Veranstaltung
10033013

Systementwicklung

Entwurf

Uwe H. Suhl und Chris Bizer

SS 2008

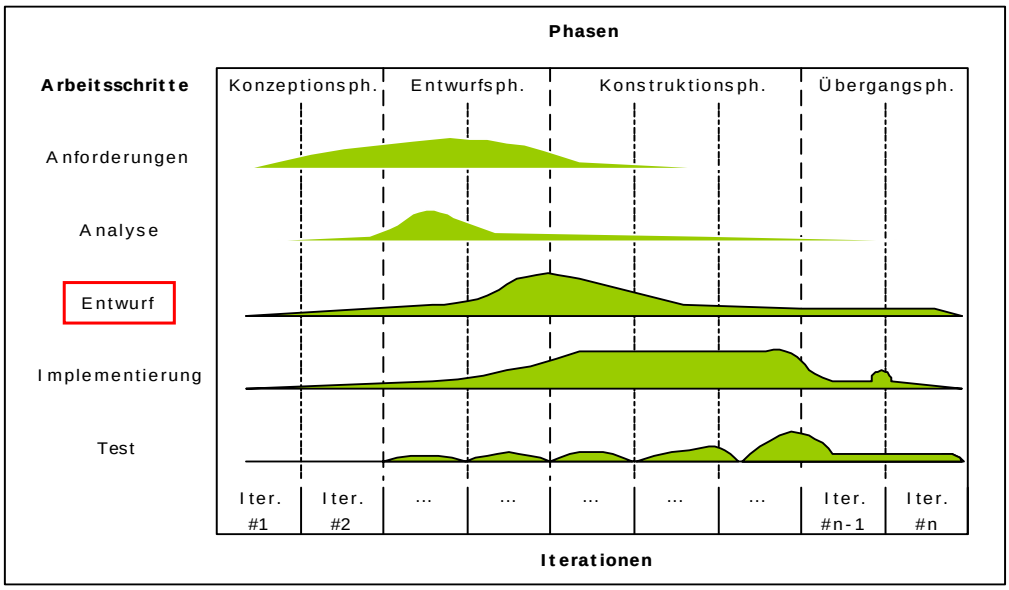
Übersicht Kapitel Entwurf

1. Ziele des Arbeitsschritts Entwurf
2. UML im Entwurf
3. Leitlinien guter Entwurf
4. Gliederung und Inhalt des Entwurfsdokuments

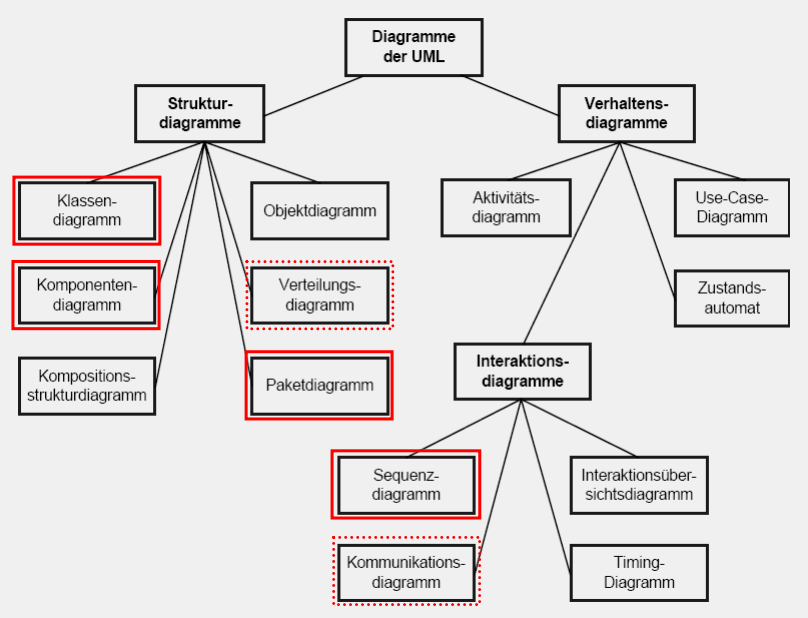
1. Ziele des Entwurfs

- **Der Entwurf beschreibt der Umsetzung der Anforderungen aus technischer Sicht.**
- **Übergang vom WAS zum WIE**
  - **WAS: fachliche Anforderungen aus der Analyse**
  - **WIE: Vorgabe für Implementierung**
- **Vorbereitung der Implementierung**
  - **Entwurfsmodelle sind primäre Arbeitsanleitung für die Implementierung.**
  - **Entwurfsmodelle können zur automatischen Codegenerierung genutzt werden.**
- **Arbeitsschritte in der Phase Entwurf**
  - **Überführung der Analysemodelle in Entwurfsmodelle.**
  - **Verfeinerung der Klassenmodelle durch Detaillierung und zusätzliche Steuerungs- und Infrastrukturklassen.**
  - **Festlegung der Subsysteme/Komponenten und deren Schnittstellen.**
- **Detaillierungsgrad und verwendete Methoden hängen vom Projekt ab.**

Phasen und Arbeitsschritte im Unified Process



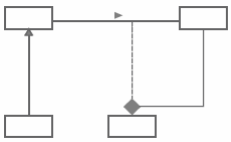
2. UML im Entwurf



Wir verwenden für den Entwurf

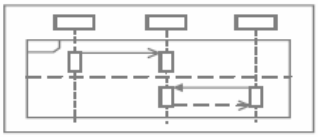
Klassendiagramm

Technische Spezifikation der Struktur des Systems.



