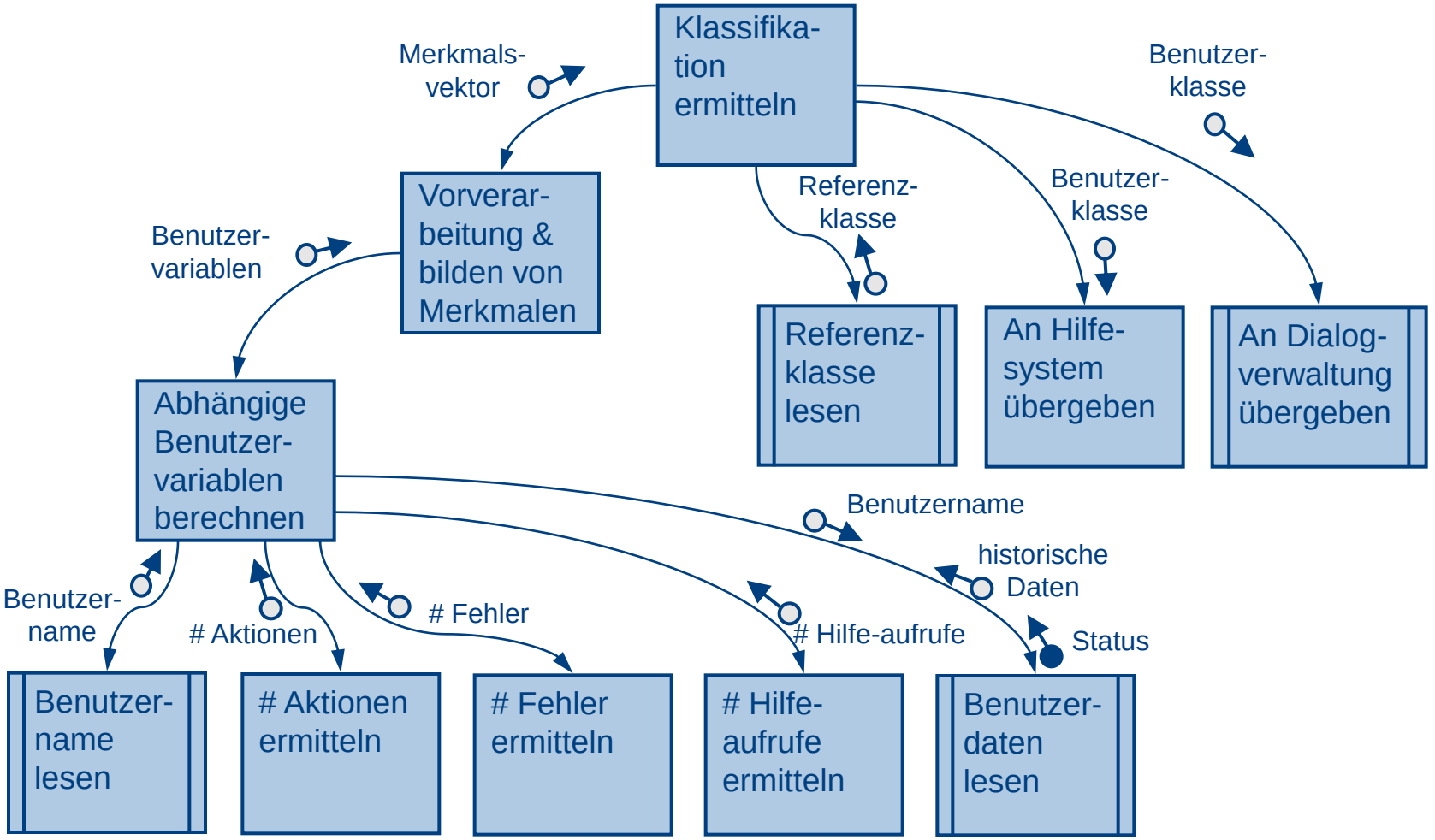
Transformation von SA nach SD

- DFD 0 für ein Benutzermodell



Transformation von SA nach SD

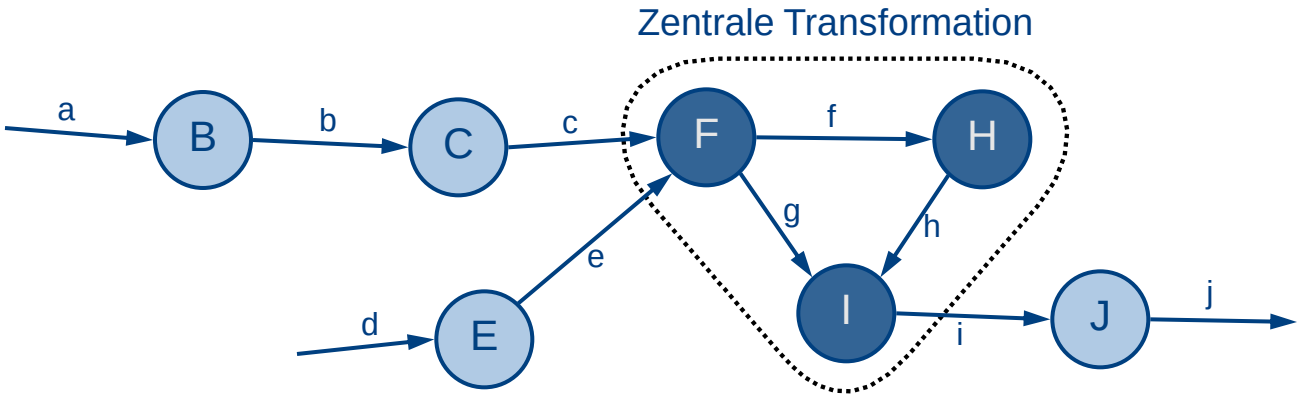
- Software-Architektur als Strukturdiagramm



