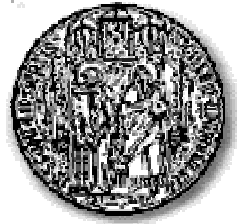


Vorlesung Softwaretechnik

Prof. Dr.-Ing.habil. Dipl.Math. Klaus-Peter Fähnrich
(Wintersemester 2001/2002)

Universität Leipzig
Institut für Informatik





14. Die Entwurfsphase: OOD-Architekturentwurf

SM & MD

1. OOD-Architekturentwurf

1. Anwendungskategorien

2. Plattformen

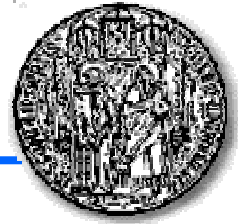
3. Entwurf der Fachkonzeptschicht

4. Entwurf der GUI und Anbindung an die Fachkonzept-Klassen

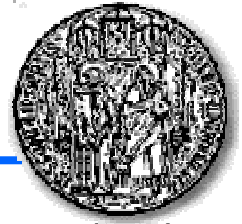
5. Entwurf der Anbindung an die Datenhaltung

2. SD & MD

14.1. Die Entwurfsphase: OOD-Architekturentwurf

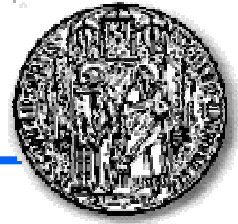


- 1. Änderungen am OOA-Modell, die für die Transformation in den Entwurf notwendig sind**
- 2. ODBC und JDBC**
- 3. GUI-Klassen für Erfassungs- und Listenfenster**
- 4. Verbindung von GUI-Klassen mit Fachkonzept-Klassen**
- 5. Singleton- und Beobachter-Muster**
- 6. Überführung einer Klassenhierarchie in eine flache Tabellenstruktur**
- 7. Konzepte zur Realisierung von nicht-funktionalen Anforderungen.**



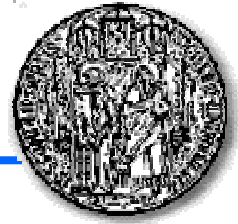
14.1. Die Entwurfsphase: OOD-Architekturentwurf

- Start mit OOA
 - OOA-Modell wird erweitert, modifiziert, optimiert und an die umgebende Architektur angepasst
 - Es ergibt sich ein OOD-Modell
- Wiederverwendung
 - Vorteil der OO-Softwareentwicklung : Ihre Konzepte – insbesondere das Vererbungskonzept – unterstützen die Wiederverwendung vorhandener Klassen und Pakete
 - Beim gesamten Entwicklungsprozess ist das Suchen nach wiederverwendbaren Teilen und das Ablegen wiederverwendbarer Teile wesentlich.
- Wahl der Programmiersprache
 - Ein fertiges OOD-Modell muss in der Implementierungsphase realisiert werden
 - Die verwendete Sprache hat daher u.U. massive Rückwirkungen auf den Architekturentwurf
 - Aus heutiger Sicht sollte immer eine OO-Sprache (Java,C++) gewählt werden
- Komponentenmodelle
 - Die Wahl der Sprache beeinflusst auch die möglichen Komponentenmodelle
- Interdependenzen : Vielfältige Abhängigkeiten bei den Entscheidungen, so dass eine sorgfältige Analyse erforderlich ist.
- Mehrere Programmiersprachen
 - Bei komplexen Unternehmenslösungen ist oft auch der Einsatz mehrerer Sprachen sowie die Einbindung von Altsystemen mit deren Sprachen erforderlich
 - Insbesondere bei Web-Architekturen werden zusätzlich noch Skript-Sprachen verwendet



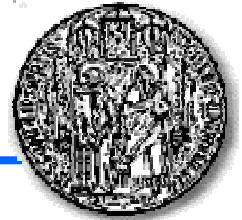
14.1.1. Die Entwurfsphase: Anwendungskategorien

- Der Architekturentwurf wird wesentlich von der Anwendungs-Kategorie determiniert, in die die zu entwickelnde Anwendung fällt
- Anwendungen lassen sich einer oder mehreren der folgenden Kategorien zuordnen:
- Desktop-Anwendungen
 - o Erlauben eine relativ einfache Systemarchitektur
 - ❖ Sie werden auf einem einzelnen Computersystem installiert
 - ❖ Es arbeitet meist nur ein Benutzer mit ihnen
 - o Drei-Schichten-Architektur erforderlich
 - o Abnehmende Bedeutung
 - ❖ Der anonyme Massenmarkt ist die traditionelle Domäne von Desktop-Anwendungen
 - ❖ Er wird zum großen Teil von wenigen großen Software-Häusern dominiert
 - ❖ Hinzu kommt, dass mit Software für Privat-Personen immer weniger Geld zu verdienen ist.
- Klassische Client/Server-Anwendungen
 - o Je eine eigenständige Anwendung auf dem Client und auf dem Server, die über ein Netzwerk miteinander kommunizieren
 - o Nutzen des jeweiligen Betriebssystems
 - ❖ Client-Seite dominiert derzeit Windows
 - ❖ Im Server-Bereich Windows und UNIX-Derivate
 - o TCP/IP-Protokoll
 - o Oft ist es daher besser, eine (komponentenbasierte) Verteilungs-Plattform einzusetzen (CORBA, EJB, COM+).



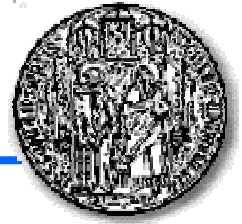
14.1.1. Die Entwurfsphase: Anwendungskategorien

- Web-Anwendungen
 - o Immer beliebter
 - o Web-Server ohnehin meist vorhanden
 - o Internet soll zur Verbesserung der eigenen Geschäftsprozesse genutzt werden, z.B. für die Kommunikation mit Händlern und Kunden
 - o Web-Browser ist Mitarbeitern & Kunden bekannt
 - o Hemmschwelle zur Nutzung ist dadurch geringer
 - o Bei kleinem Funktionsumfang
 - Serverseitige Web-Konzepte (z.B. Servlets, JSP, ASP) und einem DBS ergeben gute, skalierbare und leicht wartbare Anwendungen
 - o Steigender Funktionsumfang
 - Einsatz einer Verteilungsplattform.
- Angepasste Standard-Software
 - o Microsoft Office
 - ❖ Entwicklungsplattform für neue Anwendungen
 - o Baukastenprinzip
 - ❖ Fertige Einzelteile werden zu einer fertigen Anwendung zusammengesetzt
 - ❖ Der Architekturentwurf wird in diesem Fall sehr stark durch die Struktur von Office geprägt
 - o Nur Plattformen Windows und Apple Macintosh
 - o Kostenreduktion
 - o Verteilungsplattform
 - o Gefahr groß, ohne Analyse und Entwurf einfach »drauf los zu hacken«.



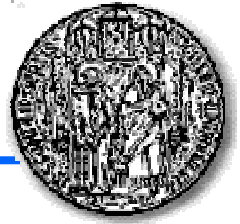
14.1.1. Die Entwurfsphase: Anwendungskategorien

- Mischformen
 - o Für jeden Teilbereich kann der jeweils beste Lösungsansatz verwendet werden:
 - o Erfassung von Massendaten : Klassische Client-Anwendung, da die Bedienungsgeschwindigkeit oder eine vollständige Tastaturbedienung eine wichtige Rolle spielen
 - o Kundenzugriff über das Internet auf firmeninterne Daten (Web-Anwendung)
 - o Um Adressen aus der unternehmensweiten Kundendatenbank automatisch in einen Musterbrief einzubetten : Keine eigene Textverarbeitung, sondern passt z.B. Word entsprechend an.



14.1.2. Die Entwurfsphase: Plattformen

- Einziger Anbieter, der für alle Kategorien Produkte anbietet und auch ein DBS im Programm hat
- Grundsätzlich können alle Produkte »aus einer Hand« gekauft werden
- + Alle Produkte (inklusive des Betriebssystems) sind relativ gut aufeinander abgestimmt
- + Inkompatibilitäten treten seltener auf
- + Außerdem bietet Microsoft eine gute technische Unterstützung für alle Produkte an und ist schon aus eigenem Interesse an einer leichten Integration seiner Produkte interessiert.
- Microsoft geht in einigen Bereichen eigene Wege
- Java spielt bei Microsoft nur eine sehr untergeordnete Rolle
 - o EJBs, JavaBeans oder Servlets werden durch hauseigene Konzepte wie COM+, ActiveX und ASP ersetzt
- Außerdem ist man von einem einzelnen Hersteller und dessen Produkt- und Preispolitik abhängig
- In eine reine Microsoft-Lösung später Produkte anderer Hersteller zu integrieren, kann sehr schwierig werden.

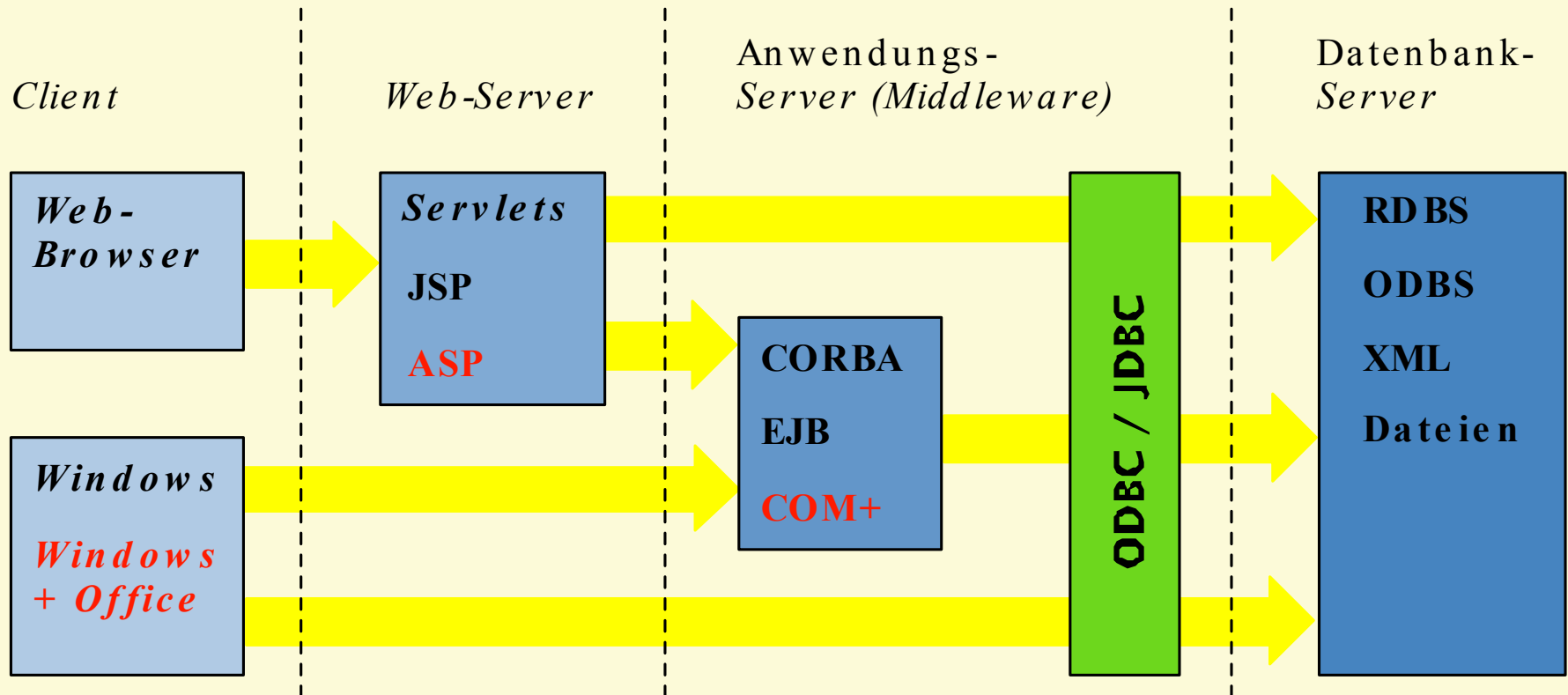


14.1.2. Die Entwurfsphase: Plattformen

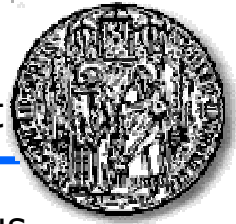
- Verschiedene Hersteller
 - o Alternative: Mischung aus Produkten und Konzepten verschiedener Hersteller
 - o Java hat sich als Sprache inzwischen fest etabliert
 - o Standards auf Java: EJBs und Servlets
- Middleware
 - o Immer mehr Standards, die auf XML-basierten Sprachen beruhen
 - ❖ z.B. X-EDI für B2B-Anwendungen
 - o XML: Von Microsoft und von den meisten anderen Herstellern als Meta-Sprache favorisiert
- Offene Standards
 - o Trend immer mehr zu offenen Standards.



