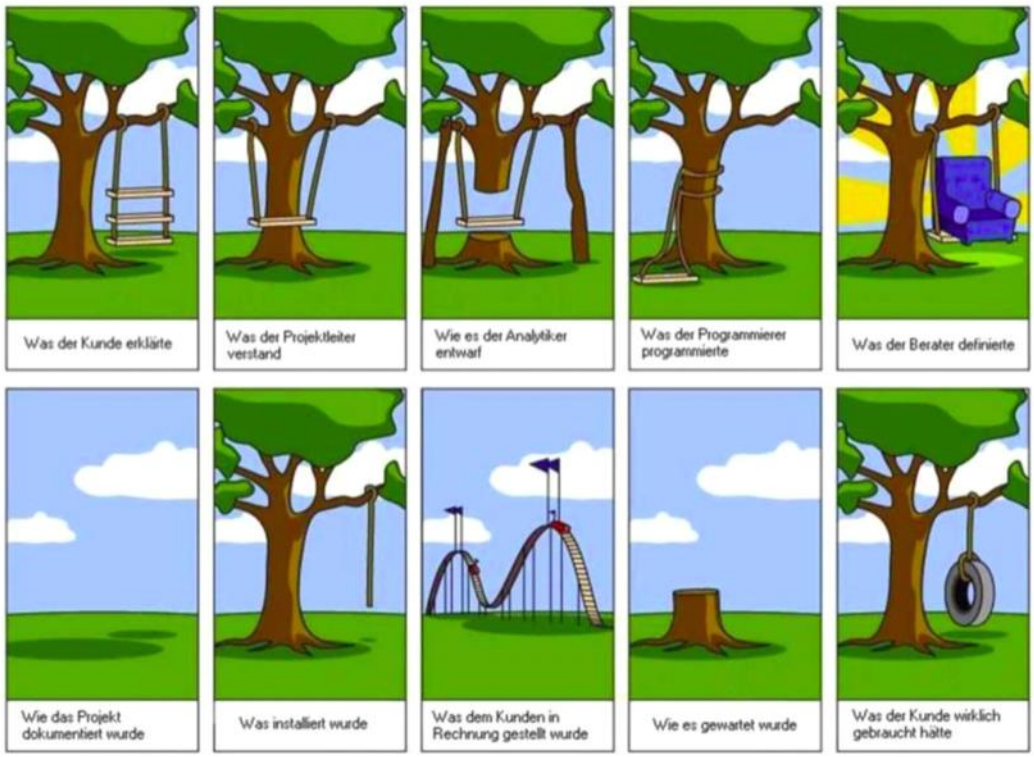
Ausdruck

Inhaltsverzeichnis

Einführung in UML

Überblick



AUFTRAG

Arbeitsauftrag

Betrachten Sie das obenstehende Bild und diskutieren Sie in der Gruppe über die Aussage des Bildes. Halten Sie Ihre Ergebnisse stichpunktartig fest!

8. September 2020

Die Unified Modeling Language (vereinheitlichte Modellierungssprache), kurz UML, ist eine grafische Modellierungssprache zur Spezifikation, Konstruktion und Dokumentation von Software-Teilen und anderen Systemen. Sie wird von der Object Management Group (OMG) entwickelt und ist sowohl von ihr als auch von der ISO (ISO/IEC 19505 für Version 2.4.1 standardisiert. Im Sinne einer Sprache definiert UML dabei Bezeichner für die meisten bei einer Modellierung wichtigen Begriffe und legt mögliche Beziehungen zwischen diesen Begriffen fest. UML definiert weiter grafische Notationen für diese Begriffe und für Modelle statischer Strukturen und dynamischer Abläufe, die man mit diesen Begriffen formulieren kann. (Quelle: Wikipedia)

Sie ist damit für die objektorientierte Programmierung das, was Struktogramme / Programmablaufpläne und ähnliche Konzepte für die strukturierte Programmierung war,

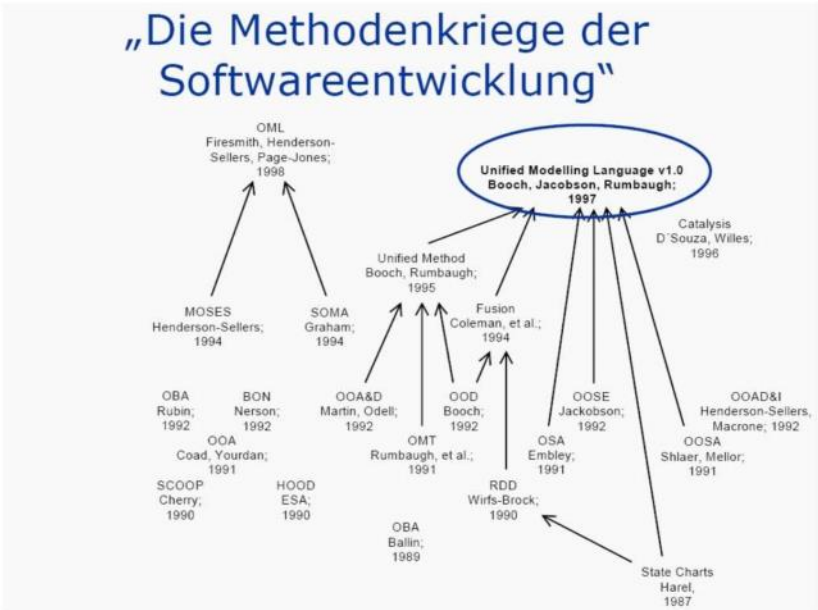
- Sie ist die verbreiteste Notation, um Softwaresysteme zu analysieren und zu entwerfen
- Sie dient zur Spezifizierung und Visualisierung komplexer Softwaresysteme unabhängig vom Fach- und Realisierunggebiet.
- Sie liefert die Notationselemente gleichermaßen für die statischen und dynamischen Modelle von Analyse, Design und Architektur und unterstützt objektorientierte Vorgehensweisen.