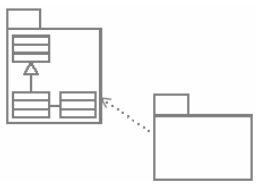
Sequenzdiagramm

Wer tauscht mit wem Informationen aus? Verifikation des Designs.



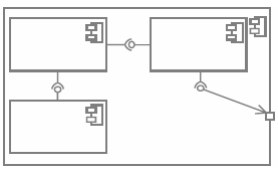
Paketdiagramm

Aufteilung des Systems in Subsysteme. Festlegung von Namensräumen.



Komponentendiagramm

Aus welchen Komponenten besteht das System? Was für Schnittstellen haben die Komponenten?



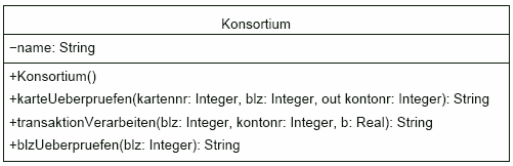
2.1 Entwurfsklassenmodell

- **Verfeinerung des Analyseklassenmodells durch Hinzufügen von**
  - **Operationen, die die Funktionalität des Systems festlegen.**
  - **Steuerungs- und Infrastrukturklassen.**
- **Das Entwurfsmodell ergibt sich aus der Integration des statischen Analysemodell (Klassendiagramm) und der dynamischen Modelle (Use Cases).**

Analyse-Modell	Entwurfs-Modell
Skizze: Teilweise unvollständig in Attributen und Operationen Datentypen und Parameter können noch fehlen Noch kaum Bezug zur Realisierungssprache Keine Überlegungen zur Realisierung von Assoziationen	Vollständige Angabe aller Attribute und Operationen Vollständige Angabe von Datentypen und Parametern Auf Umsetzung in gewählter Programmiersprache bezogen Navigationsangaben, Qualifikation, Ordnung, Verwaltungsklassen Entscheidung über Datenstrukturen Vorbereitung zur Anbindung von Benutzungsoberfläche und Datenhaltung an fachlichen Kern

Darstellung von Operationen in Klassendiagrammen

- **Operationen legen die Funktionalität des Systems fest.**



- **Spezifikation von Operationen:**

  `[Sichtbarkeit] Name (Parameterliste) : Rückgabetyp`

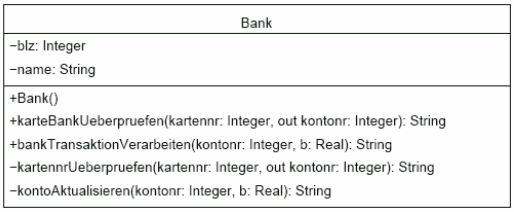
  - **Operationen werden mindestens durch ihren Namen dargestellt.**
- **Parameterliste:**

  `Name : Typ [´[´Multiplizität´]´][= Vorgabewert]`

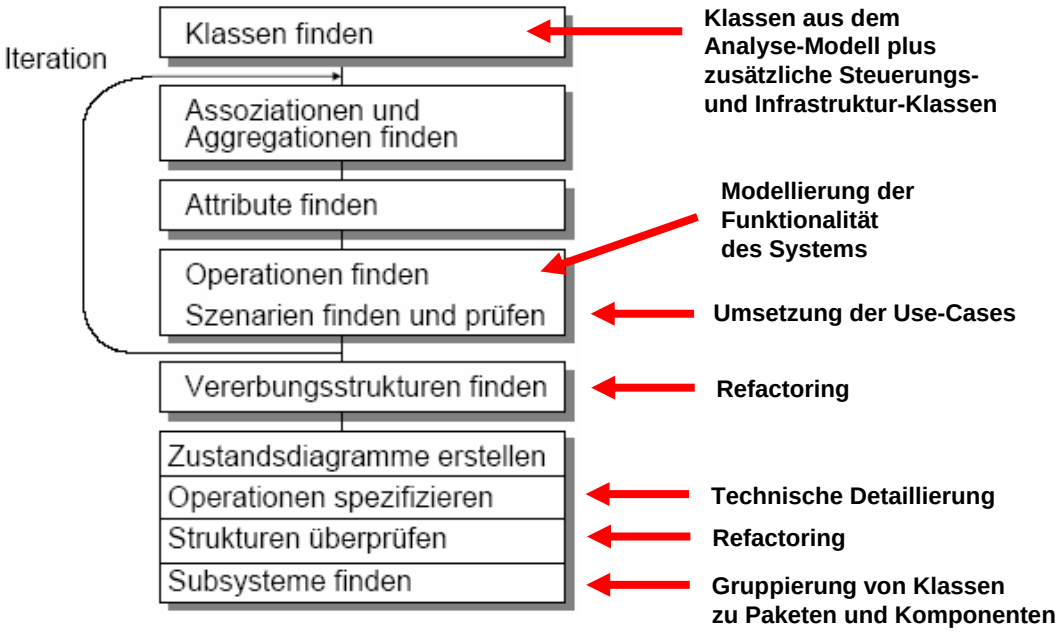
  - **Typ: Elementarer Datentyp oder Klasse**
  - **Multiplizität: viele Inhalte umfasst der Parameter [1..*], [2..4]. Nur notiert, wenn nicht [1].**