14.1.2. Die Entwurfsphase: Plattformen

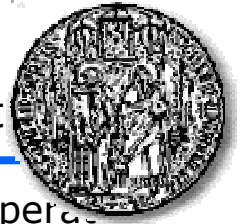


14.1.3. Die Entwurfsphase: Entwurf der Fachkonzept-Schicht



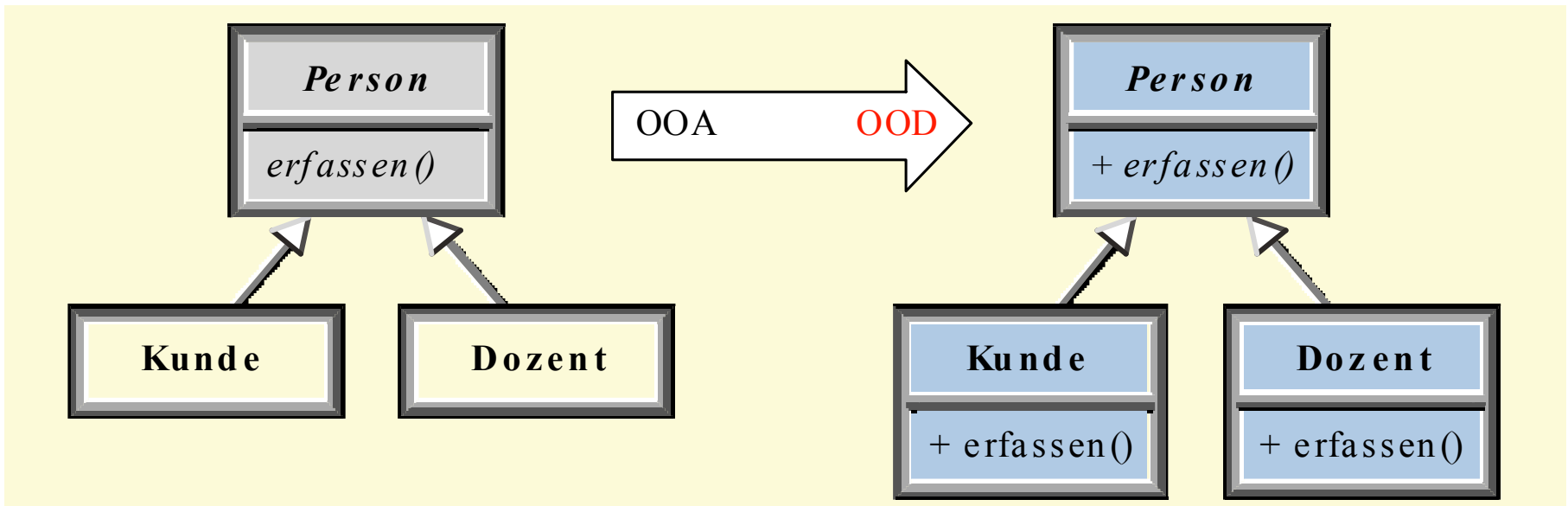
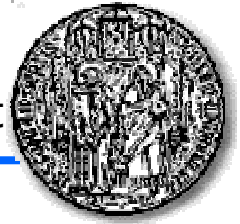
- Architekturentwurf für eine Desktop-Anwendung sieht folgendermaßen aus
 - o Ausgangspunkt OOA-Modell : 1. Version der Fachkonzeptschicht, die unter den Aspekten des Entwurfs verfeinert und überarbeitet wird
- 1. Modifikation der Klassenstruktur
 - o Objektverwaltung durch Container-Klassen
 - o Analyseklassen 1:1 in die Fachkonzeptschicht
 - ❖ Funktionale Komplexität einer Klasse zu hoch → Teilaufgaben an detailliertere Klassen delegieren
 - ❖ Erreichen der Performance → Klassen mit starker Interaktion – d.h. mit einer hohen Kopplung – zusammenfassen.
 - ❖ Hinzufügen weiterer Klassen (z.B. um Zwischenergebnisse zu modellieren, d.h. mehrere abgeleitete Attribute in einer neuen Klasse zu »bündeln«).
 - ❖ Assoziative Klassen in »normale« Klassen auflösen
- 2. Verfeinern der Attribute
 - o Für »abgeleitet« Attribute prüfen, ob die Werte zu speichern sind oder ob sie jeweils aktuell berechnet werden sollen
 - ❖ Handelt es sich um viele Attribute, so ist es sinnvoll, eine neue Klasse dafür in das Modell einzufügen.
- 3. Verfeinern der Operationen
 - o Spezifizierte Operationen sind aus Entwurfssicht detaillierter zu beschreiben
 - o Komplexe Operationen sind in Teiloperationen zu gliedern

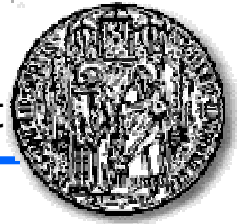
14.1.3. Die Entwurfsphase: Entwurf der Fachkonzept-Schicht



- o Besitzt die Klasse einen Lebenszyklus, so ist eine auszuführende Operation von dem jeweiligen Objektzustand abhängig
 - ❖ Dann muss der Algorithmus entsprechende Abfragen enthalten oder es ist das Zustandsmuster anzuwenden.
- 4. Verfeinern von Assoziationen
 - o Prüfen, ob eine Navigationsrichtung ausreicht (Richtung im Klassendiagramm durch einen Pfeil kennzeichnen)
 - o Assoziationen unter dem Gesichtspunkt des optimalen Zugriffs auf Objekte modellieren
 - o Für jede Operation prüfen, welche Assoziationen sie »durchlaufen« muss, um an die benötigten Informationen zu gelangen
 - o Beispiel : Assoziation zwischen Klassen A & B wird nur von A nach B als Zeiger in A implementiert
 - ❖ Bei einem Zugriff in der Gegenrichtung müssen alle Objekte von A betrachtet und gefiltert werden.
- 5. Verfeinern der Vererbung
 - o Beispiel
 - ❖ Im OOA-Modell: erfassen() gilt für alle Objekte ihrer Unterklassen
 - ❖ Im OOD-Modell: Erfassen bei beiden Unterklassen unterschiedlich. Jede Unterklasse enthält diese Operation

14.1.3. Die Entwurfsphase: Entwurf der Fachkonzept-Schicht



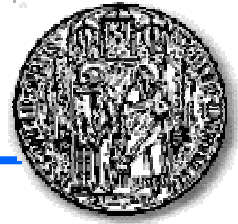


14.1.3. Die Entwurfsphase: Entwurf der Fachkonzept-Schicht

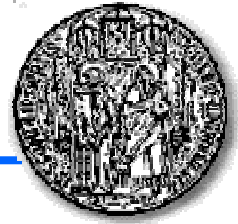
- Ergebnis: OOD-Klassendiagramm
 1. Modifikation der Klassenstruktur
 - o Hinzufügen von Container-Klassen und Zerlegen komplexer Klassen
 - o Zusammenfassen von Klassen mit starker Interaktion
 - o Hinzufügen von Klassen zum Modellieren von Zwischenergebnissen
 - o Transformation von assoziativen Klassen in »normale« Klassen
 2. Verfeinern der Attribute
 - o Abgeleitete Attribute des OOA-Modells übernehmen oder in Operationen wandeln
 - o Neue abgeleitete Attribute einführen.
 3. Verfeinern der Operationen
 - o Operationen präzisieren
 - o Komplexe Operationen in einfachere, interne Operationen zerlegen
 - o Transformieren einfacher Lebenszyklen in Algorithmen
 - o Transformieren komplexer Lebenszyklen mittels Zustandsmuster
 4. Verfeinern von Assoziationen
 - o Navigationsrichtung bei allen Assoziationen prüfen und unidirektionale Assoziationen durch Pfeile kennzeichnen
 - o Zugriffspfade optimieren.
 5. Verfeinern der Vererbungsstruktur
 - o Abstrakte Operationen für einheitliche Schnittstellen hinzufügen
 - o Hinzufügen von abstrakten Oberklassen
 - o Maximierung des Polymorphismus
 - o Komprimieren von Vererbungsstrukturen
 - o Wiederverwenden existierender Klassen

Checkliste Entwurf der Fachkonzeptschicht

14.1.4. Die Entwurfsphase: Entwurf der GUI-Schicht und ...

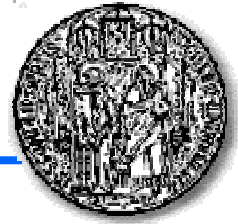


- Abgrenzung Fachkonzept – GUI
 - o Faustregel: Alle Objekte, die unabhängig von dem verwendeten GUI dargestellt werden müssen, zählen zum Fachkonzept
- GUI-Schicht
 - o Die Architektur der Benutzungsoberfläche wird durch das verwendete GUI-System geprägt
 - o Eine GUI-Bibliothek besteht meistens aus einem oder mehreren größeren Bäumen
 - Für jedes Interaktionselement gibt es eine Blattklasse
 - Für die Fenster gibt es eine Oberklasse, von der dann die individuellen Fenster abgeleitet werden.
- Elementare Klassen
 - o Grenze zwischen Architekturklassen und elementaren Klassen sorgfältig ziehen
 - o Architekturklassen werden in das Klassendiagramm eingetragen
 - o Elementare Klassen werden im Sinne von Attributtypen behandelt
 - o Architekturklassen : Alle Klassen, die Fenster realisieren
 - o Elementare Klassen : Klassen, die Interaktionselemente implementieren.



14.1.4. Die Entwurfsphase: Entwurf der GUI-Schicht und ...

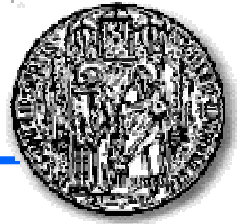
- Bei umfangreicher Anwendung
 - Trennung von GUI- und Fachkonzeptschicht durch eine Zugriffsschicht
- Fassade ist ein Entwurfsmuster, das ein Subsystem hinter einer einfachen Schnittstelle verbirgt
- Die GUI-Klasse kennt dann die Typen der Fachkonzept-Klasse nicht
- GUI-Schicht ist vollständig von der Fachkonzeptschicht isoliert
- Eine Fassade hat folgende Aufgaben:
 - Sie reagiert auf Eingaben und andere Ereignisse
 - Sie öffnet Fenster, die Informationen der Fachkonzeptobjekte darstellen.
 - Sie verwaltet Transaktionen, z.B. commit und rollback
- Sollen die Informationen der Anwendung in verschiedenen Fenstern angezeigt werden (dependent windows), dann führt die Fassade noch weitere Aufgaben durch:
 - Sie ermöglicht es, in mehreren Fenstern gleichzeitig die Informationen eines Fachkonzeptobjekts anzuzeigen
 - Sie informiert abhängige Fenster, wenn sich die Information ihres Fachkonzeptobjekts geändert hat und eine Aktualisierung der Darstellung nötig ist.



14.1.4. Die Entwurfsphase: Entwurf der GUI-Schicht und ...

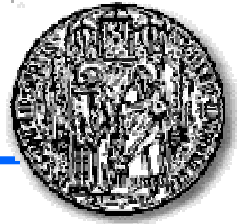
- Checkliste: Entwurf der GUI-Schicht und Anbindung an die Fachkonzeptschicht
- Ergebnisse: OOD-Klassendiagramm, Sequenzdiagramme
- GUI-System auswählen (Java-GUI mit Swing)
- Jedes GUI-Fenster: Unterklasse von JDialog
- GUI-Klasse für Erfassungsfenster enthält:
 - einfache Assoziation zur Fachkonzept-Klasse
 - eine einfache Assoziation zur Container-Klasse
 - die Operation `speichere()` zum Übergeben der Werte an die Fachkonzept-Klasse
 - die Operation `aktualisiere()` zum Anzeigen der fachlichen Werte im Erfassungsfenster.
- GUI-Klasse für Listenfenster enthält:
 - eine einfache Assoziation zur Container-Klasse (`alleObjekte`)
 - die Operation `aktualisiere()` zum Anzeigen aller Objekte dieser Klasse
- Container-Klasse (Fachkonzept-Klasse), von der es nur ein Exemplar gibt (Singleton-Muster), enthält:
 - das Klassenattribut `einzigesObjekt`, das die Referenz auf das einzige Objekt enthält
 - die Klassenoperation `getObjektreferenz()`, die auf diese Referenz zugreifen kann und beim ersten Aufruf das Objekt erzeugt.

Checkliste Anbindung der GUI-Schicht an der Fachkonzept-Schicht (1)



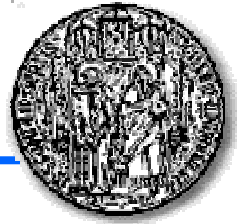
14.1.4. Die Entwurfsphase: Entwurf der GUI-Schicht und ...

- Container-Klasse informiert ihre GUI-Listenklassen (Beobachter) mittels Beobachter-Muster und enthält:
 - eine *-Assoziation zur GUI-Listenklasse (Beobachter)
 - die Operation meldeAn(), die eine Verbindung zu einem Beobachter-Objekt aufbaut
 - die Operation meldeAb(), die eine Verbindung zu einem Beobachter-Objekt abbaut
 - die Operation benachrichtige(), die alle Beobachter über eine Veränderung benachrichtigt
- Stärkere Entkopplung von GUI- und Fachkonzeptschicht durch Fassadenklassen.



14.1.5. Die Entwurfsphase: Entwurf der Anbindung an die ...

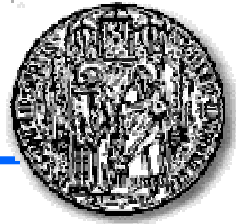
- Anbindung an die Datenhaltung hängt im Wesentlichen davon ab, ob die Datenhaltung durch...
 - o eine objektorientierte Datenbank
 - o eine relationale Datenbank
 - o ein Dateisystemerfolgt.
- Anbindung an ein OO-DBS
 - o Anbindung an die Fachkonzeptschicht
 - o Relativ einfach
 - o Exemplarisch : ODBS Poet und seine Java-Sprachanbindung
 - o Alle Klassen, deren Objekte persistent gemacht werden sollen, müssen in der Konfigurationsdatei ptjavac.opt aufgeführt werden
 - o Typkonvertierung
 - ❖ Im Übergang von der Analyse zum Entwurf sich auf die Datentypen des ODMG-Standards beschränken
 - ❖ Erspart aufwändige Typkonvertierungen
 - ❖ Oftmals erzwingt Einsatz von Bibliotheken, z.B. die MFC, den Einsatz nicht-konformer ODMG-Typen
 - ❖ Konvertierung der Typen – wenn möglich – durchgängig in der Schicht vorgenommen werden, die diese Sondertypen benutzt (im MFC-Fall also die GUI-Schicht) , um die Auswirkungen auf die anderen Schichten möglichst gering zu halten.



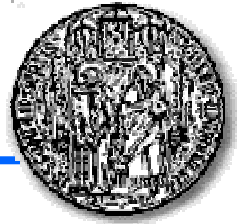
14.1.5. Die Entwurfsphase: Entwurf der Anbindung an die ...