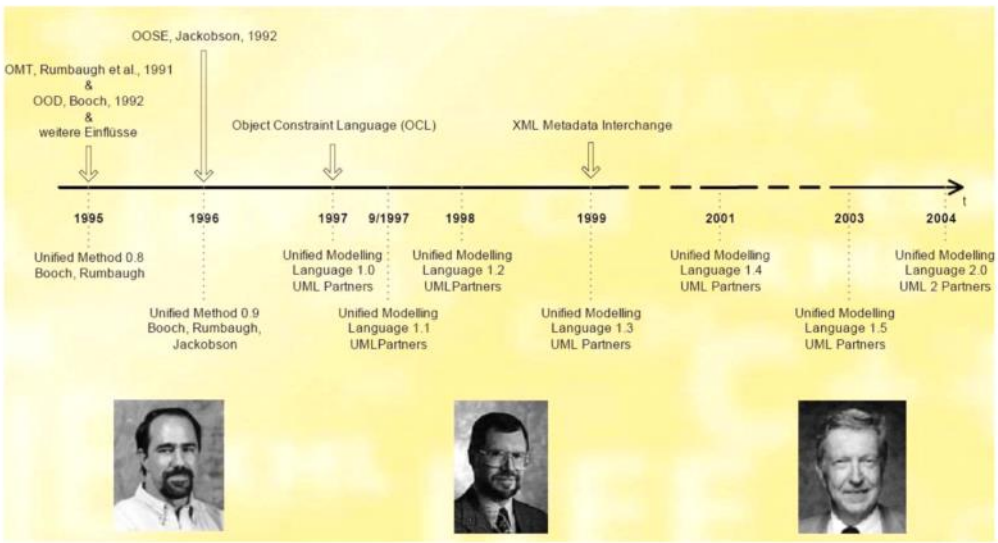
Die UML ist

- nicht perfekt
- nicht vollständig
- keine rein formale Sprache
- nicht spezialisiert auf ein Anwendungsgebiet
- kein vollständiger Ersatz für Textbeschreibung
- keine Methode oder ein Vorgehensmodell

UML - historisch gesehen

Im Zuge der Entwicklung objektorientierter Programmiersprachen und deren Möglichkeiten, kam es besonders zum Anfang zu einer Vielzahl unterschiedlicher Notationsmodelle. Um diese zu vereinheitlichen, setzten sich zu Beginn der 90er Jahre mehrere Vertreter unter Führung der Firma RATIONAL zusammen, um eine gemeinsame Sichtweise (Unified) zu schaffen. Diese Arbeiten mündeten dann 1997 in die Verabschiedung der UML-Version 1.0.