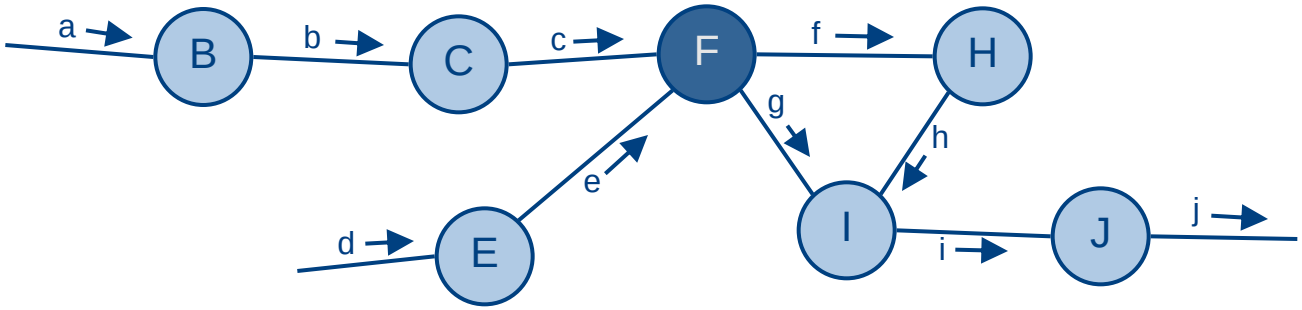
Transformation von SA nach SD Methode

- Transformations-Analyse in fünf Schritten
  - Zeichnen eines DFD pro Transaktionstyp
  - Finden der zentralen Funktionen bzw. Prozesse des DFD
  - Konvertieren des DFD in ein »erstes« Strukturdiagramm
  - Verfeinerung des Strukturdiagramms entsprechend den Entwurfskriterien
  - Überprüfung, ob das endgültige Strukturdiagramm die Anforderungen des ursprünglichen DFD erfüllt

- Verfolgen jedes Eingabedaten-Stroms vom Äußeren des DFD zur Mitte hin
- Verfolgen jedes Ausgabedaten-Stroms vom Äußeren des DFD zur Mitte hin
- Alle Markierungen werden durch eine geschlossene Kurve miteinander verbunden
- DFD-Notation



- Eine Funktion der zentralen Transformation wird zur »Steuerungszentrale« (hier: F)
- Übergang zum Strukturdiagramm



Transformation von SA nach SD

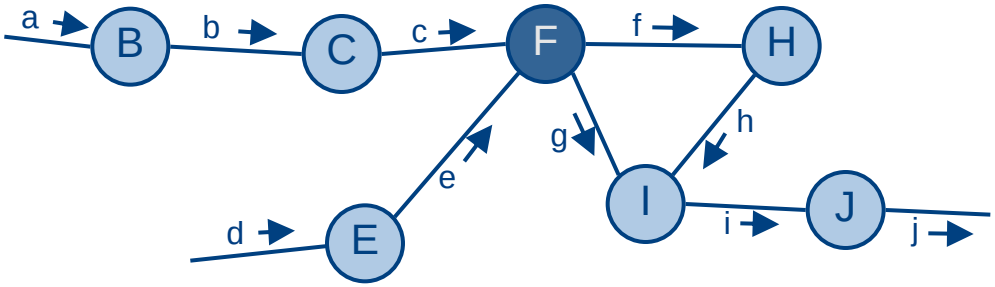
- Eigenschaft einer Steuerungszentrale
  - Wenig Verarbeitung – Viel Koordination
  - Falls mehrere Funktionen als Steuerungszentrale geeignet erscheinen, sollte jede Variante ausprobiert werden
  - Falls keine Kandidaten existieren, so kann die zentrale Transformation unter eine "synthetische" Steuerungszentrale "gehängt werden"
- Vergleich
  - DFD: Menge von Tischtennis-Bällen, die durch eine Schnur miteinander verbunden sind
  - Ein Ball wird aus der zentralen Transformation als Steuerungszentrale ausgewählt
  - Hebt man diesen Ball hoch, dann »baumeln« die anderen Bälle an den Schnüren

