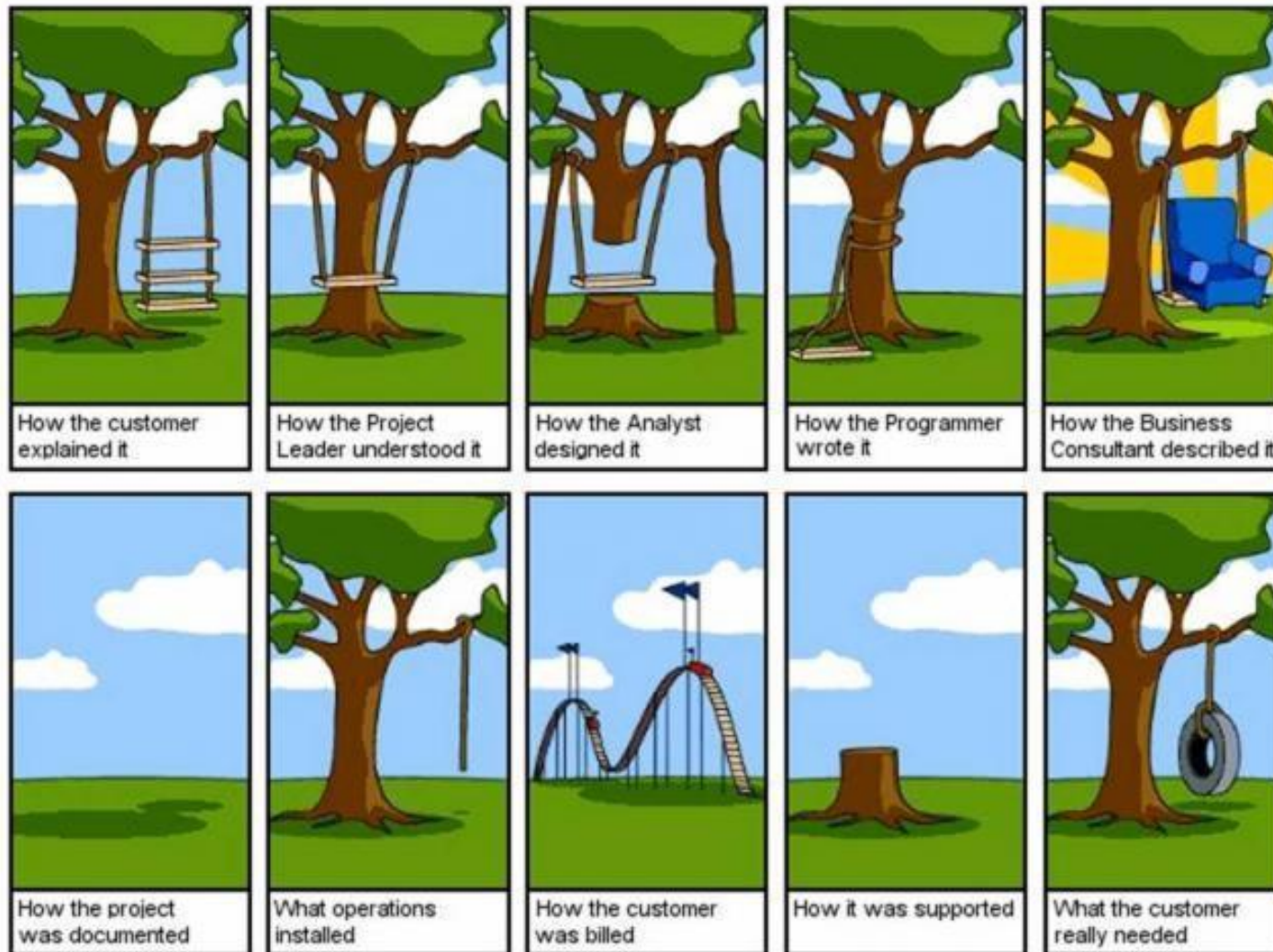


Softwaretechnik und Programmierparadigmen

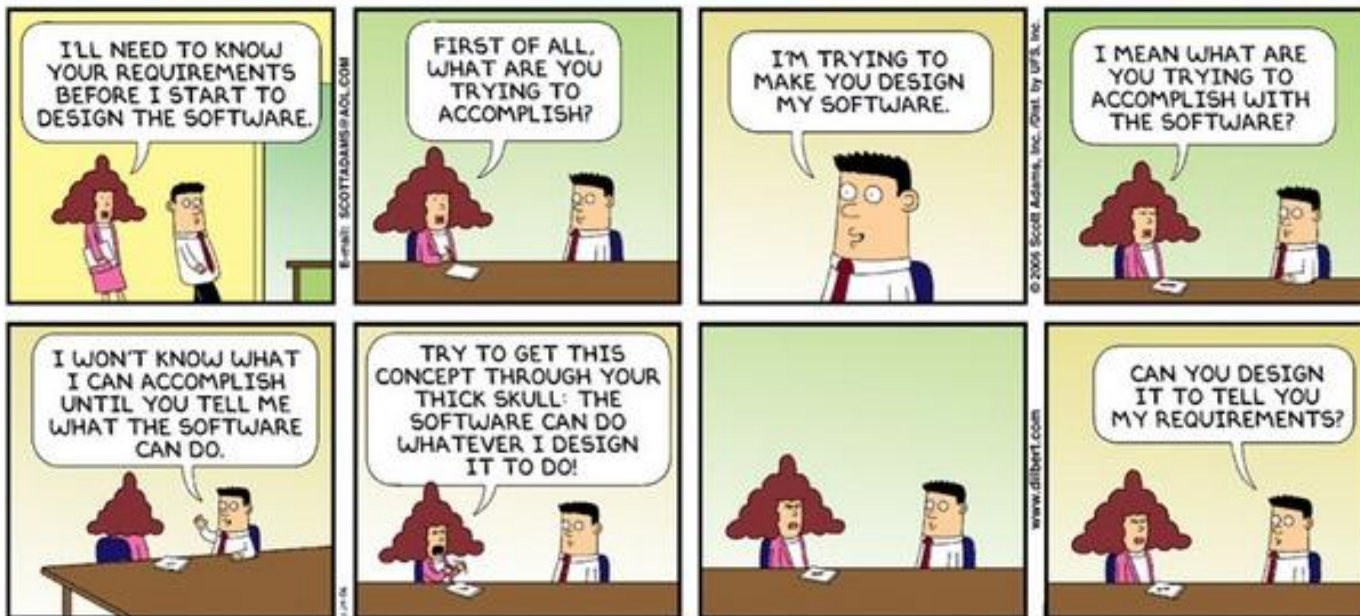
VL03: Requirements Engineering

Prof. Dr. Sabine Glesner
FG Programmierung eingebetteter Systeme
Technische Universität Berlin

Wozu Requirements?



Wozu Requirements Engineering?



© Scott Adams, Inc./Dist. by UFS, Inc.

Wozu Requirements Engineering?

- Wie erhalte ich Requirements vom Benutzer/Auftraggeber?
- Wie dokumentiere ich Requirements?
- Wie sollten Requirements aussehen?

Fallbeispiel für die Vorlesung (1)

- Onlineshop (Auszug!)