Sichtbarkeit / Zugriffsrechte bestimmen

- Häufig sollen nur bestimmte Merkmale der Objekte einer Klasse von außen zugreifbar sein ("Kapselungsprinzip„).
- Zur Zugriffskontrolle verwendet man Sichtbarkeits-markierungen für Attribute und Operationen:
  - **+name** d.h. öffentlich zugreifbar ("public")
  - **-name** d.h. nur innerhalb der Klasse verwendbar ("private")
  - **#name** d.h. nur innerhalb der Klasse und in allen Subklassen verwendbar ("protected")
- Beispiel:



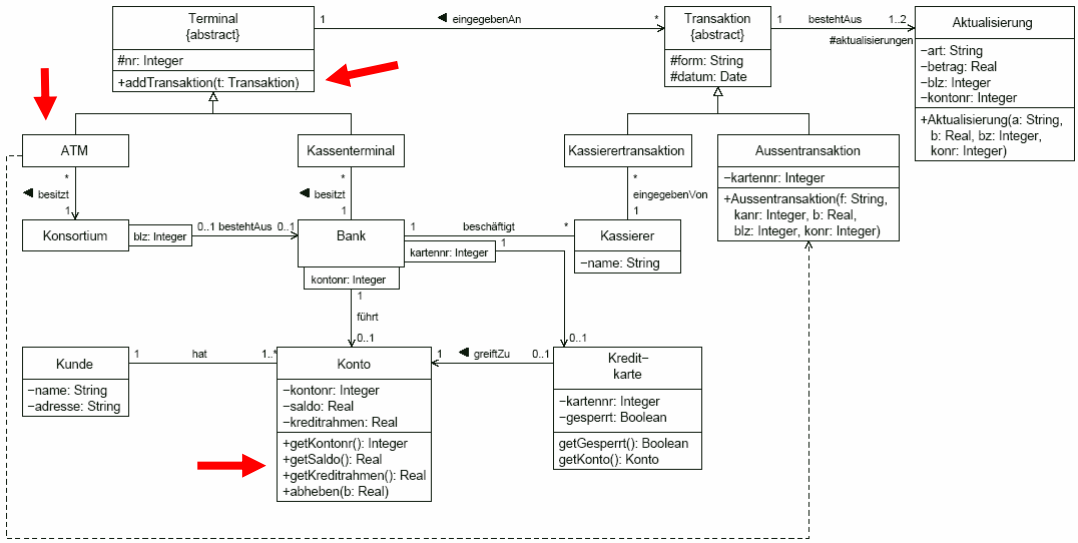
Erstellung des Entwurfsklassenmodells



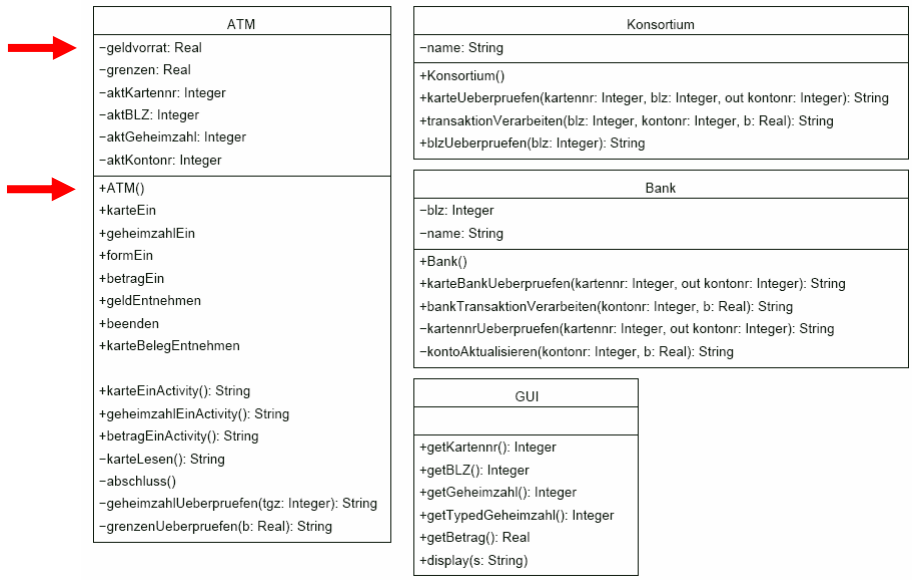
Erstellung des Entwurfsklassenmodells

- **Attribute**
  1. Assoziationen ausrichten / Referenz-Attribute einfügen
  2. Festlegung von Datentyp und Sichtbarkeit
- **Operationen**
  1. Operationen hinzufügen
  2. Parameter und Rückgabewerte festlegen
  3. Zugriffsrechte bestimmen / Schnittstellen definieren
- **Steuerungs- und Infrastrukturklassen**
  1. Controller-Klassen zur Steuerung von Workflows.
  2. Zusätzliche aus technischer Sicht notwendigen Klassen zur Einbindung in Infrastruktur, Altsysteme (z.B. Datenbankzugriff, Web-Serivces, ...) modellieren.
     - Wiederverwendung existierender Klassen durch Inklusion oder Vererbung (z.B. .net Framework).