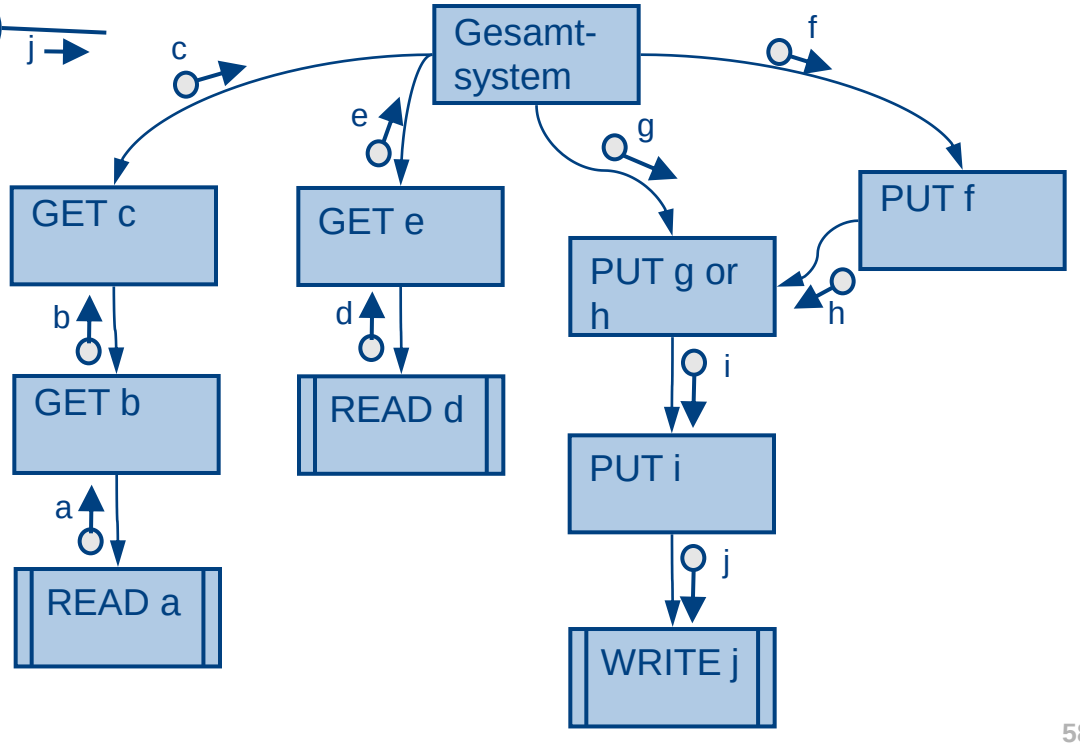
Transformation von SA nach SD

- Anschließend notwendige Änderungen durchführen
  - Pfeile aus dem DFD entfernen, da die Richtung des Datenflusses nicht mit der Aufrufrichtung der Module übereinstimmt
  - Aufrufpfeile hinzufügen
  - Kreise in Rechtecke umzeichnen
  - Modulnamen so modifizieren, dass sie die Aktivitäten der aufgerufenen Module insgesamt repräsentieren. (Der Funktionsname in einem DFD beschreibt nur die eigene Aktivität)
  - Hinzufügen von "technischen" Modulen (z.B. Schreib- und Lesemodulen), da das DFD ja nur die fachliche Sicht darstellt

Transformation von SA nach SD

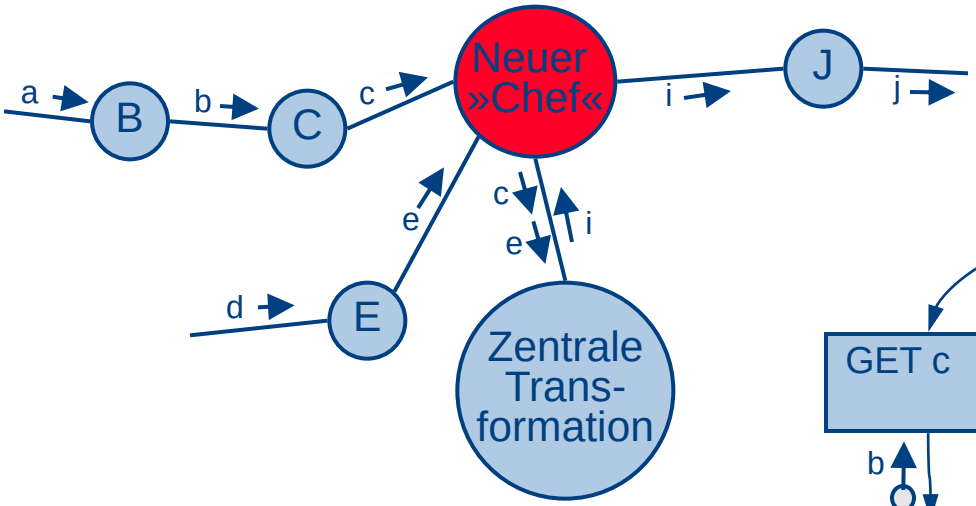
- Beispiel: Resultierende Strukturdiagramme