- Anbindung an ein RDBS
 - o Hauptaufgabe bei der Anbindung an eine relationale oder objektrelationale Datenbank
 - ❖ Abbildung des Objektmodells auf Tabellen
 - ❖ Jede Tabelle wird – unabhängig davon, ob ein fachliches Schlüsselattribut vorhanden ist – um ein OID-Attribut erweitert, das die Rolle des Schlüsselattributs spielt
 - o Typen
 - ❖ Sollen objektorientierte Systeme mit relationalen Datenbanken arbeiten, dann ist die Abbildung der OOA-Datentypen auf SQL-Datentypen notwendig.

14.1.5. Die Entwurfsphase: Entwurf der Anbindung an die ...



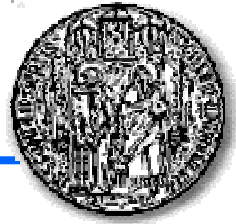
- SQL
 - o Nur Format und Semantik von Abfragen spezifiziert
 - o Um Daten von einem DB-Server abfragen zu können, muss eine Verbindung zwischen Client und Server aufgebaut werden
 - o Ergebnisse einer Anfrage werden immer als Mengen von Datensätzen zur Verfügung gestellt
- Proprietäre Schnittstellen
 - o DB-Hersteller bieten produktspezifische und damit proprietäre Schnittstellen an, um den Zugriff auf technischer Ebene zu ermöglichen.
- Embedded SQL
 - o Erweiterung einer Sprache um spezielle Konstrukte
 - o Sie ermöglichen es, direkt aus der Anwendung über SQL-Befehle auf die Datenbank zuzugreifen
 - o Ein Präprozessor wandelt die SQL-Befehle in Aufrufe von Bibliotheksprogrammen um
 - o Das so entstandene Programm kann dann mit einem herkömmlichen Compiler übersetzt werden
 - o Damit ist die Abbildung des mengenorientierten SQL auf gewöhnliche Sprachen leichter geworden
 - o Jedoch auch eine proprietäre Schnittstelle, denn jeder Hersteller unterstützt seine eigene Embedded SQL-Implementierung.



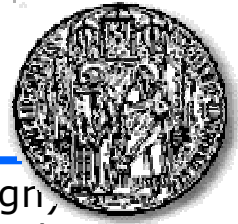
14.1.5. Die Entwurfsphase: Entwurf der Anbindung an die ...

- ODBC (Open Database Connectivity)
 - o Standardisierte Schnittstelle für Zugriff auf RDBS
 - o Ursprung Microsoft, inzwischen aber betriebssystemübergreifender de facto-Standard
 - o Zugriff auf jede Datenbank, für die ein ODBC-Treiber existiert
 - o Spezifiziert ein API, das Funktionen zum Öffnen und Verwalten einer DB-Verbindung, zum Senden von Anfragen und zur Auswertung der gelieferten Daten zur Verfügung stellt
 - o Für die Anfragen stützt sich ODBC auf eine standardisierte SQL-Version
 - o Die technische Realisierung der Verbindung wird durch einen ODBC-Treiber für ein bestimmtes Datenbanksystem realisiert
 - o Anwendung kann dadurch unabhängig von einem bestimmten Datenbanksystem entwickelt werden
- ADO (ActiveX Data Objects)
 - o ODBC: funktionales API
 - OO- Anwendung muss selbst die Abbildung auf API-Funktionen vornehmen
 - o OO-Schalen um ODBC liefert Microsoft in Form des COM-basierten ADO
 - o Nur auf Windows-Plattformen verfügbar.

14.1.5. Die Entwurfsphase: Entwurf der Anbindung an die ...



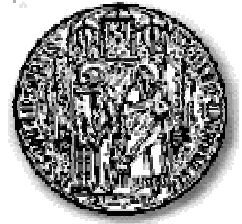
- JDBC (Java Database Connectivity)
 - o Sun hat einen Standard definiert, um aus Java-Programmen heraus auf RDBS zugreifen zu können
 - o JDBC ist vollständig objektorientiert und plattformunabhängig
 - o Wie bei ODBC realisiert ein Treiber den technischen Zugriff auf ein konkretes DBS
 - o JDBC-ODBC-Bridge ermöglicht es, JDBC auf ODBC aufzusetzen.
- XML
 - o Zugriff über die Kombination HTTP–XML
 - o Eine DB-Anfrage wird über HTTP an den DB-Server geschickt
 - o Der DB-Server stellt das Ergebnis als XML-Dokument zur Verfügung und schickt dieses an den Client zurück
 - o Die Struktur des XML-Dokuments wird dabei durch die Anfrage festgelegt
 - o Diese Art der Kommunikation mit Datenbanken wird in Zukunft wahrscheinlich weiter zunehmen
 - o Insbesondere, wenn es mehr DBS gibt, die für die Speicherung großer Mengen von XML-Dokumenten optimiert sind.



14.1. Die Entwurfsphase: OOD-Architekturentwurf

- Am Anfang eines objektorientierten Entwurfs (OOD, object oriented design), muss geprüft werden, zu welcher Anwendungskategorie das zu entwerfende Produkt gehört und auf welchen Plattformen es laufen soll. Bei den Plattformen stellt sich die Frage, ob sie bereits vorhanden sind oder ob mit dem Produkt auch die Plattformen einzuführen sind. Im zweiten Fall ist sorgfältig zu überlegen, welche Plattformen verwendet werden, da diese Entscheidung das Umfeld für weitere Produkte bestimmt.
- In Abhängigkeit von der Anwendungskategorie und den Plattformen ist dann der Architekturentwurf durchzuführen.
- Bei einer Desktop-Anwendung sind folgende Aktivitäten durchzuführen.
 - o Entwurf der Fachkonzept-Schicht: ausgehend vom OOA-Modell wird ein OOD-Modell erstellt wobei (unter Entwurfsgesichtspunkten) das OOA-Modell verfeinert und überarbeitet wird.
 - o Entwurf der GUI-Schicht und Anbindung an die Fachkonzeptklassen: Nach Auswahl des GUI-Systems müssen die GUI-Klassen entworfen und durch unidirektionale Assoziationen an die Fachkonzept-Klassen angebunden werden.
 - o Entwurf der Anbindung an die Datenhaltung : Die Datenhaltungsschicht wird bestimmt durch die Wahl des Datenbankmodells. Bei einer objektorientierten Datenbank sind nur geringe Anpassungsarbeiten vorzunehmen, während beim Einsatz einer relationalen DB eine objektrelationale Abbildung vorzunehmen ist. Außerdem ist zu prüfen, ob über ODBC, JDBC oder XML auf die DB zugegriffen werden soll

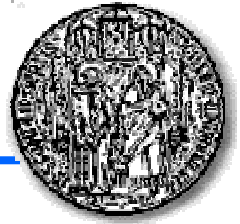
Zusammenfassung



14. Die Entwurfsphase: OOD-Architekturentwurf

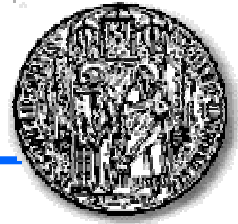
SM & MD

- 1. OOD-Architekturentwurf**
- 2. SD & MD**
 - 1. Strukturierter Entwurf**
 - 1. Funktionale Abstraktion**
 - 2. Strukturdiagramme**
 - 3. Strukturierte Software-Architektur**
 - 4. Transformation von SA nach SD**
 - 2. Modularer Entwurf**
 - 1. Datenabstraktion**
 - 2. Abstrakte Datenobjekte**
 - 3. Abstrakte Datentypen**
 - 4. Algebraische Spezifikation**
 - 5. Modulare Entwurfsmethoden**
 - 6. Modularer Entwurf vs. Objektorientierter Entwurf**



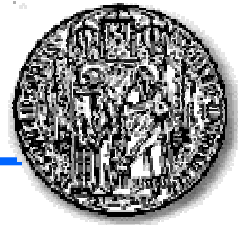
14.2. Die Entwurfsphase: SD & MD

- 1. Notation und Symbolik von Strukturdiagrammen**
- 2. Eigenschaften funktionaler Abstraktionen und funktionaler Modulen**
- 3. Charakteristika des strukturierten Entwurfs**
- 4. Unterschiede zwischen funktionaler Abstraktion und Datenabstraktion**
- 5. Algebraische Spezifikation**
- 6. Top-down und bottom-up-Entwurf**
- 7. Unterschiede zwischen dem modularen und dem OOD**
- 8. Strukturierung von Entwürfen durch Strukturdiagrammen**



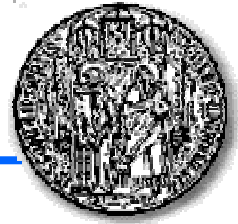
14.2.1. Die Entwurfsphase: Strukturierter Entwurf

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



14.2.1.1. Die Entwurfsphase: Funktionale Abstraktion

- Entwurf eines Software-Systems
 - o Reduzierung der Problemkomplexität durch Wahl geeigneter Abstraktionen
- Funktionale Abstraktion
 - o Stellt eine Leistung in Form einer abstrakten
 - ❖ Funktion
 - ❖ Operation oder
 - ❖ Prozedur zur Verfügung
 - o Historisch erste Form der Abstraktion
 - o Seit 1954 in Fortran, später in allen klassischen Programmiersprachen vorhanden
 - o Oft operationale oder prozedurale Abstraktion genannt.
- Beispiel
 - o Berechnung der Quadratwurzel über Newtonsche Iterationsformel
 - ❖ $y_1 = (y_0 + x/y_0) / 2$
 - ❖ y_0 = gegebener Näherungswert für x
 - ❖ x reell , $x \geq 0$, $y_0 \geq 0$
 - o In vielen Problemen benötigt
 - ❖ In den meisten Programmiersprachen Standardfunktion (C: `sqrt(double)`)
 - o Steht diese Funktion nicht zur Verfügung, dann schreibt man ein Programm in Form einer Funktion.

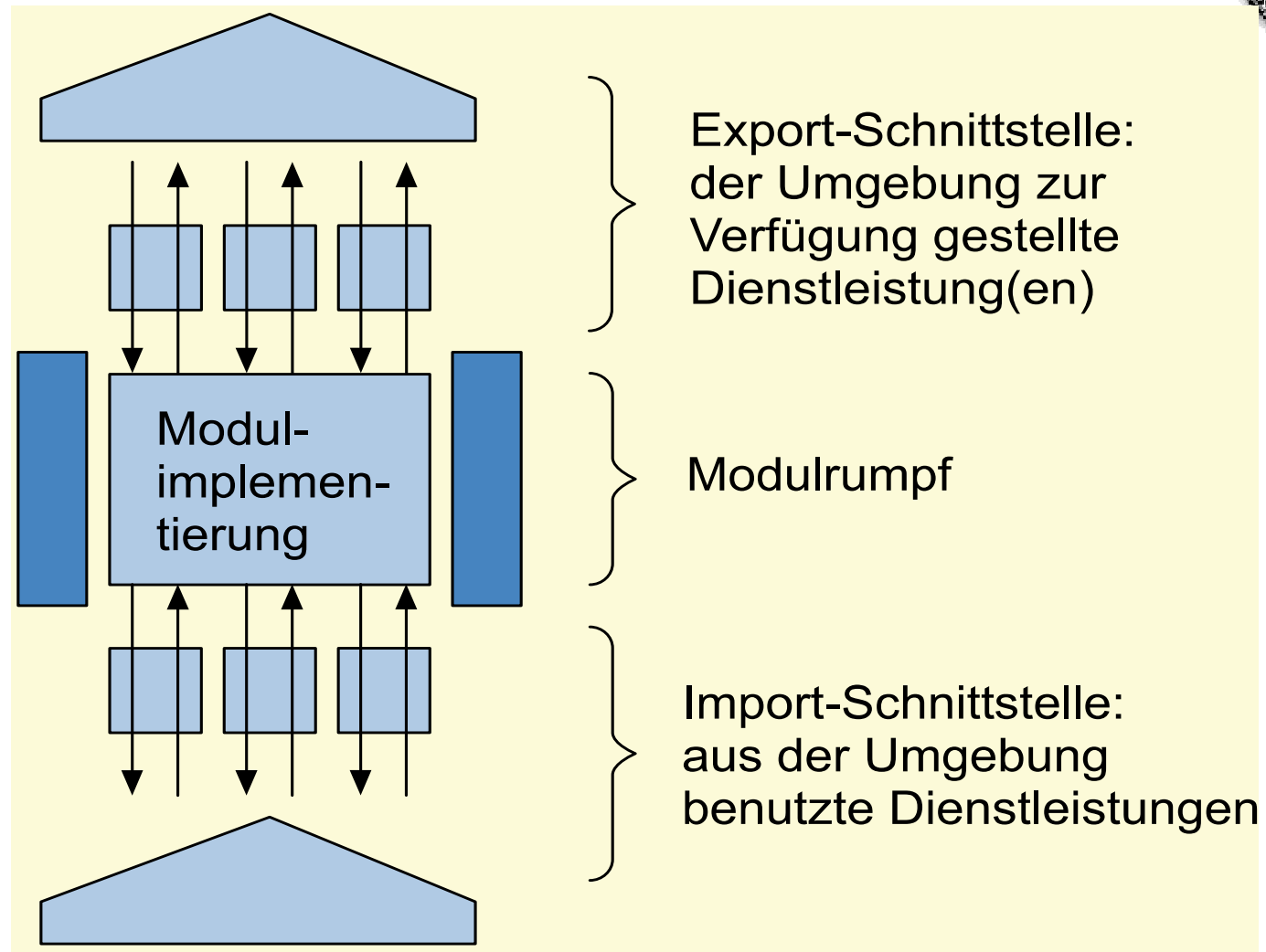


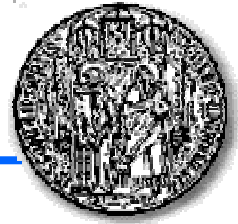
14.2.1.1. Die Entwurfsphase: Funktionale Abstraktion