Beispiel: Entwurfs-Klassendiagramm 1



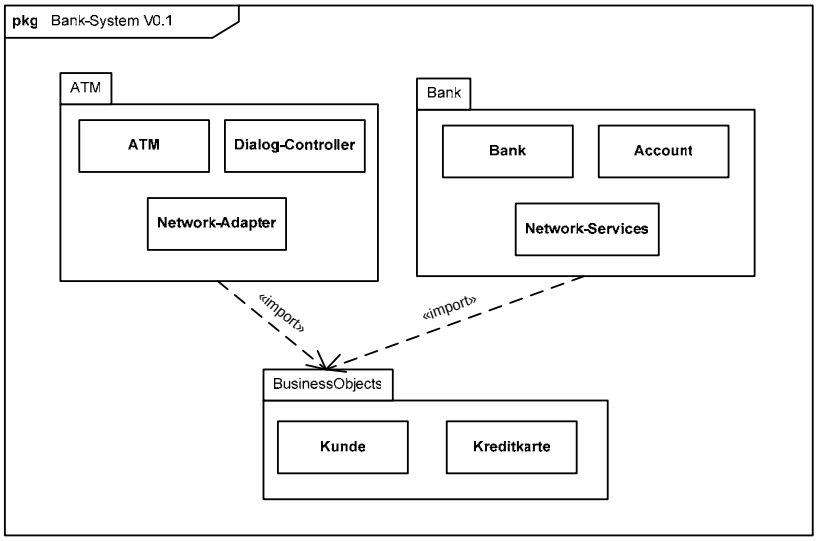
Beispiel: Entwurfs-Klassendiagramm 2



2.2 Paketdiagramm

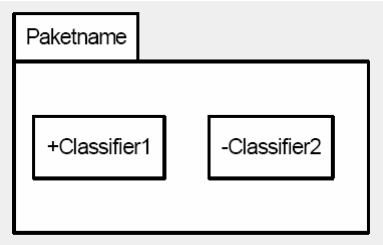
- **Pakete bündeln Modellelemente mit gleicher Thematik zu größeren Einheiten.**
- **Pakete dienen der**
  - **Gliederung von Klassendiagrammen (vereinfachte, abstrakte Sicht auf System)**
  - **Entkopplung von Subsysteme** (siehe Kapitel Architektur)
  - **der Festlegung von Namensräumen. Paketname.Klassenname Z.B. system.data.sql**
- **Ziel: Starke Bindung (Kohäsion) innerhalb des Pakets**
  - **Einheitlicher Themenbereich.**
  - **Vererbung soweit wie möglich nur innerhalb des Pakets.**
- **Ziel: Schwache Kopplung zwischen Paketen**
  - **Möglichst wenig Assoziationen über Paketgrenzen hinweg.**
  - **Festlegung wohldefinierter Schnittstellen.**
- **Faustregeln für ein sinnvolles Paket:**
  - **10-15 Klassen**
  - **1 DIN A4 Seite für ein Diagramm**

Beispiel: Paketdiagramm



Notationselemente Paketdiagramm

- **Das Paketdiagramm hat folgende Notationselemente :**
- **Paket**
  - **Pakete beinhalten Klassen sowie ggf. weitere Pakete.**
  - **Jede Klasse kann nur in einem oder keinem Paket enthalten sein.**

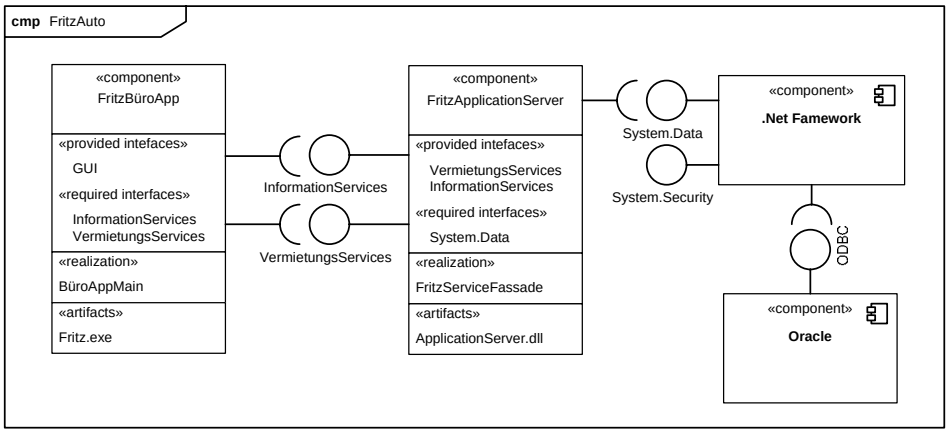


- **Import-Beziehung**
  - **Zeigt den Zugriff eines Pakets auf ein anderes Paket.**
  - **z.B. Paket `Sonnenschein.UserInterface` importiert Paket `System.Web.UI.HtmlControls`**