Sie werden gebeten für ein kleines Unternehmen, das Schuhe und Kleidung verkauft, die Verwaltungssoftware eines Online-Shops zu entwickeln. Der Onlineshop soll es dem Kunden ermöglichen, Produkte in einen Warenkorb zu legen und diesen zu bezahlen. Als Bezahlmethoden sind zunächst Bankeinzug und Kreditkartenzahlung vorgesehen. Bevor die Bestellung aufgegeben wird, muss sichergestellt werden, dass die Bezahlung tatsächlich erfolgen kann.

Weiterhin soll das System gleichzeitig auch die Mitarbeiter verwalten. Sowohl Kunden als auch Mitarbeiter sollen registriert werden können. Auf Sicherheit soll entsprechend geachtet werden.

Fallbeispiel für die Vorlesung (2)

Produkte sollen über das Webinterface auch gesucht werden können. Dabei sollen Rechtschreibfehler toleriert werden und innerhalb von einer akzeptablen Zeit sollen passende Produkte angezeigt werden.

Alle Funktionen sollen von Nicht-Entwicklern ausgiebig getestet werden.

Übersicht

- Kategorisierung von Anforderungen
- Beschreibung von Requirements
 - Natürliche Sprache
 - Strukturierte Natürliche Sprache
 - Design Beschreibungssprache Bsp. UML, für Hardware System-C
 - Mathematische Spezifikation
- Grafische Notation von Requirements
 - Übersicht UML
 - Use Case Diagramme
 - Sequenzdiagramme

Übersicht

- Kategorisierung von Anforderungen
- Beschreibung von Requirements
 - Natürliche Sprache
 - Strukturierte Natürliche Sprache
 - Design Beschreibungssprache
 - Mathematische Spezifikation
- Grafische Notation von Requirements
 - Übersicht UML
 - Use Case Diagramme
 - Sequenzdiagramme

User Requirements vs System Requirements

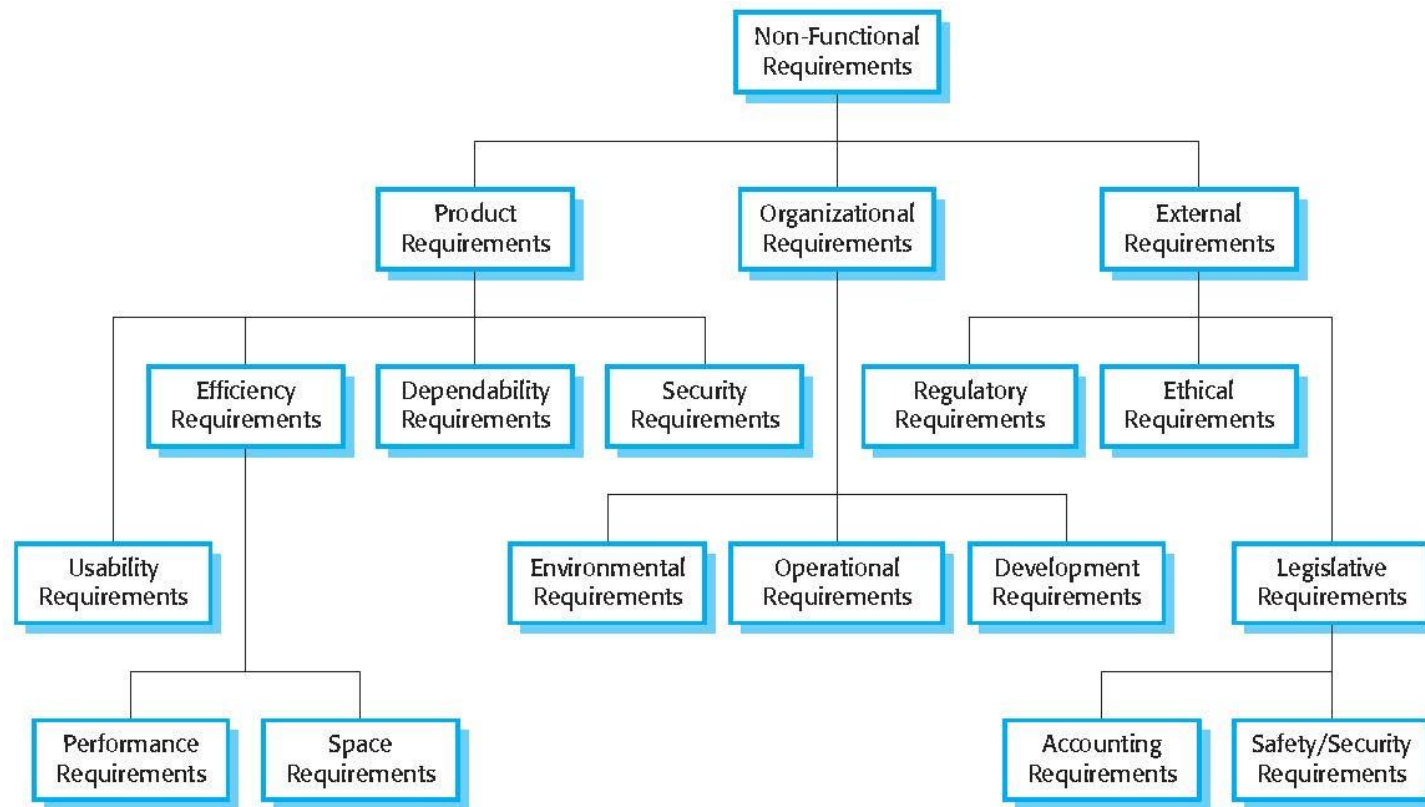
- User Requirements sind alle Anforderungen, die der Benutzer an das System als Blackbox stellt.
- System Requirements sind abgeleitete Requirements, die das System erfüllen muss, um die User Requirements zu erfüllen.

Funktionale und nicht-funktionale Requirements

- Funktionale Anforderungen beschreiben, **was** das System bei bestimmten Eingaben (ggf. nicht) tun soll. Ein- und Ausgabewerte (wie bei mathematischer Funktion)
- Nicht-funktionale Anforderungen beschreiben, **wie** (im Sinne von Eigenschaften) das System bestimmte Dinge machen sollte. Oft betreffen nicht-funktionale Eigenschaften nicht nur einzelne Module, sondern betreffen das gesamte Systeme.