- Realisiert eine funktionale Abstraktion
- Eigenschaften
 - o Aktiv bzw. aktionsorientiert, es »tut« etwas
 - o Besitzt ein Transformationsverhalten
 - ❖ Eingabedaten werden in Ausgabedaten transformiert
 - o Identische Eingabedaten führen immer zu identischen Ausgabedaten
 - ❖ Kein internes Gedächtnis.
- Aufgaben
 - o Steuerungs- und Koordinationsaufgaben
 - ❖ Beispiel: Hauptprogramm
 - o Transformationsaufgaben unterschiedlicher Komplexität
 - ❖ Beispiel: Compiler
 - o Auswertungsaufgaben, die man als Spezialfall von Transformationen ansehen kann
 - ❖ Beispiel: mathematische Routinen
 - o Hilfsaufgaben.
- Ausprägungen
 - o Funktionales Modul = Funktionale Abstraktion
 - o Funktionales Modul = mehrere, logisch, d.h. semantisch, zusammengehörende funktionale Abstraktionen



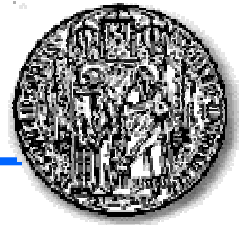
14.2.1.1. Die Entwurfsphase: Funktionale Abstraktion





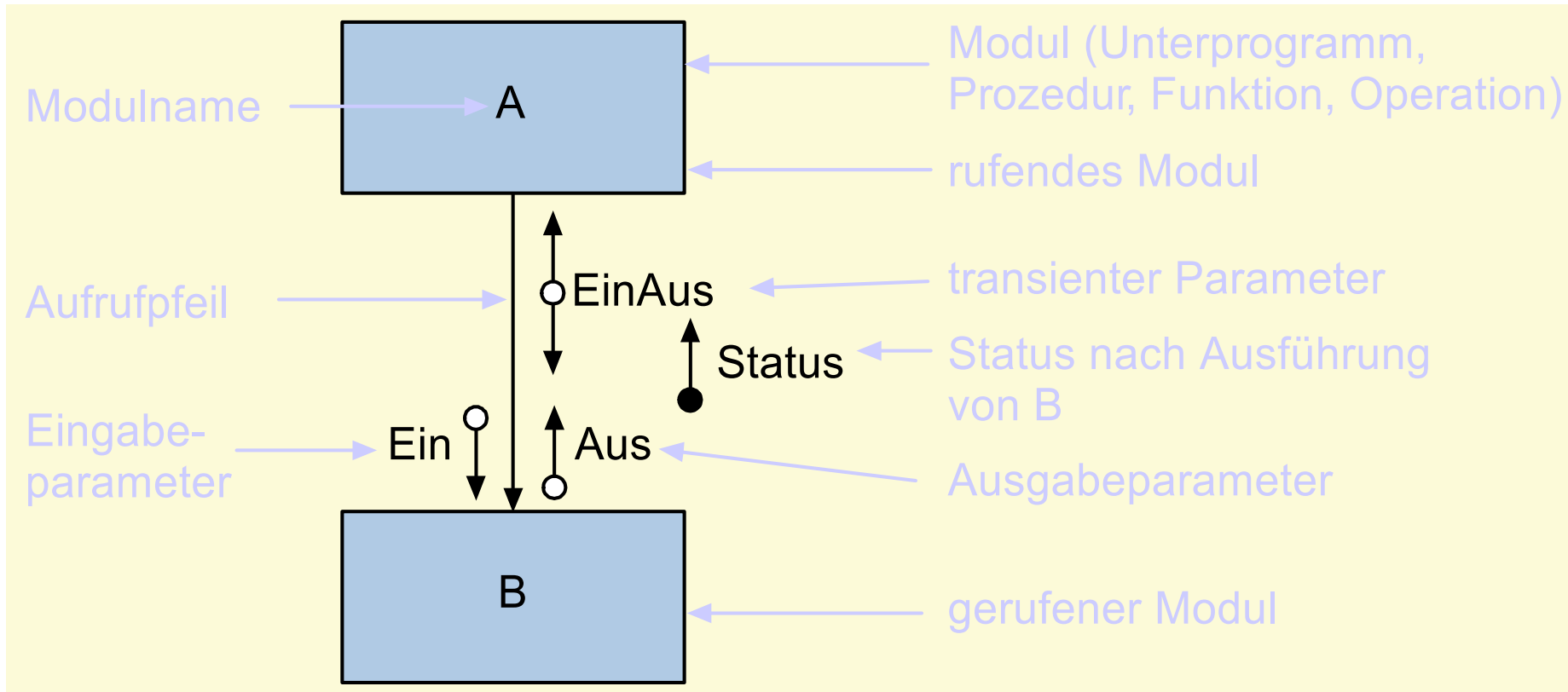
14.2.1.1. Die Entwurfsphase: Funktionale Abstraktion

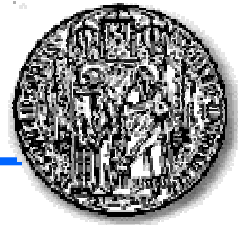
- Spezifikation der Schnittstelle
 - o Textuell oder grafisch
 - o Informal / Semiformal / Formal (E/A-Parameter):
 - ❖ Aufgabenbeschreibung
 - ❖ Eingabeparameter einschl. Datentyp und Beziehungen untereinander
 - ❖ Ausgabeparameter einschl. Datentyp und ihre Abhängigkeit von den Eingabeparametern
 - ❖ Voraussetzungen und Vorbedingungen
 - ❖ Bedingungen, die nach der Anwendung gelten
 - ❖ Verhalten bei inkorrekten Eingabewerten oder Fehlfunktion des Basissystems
 - ❖ Leistungsmerkmale.
- Programmiersprachen
 - o Durch Unterprogrammkonzepte realisiert
 - ❖ Prozeduren / Funktionen / Operatoren
 - o C++: Durch Funktionen implementiert
 - ❖ Prozeduren sind Sonderfälle von Funktionen
 - o Java: Keine eigenständigen funktionalen Module
 - ❖ Realisierung durch Operationen, die zu Klassen gehören
- Softwarearchitektur
 - o Funktionale Module und Datenabstraktionsmodule benötigt
 - o Nur funktionale Module: Unübersichtliche und änderungsunfreundliche Architekturen.



14.2.1.2. Die Entwurfsphase: Strukturdiagramme

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

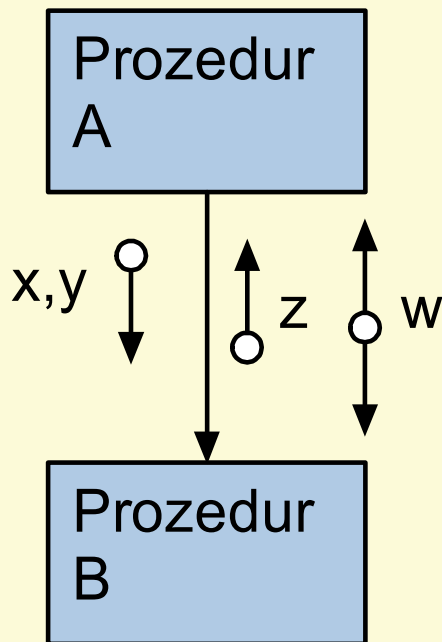




14.2.1.2. Die Entwurfsphase: Strukturdiagramme

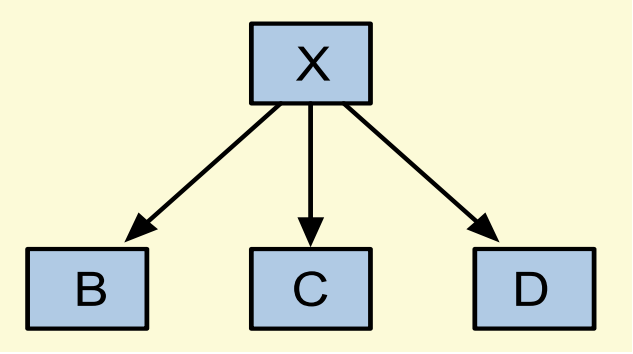
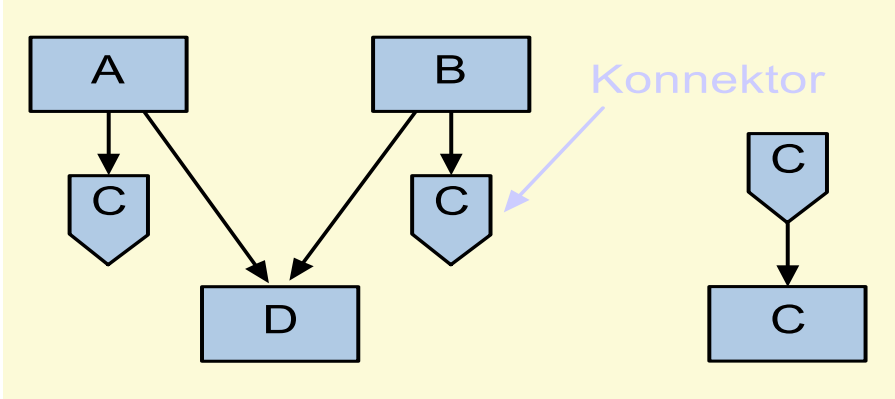
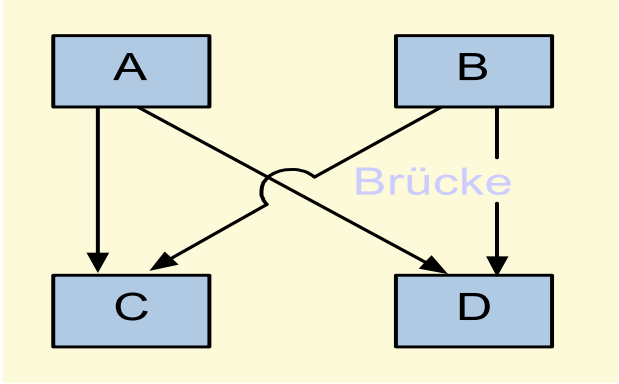
- Schnittstellentabelle
 - Bei großer Anzahl an Datenflüssen
- Datenflüsse grafisch oder als Tabelle

a Datenflüsse grafisch

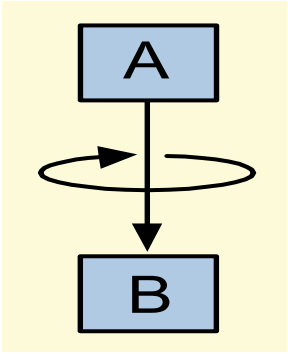


b Datenflüsse in Schnittstellentabelle

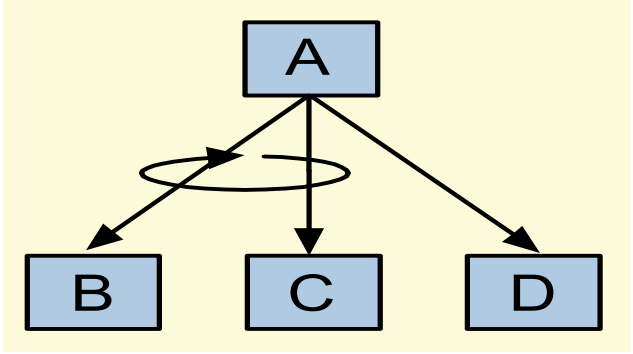
Nr	in	out	inout
1	x,y	z	w
...