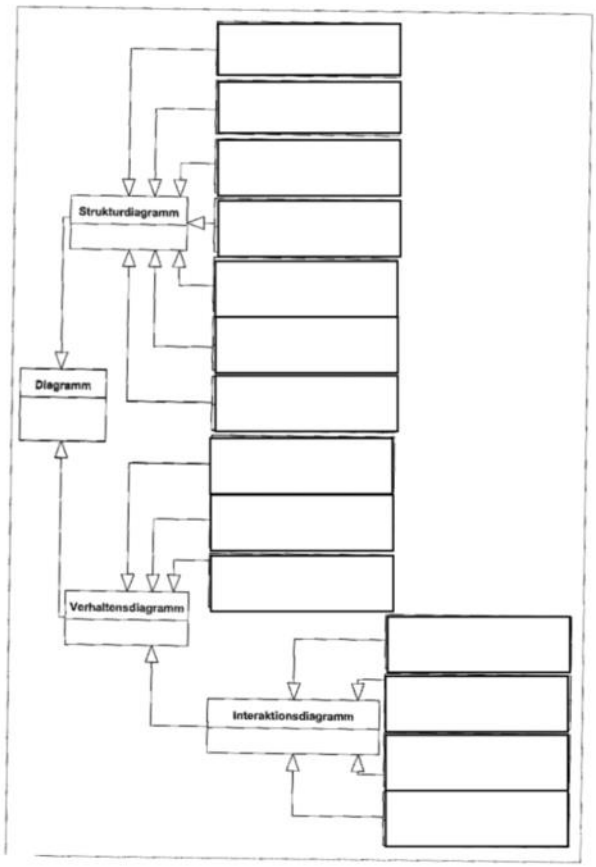


UML-Diagramme im Überblick

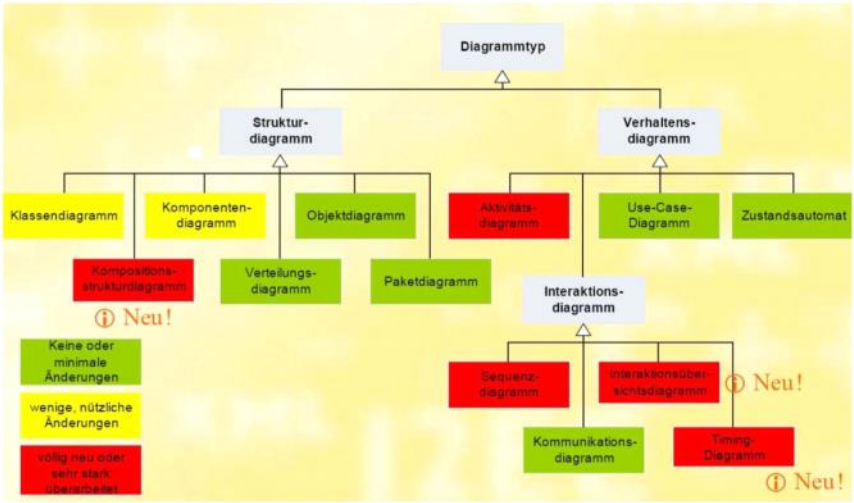
Die bisher 7 Diagramme der UML 1.x - Notation wurden in der Version 2.x auf 13 Diagramme erweitert. Darüber hinaus wurden manche (alte) Diagramme mit neuen Inhalten versehen.

Arbeitsauftrag

Die Unified Modeling Language (UML) bietet verschiedene Diagramme zur Spezifikation, Konstruktion und Dokumentation von Software(-teilen). Die untenstehende Übersicht zeigt die verschiedenen Diagrammarten. Sortieren Sie die aufgeführten Diagramme ihren übergeordneten Diagrammarten zu.



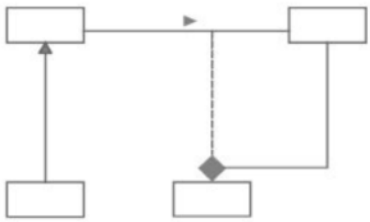
Verteilungsdiagramm – Sequenzdiagramm – Profildiagramm – Komponentendiagramm – Zustandsdiagramm – Aktivitätsdiagramm – Klassendiagramm – Kommunikationsdiagramm – Paketdiagramm – Timing-Diagramm – Kompositionsstrukturdiagramm – Interaktionsübersichtsdiagramm – Anwendungsfalldiagramm – Objektdiagramm


Diagrammtyp
Struktur-diagramm
Verhaltens-diagramm
Klassendiagramm
Komponenten-diagramm
Objektdiagramm
Aktivitäts-diagramm
Use-Case-Diagramm
Zustandsautomat
Kompositions-strukturdiagramm
Verteilungs-diagramm
Paketdiagramm
Interaktions-diagramm
Neu!
Keine oder minimale Änderungen
wenige, nützliche Änderungen
völlig neu oder sehr stark überarbeitet
Sequenz-diagramm
Interaktionsüber-sichtsdiagramm
Neu!
Kommunikations-diagramm
Timing-Diagramm
Neu!

UML - Tabellarischer Überblick

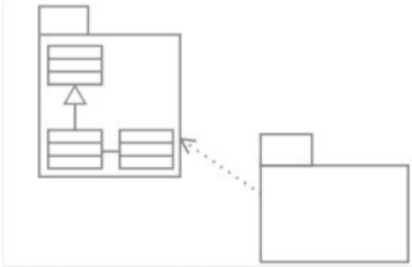
Folgende Darstellung gibt einen Überblick über die Schwerpunkte der einzelnen Diagrammarten.

Klassendiagramm



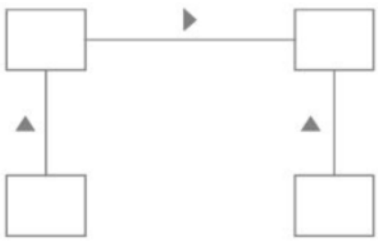
Es beschreibt die Klassen des System und wie sie miteinander in Beziehung stehen. Es ist das wichtigste Diagramm in der UML.

Paketdiagramm



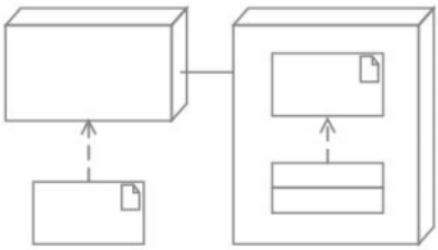
Es bündelt die Klassen zu überschaubaren Paketen und beschreibt die Beziehungen zwischen den Paketen.

Objektdiagramm



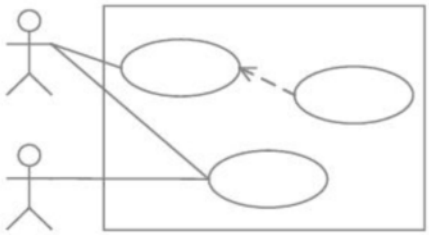
- Welche innere Struktur besitzt mein System zu einem bestimmten Zeitpunkt zur Laufzeit (Klassendiagrammschnappschuss)
- Zeigt Objekte und Attributbelegungen zu einem bestimmten Zeitpunkt
- Verwendung beispielhaft zur Veranschaulichung