Bps: Das System soll immer Antwortzeiten <10ms haben

Beispiele: Nicht-funktionale Anforderungen

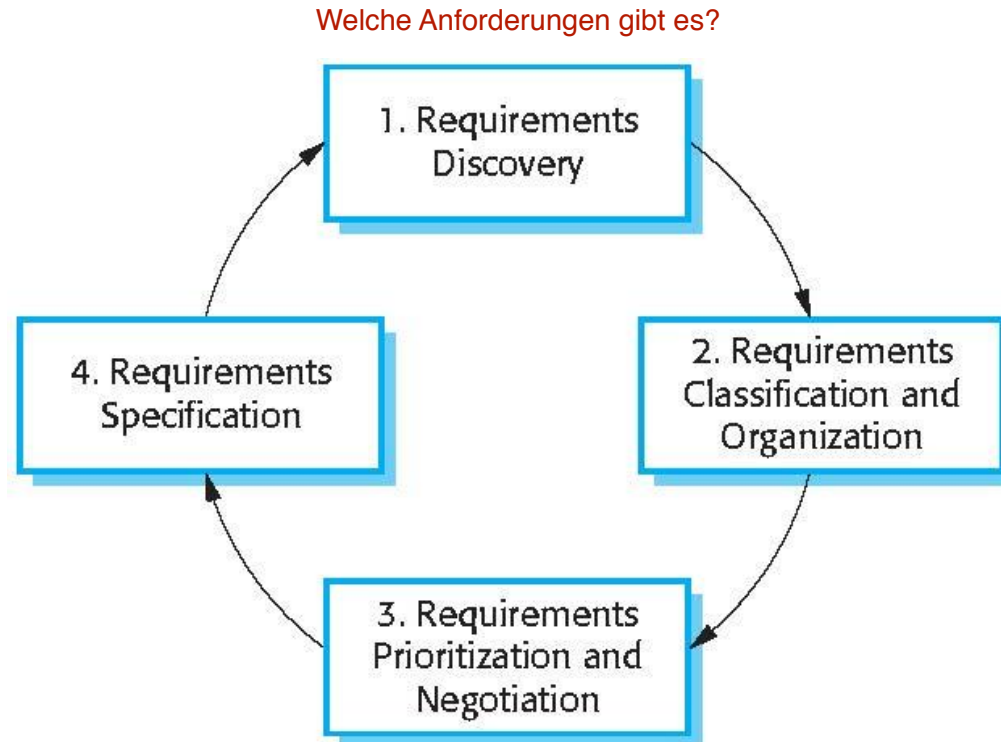


[Sommerville]

Beispiele für (nicht-)funktionale Anforderungen an den Online-Shop

- Funktionale Anforderungen
 - Webinterface für Online-Shop
 - Anlegen eines Warenkorbs
 - Bezahlen mit EC- oder Kreditkarte
- Nicht-funktionale Anforderungen
 - „Auf Sicherheit soll geachtet werden“
 - Suchergebnisse sollen in „akzeptabler Zeit“ vorliegen
 - Testen von einem unabhängigen Team

Requirements Engineering Prozess



Priorisierung der Anforderungen,
eventuell Verhandeln

[Sommerville]

Requirements Engineering Prozess

- Requirements discovery: Sammeln von Anforderungen von allen Projektbeteiligten
- Requirements classification and organization: Gruppieren von Anforderungen, Identifizierung von Subsystemen
- Requirements prioritization and negotiation: Priorisierung von Anforderungen, Kompromissfindung bei Konflikten
- Requirements specification: Dokumentierung von Anforderungen (formal oder informell)

Dokumentation von Requirements

- Qualitätskriterien der „Software Requirements Specification“ gemäß ANSI/IEEE 830-1984
 - Korrekt
 - Eindeutig
 - Vollständig
 - Konsistent
 - Bewertet nach Wichtigkeit
 - Verifizierbar
 - Modifizierbar
 - Verfolgbar

Verifizierbarkeit von Requirements

- Funktionale Requirements können meistens unmittelbar geprüft zu werden: Prüfen, ob Funktionalität vorhanden ist oder nicht
- Nichtfunktionale Requirements können nicht immer direkt überprüft werden

Beispiele


- „ Sowohl Kunden als auch Mitarbeiter sollen registriert werden können. **Auf Sicherheit soll entsprechend geachtet werden.**“
- „Dabei sollen Rechtschreibfehler toleriert werden und innerhalb von einer **akzeptablen Zeit** sollen passende Produkte angezeigt werden.“

-> nicht eindeutig!

Beispiele (verbessert)

- „ Sowohl Kunden als auch Mitarbeiter sollen registriert werden können. **Es soll über Authentifizierungsmethoden sichergestellt werden, dass nur berechtigte Personen neue Mitarbeiter anlegen können.**“
- „Dabei sollen Rechtschreibfehler toleriert werden und innerhalb von **höchstens 5 Sekunden** sollen passende Produkte angezeigt werden. **Gegebenenfalls soll nur ein unvollständiger Teil der möglichen Ergebnisse angezeigt werden.**“

Requirements Document

- Besteht im wesentlichen aus
 - C-Requirements (Customer Requirements, auch User Requirements - Lastenheft)
 - D-Requirements (Development Requirements, auch System Requirements - Pflichtenheft)
- Lastenheft
 - Anforderungen aus Sicht des Kunden/ Endanwenders
 - „Was soll die Software können“
- Pflichtenheft
 - Anforderungen aus Sicht des ~~Auftraggebers~~ ^{Auftragnehmers} 
 - „In welchem Umfang und unter welchen Bedingungen wird die Software eingesetzt.“

Übersicht

- Kategorisierung von Anforderungen
- Beschreibung von Requirements
 - Natürliche Sprache
 - Strukturierte Natürliche Sprache
 - Design Beschreibungssprache
 - Mathematische Spezifikation
- Grafische Notation von Requirements
 - Übersicht UML
 - Use Case Diagramme
 - Sequenzdiagramme

Natürlichsprachliche Requirements

- Ausführliche textuelle Beschreibung der Anforderungen an ein System bzw. eine Funktionalität
- Problem: Häufig entsteht zu viel Interpretationsspielraum, da natürliche Sprache nicht eindeutig ist.

Vorteil: Jeder versteht es

Nachteil: Interpretationsspielraum

Beispiel

- Der zu entwickelnde Online-Shop soll es ermöglichen Produkte in den Warenkorb abzulegen. Wenn der Kunde sich entschließt den Warenkorb zu kaufen, werden seine Zahlungsinformationen abgefragt. Dabei soll der Online-Shop mindestens die Zahlungsweise „EC“-Karte oder „Kreditkarte“ anbieten.

Strukturierte Requirements

- Bessere Erfassung von Anforderungen im Vergleich zu natürlicher Sprache
- Tabellarische Erfassung von Requirements mit einheitlichen Eckdaten wie z.B.
 - Funktion
 - Beschreibung
 - Inputs
 - Outputs
 - Aktion
 - Pre-Condition
 - Post-Condition

Hoare-Kalkül:

Für ein Stück Code kann eine Vorbedingung festgelegt werden.

Aus der Erfüllung der Vorbedingung (und Korrektheit des Codes) folgt die Nachbedingung.