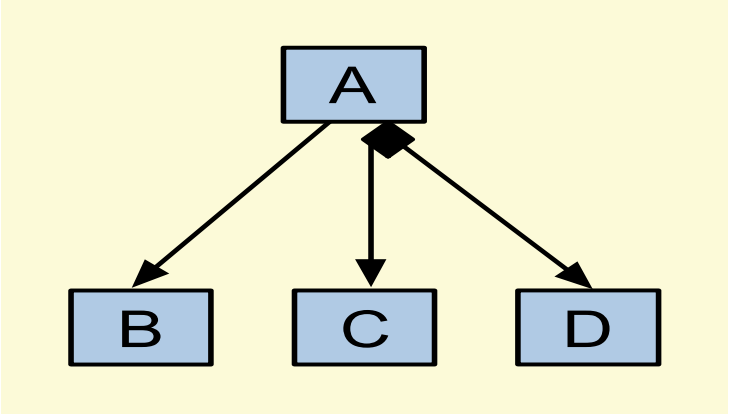
Sequenz



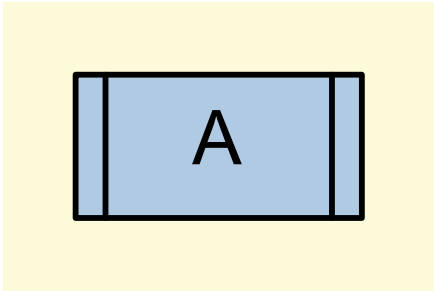
Wiederholung von B



Wiederholung von B und C



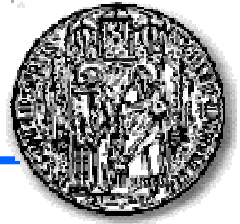
Auswahl



Vorhandenes Modul

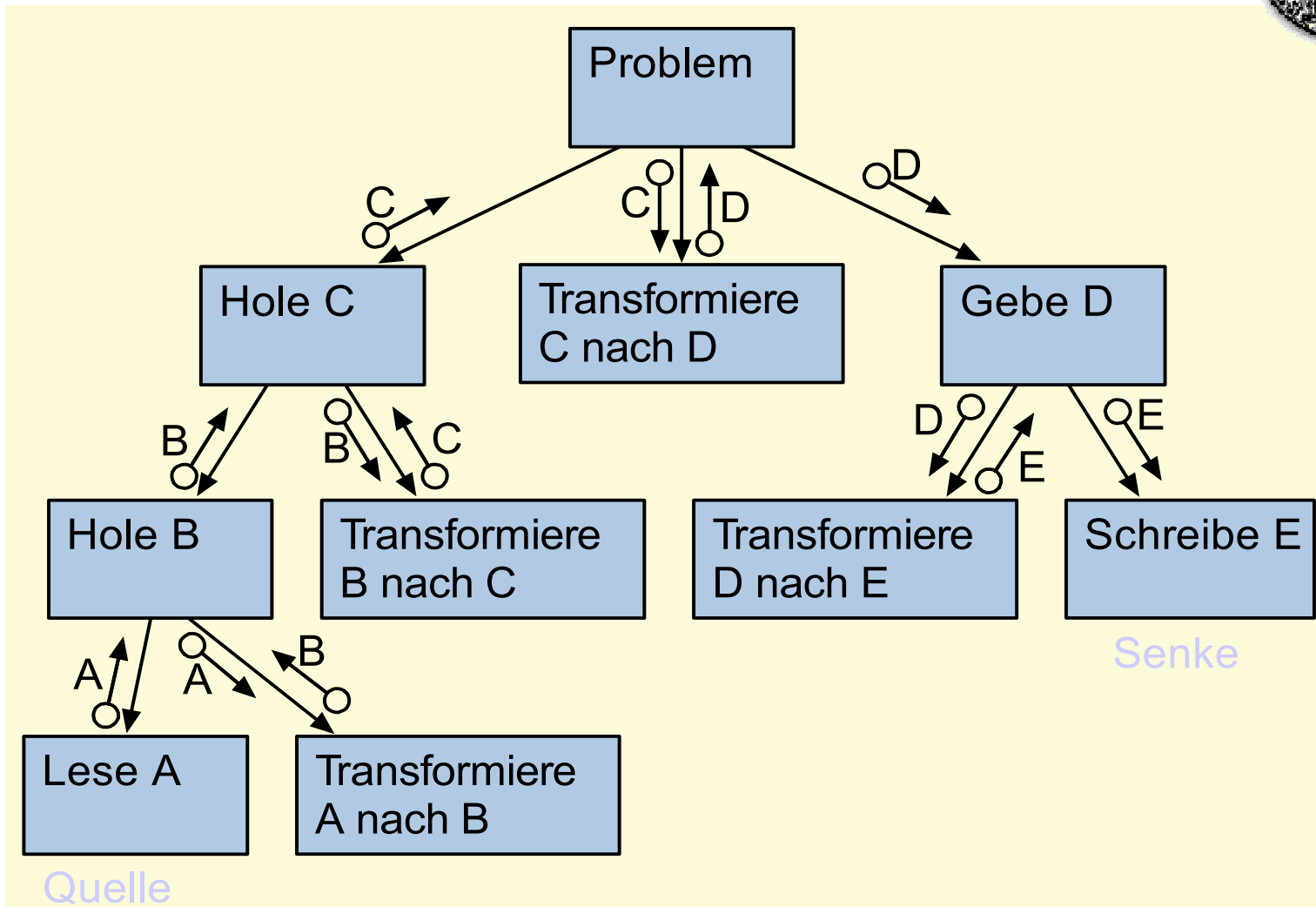
Darstellung von Datenflüssen

14.2.1.3. Die Entwurfsphase: Strukturiere Software-Archi ...

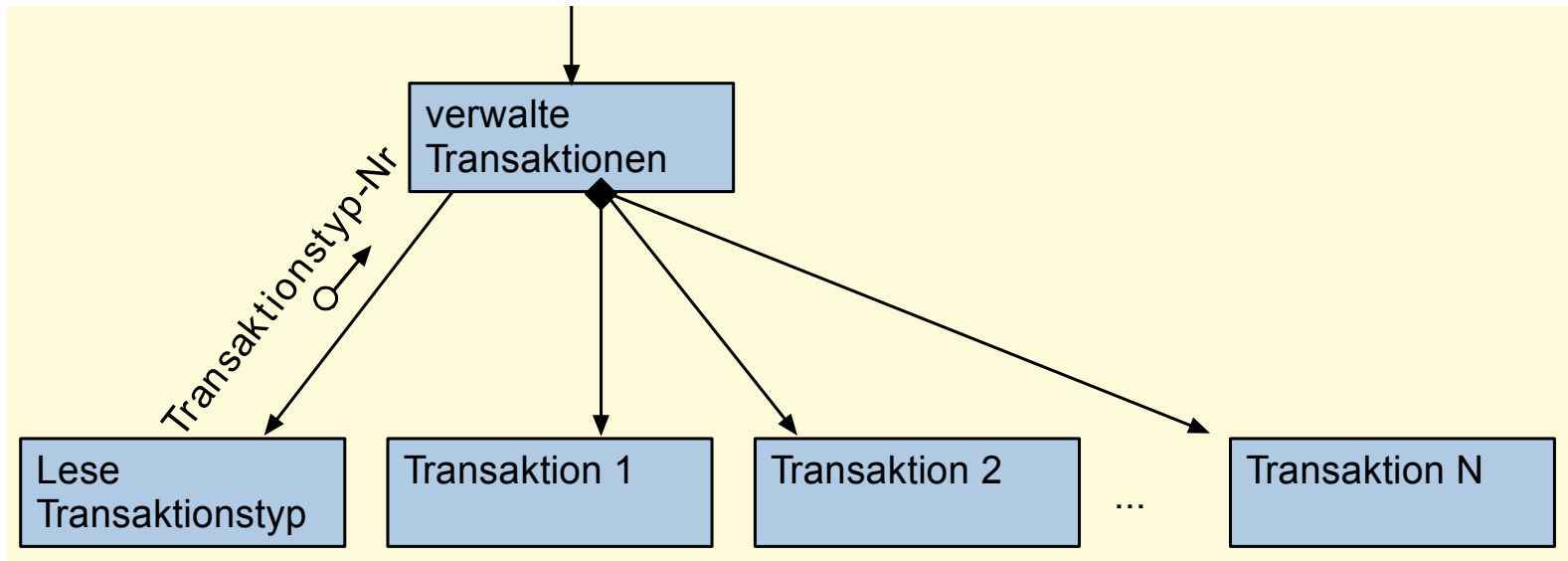
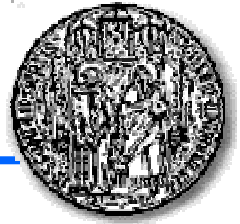


- 2 Grundarchitekturen
 - o Allgemeine Transformationsstruktur
 - o Transaktionsstruktur
 - o Kombination
- Allgemeine Transformationsstruktur
 - o Daten kommen von einer Quelle, werden transformiert und verschwinden in einer Senke
 - o Merkmale
 - ❖ Module weitgehend kontextunabhängig
 - ❖ Zwischen den Kindern eines Vaterknotens bestehen keine Kommunikationsbeziehungen
 - ❖ Modul kann von mehreren Modulen aus aufgerufen werden.
- Transaktionsstruktur
 - o Transaktion besitzt 5 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)

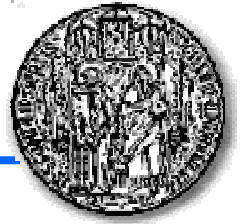
14.2.1.3. Die Entwurfsphase: Strukturiere Software-Archi ...



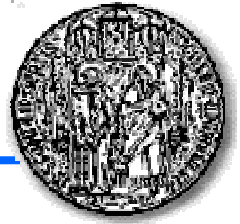
14.2.1.3. Die Entwurfsphase: Strukturiere Software-Archi ...



14.2.1.3. Die Entwurfsphase: Strukturiere Software-Archi ...

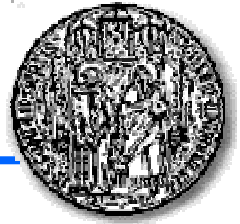


- Qualitätsziele
 - o Starke Bindung (strong cohesion)
 - o Lose Kopplung (loose coupling)
- Strukturierter Entwurf
 - o Enthält von Konzept und Methode her nur funktionale Module
 - o Erst Page-Jones führt auch abstrakte Datenobjekte ein, genannt Informationale Cluster
 - o Module, die mehrere funktionale Abstraktionen zur Verfügung stellen, sind nicht vorgesehen
- Bewertung
 - o Nur noch in Sonderfällen sinnvoll
 - o Baumhierarchie oder azyklisches Netz stellen nicht für jedes Problem die geeignete Software-Struktur dar



14.2.1.3. Die Entwurfsphase: Transformation von SA nach SD

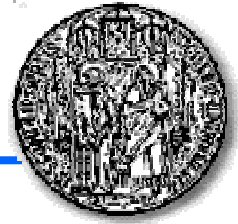
- Transformation von SA nach SD
 - o 2 Methoden
 - Transaktions-Analyse
 - Transformations-Analyse
 - o Ableitung in 3 Schritten
 1. Aufbrechen des SA-Modells in handhabbare Einheiten durch die Transaktions-Analyse
 2. Konvertieren jeder Einheit in ein gutes Strukturdiagramm durch die Transformations-Analyse
 3. Zusammenfügen der getrennten Einheiten zu einem Gesamtsystem.
- Transaktions-Analyse
 - o Die Transaktionstypen eines Systems werden identifiziert
 - o Jeder Transaktionstyp wird als Entwurfseinheit verwendet und für sich entworfen
- Transformations-Analyse
 - o Der Teil eines DFD, der durch die Transaktions-Analyse isoliert wurde, wird in ein Strukturdiagramm konvertiert.



14.2.1.3. Die Entwurfsphase: Transformation von SA nach SD

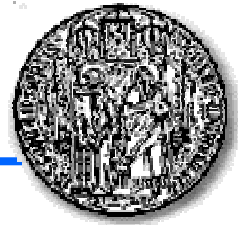
- Methode
 - o Transformations-Analyse aus 5 Schritten
 1. Zeichnen eines DFD pro Transaktionstyp
 2. Finden der zentralen Funktionen bzw. Prozesse des DFD
 3. Konvertieren des DFD in ein »erstes« Strukturdiagramm. Eine Funktion der zentralen Transformation wird zur »Steuerungszentrale«
 4. Verfeinerung des Strukturdiagramms entsprechend den Entwurfskriterien
 5. Überprüfung, ob das endgültige Strukturdiagramm die Anforderungen des ursprünglichen DFD erfüllt.

- Eigenschaft einer Steuerungszentrale
 - o Wenig Verarbeitung – Viel Koordination
 - o 2 oder 3 Funktionen, die als Steuerungszentrale geeignet erscheinen
 - ❖ Jede Variante ausprobieren
 - o Keine Kandidaten, dann zentrale Transformation unter eine neue Steuerungszentrale hängen



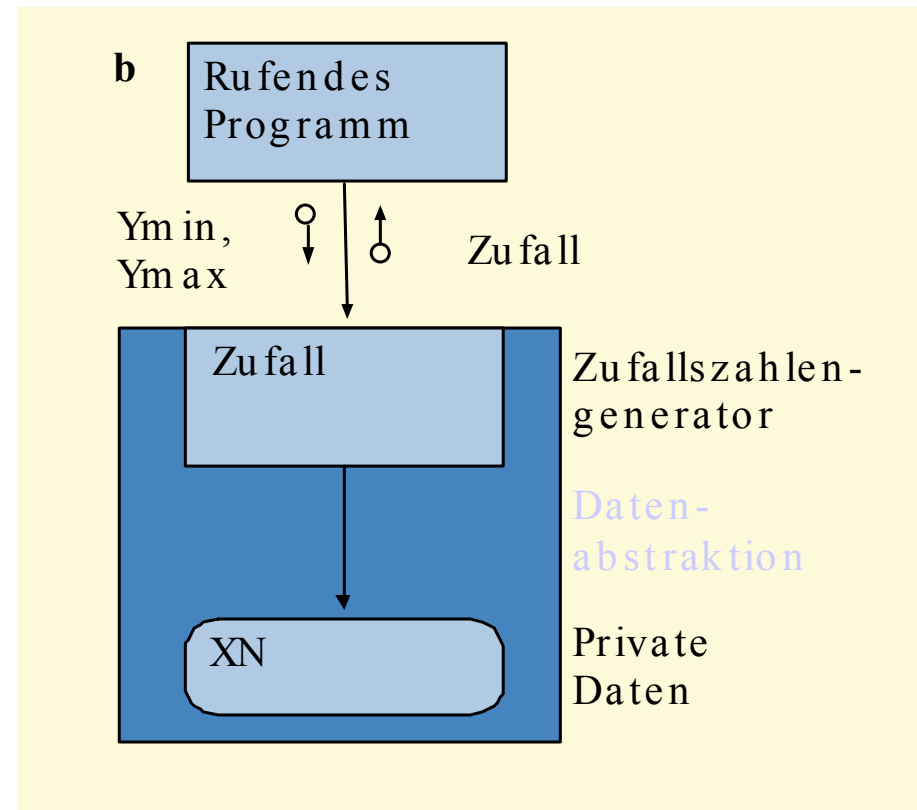
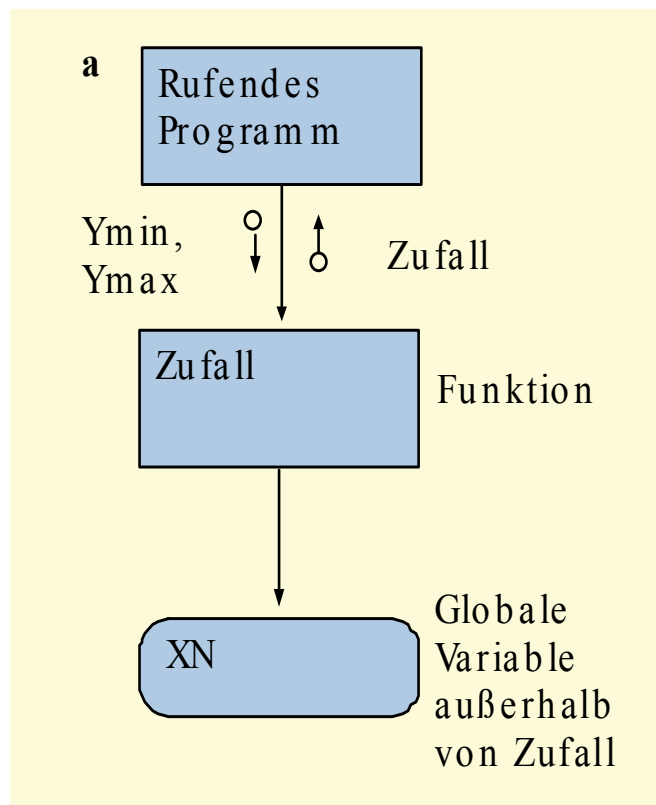
14.2.2. Die Entwurfsphase: Modularer Entwurf