Verteilungsdiagramm

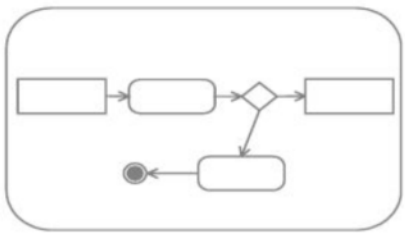


Wie sieht

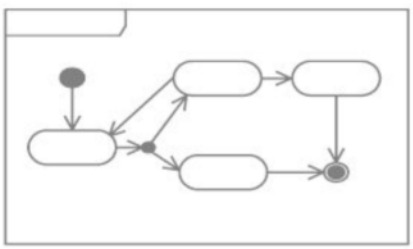
UseCase-Diagramm



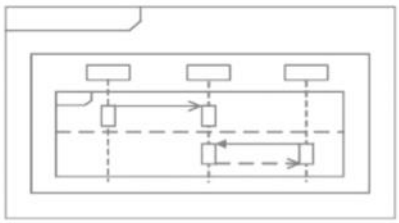
Aktivitätsdiagramm



Zustandsdiagramm



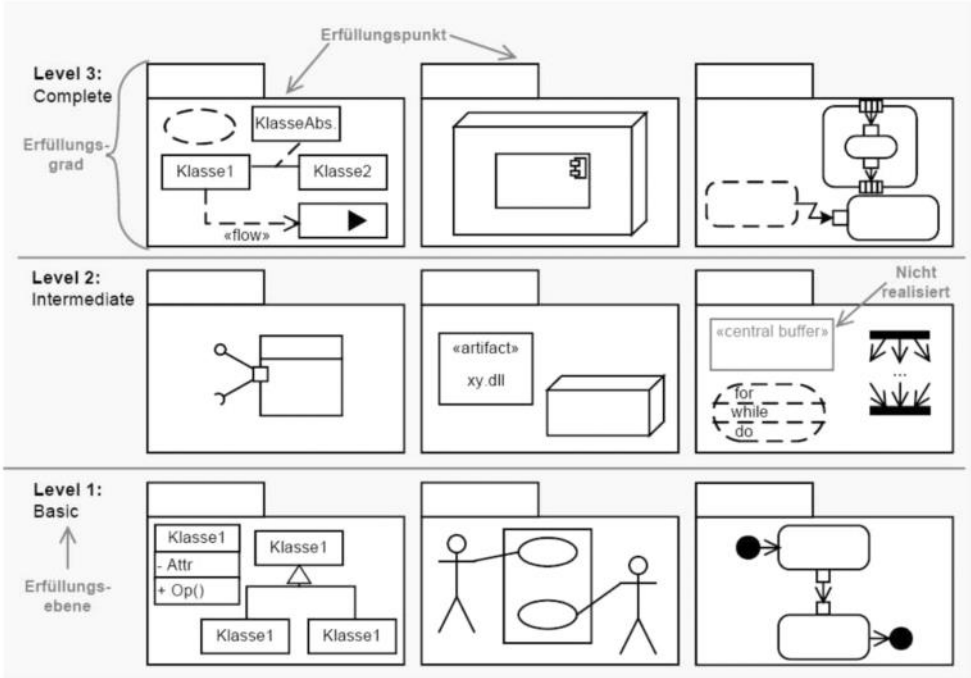
Sequenzdiagramm



- Wer tauscht mit wem welche Informationen in welcher Reihenfolge aus.
- Darstellung des Informationsaustausches zwischen Kommunikationspartnern
- Sehr präzise Darstellung der zeitlichen Abfolge auch mit Nebenläufigkeiten

UML-Erfüllungsebenen

UML erlaubt durch seine Vielzahl seiner Diagramme eine sehr detaillierte Darstellung der Prozesse. Dies ist nicht immer notwendig und machbar. Folgende Grafik veranschaulicht Priorisierungen und Variationen des Detaillierungsgrades.



Verständnisfragen

- Vor welchem Problem stand anfangs die Modellierungssprache UML ?
- Welche Bedeutung haben die Begriffe Statisch/Dynamisch im Zusammenhang mit Objekten und der UML?
- Gibt es neben der UML andere Modellierungssprachen? Welche Bereiche decken diese Sprachen ab ?

UseCase-Diagramm

Die Frage **Was soll mein geplantes System eigentlich leisten** sollte am Beginn jeder Systementwicklung stehen. **Das Use-Case-Diagramm zeigt das externe Verhalten eines Systems aus der Sicht der Nutzer.**

Das Ziel jedes Softwareentwicklungsprozesses ist es, eine Software zu entwickeln, die ganz bestimmte Anforderungen erfüllt. Die Entwicklung einer Software fängt mit der Zielsetzung an: Die Software soll, wenn fertiggestellt, die zu Beginn des Entwicklungsprozesses festgelegten Anforderungen/Ziele erfüllen.