Auch: Suche nach Nachbedingungen, die genauso stark sind, dass sie von den gewünschten Vorbedingungen impliziert werden (aber nicht zu stark oder schwach)

Beispiel

- Funktion: Bezahlen
- Beschreibung: Auswahl zwischen Zahlungsmethode „EC-Karte“ und „Kreditkarte“ und Abbuchen des entsprechenden Betrags
- Inputs: Summe des Warenwerts der Produkte des Warenkorbs
- Outputs: Nachricht, ob Bezahlung erfolgreich war oder nicht
- Aktion: Abbuchung bei der Bank initiieren; Versand vorbereiten
- Pre: Kunde ist registriert
- Post: Warenkorb ist geleert und Bestelldaten werden hinterlegt

Mathematische Beschreibung von Requirements

- Anforderungsbeschreibungen können missverständlich sein
- Natürliche Sprache lässt Interpretationsspielraum zu
- Lösung: Nutzen eines mathematischen Formalismus zur Beschreibung der Anforderung
- Formalismen: Automaten, (Object) Z, OCL, Prozesskalküle, ...
- Details in späterer Vorlesung

Übersicht

- Kategorisierung von Anforderungen
- Beschreibung von Requirements
 - Natürliche Sprache
 - Strukturierte Natürliche Sprache
 - Design Beschreibungssprache
 - Mathematische Spezifikation
- Grafische Notation von Requirements
 - Übersicht UML
 - Use Case Diagramme
 - Sequenzdiagramme

UML: Einleitung

Formalisierung intuitiver
Skizzen von
Programmzusammenhängen

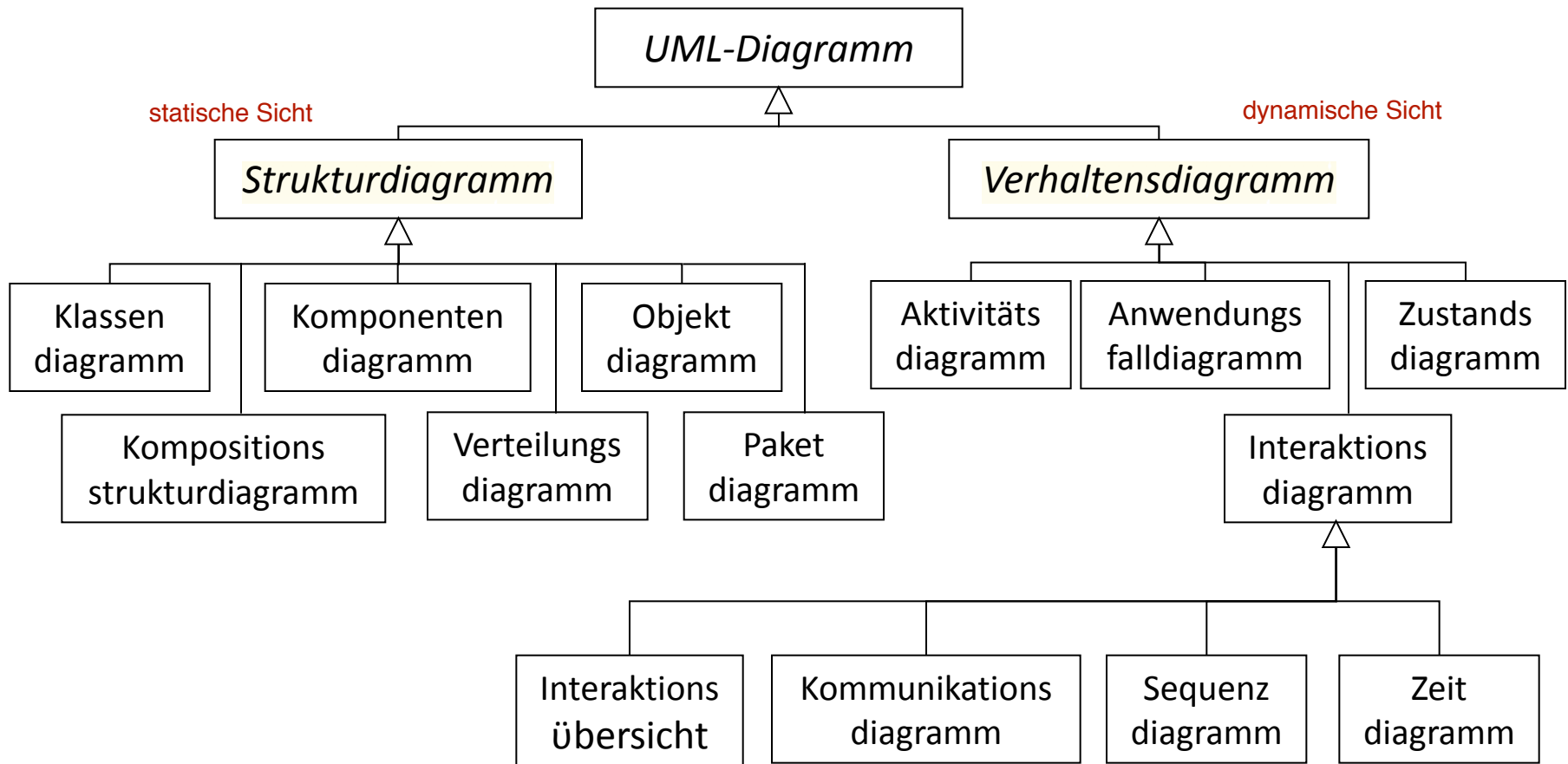
- UML = Unified Modeling Language
 - Erste Ansätze Mitte der 90er von Grady Booch, James Rumbaugh und Ivar Jacobson (Rational Software Cooperation)
 - Ziel: „Unified Method“ verschiedener objektorientierter Modellierungsmethoden Fokus auf objektorientierten Sprachen, nicht für funktionale Sprachen
 - als einheitliche Modellierungssprache seit 1997 von der OMG (Object Management Group) standardisiert
 - Aktuelle Version (seit Oktober 2009): UML 2.2

UML: Spezifikation

- UML Infrastructure:
 - Definiert den Sprachkern der UML (z.B. Konzepte wie Klasse, Assoziation, Attribut und Methode)
 - Erweiterbar durch Erweiterungsmechanismen auf Nutzerebene und Profile
- UML Superstructure:
 - Erweitert den Sprachkern auf den vollständigen UML-Sprachumfang
 - Definiert Modellelemente, Notationen und Diagrammtypen
 - Definiert welche Eigenschaften Sprachelemente haben dürfen und welche Beziehungen zwischen Sprachelementen zulässig sind
- UML Object Constraint Language: separat von den anderen Bestandteilen von UML
 - Zur Spezifikation von Invarianten und Bedingungen
 - Metamodell-basierte Definition
 - Konsistent zum UML-Metamodell

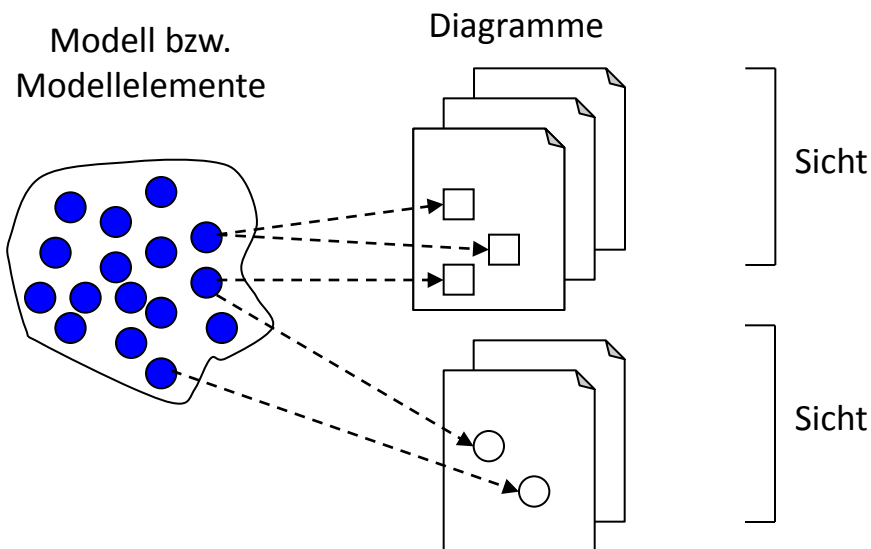
Invarianten: Bedingungen, die immer gelten (außer während einer Transaktion/Aktion/Schleifendurchlauf)