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

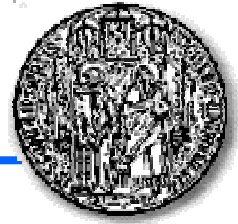


14.2.2.1. Die Entwurfsphase: Datenabstraktion

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

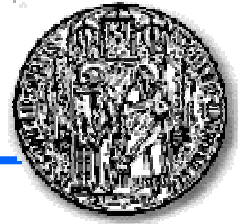


Beispiel



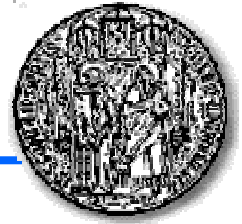
14.2.2.2. Die Entwurfsphase: Abstrakte Datenobjekte

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

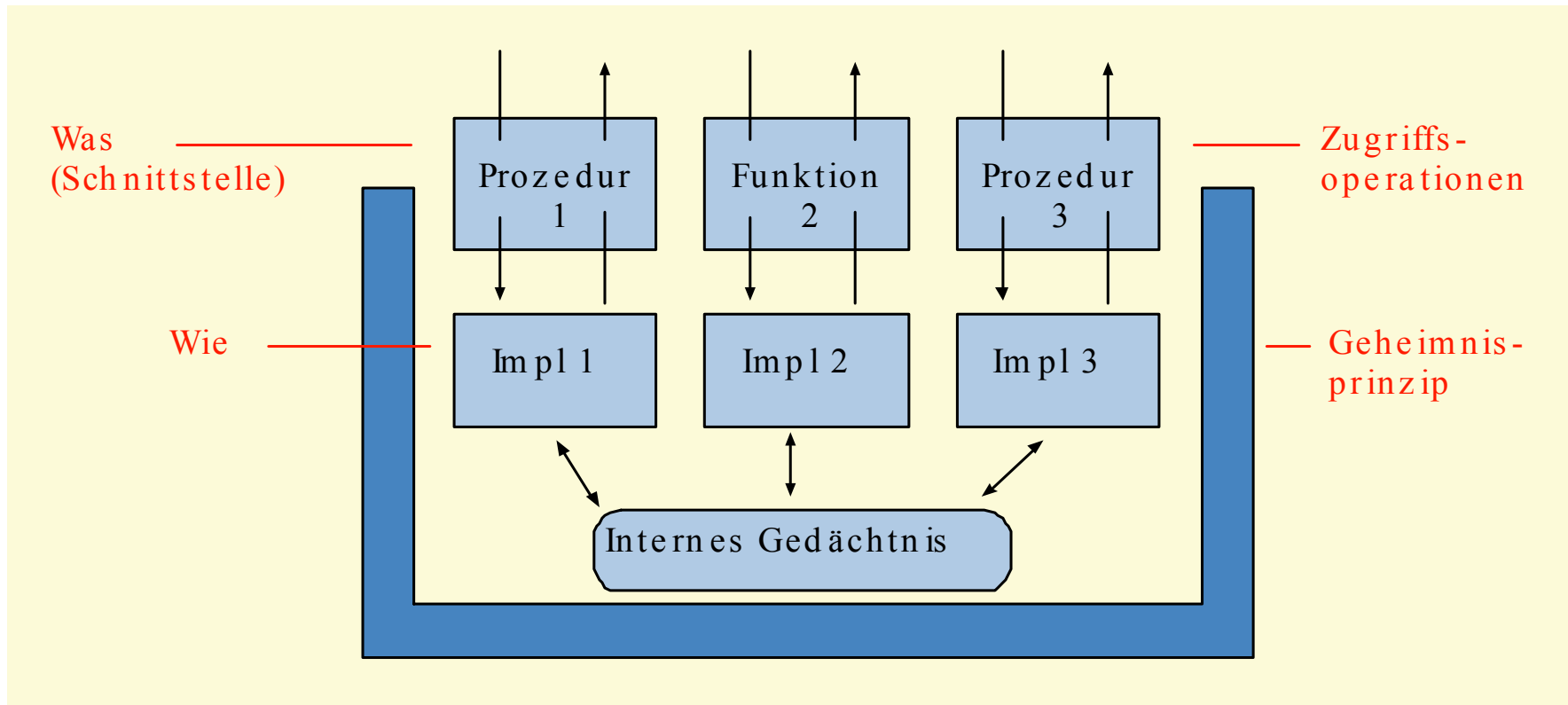


14.2.2.2. Die Entwurfsphase: Abstrakte Datenobjekte

- 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.
- Vergleich: Objekt in der objektorientierten Softwareentwicklung
 - o Internes Gedächtnis entspricht den Attributen eines Objekts
 - o Zugriffsoperationen entsprechen den Objektoperationen

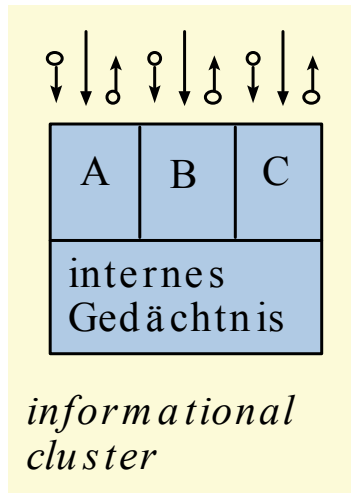


14.2.2.2. Die Entwurfsphase: Abstrakte Datenobjekte

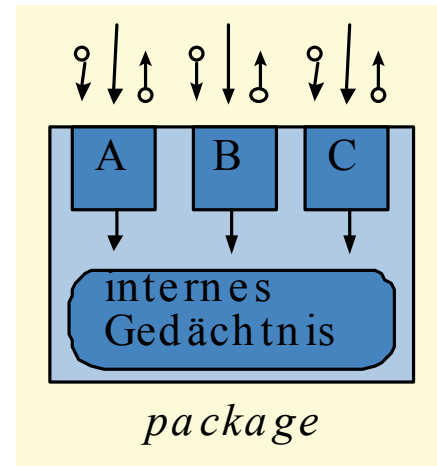


Nächste Folie : Vergleich von Notationen für abstrakte Datenobjekte

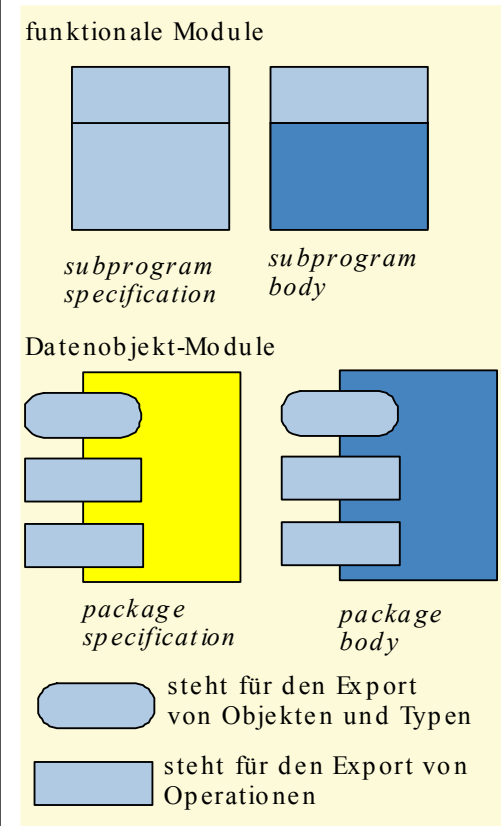
Aufbau eines Datenobjektmoduls



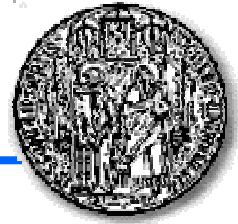
- *informational cluster*
- Zusätzlich zu Strukturdiagrammsymbolen



- Zusätzlich zu Strukturdiagrammsymbolen
- Die Symbole sind schachtelbar
- Extra Symbole für nebenläufige Module
- Die Operationen können auch an die Seiten gezeichnet werden

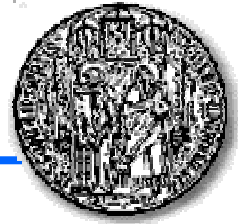


- Aufrufpfeile zeigen nur auf Module nicht auf Operationen
- Datenflüsse werden nicht angegeben
- Notation alternativ zu Strukturdiagrammen
- Symbole schachtelbar
- Extra Symbole für nebenläufige Module und abstrakte Datentypen
- Terminologie auf Ada zugeschnitten



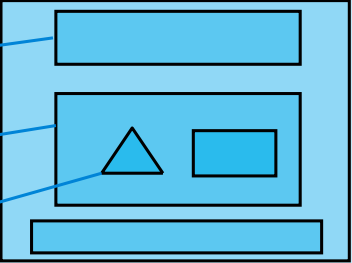
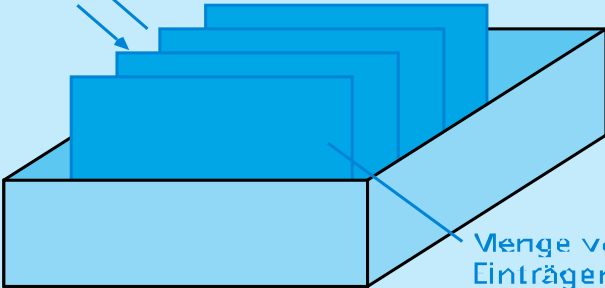
14.2.2.2. Die Entwurfsphase: Abstrakte Datenobjekte

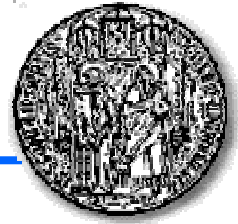
- Aufgabenbeschreibung
- Modultyp (Datenobjekt-Modul oder ADT-Modul)
- Leistungsmerkmale
 - o Geschwindigkeit
 - o Speicherplatz
 - o Genauigkeit (bei numerischen Operationen)
- Voraussetzungen und Vorbedingungen für die Anwendung des Moduls
- Bedingungen, die nach der Anwendung des Moduls gelten
- Alle bereitgestellten Zugriffsoperationen.
- Erforderliche Angaben für jede Zugriffsoperation:
 - o Eingabeparameter einschl. Datentyp und Beziehungen untereinander
 - o Ausgabeparameter einschl. Datentyp und ihre Abhängigkeit von den Eingabeparametern und anderen Daten bzw. von internen Zuständen
 - o Wirkung bzw. Effekte der Zugriffsoperation
 - o Anwendungsvoraussetzungen und Vorbedingungen
 - o Gültige Bedingungen nach der Anwendung
 - o Verhalten bei inkorrekten Eingabewerten oder Fehlfunktion des zugrundeliegenden Basissystems.



14.2.2.2. Die Entwurfsphase: Abstrakte Datenobjekte

- Abhängigkeiten und Relationen zwischen den Zugriffsoperationen:
 - o Statische Relationen
 - o Gegenseitige Ausschlussbedingungen
 - o Zulässige Aufruffolgen
- Um die Abhängigkeiten zwischen den Zugriffsoperationen zu beschreiben, eignet sich die algebraische Spezifikation
- Gut geeignet sind auch Zustandsautomaten
 - o Analog zur objektorientierten Welt können Objekt-Lebenszyklen auch bei abstrakten Datenobjekten beschrieben werden.
- Vorteile einer Schnittstellenspezifikation:
 - o Verhalten und Anwendung eines Moduls kann allein aus der Schnittstellenspezifikation entnommen werden
 - o Konsequente Verwirklichung des Geheimnisprinzips nur durch eine vollständige Schnittstellenspezifikation möglich
 - o Anhand der Schnittstellenspezifikation kann das Modul implementiert werden, ohne dass weitere Informationen eingeholt werden müssen
 - o Implementierung kann gegen die Spezifikation geprüft werden.

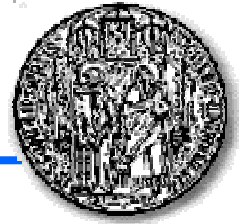
Aufgaben des abstrakten Datenobjekts	Anwendersicht	Implementierungssicht
Komplexen Einzeleintrag verwalten	<p>Gesamtobjekt</p>  <ul style="list-style-type: none"> ■ Lese- und Schreibzugriff auf die einzelnen »logischen« Komponenten z.B. Ort bei Adresse 	<ul style="list-style-type: none"> ■ Komponentenreihenfolge hintereinander oder verstreut? ■ Realisierung der Reihenfolge durch sequentielle Ablage, verkettet, über Indizes oder Zeiger? ■ Einzelkomponenten verdichtet, einzelne Komponente muß erst berechnet werden o.ä.? ■ Zusätzliche Komponenten für interne Zwecke, z.B. Statistik?
Sammlung von Einträgen verwalten	<ul style="list-style-type: none"> ■ Ablegen, Auffinden, Ändern und Löschen von Elementen der Sammlung ■ Festlegung der Zugriffsart (FIFO, LIFO, RANDOM, usw.) <p>Zugriffsart</p>  <p>Menge von Einträgen</p>	<ul style="list-style-type: none"> ■ Wie wird die Menge/geordnete Menge realisiert: sequentiell, verkettet über Indizes oder Zeiger usw.? ■ Wo wird die Menge abgelegt: im statischen Speicherbereich, im Laufzeitkeller, auf der Halde, auf dem Sekundärspeicher, auf einem Knoten im Netz o.ä.? ■ Wie wird die Zugriffsart realisiert: über Berechnung oder Einrichtung von Zugriffspfaden? ■ Zusätzliche Komponenten für interne Zwecke erforderlich?