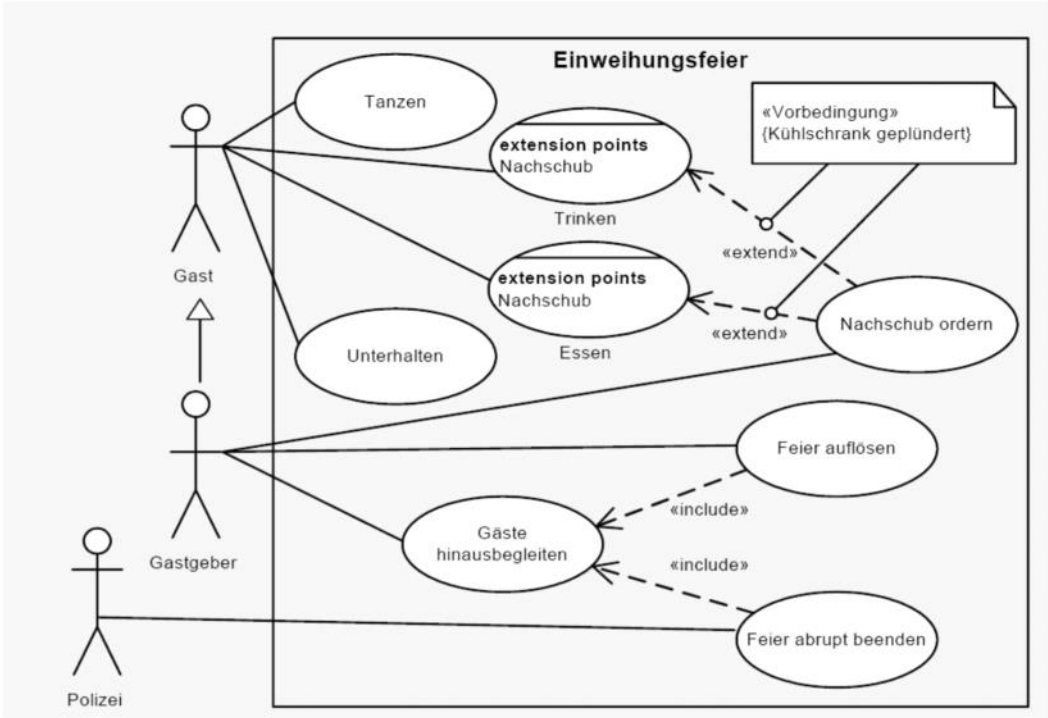
Leider sind diese Ziele nicht immer klar definiert. Man nimmt sich zum Beispiel vor, einen Online-Shop zu entwickeln, und stellt dann während des Entwicklungsprozesses fest, dass es unendlich viele unterschiedliche Funktionen in einem Online-Shop geben kann und man sich eigentlich nie klar gemacht hat, was man denn nun für Funktionen im Detail braucht. Man muss den Entwicklungsprozess daher wiederholt unterbrechen und inne halten, um sich zu überlegen, welche Funktionen, die einem bei der Entwicklung gerade eingefallen sind, notwendig sind und welche nicht.

Besonders schwierig wird die Situation, wenn der Auftraggeber des Entwicklungsprozesses nicht gleichzeitig der Entwickler ist. In diesem Fall kann der Entwickler nicht entscheiden, welche Funktionen notwendig sind - dies weiß nur der Auftraggeber. Dies führt zu einem ständigen Frage-Antwort-Spiel zwischen Entwickler und Auftraggeber, wenn der Entwickler nicht - noch schlimmer - die Entscheidungen selbst trifft und hofft, dies jeweils im Sinne des Auftraggebers zu tun.

Wenn Anforderungen an die zu entwickelnde Software nicht zu Beginn des Entwicklungsprozesses klipp und klar sind, wird der Entwicklungsprozess an sich unnötig erschwert. Denn das, was Sie entwickeln, richtet sich nach den bekannten Anforderungen. Jede Anforderung, die Ihnen oder Ihrem Auftraggeber später einfällt, führt dazu, dass Sie das, was Sie bisher entwickelt haben, ändern müssen. Denn die neue Anforderung hatten Sie logischerweise in Ihrer bisherigen Entwicklung nicht berücksichtigt. Grundsätzlich gilt, dass je später Anforderungen in einem Entwicklungsprozess bekannt werden, umso aufwändiger und daher teurer der Entwicklungsprozess wird. Anders gesagt: Wenn alle Anforderungen von Anfang an bekannt sind, bevor der Entwicklungsprozess gestartet wird, wäre das ideal.

Im Zusammenhang mit Anforderungen setzt man als erstes Hilfsmittel das Use-Case-Diagramm ein.

- Use-Case-Diagramme sind, auch bedingt durch die geringe Anzahl der Modellelemente, eingängig und übersichtlich.
- In der Projektrealität trifft man häufig eine sehr einfache skizzenhafte Verwendung von Use-Cases für erste Diskussionen.
- Sie enthalten die grafische Darstellung
  - des Systems
  - der Use-Cases
  - der Akteure außerhalb des Systems
  - der Beziehungen zwischen Akteur und Use-Case, der Akteure untereinander oder Use-Cases untereinander



Modellelemenente

Aufgabe des Use-Case-Diagramms ist es, eine grobe Ordnung in die vielen zum Teil sehr detaillierten Anforderungen zu bringen. Das Use-Case-Diagramm soll uns einen Überblick über das verschaffen, was die zu entwickelnde Software eigentlich im Wesentlichen können muss. Es werden also wichtige Funktionen der Software herausgearbeitet und zueinander in Beziehung gesetzt. Die technische Implementierung spielt bei diesem Überblick keine Rolle - vergessen Sie alles, was irgendetwas mit Programmiersprachen zu tun hat. Es geht um das Was, nicht um das Wie.

Das Use-Case-Diagramm ist ein recht einfacher Diagrammtyp, der schön anschaulich ist. Im Folgenden sehen Sie die grundlegenden Bausteine, mit denen alle Use-Case-Diagramme aufgebaut sind.

Das Use-Case-Diagramm verfügt über folgende Notationselemente:

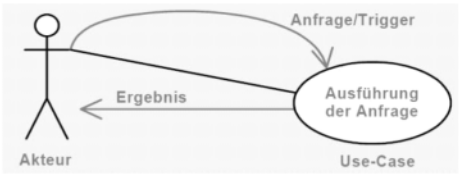
- UseCase
- System
- Akteur
- includes-Beziehung
- extends-Beziehung

Use-Case = System + Anwendungsfall + Akteur

Im Use-Case-Diagramm gibt es den Akteur und das System. Der Akteur ist der Anwender, das System die zu entwickelnde Software. Im System, das als Rechteck dargestellt wird, werden verschiedene wesentliche Anforderungen platziert. Man schreibt hierzu sehr knappe Funktionsbeschreibungen in Ellipsen, wobei jede Ellipse genau eine Funktion darstellt. Die Ellipsen sind die Use-Cases - daher hat das Use-Case-Diagramm seinen Namen. Wenn Sie einen deutschen Begriff verwenden möchten: Use-Case wird für gewöhnlich mit Anwendungsfall übersetzt.