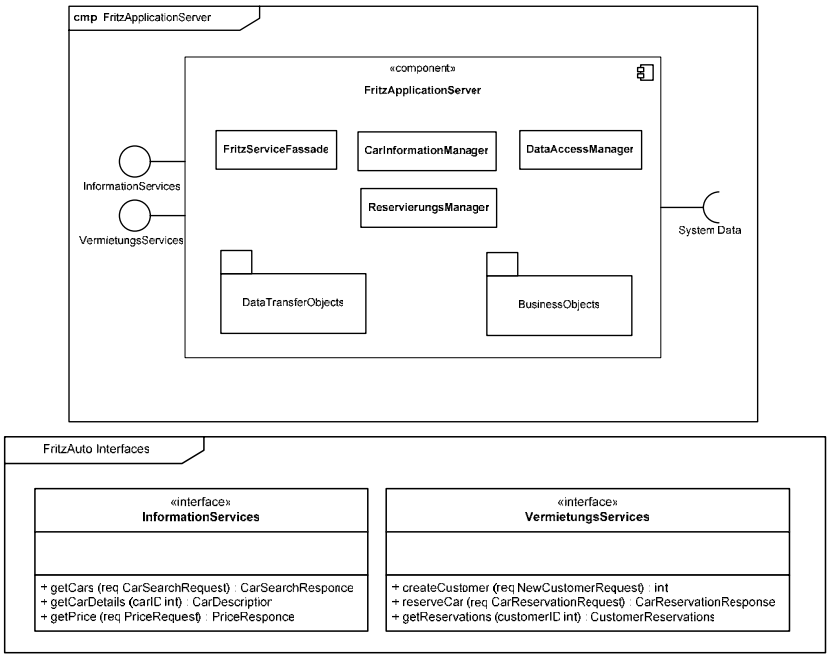
2.3 Komponentendiagramm

- **Komponentendiagramme stellten die Bestandteile des Systems zur Laufzeit dar.**
- **Komponente**
  - **modularer Systemteil mit transparenter Kapselung seines Inhalts.**
  - **können aus mehreren Klassen, Paketen sowie weiteren Komponenten bestehen.**
  - **lassen sich ausschließlich über ihre öffentlichen Interfaces beschreiben.**
  - **Komponenten werden oft als eine .exe, .dll oder mittels (mehrerer) PHP Scripte realisiert.**
- **Interface**
  - **Schnittstelle, die eine Komponente anderen Komponenten zur Verfügung stellt.**
  - **Interfaces werden durch die Signaturen der öffentlich angebotenen Operationen beschrieben.**
  - **Interfaces werden später von einer konkreten Klasse implementiert.**

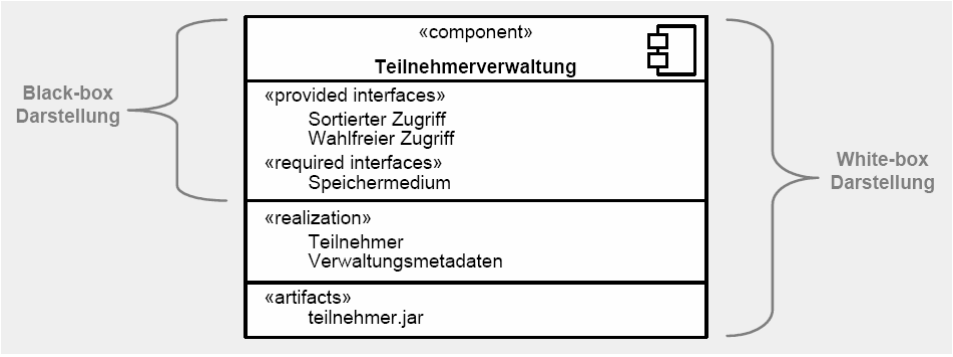
Beispiel Komponentendiagramm

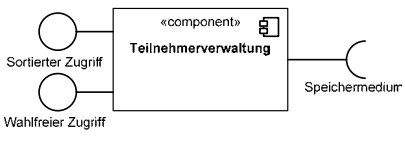


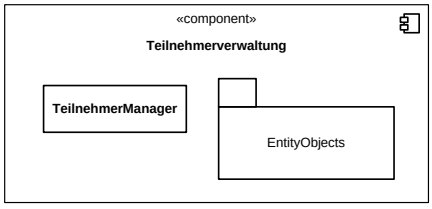
Beispiel Komponentenstruktur und Interfaces



Alternative Notationen für Komponenten





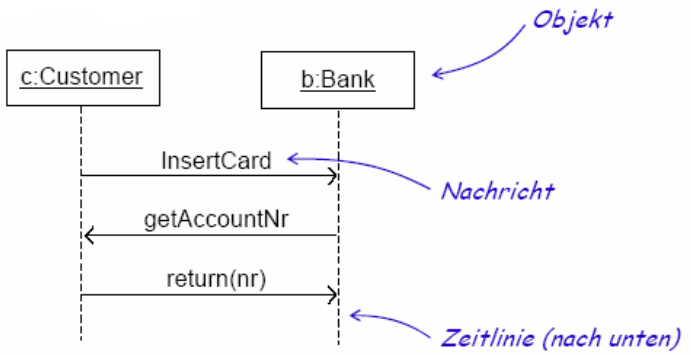


2.4 Interaktionsmodelle

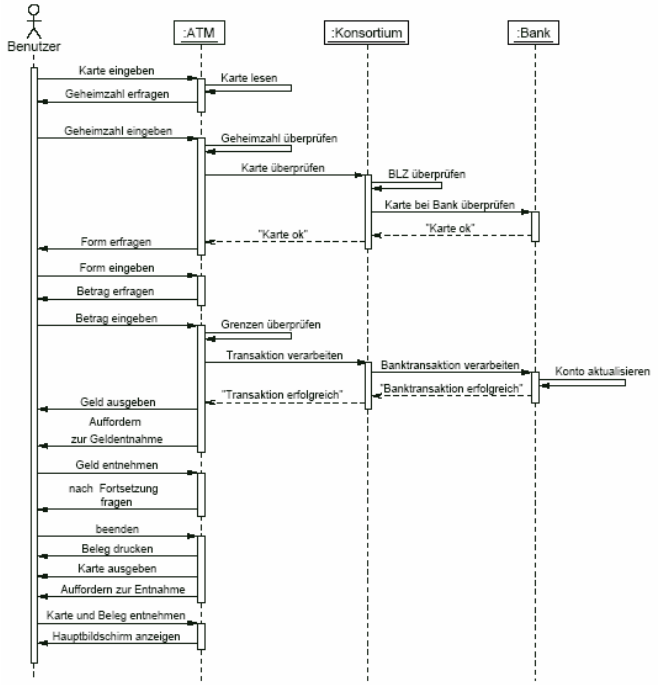
- **Interaktion = spezifisches Muster der Zusammenarbeit und des Nachrichtenaustauschs zwischen Objekten zur Erledigung einer bestimmten Aufgabe (z.B. eines Anwendungsfalls).**
- **UML bietet die Diagrammtypen Sequenz- und Kommunikations-diagramm zur Visualisierung implementierungsnaher Abläufe.**
  - **Wir verwenden Sequenzdiagramme.**
  - **Sequenzdiagramme sind exemplarisch und können daher keine vollständigen Verhaltensbeschreibungen sein.**
- **Die Diagramme dienen**
  - **der implementierungsnahen Visualisierung von Abläufen im System.**
  - **der Überprüfung ob die Objekte des Design-Klassenmodells korrekt zusammenarbeiten.**