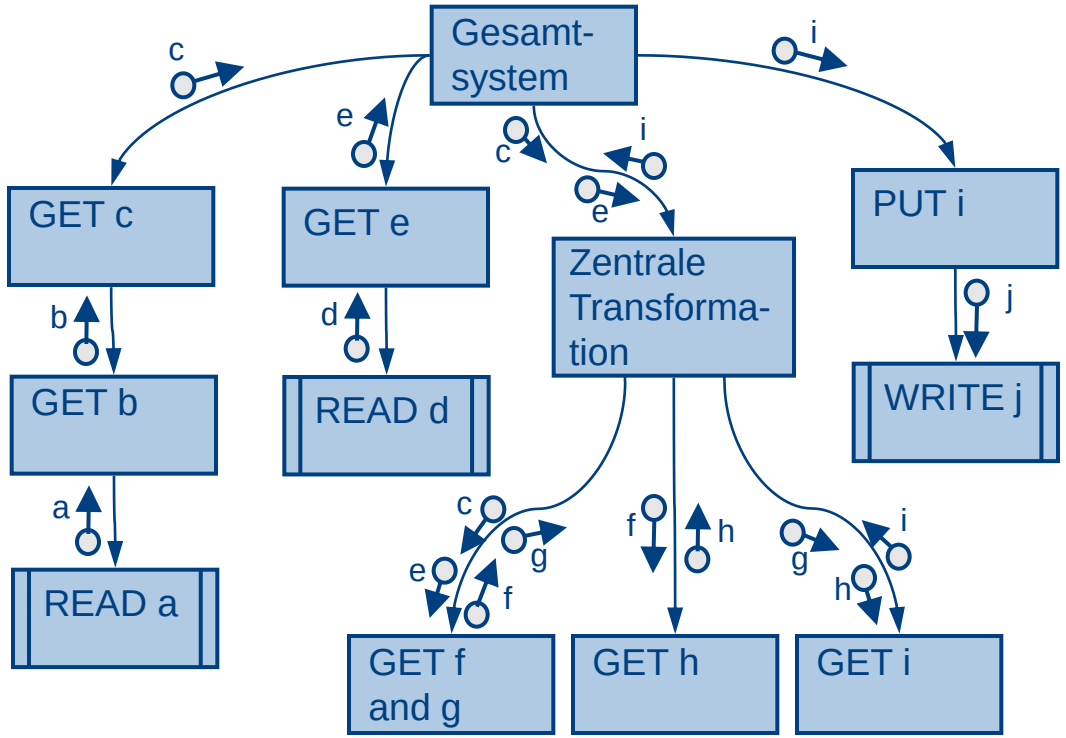




Transformation von SA nach SD

- Beispiel: Resultierende Strukturdiagramme





Transformation von SA nach SD
Schritt 4: Verfeinerung des Strukturdiagramms

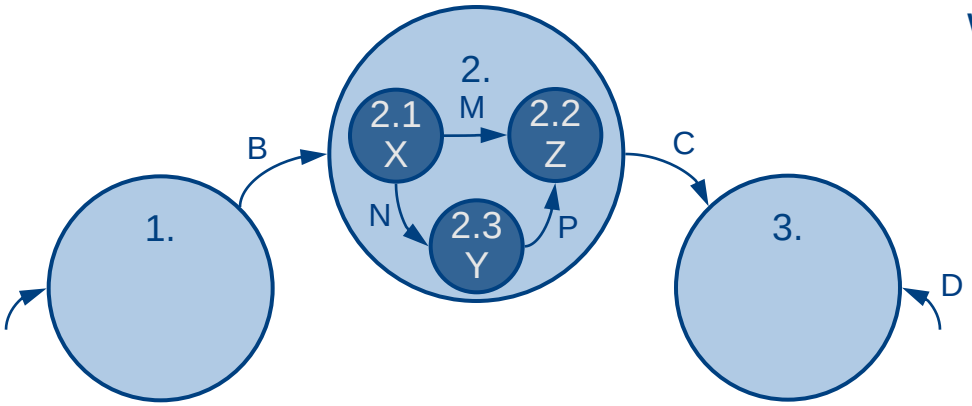
- Hinzufügen von Lese- und Schreibmodulen, um auf Quellen, Senken und Speicher zuzugreifen (im Beispiel bereits erfolgt)
- Überprüfung entsprechend der Entwurfskriterien Bindung und Kopplung und evtl. Umstrukturierung des Diagramms
- Hinzufügen von Fehlerbehandlungsmodulen
- Wenn erforderlich, Initialisierungs- und Terminationsdetails hinzufügen
- Sicherstellen, dass alle Module die Namen entsprechend der hierarchischen Position haben
- Hinzufügen von Status-Informationen