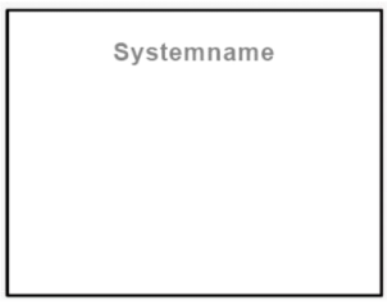
Da es beim Use-Case-Diagramm um einen ersten groben Überblick geht, beschränkt man sich auf die Darstellung wesentlicher Funktionen - alles, was nicht zum Überblick beiträgt, wird weggelassen. Beim Erstellen eines Use-Case-Diagramms geht es also nicht darum, möglichst viele Ellipsen in das Rechteck zu malen. Es geht darum, wesentliche

Anforderungen zu finden und diese zusammenhängend als Use-Cases in das System einzuzeichnen. Die Zusammenhänge zwischen Use-Cases und dem Akteur als auch zwischen Use-Cases untereinander werden durch Verbindungslinien dargestellt.

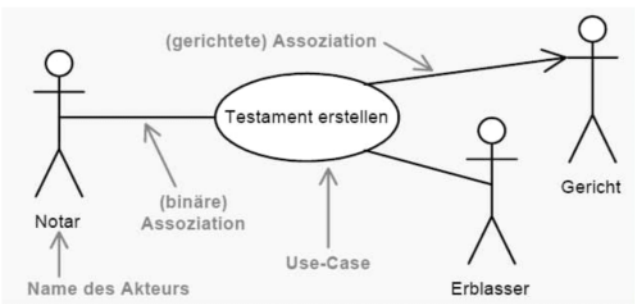


- Beschreibt eine Reihe von Aktionen, die nacheinander ein Verhalten formen
- Wird immer von einem Akteur ausgelöst
- Hat immer ein Ergebnis
- Kann gleichzeitig mehrfach instanziiert werden

System



Akteur



Beziehungen

Die Zusammenhänge zwischen Use-Cases und dem Akteur als auch zwischen Use-Cases untereinander werden durch Verbindungslinien dargestellt.

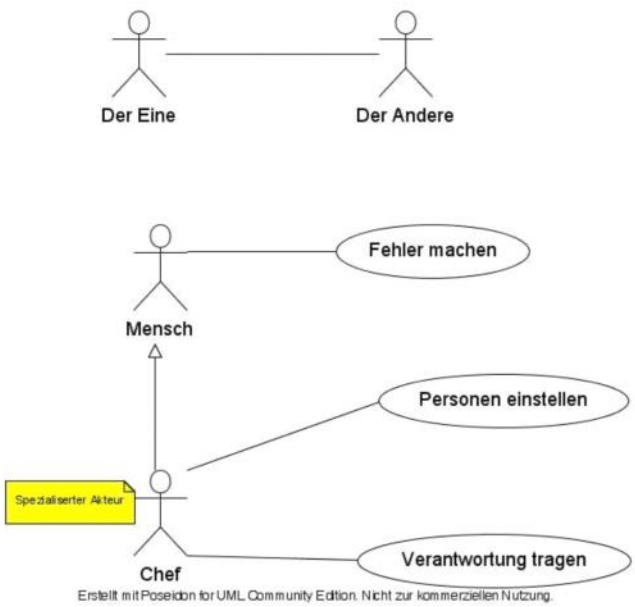
Verbindungslinien in UML-Diagrammen werden Assoziationen genannt. Sie stellen Zusammenhänge zwischen Elementen an den Enden von Assoziationen dar. Welche Art von Assoziation ausgedrückt wird, hängt von der Darstellung der Verbindungslinie und von Schlüsselwörtern ab, die an der Verbindungslinie stehen können.

Im Use-Case-Diagramm gibt es zwei Arten von Verbindungslinien: Die durchgezogene Verbindungslinie stellt eine Assoziation zwischen dem Akteur und einem Use-Case dar. Sie bedeutet, dass der Akteur den Use-Case in irgendeiner Form anwendet. Der Akteur tauscht also Informationen mit dem Use-Case aus. Das kann zum Beispiel bedeuten, dass er eine Funktion des Systems startet oder ihm von einer Funktion des Systems Daten ausgegeben werden.

Die gestrichelte Verbindungslinie stellt eine Assoziation zwischen zwei Use-Cases dar. Da es zwei verschiedene Arten von Assoziationen zwischen Use-Cases gibt, wird neben die gestrichelte Verbindungslinie ein Schlüsselwort gesetzt. Schlüsselwörter in der UML, die zur Spezifizierung von Verbindungslinien oder anderen geometrischen Formen verwendet werden, werden immer zwischen doppelte spitze Klammern gestellt. Diese Schlüsselwörter werden in der UML Stereotypen genannt.