UML Sequenzdiagramm

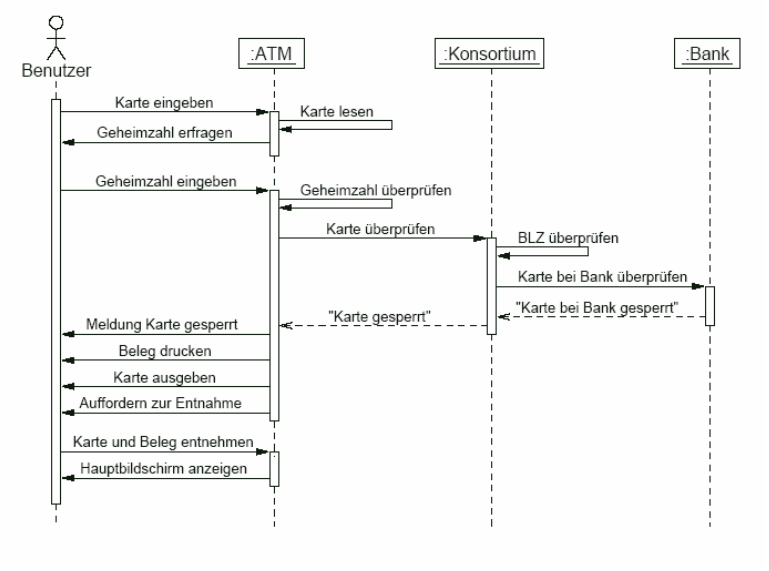
- **Sequenzdiagramme beschreiben den Nachrichtenaustausch zwischen mehreren Objekten.**
- **Heben die zeitliche Reihenfolge hervor, in der Nachrichten zwischen Objekten ausgetauscht werden.**
- **Nachrichten entsprechen Operationsaufrufen.**



Sequenzdiagramm: Geld abheben

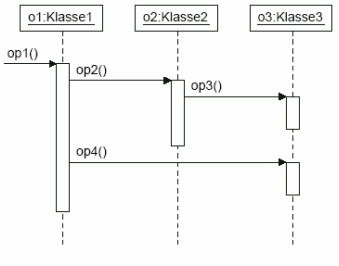


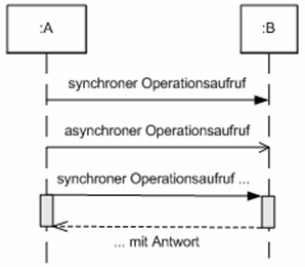
Sequenzdiagramm: Karte gesperrt



Nachrichten

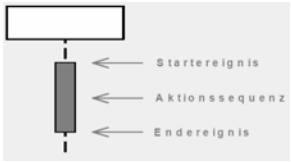
- **Nachrichten visualisieren den Informationsfluss zwischen Objekten durch Operationsaufrufe.**
- **Antwortnachrichten zeigen an,**
  - **wann der Kontrollfluss zum Aufrufer zurück geht.**
  - **welches Ergebnis dabei übertragen wird.**
- **Kommunikationsarten**
  - **synchron: Sender wartet auf Antwortnachricht.**
  - **asynchron: Sender fährt direkt nach Sendeereignis mit Abarbeitung fort ohne auf Empfängersignal zu warten.**

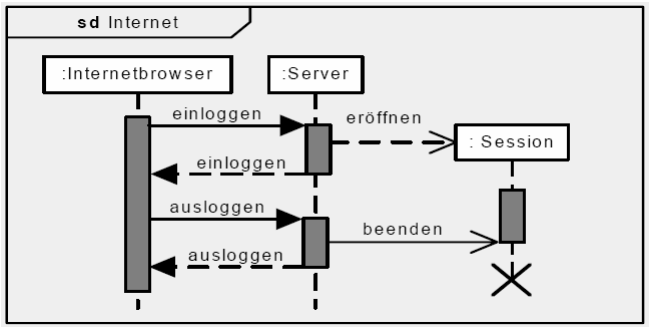




Aktionssequenz

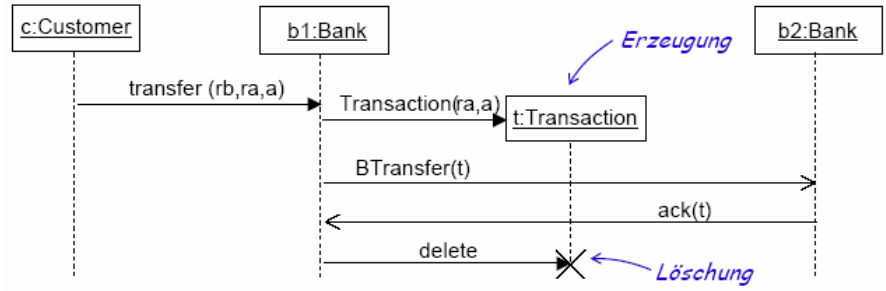
- **Zeitspanne, innerhalb der ein Objekt aktiv ist, weil**
  - **es gerade selbst tätig ist, oder**
  - **auf die Beendigung einer Aktivierung eines (anderen) Objekts wartet, dem es eine (synchrone) Nachricht gesendet hat ("Rückgabe des Steuerungsfokus")**