14.2.2.2. Die Entwurfsphase: Abstrakte Datenobjekte

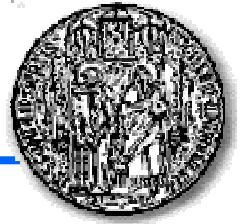
- + Abstrakte Datenobjekte können allein durch Anwendung der definierten Zugriffsoperationen benutzt werden, ohne irgendwelche Kenntnisse der Implementation
 - + Durch die Datenabstraktion wird das »Was« vom »Wie« getrennt
 - + Die Zugriffsoperationen können ohne Rücksicht auf die Struktur des internen Gedächtnisses (z.B. Datei-, Satz- und Datenstrukturaufbau) von den Anforderungen der Anwendung her festgelegt werden.
 - + Änderungen an einer Zugriffsoperation haben i.Allg. keine Auswirkungen auf andere Zugriffsoperationen
 - o 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 i.Allg. keine Auswirkungen auf die Anwendung der Zugriffsoperation. Je weniger Daten in der Parameterliste einer Zugriffsoperation übertragen werden
 - o desto änderungsfreundlicher ist eine Datenabstraktion
 - o 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
-

Vorteile eines Datenobjektmoduls



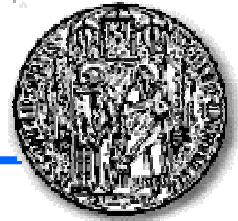
14.2.2.3. Die Entwurfsphase: Abstrakte Datentypen

- Abstrakter Datentyp (ADT)
 - o Trennung von Typdeklaration und Variablendeklaration
 - o Beliebig viele Exemplare können erzeugt werden
 - o Jedes Exemplar besitzt sein eigenes Gedächtnis
 - o Vergleich: Klasse – ADT
 - ❖ ADT entspricht weitgehend eine Klasse, aber ohne Vererbung
- Variablensemantik
 - o Automatische Verwaltung durch das Laufzeitsystem der Programmiersprache
 - o Das Exemplar wird...
 - ❖ erzeugt, wenn die Datenobjektdeklaration abgearbeitet ist
 - ❖ gelöscht, wenn der Gültigkeitsbereich der Deklaration verlassen wird
 - o Wird ein Exemplar einem anderen zugewiesen, dann wird es kopiert.
- Zeigersemantik
 - o Beim Erzeugen über eine Kreierungsoperation entsteht eine Bezeichnung für ein neu geschaffenes abstraktes Datenobjekt
 - o Die Verwaltung (Löschen, Erzeugen) liegt in der Verantwortung des anwendenden Moduls
 - o Es existiert im allgemeinen eine Lösch-Operation
 - o Die Exemplare existieren als eigenständige Objekte
 - o Der Implementierer muss die Gesamtheit aller Objekte verwalten
 - ❖ Haldenverwaltung.



14.2.2.3. Die Entwurfsphase: Abstrakte Datentypen

- Abstrakte Datentypen mit formalen Parametern
 - o Bisher: Es können beliebig viele identische Exemplare eines ADT erzeugt werden
 - o In der Praxis: Oft werden leicht modifizierte Exemplare eines ADT benötigt
 - o ADA: Formale Parameter werden in der generischen Parameterdeklaration aufgeführt, die nach dem Wortsymbol generic steht.
- Abstrakte Datentypen mit Typen als formalen Parametern
 - o Exemplare
 - ❖ Im Verhalten identisch
 - ❖ Unterschied im Typ des internen Gedächtnisses
 - o Ziel: Konstruktion typunabhängiger ADTs
 - o 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.



14.2.2.3. Die Entwurfsphase: Abstrakte Datentypen

funktionale Module

- Steuer- oder Koordinationsaufgabe
- Transformationsaufgabe
- komplizierte Auswertung
- Hilfsdienste auf verschiedenen Datenstrukturen (funktionale Zwischenschicht zwischen Datenabstraktions-Schichten)

Datenobjektmodule

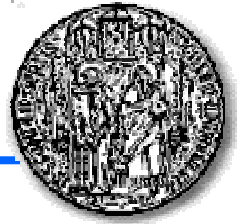
- einzelner, komplex aufgebauter Eintrag
- einzelne Sammlung mit bestimmten Zugriffsoperationen

Datentypmodule

- Handhabung von komplex aufgebauten Einzeleinträgen
- Handhabung von Sammlungen mit bestimmten Zugriffsoperationen

Legende: Blaue Verwendungszwecke sind besonders wichtig

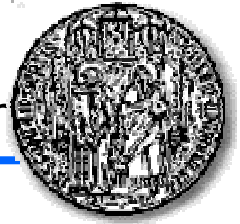
Verwendungszwecke der Modularten nach /Nagl 90/



14.2.2.4. Die Entwurfsphase: Algebraische Spezifikation

- Betrachtet abstrakte Datentypen als algebraische Strukturen, d.h. als Mengen mit darauf definierten Operationen, für die bestimmte Axiome gelten
- Ziele
 - o Formale Spezifikation einer Modulschnittstelle
 - o Verifikation der Implementierung gegen die Spezifikation.
- Syntax
 - o Der Syntaxteil besteht aus einer Menge von Vereinbarungen von Zugriffsoperationen
 - o Jede Zugriffsoperation hat die Form
 - ❖ Operationsname: Definitionsbereiche --> Wertebereich
 - ❖ Definitionsbereiche legen die Eingabe- bzw. Argumentenbereiche der Operation fest
 - ❖ Verwendete Datenbereiche werden als Sorten bezeichnet
- Semantik
 - o Menge von Gleichungen, die die Beziehungen zwischen den einzelnen Zugriffsoperationen in Form von Axiomen beschreiben.
- Schwierigkeiten entstehen...
 - o wenn die Zugriffsoperationen nicht Funktionen, sondern Prozeduren mit mehreren Wertebereichen sind
 - o wenn Ausnahme- und Fehlerbehandlungen spezifiziert werden müssen
 - o Lösung: Einführung verdeckter Operationen

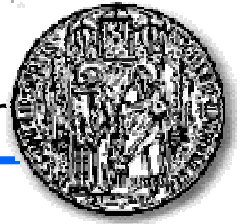
14.2.2.5. Die Entwurfsphase: Modulare Software-Architektur



- Modulare Architektur
 - o Funktionale Module
 - o Abstrakte Datenobjektmodule
 - o Abstrakte Datentypmodule
 - o Bilden die Systemkomponenten, aus denen eine modulare Software-Architektur aufgebaut wird.

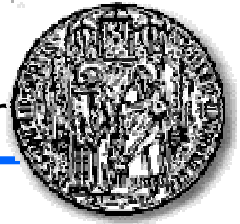
- Modul = Schnittstelle + Rumpf
 - o Jedes Modul gliedert sich in eine Schnittstelle und einen Rumpf
 - o Schnittstelle
 - o Spezifikation, welche Dienstleistungen das Modul exportiert, d.h. seiner Umgebung zur Verfügung stellt
 - o Rumpf
 - ❖ Enthält die Implementierung der in der Schnittstelle spezifizierten Dienstleistungen
 - o Zur Realisierung der Implementierung
 - ❖ Ein Modul kann die Dienstleistungen anderer Module verwenden, d.h. diese Dienstleistungen importieren.

14.2.2.5. Die Entwurfsphase: Modulare Software-Architektur

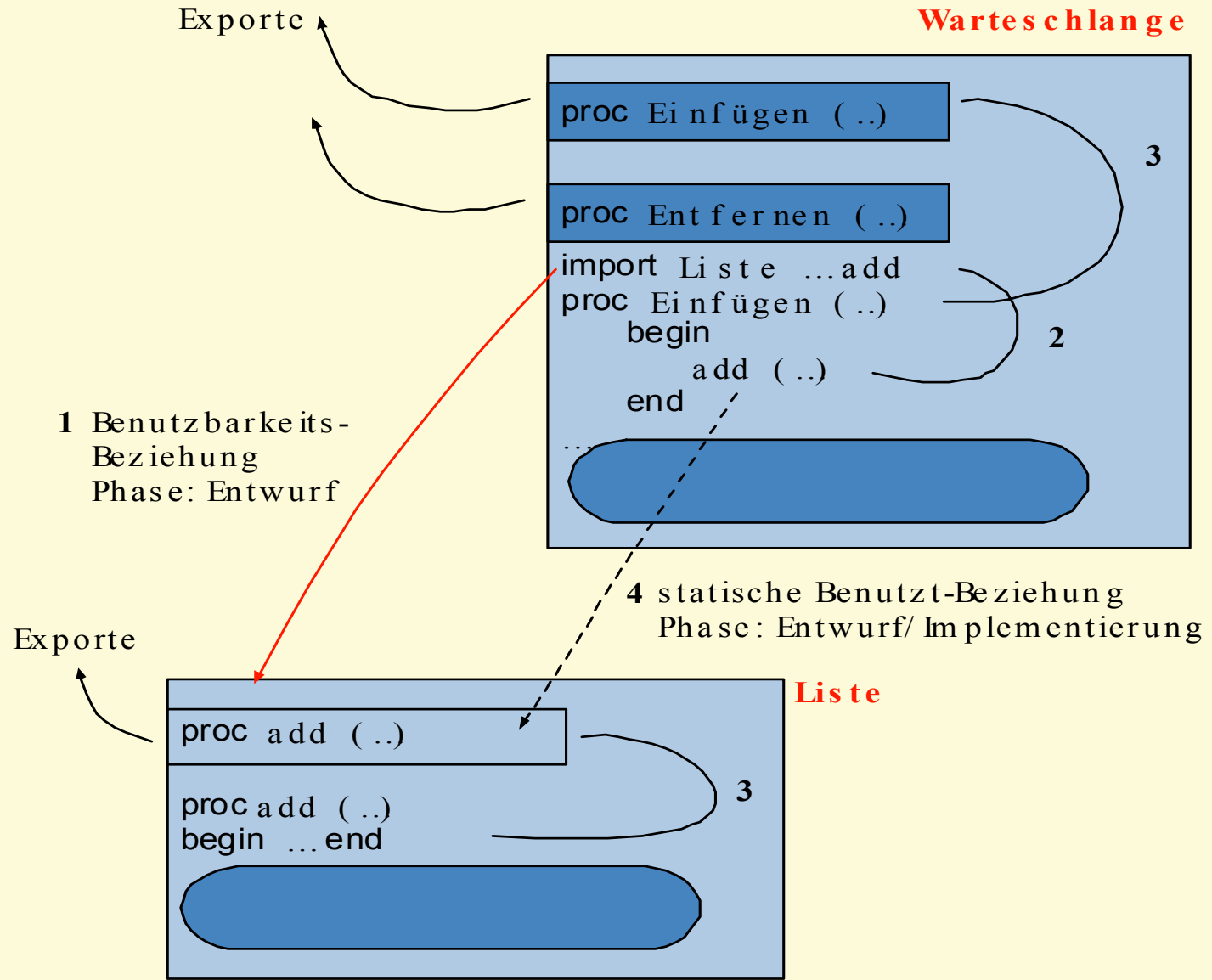


- Arten von Beziehungen
 - **Importbeziehung oder Benutzbarkeitsbeziehung** : Modul A kann die Dienstleistung von Modul B in Anspruch nehmen, muß es aber nicht
 - **Statische Benutzt-Beziehung** : Modul A benutzt die Dienstleistung von Modul B im Programmtext
 - **Dynamische Benutzt-Beziehung** : Modul A verwendet die Dienstleistung von Modul B zur Laufzeit
 - **Benutzbarkeitsbeziehung** : Voraussetzung für die statische Benutzt-Beziehung und diese wiederum für die dynamische Benutzt-Beziehung.
- Benutzbarkeitsbeziehungen
 - Für den Entwurf relevant & müssen festgelegt werden
 - 2 Arten von Benutzbarkeitsbeziehungen:
 - ❖ Lokale Benutzbarkeit
 - ❖ Allgemeine Benutzbarkeit.
- Lokale Benutzbarkeit
 - Leitet sich aus dem Schachtelungskonzept blockstrukturierter Programmiersprachen ab
 - Prozedur oder Funktion kann an beliebiger Stelle innerhalb der Schachtelungshierarchie deklariert werden
 - Damit wird die Benutzbarkeit auf den Gültigkeitsbereich eingeschränkt
 - Ausprägung des **Lokalitätsprinzips**.