Transformation von SA nach SD
Schritt 5: Überprüfung

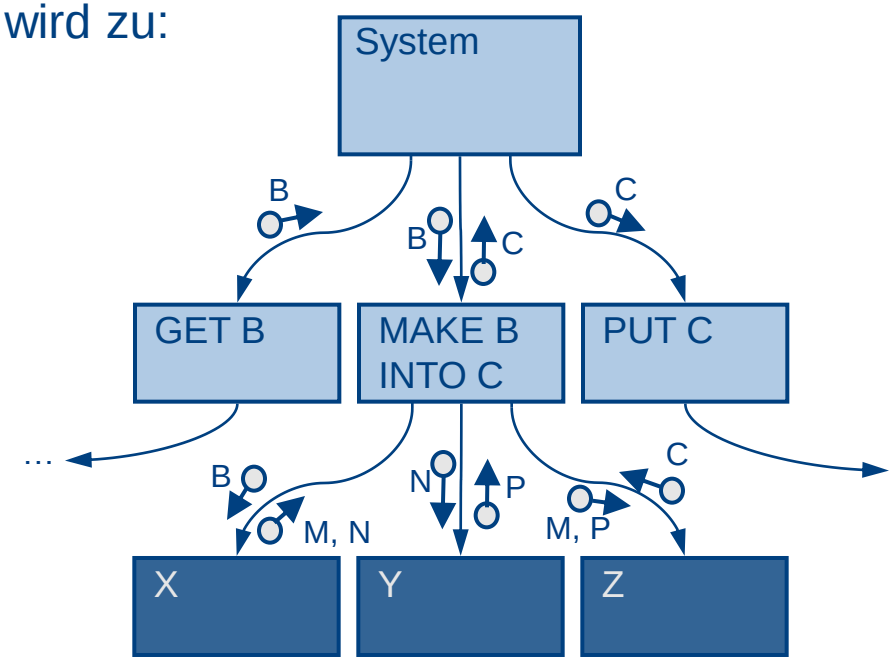
- Nach der Erstellung des endgültigen Strukturdiagramms ist zu prüfen, ob es die Anforderungen, d.h. das DFD, korrekt realisiert und ob es den Entwurfskriterien genügt
- Wünschenswert ist, dass der Autor des DFD das Strukturdiagramm überprüft
- Berücksichtigung mehrerer DFD-Ebenen
  - In der Regel besitzt ein SA-Modell mehrere DFD-Ebenen
  - Die verschiedenen DFD-Ebenen können bei der Transformation berücksichtigt werden

Transformation von SA nach SD

- Berücksichtigung mehrerer DFD-Ebenen



wird zu:

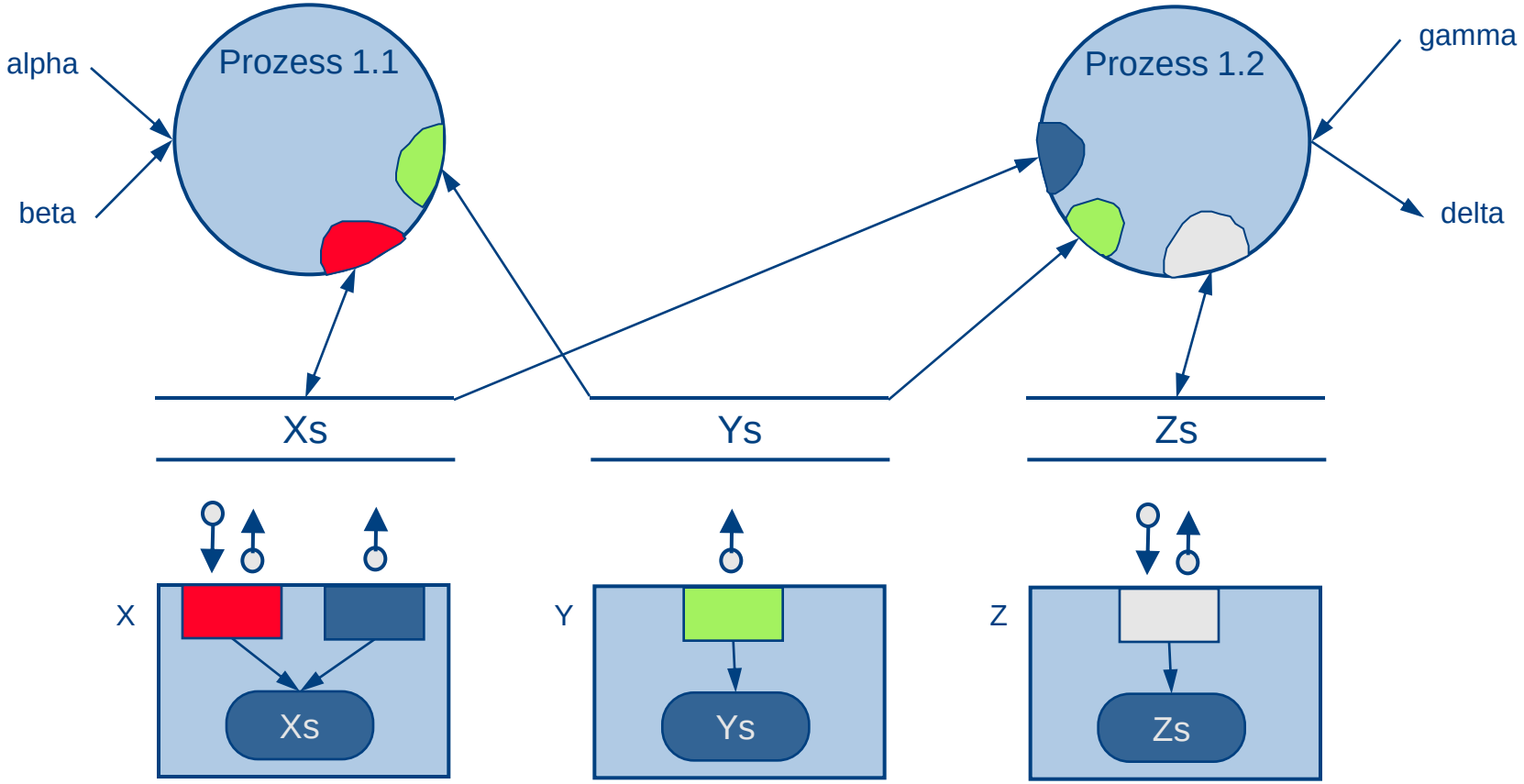


Transformation eines SA-Modells

- Transformation eines SA-Modells in einen Entwurf mit Datenabstraktionen
  - Ist nicht ohne Strukturbruch möglich, da DFDs nicht datenabstraktionsorientiert sind
  - Aus jedem Speicher eines DFD wird ein abstraktes Datenobjekt, das Zugriffsoperationen für die Prozesse zur Verfügung stellt
  - Die Prozesse werden in funktionale Module mit funktionaler Abstraktion transformiert

Transformation eines SA-Modells

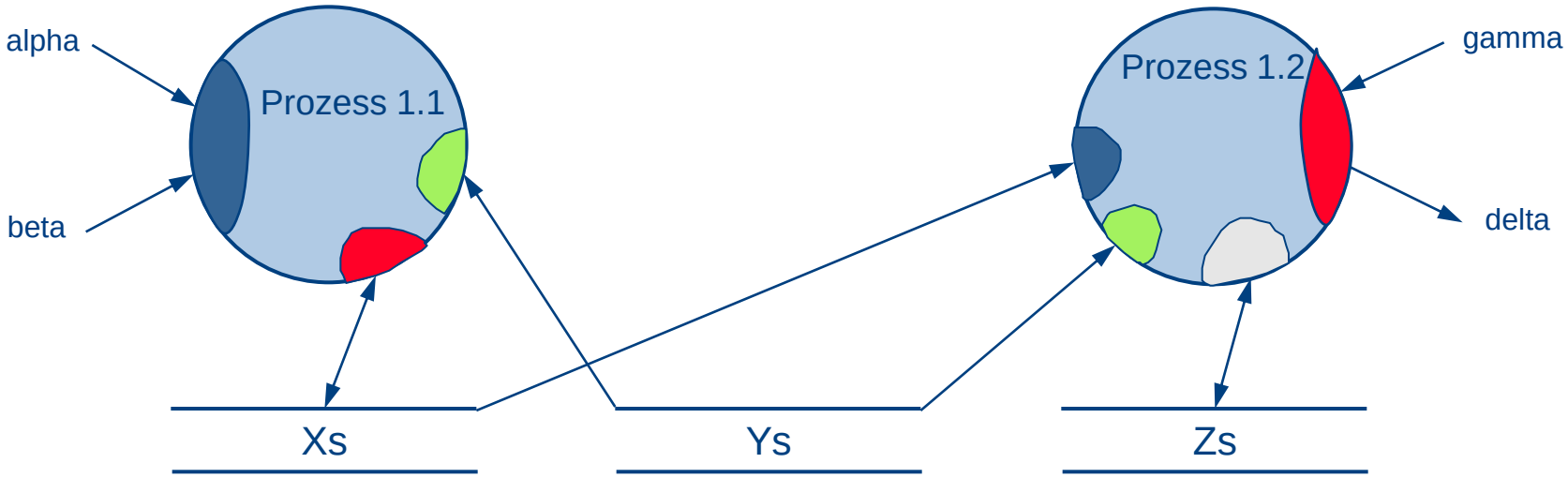
- Transformation von DFD-Elementen in abstrakte Datenobjekte