Beziehungen zwischen Akteuren

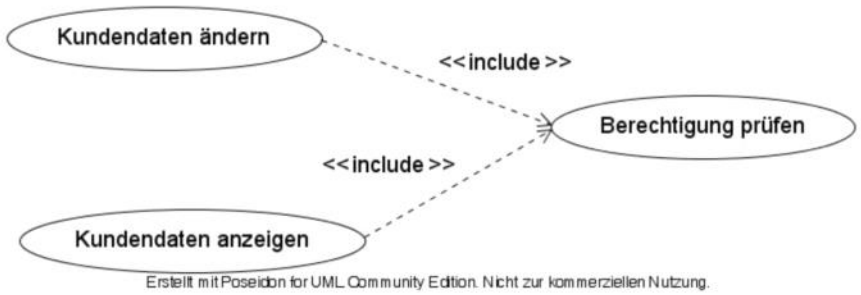
Akteure dürfen untereinander in Beziehung stehen, es sind sogar Vererbungsbeziehungen erlaubt. Der spezialisierte Akteur ist dann an den gleichen UseCase-Abläufen beteiligt wie der vererbende Akteur.

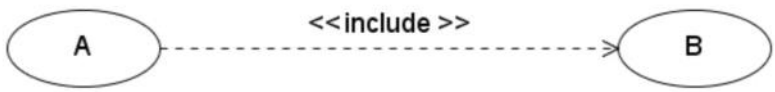


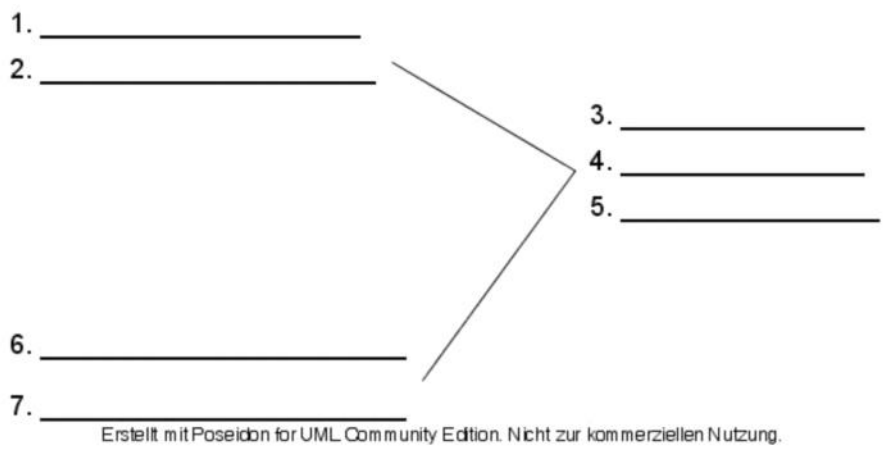
includes-Beziehung

Darstellung durch Abhängigkeitsbeziehung mit Stereotyp include. Der *linke*linke Use-Case importiert das Verhalten des *rechten* Use-Case.

Eine include-Beziehung ist nicht optional; das Verhalten wird immer eingeschlossen. Erst durch die Inklusion ergibt sich ein (sinnvolles) Gesamtverhalten. Rekursive Include-Beziehungen sind nicht erlaubt.

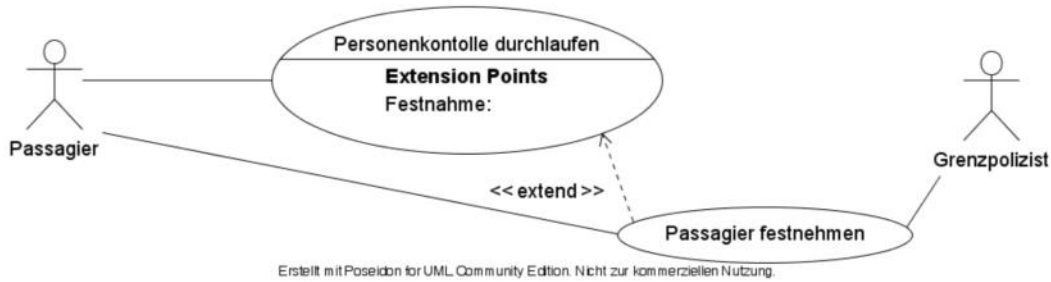






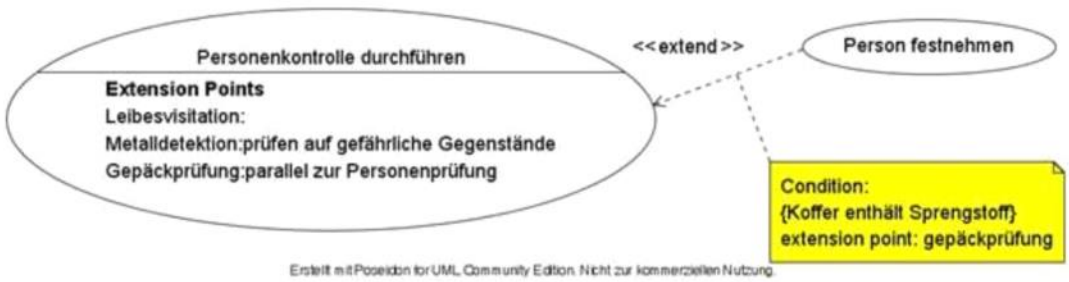
extends-Beziehung

Eine extend-Beziehung zeigt an, dass das Verhalten eines Use-Case (D) durch einen anderen Use-Case (C) erweitert werden kann, aber nicht muss.