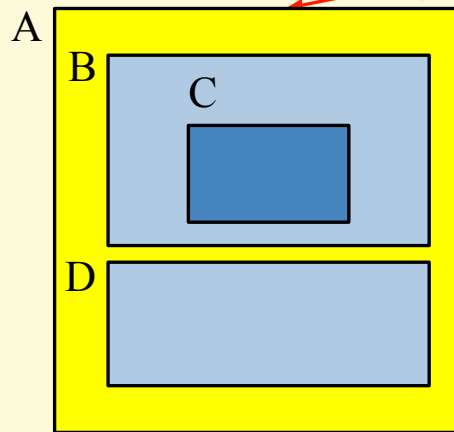
14.2.2.5. Die Entwurfsphase: Modulare Software-Architektur



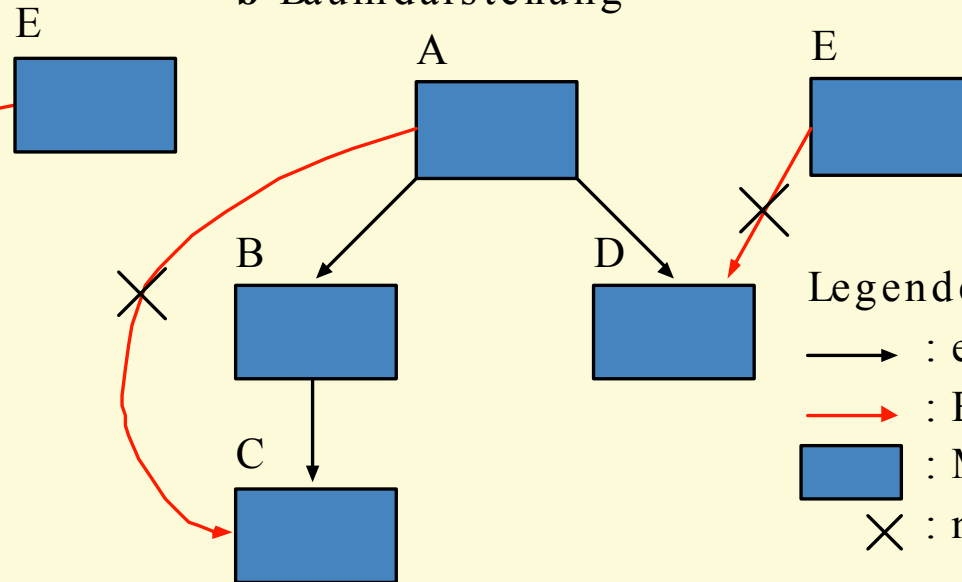
- Allgemeine Benutzbarkeit
 - Führt zu einer Schichtenstruktur
 - In einer Schicht vorkommende Teilsysteme werden zu Knoten
 - Allgemeine Benutzbarkeiten können sich über mehrere Schichten nach unten erstrecken
 - Zugriff auf Schichten, die wesentlich tiefer liegen, deutet auf falsche Modellierung hin
 - Es können Baumstrukturen modelliert werden
 - Legt fest, dass ein allgemein verwendbares Modul im Abstraktionsniveau tiefer anzusiedeln ist.
- Enthaltenseins-Beziehung
 - Durch Enthaltenseins-Beziehungen ergibt sich eine Baumstruktur
 - Module werden in den Baum eingehängt und damit nur in einem Teilbereich nutzbar
 - Interna einer Baumstruktur bleiben nach außen hin verborgen
 - ❖ Es gilt das Geheimnisprinzip.
 - Ein Modul hat Zugriff auf
 - ❖ seine Söhne (Standardfall)
 - ❖ seine Brüder
 - ❖ sich selbst (bei direkter Rekursion)
 - ❖ seine Vorfahren (bei indirekter Rekursion)
- Ein Modul mit darunter hängendem Enthaltenseins-Baum wird Teilsystem genannt.



a geschachtelte Darstellung



b Baumdarstellung



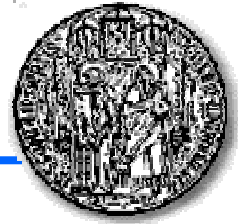
Legende:

→ : enthält

→ (red) : Benutzbarkeit

■ (blue) : Modul

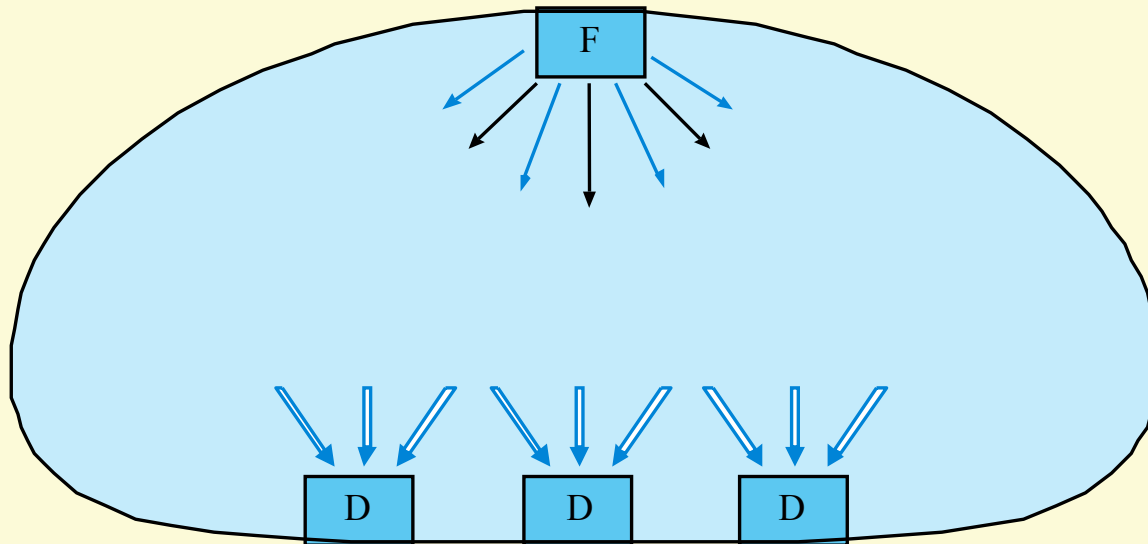
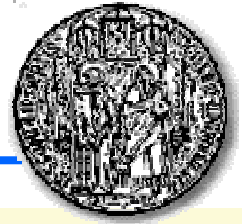
× : nicht erlaubt



14.2.2.6. Die Entwurfsphase: Modulare Entwurfsmethoden

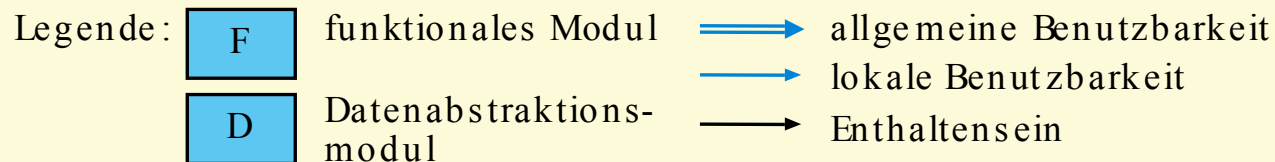
- MD verwendet
 - o funktionale Module
 - o abstrakte Datenobjektmodule
 - o abstrakte Datentypmodule
 - o für die Softwarearchitektur
- Liegt ein SA-Modell vor, dann wird dieses Modell meist nur als informelle Vorlage verwendet, und der Entwurf wird völlig neu erstellt
- Liegt ein ER- oder OOA-Modell vor, dann können diese Modelle als Basis für Datenabstraktionen verwendet werden. Nimmt man die Vorgaben aus der Definitionsphase nur als informelle Vorgaben, dann kann man einen **top-down-Entwurf** oder einen **bottom-up-Entwurf** durchführen
- top-down-Entwurf
 - o Entwurf wird auf der höchsten abstrakten Ebene oder Schicht begonnen
 - o Von oben nach unten wird jede Ebene/Schicht weiter verfeinert
 - o Verfeinert wird solange, bis Basismodule erreicht werden, deren Leistungen durch zugrunde liegende Basissoftware realisiert werden können.
- bottom-up-Entwurf
 - o Begonnen wird mit der Modulen der niedrigsten abstrakten Ebene
 - o Aufbauend auf diesen Modulen werden dann die Module der nächsthöheren Schicht entwickelt
 - o Sind Module aus vorherigen Entwicklungen vorhanden, dann werden aus diesen Modulen nach und nach höhere abstrakte Ebenen zusammengebaut
- Entwurf und Implementierung werden bei beiden Methoden oft verzahnt abgewickelt.

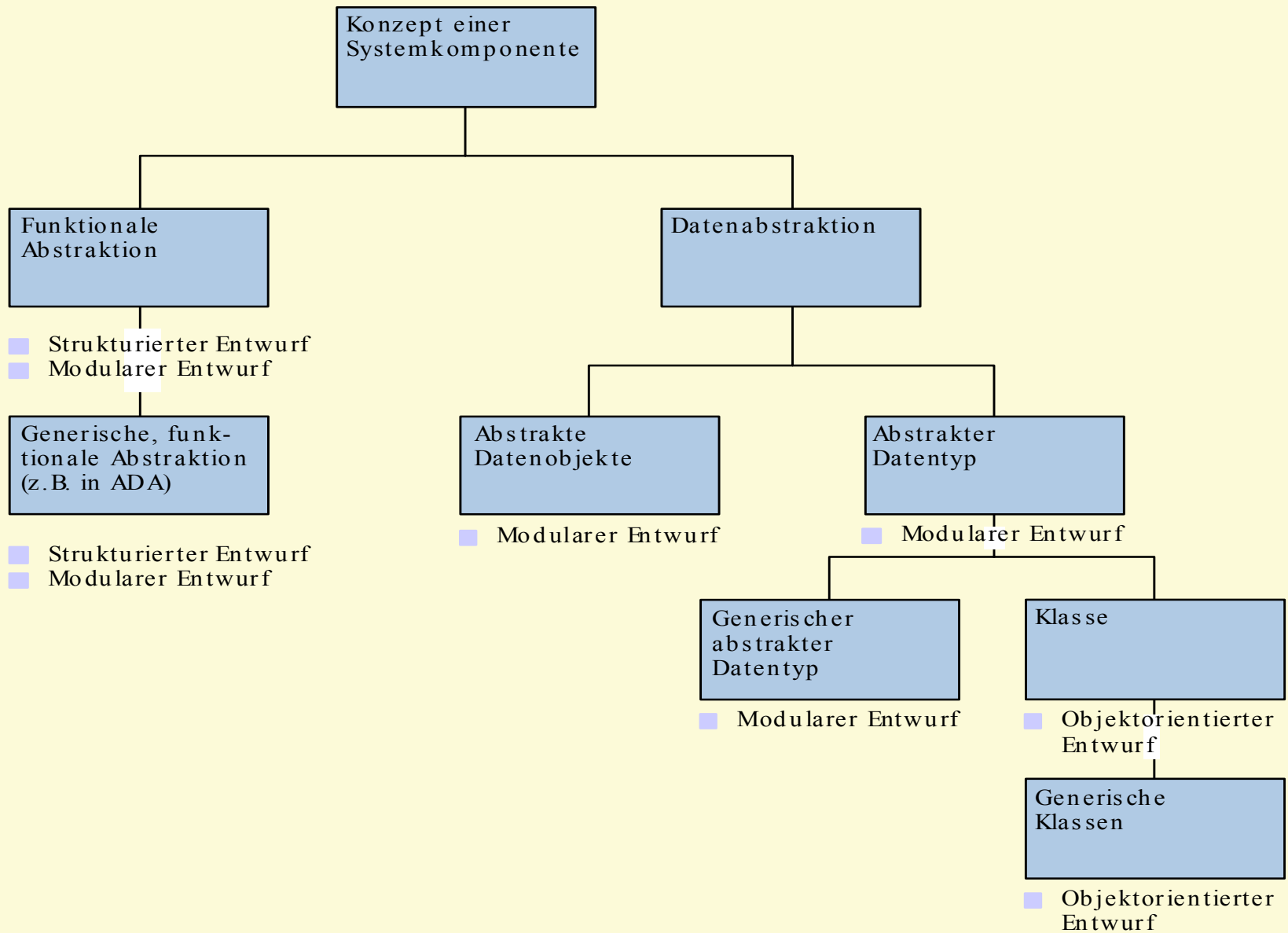
14.2.2.6. Die Entwurfsphase: Modulare Entwurfsmethoden

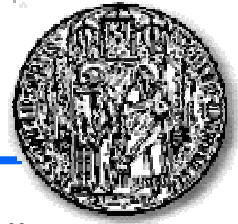


verstärktes Auftreten
von funktionalen Modulen
und lokaler Benutzbarkeit

verstärktes Auftreten von
Datenabstraktionsmodulen
und allgemeiner Benutzbarkeit

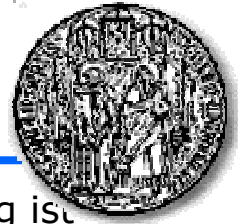






14.2. Die Entwurfsphase: SD & MD

- Der strukturierte Entwurf (structured design, SD) gibt eine Methode an, wie eine Software-Architektur bestehend aus funktionalen Modulen erstellt werden kann.
- Funktionale Module realisieren eine oder mehrere funktionale Abstraktionen und sind in der Lage, Steuerungs- und Koordinationsaufgaben, Transformations- und Auswertungsaufgaben sowie Hilfsdienste zu erledigen.
- Strukturdiagramme (structure charts) erlauben es, Software-Architekturen, bestehend aus funktionalen Modulen, grafisch und textuell zu beschreiben. Die Grafik stellt die Software-Architektur übersichtlich dar.
- Legt aus der Definition ein SA-Modell, dann kann durch eine Transformationsanalyse eine Umsetzung in einen strukturierten Entwurf erfolgen.
- Der modulare Entwurf verwendet :
 - o funktionale Module,
 - o Datenobjekt-Module, und
 - o Datentyp-Moduleum die Systemkomponenten einer Software-Architektur zu modellieren.
- Datenobjekt-Module und Datentyp-Module ermöglichen in unterschiedlicher Weise die Datenabstraktion.
- Ein Datenobjekt-Modell realisiert ein abstraktes Datenobjekt, das folgende Eigenschaften besitzt :
 - o Es ist passiv,
 - o Es verwaltet die Ablage von Daten (internes Gedächtnis), auf die über Zugriffsoperationen zugegriffen wird



14.2. Die Entwurfsphase: SD & MD

- o Es wird durch seine Beschreibung kreiert, d.h. mit der Beschreibung ist gleichzeitig die Deklaration genau eines Exemplars verbunden
- Ein Datentyp-Modul realisiert einen abstrakten Datentyp (ADT), der folgende Charakteristika hat
 - o Er ist passiv,
 - o Typdeklaration und Variablendeklaration sind getrennt
 - o Es können beliebig viele Exemplare erzeugt werden
 - o Er kann mit formalen Parametern versehen werden
- Die Abhängigkeit zwischen den Zugriffsoperationen können m.H. der algebraischen Spezifikation beschrieben werden.
- Die Systemkomponenten einer Software-Architektur stehen untereinander in Beziehung. Beim modularen Entwurf lassen sich folgende Beziehungstypen unterscheiden :
 - o Benutzbarkeitsbeziehung (Importbeziehung)
 - o allgemeine Benutzbarkeit
 - o lokale Benutzbarkeit
 - o Enthaltenseins-Beziehung (bei blockstrukturierten Programmiersprachen)
- Die Erstellung einer modularen Software-Architektur kann durch einen
 - o top-down-Entwurf oder einen
 - o bottom-up-Entwurferfolgen.