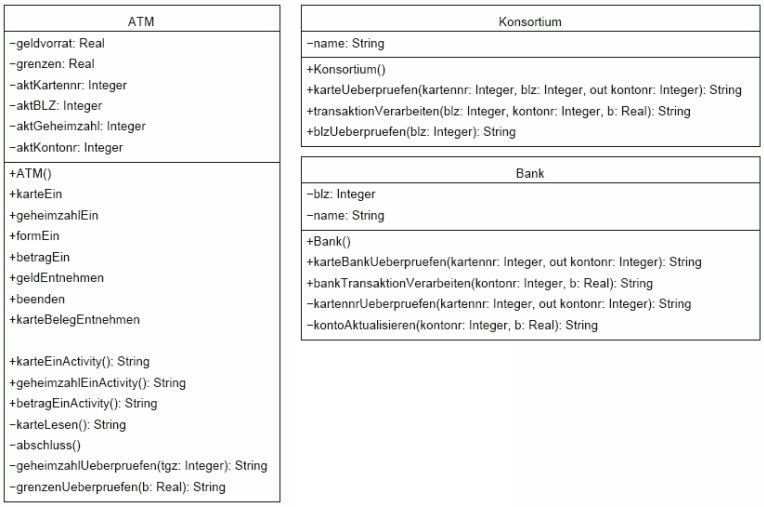
Objekterzeugung und Lösung

- Objekte die erzeugt werden, werden an der Erzeugungsstelle angegeben.
- Eine create()-Nachricht zeigt direkt auf das Objekt.
- Objektlöschung wird durch ein Kreuz am Ende der Zeitlinie angezeigt.



Übung 1: Transaktion gescheitert weil Konto leer

- Zeichnen Sie eine Sequenzdiagramm das den Ablauf einer gescheiterten Transaktion visualisiert.
  1. Start Operationsaufruf: ATM.betragEin(2000)
  2. Grenzen prüfen
  3. Konsortium Transaktion verarbeiten
  4. Bank Transaktion verarbeiten
  5. Rückmeldung an Nutzer



3. Leitlinien guter Entwurf

- Wie soll ich mein System nun tatsächlich modellieren?
  - Was für Klassen?
  - Welchen Klassen soll ich welche Operationen zuordnen?
  - Wie soll ich Klassen zu Paketen zusammenfassen?
  - Wie gliedere ich mein System in Komponenten?
- Leitlinien
  - Allgemeinen Entwurfsprinzipien
  - Entwurfsmuster
- Guter Entwurf setzt immer
  - Erfahrung und
  - gute Technologiekenntnis voraus!

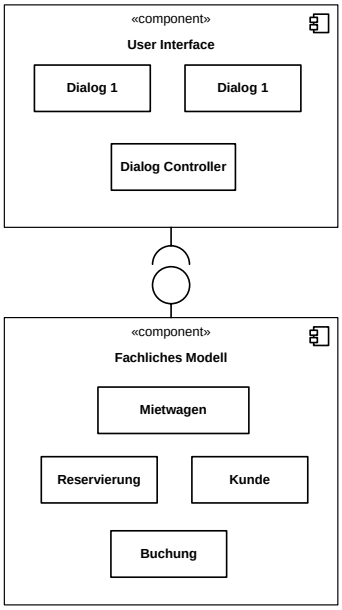
Hohe Kohäsion anstreben!

- Dinge die inhaltlich zusammengehören sollen in einer gemeinsamen Struktureinheit (Komponente, Paket, Klasse) zusammengefasst werden.
- Kohäsion
  - Kohäsion ist ein Maß für die Zusammengehörigkeit der Bestandteile eines Ganzen.
  - z.B. ein schwarzes Schaft verringert Kohäsion einer Herde weißer Schafe.
- Hohe Kohäsion von Komponenten, Paketen und Klassen erleichtert
  - das Verständnis,
  - die Wartung und
  - die Anpassung,
  - da alle betroffenen Elemente in der Struktureinheit zu finden sind.

Geringe Kopplung anstreben!