UML: Diagrammübersicht



UML: Diagramme

- UML ist eine visuelle Sprache zur Spezifikation, Konstruktion und Dokumentation der technischen Aspekte von Systemen
- UML definiert eine Menge von Diagrammtypen
- Mehrere Diagramme können gemeinsam eine Sicht auf ein UML-Modell definieren



Flexibilität in UML

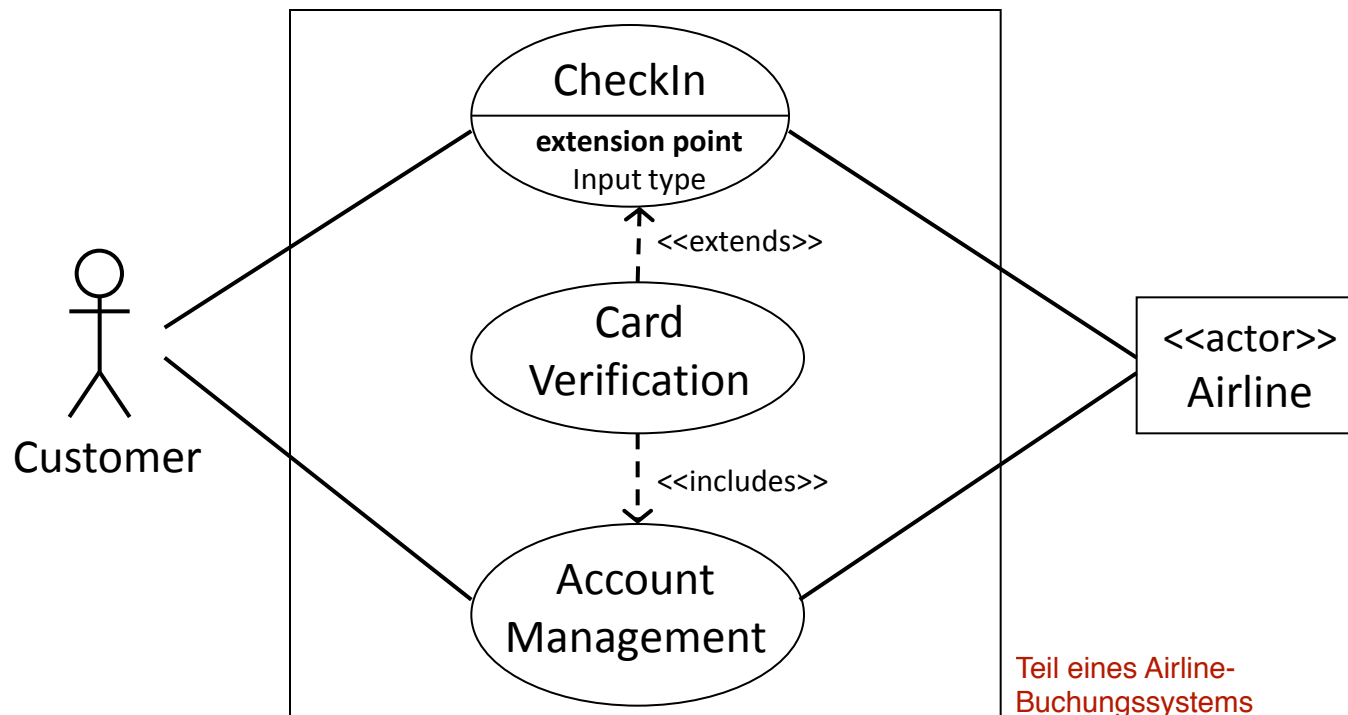
- Durch unterschiedliche Perspektiven auf ein System:
UML-Modell hat keinen Vollständigkeitsanspruch
 - Dass bestimmte Modellteile nicht aufgeführt werden heißt nicht, dass sie nicht da sind!
- Modelle können schrittweise erweitert werden

Verhaltensmodellierung mit UML

- Verhaltensmodellierung:
- Beschreibung dynamischer Aspekte
- Verhalten ist beobachtbar in Veränderungen
 - der Eigenschaften der beteiligten Elemente (Zustandsänderungen)
 - der Struktur des Gesamtsystems
- Die Veränderungen werden durch Ereignisse ausgelöst
- Grundformen der Verhaltensbeschreibung zur Unterstützung verschiedener Sichten auf das Verhalten:
 - Anwendungsfälle
 - Zustandsautomaten
 - Aktivitäten
 - Interaktionen

Use Case Diagramme

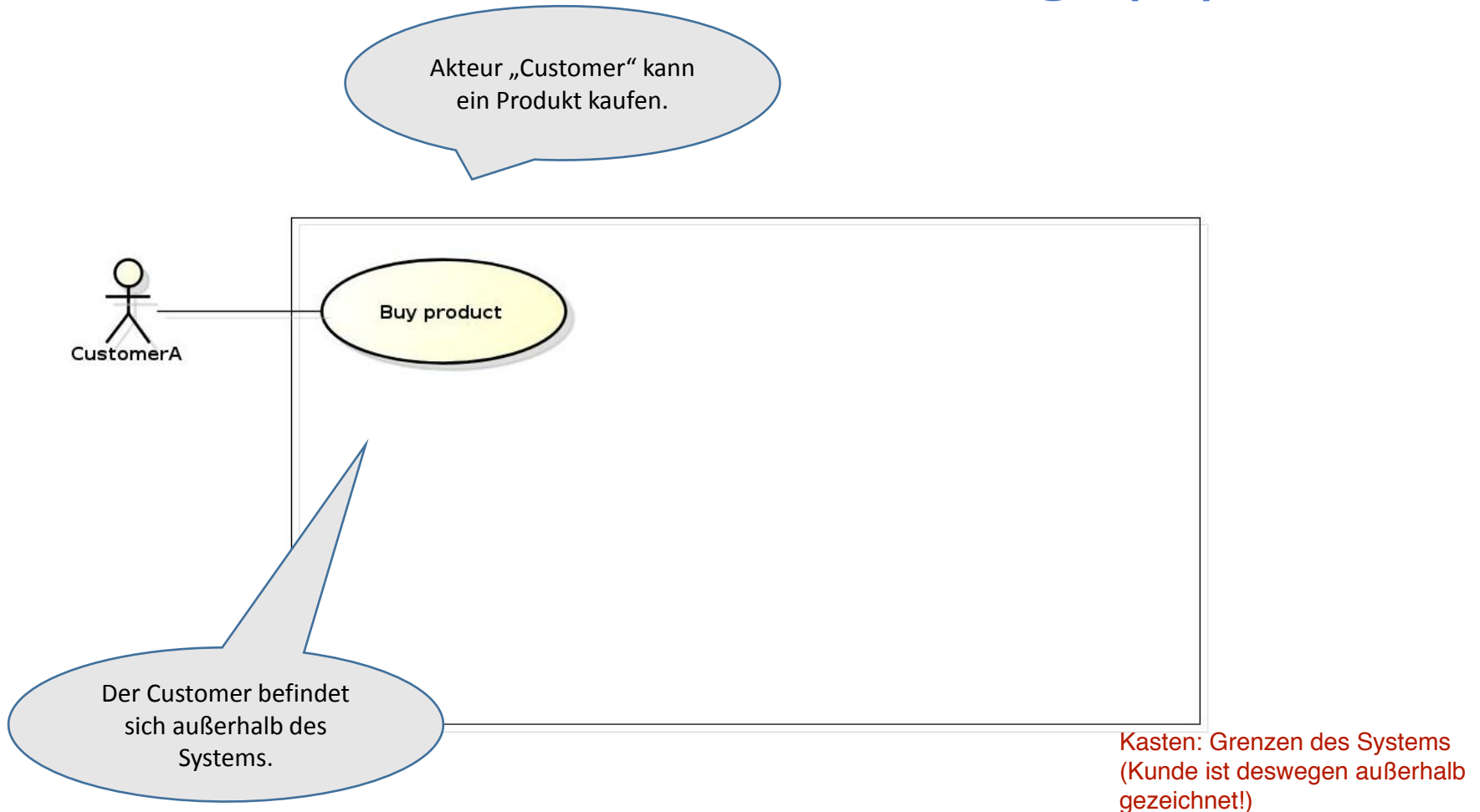
- Anwendungsfälle (Use Cases) sind sichtbares Verhalten und dienen der Spezifikation von Anforderungen an das System
- Anwendungsfälle werden von Akteuren angestoßen (externe Nutzer des Systems)