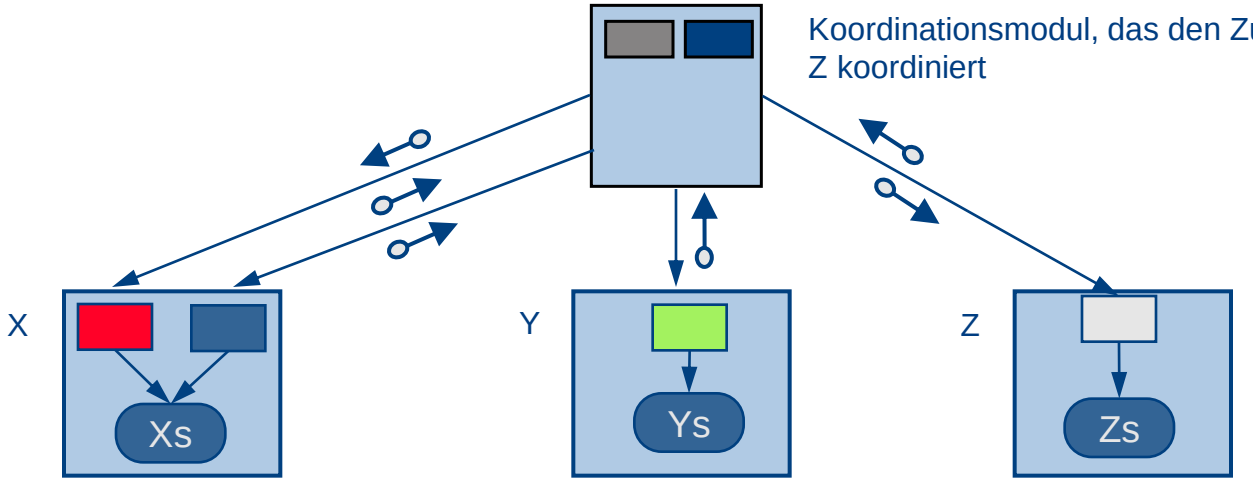


Transformation eines SA-Modells

- Koordinationsmodul
  - Die Abbildung der Datenflüsse auf funktionale Abstraktionen ist oft nicht trivial
  - Daher werden manchmal mehrere Prozesse zu einem Koordinationsmodul zusammengefasst

Transformation eines SA-Modells
Einführung eines Koordinationsmoduls



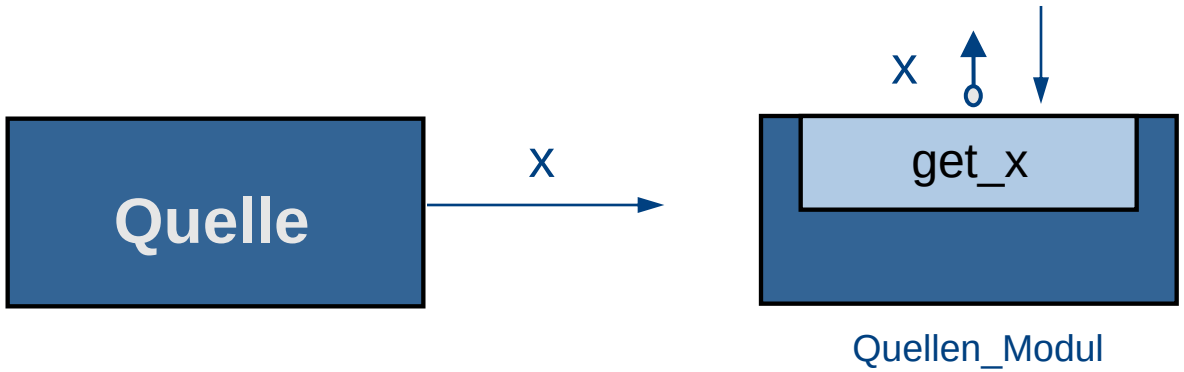


Koordinationsmodul, das den Zugriff auf X, Y, und Z koordiniert

Transformation eines SA-Modells

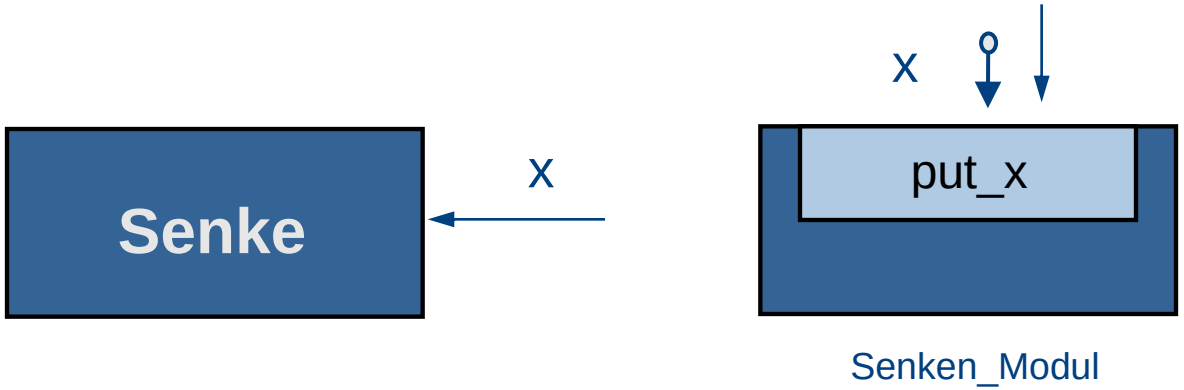
- Anhaltspunkte des SA-Modells für den Entwurf
  - DFDs können für die Transformation in funktionale Module und Datenabstraktions-Module ausgewertet werden
  - Aus dem DD können die Datenstrukturen spezifiziert werden
  - Minispecs können die Spezifikation der Zugriffsoperationen ergeben