- **Es sollte immer eine möglichst geringe Kopplung zwischen Struktureinheiten angestrebt werden.**
- **Kopplung**
  - **Kopplung ist ein Maß für die Abhängigkeiten zwischen Struktureinheiten.**
  - **Niedrige Kopplung zwischen Komponenten und Paketen**
    - **erleichtert die Wartbarkeit, da Änderungen meist nur bzgl. einer Komponente.**
    - **ermöglicht den Austausch von Komponenten (z.B. verschiedene DBMS)**
  - **Strukturkopplung über Komponenten, Paketgrenzen hinweg vermeiden !**
    - **z.B. keine Vererbung über Paketgrenzen hinweg.**
  - **Schnittstellenkopplung ist akzeptabel**
    - **Klar definierte Schnittstellen, die von verschiedenen Komponenten implementiert werden können (z.B. ODBC).**
    - **öffentliche get/set-Methoden statt direktem Attributzugriff**

Beispiel: Hohe Kohäsion + Geringe Kopplung



- **Hohe Kohäsion:**
  - **Komponente A darf keine Information und Funktionalität enthalten, die zum Zuständigkeitsbereich von B gehört und umgekehrt.**
- **Geringe Kopplung:**
  - **Es muss möglich sein, Subsystem A weitgehend auszutauschen oder zu verändern, ohne Subsystem B zu verändern.**
  - **Änderungen von Subsystem B sollten nur möglichst einfache Änderungen in Subsystem A nach sich ziehen.**

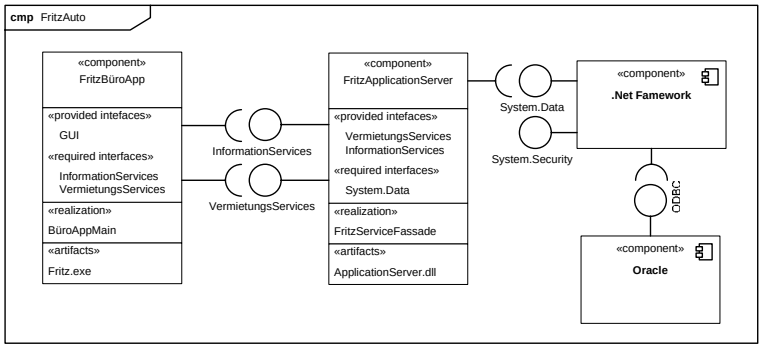
Weitere Leitlinien für einen guten Entwurf

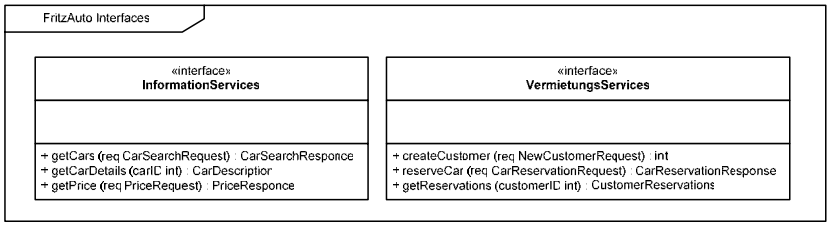
- **Wiederverwendung von Komponenten, Paketen und Klassen**
  - verringert Redundanz
  - erhöht Fehlerfreiheit und Stabilität
  - erhöht allerdings auch die Kopplung
  - erst einmal schauen, ob es Klasse im .Net Framework schon gibt bevor man sie selber programmiert.
- **Ressourcenschonung**
  - So designen, dass nicht unnötig Ressourcen (Speicher, Rechenleistung, Netzwerklast) beansprucht werden.
- **Verständlichkeit**
  - Verwendung aussagekräftiger Paket-, Klassen-, Operations- und Attributnamen.
  - Positivbeispiel: Rechnung.getRechungspositionen()
  - Negativbeispiele: paket1, ROj.opGRP
  - gute Dokumentation in den Diagrammen, später im Code

4. Gliederung Entwurfsdokument

1. **Komponentendiagramm**
   - Aufteilung des Systems in Komponenten
2. **Komponenten-spezifisches Entwurfsklassenmodell**
   - Verfeinerung des Analyseklassenmodells
   - alle Klassen inkl. aller Eigenschaften und Methoden
3. **Paketdiagramm**
   - Aufteilung des Klassendiagramms in Pakete
4. **Komponenten-spezifische Schnittstellendefinition**
   - Definiere ausgehend von den Anwendungsfällen für jede Komponente die notwendigen Schnittstellen.
5. **Sequenzdigramme zu jedem Anwendungsfall**
   - Visualisieren die Zusammenarbeit zwischen den Objekten
6. **Testfälle zu jedem Anwendungsfall**
   - Ermöglichen die spätere Verifikation der Implementierung

Komponentendiagramm und Schnittstellen-Spezifikation





Entwurf von Sequenzdiagrammen

- **Input**
  - **Use-Case Beschreibungen**
  - **Aktivitätsdiagramme**
  - **Entwurfs-Klassendiagramm**
- **Ziel**
  - **Modellierung der Zusammenarbeit von Objekten innerhalb eines Use-Cases.**
  - **Überprüfung des Entwurfs-Klassendiagramms**
- **Vorgehensweise**
  - **Identifiziere die Nachrichten, die innerhalb eines Use-Cases ausgetauscht werden und die Objekte, die die Nachrichten senden und empfangen. (Nachricht = Operationsaufruf)**
  - **Konstruiere Sequenzdiagramme für jeden Use-Case.**
    - **Standardablauf und**
    - **Wichtige Ausnahmen**

Definition von Test Cases

- **Definiere eine Reihe von Test Cases zu jedem Anwendungsfall.**
- **Testfall: Inputdaten mit Spezifikation des zu erwarteten Resultates.**
- **Darstellung als Tabelle:**

UC-1: Geld abheben		
Nr.	**Test**	**Resultat**
T1	Aufruf der Methode Konsortium.kartePrüfen() mit der ungültigen Kartennummer 5464 3563 435 2354	Rückgabewert „Karte gesperrt“
T2	Aufruf der methode ATM.grenzePrüfen() mit dem Werten 0 und 100 000 000.	Rückgabewert „Ungültiger Wert“
..	…	….