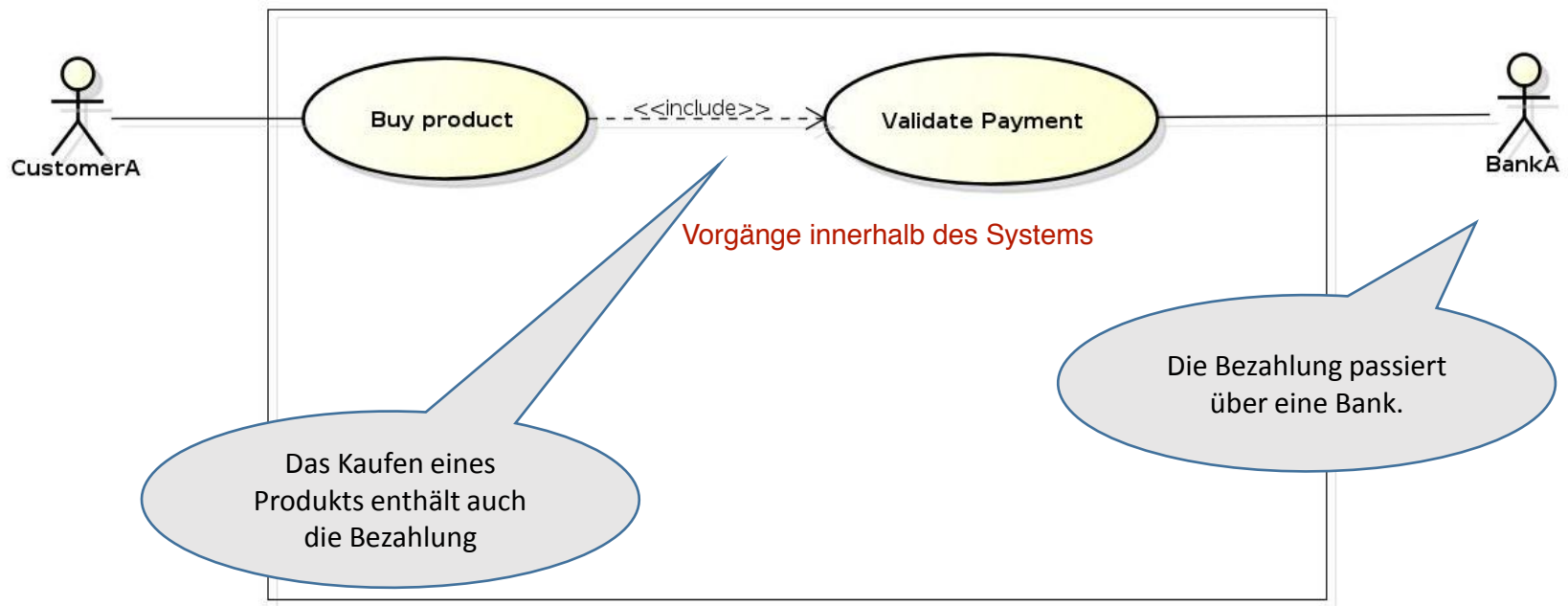
Use Case Diagramme

- Grafische Erfassung von Akteuren und Anwendungsfällen
- Bestandteil der UML
- Modellierungselemente
 - Akteure
 - Kommunikationslinien
 - ggf. Generalisierung, <<extend>>, <<include>>
 - Anwendungsfälle
 - ggf. mit Erweiterungspunkten, zusätzlichen Bedingungen
- Grundlage für detailliertere Sequenzdiagramme

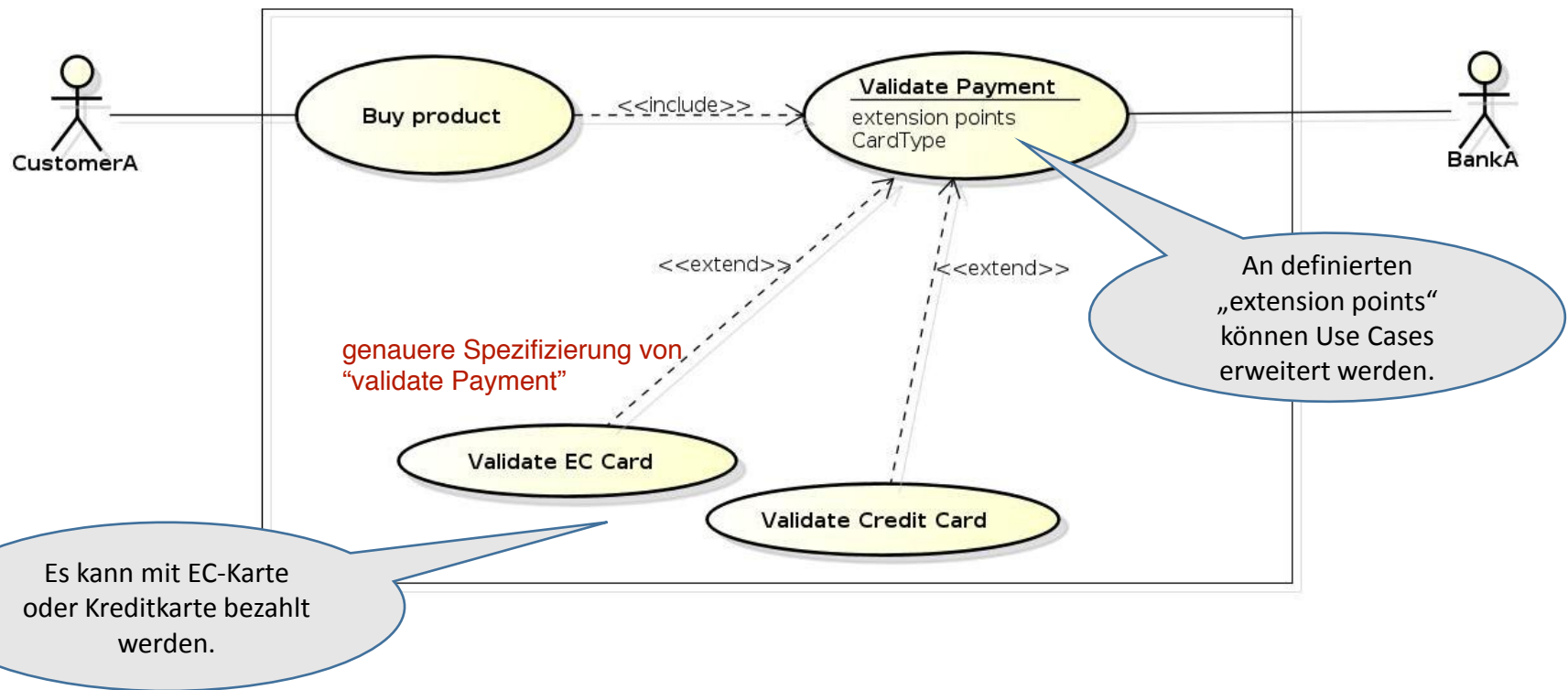
Use Case Modellierung (1)