- Transformation 1: Schnittstelle, die als Quelle dient

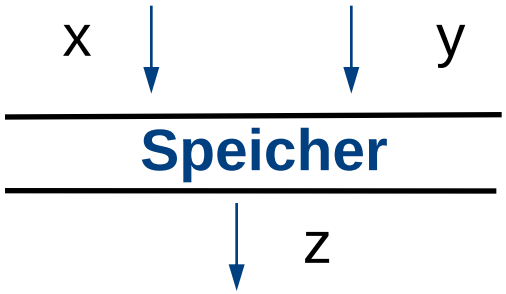


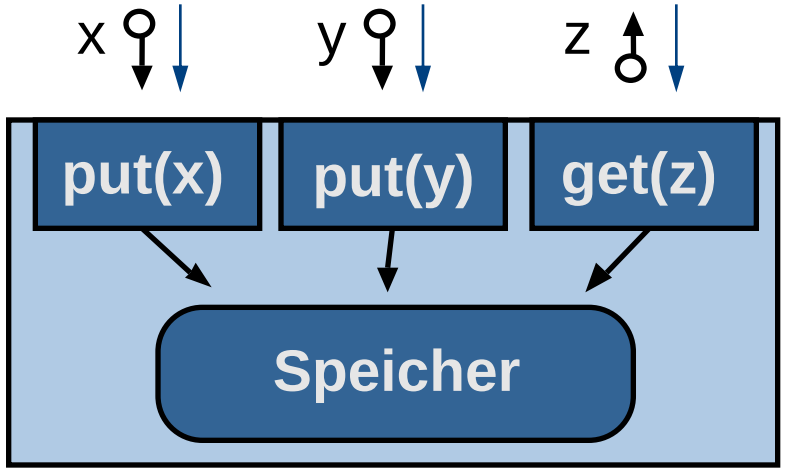
get_x ist eine Funktion, die einen Wert zurückgibt

- Transformation 2: Schnittstelle, die als Senke dient



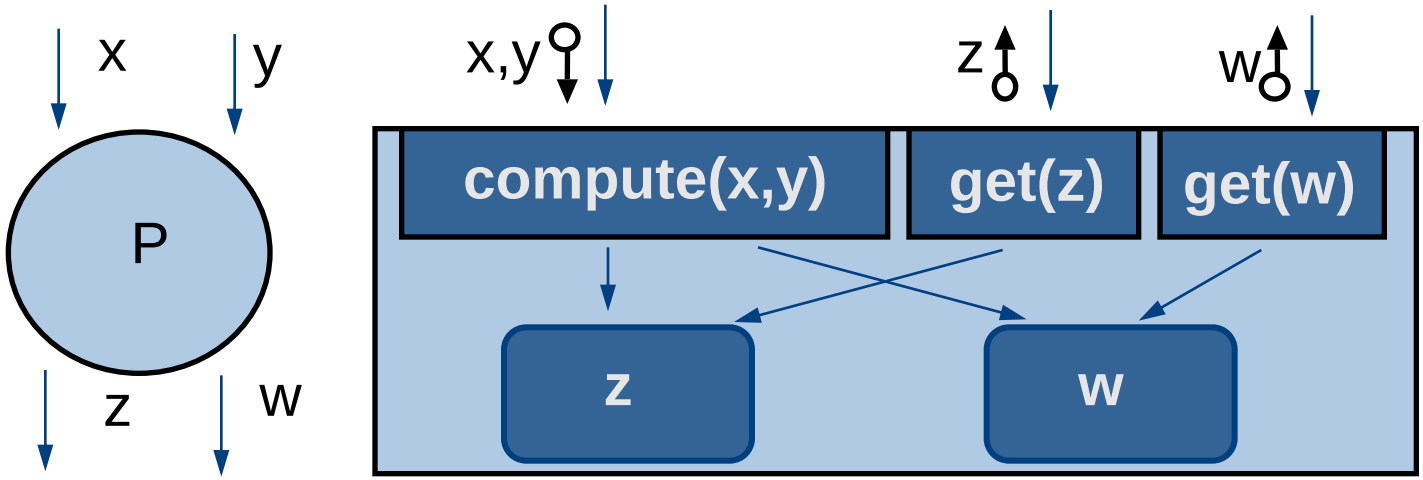
- Transformation 3: Speicher





Speicher_Modul

- Transformation 4: Prozess



compute(x,y) verändert die internen Gedächtnisse z und w

Prozess_Modul

Transformation eines SA-Modells

- Die Transformationen können in beliebiger Reihenfolge durchgeführt werden, da sie orthogonal zueinander sind
- Die entstehende Architektur besteht aus unverbundenen Modulen
- Die Architektur kann nun manuell durch ein Steuerungsmodul ergänzt werden, das die Ausführung des DFD simuliert, indem es die Module aufruft