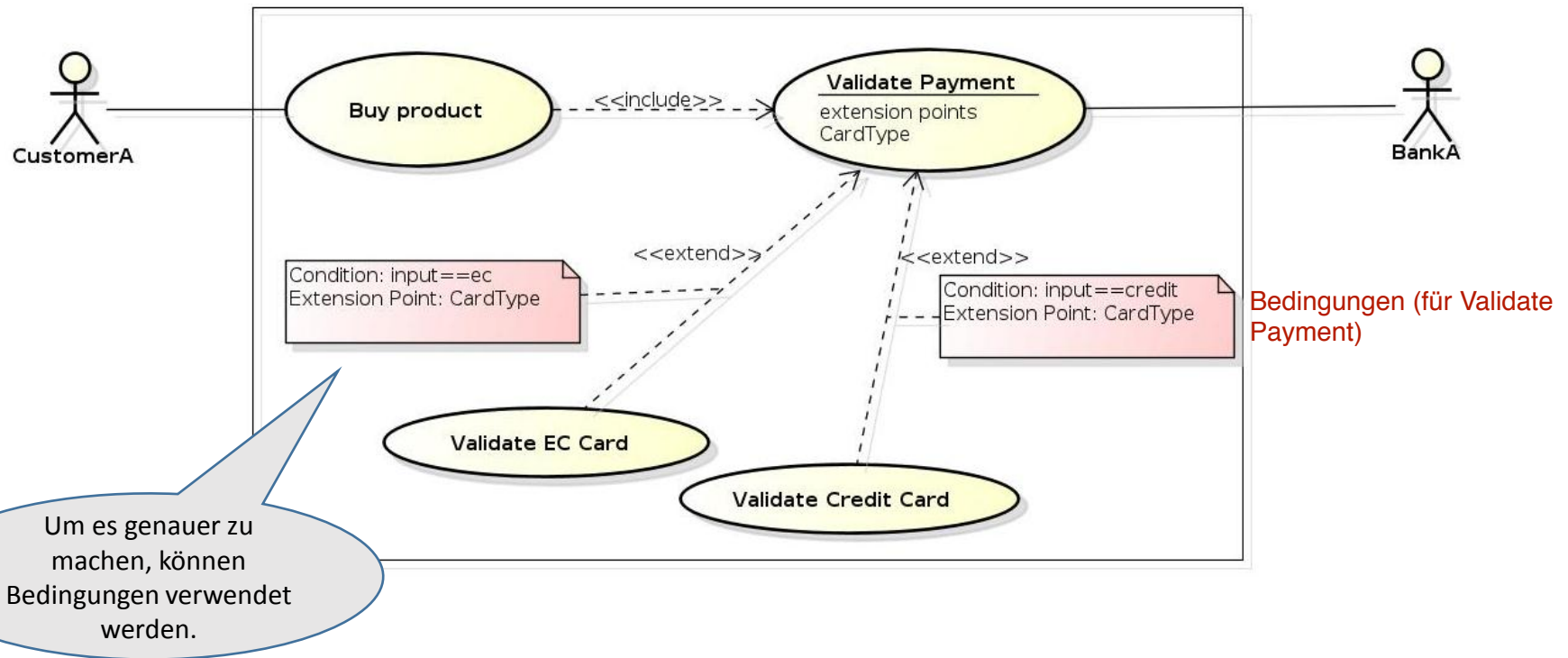
Use Case Modellierung (2)



Use Case Modellierung (3)



Use Case Modellierung (4)



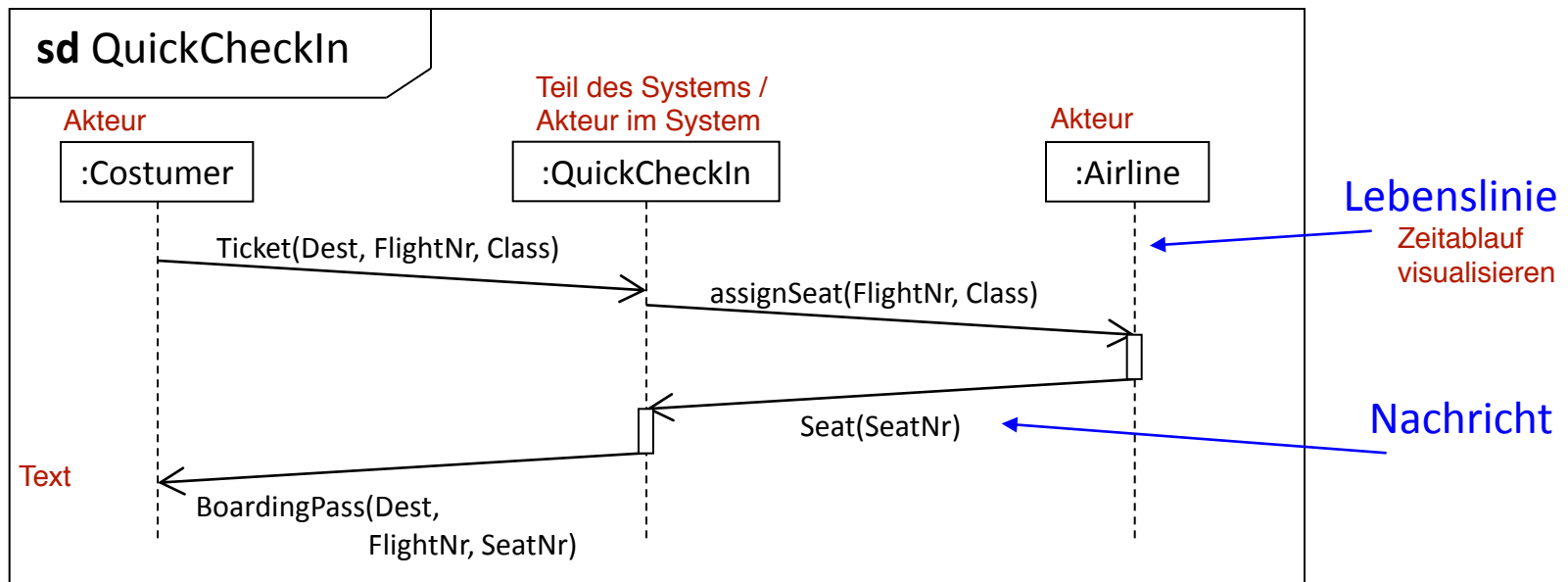
Sequenzdiagramme

- Beschreibung von typischen Szenarien von Anwendungsfällen
- Bestandteil der UML
- Grundlage für spätere detailliertere Sequenzdiagramme, die die Interaktion mit System-Objekten darstellen

Sequenzdiagramme

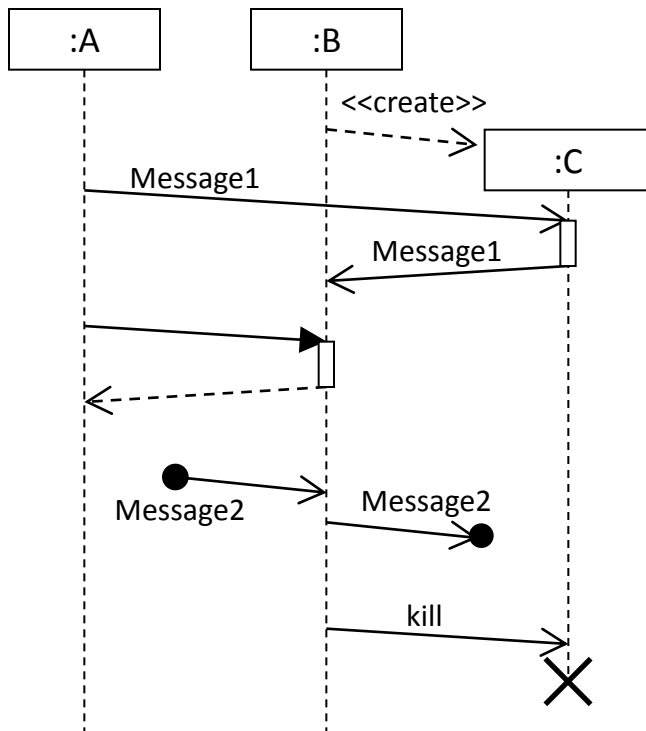
Systemverhalten darstellen

- Sichtbares Verhalten als Kommunikationssequenz
- Exemplarisch auf der Grundlage von konkreten Instanzkonfigurationen
- Darstellung gültiger oder nicht zulässiger Abläufe



Sequenzdiagramme

- Nachrichtenformen in Sequenzdiagrammen:

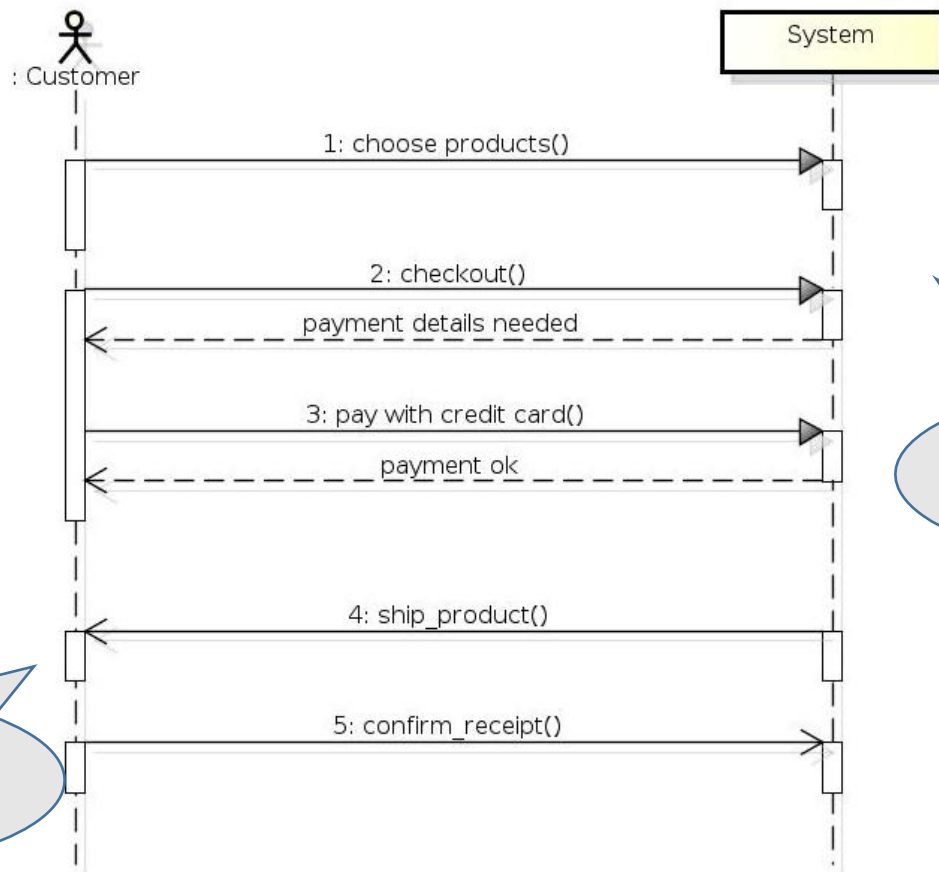


- } Instanzerzeugung
- } Asynchroner Nachrichtenaustausch
^v wichtig: unterschiedliche Pfeiltypen!
- } Synchroner Nachrichtenaustausch
- } gefundene & verlorene Nachrichten
- } Instanzzerstörung

Modellierungselemente

- Akteure/ Objekte mit Timeline
- Nachrichten
 - synchron
 - asynchron
- Kombinierte Fragmente
 - Alternativen (alt)
 - Schleifen (loop)
 - ...
- Objekterzeugung/-zerstörung

Beispiel (1)



Synchrone
Kommunikation

Asynchrone
Kommunikation

Beispiel (2)

Schleife

System

: Customer

loop

1: choose_product()

2: checkout()

payment details needed

Block mit Alternativen

alt

3: pay with credit card()

payment ok

4: pay with ec card()

payment ok

erste Alternative

zweite Alternative

Zusammenfassung

- Requirements müssen sorgfältig erfasst werden, um „richtiges“ Produkt entwickeln zu können
- Qualitätskriterien für Requirements-Beschreibung einhalten
- Beschreibungsmöglichkeiten
 - Natürliche Sprache
 - Strukturierte natürliche Sprache
 - Mathematische Beschreibung
 - Grafische Beschreibung
- Use Cases
- Sequenzdiagramme