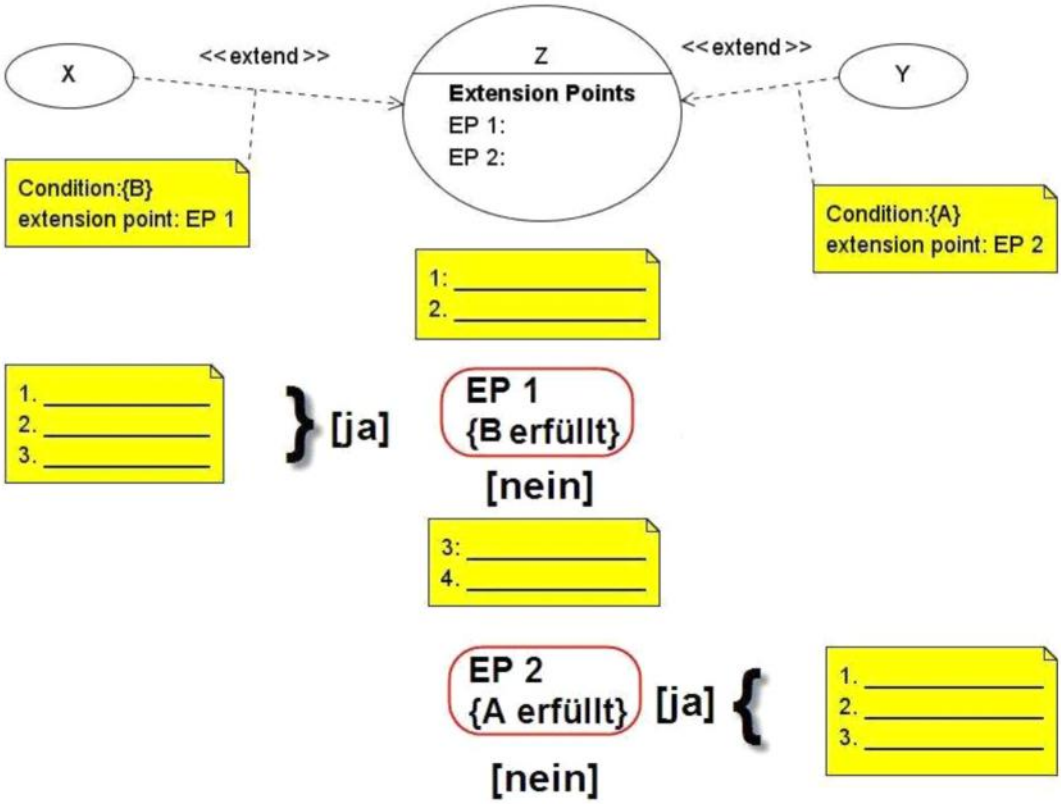


Obige Abbildung zeigt, dass der UseCase *Personenkontrolle durchlaufen* in bestimmten Fällen durch *Passagier festnehmen* erweitert wird.

Der Zeitpunkt, an dem ein Verhalten eines UseCases erweitert werden kann, wird als **Extension point** bezeichnet.

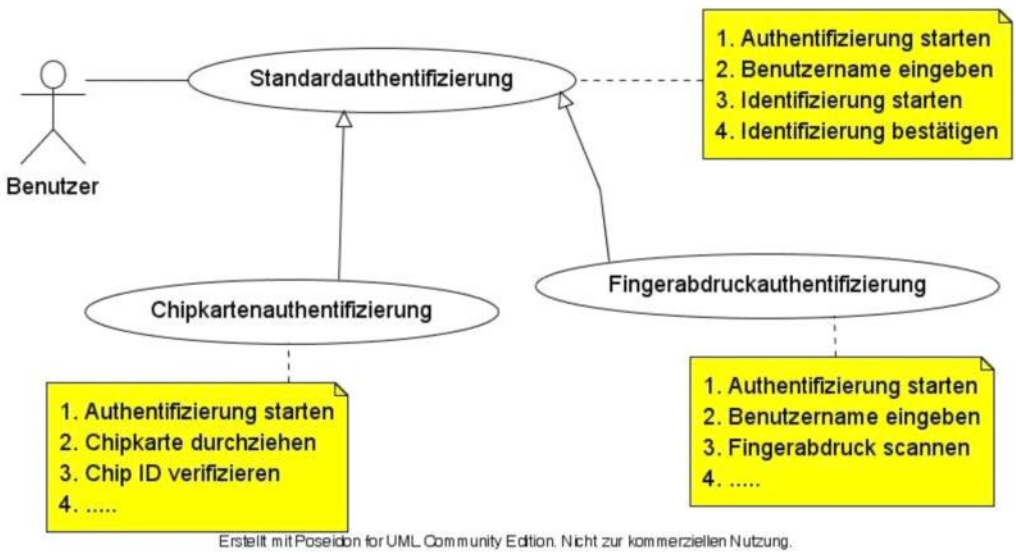


Neben dem Erweiterungspunkt kann eine Bedingung für die Erweiterung angegeben werden. Die Bedingung wird bei Erreichen des Erweiterungspunktes geprüft. Ist die Bedingung wahr, wird der Ablauf erweitert; ist die Bedingung nicht erfüllt, läuft der UseCase *normal* weiter



Vererbung

Zwischen UseCases kann auch eine Vererbungsbeziehung modelliert werden. Sie entspricht von Konzept her der Vererbung im Klassendiagramm.



Unterschied Vererbung – Extends

In der Praxis werden Generalisierungs- und Extends-Beziehungen häufig falsch eingesetzt.

Unterschiede include-extend

<<include>>

```
  includes
A --------> B
```

<<extends>>

```
     extends
A <---------- B
```

Geschäftsprozessschablone

Wie werden Use Cases erstellt? Es werden für ein definiertes System alle Akteure festgestellt (hierin liegt das größte Problem, dass ein Akteur übersehen und/oder vergessen wird). Alle Anforderungen der jeweiligen Akteure werden festgehalten und der Reihe nach in Schablonen übertragen. Die Schablonen werden nummeriert. Nicht alle Werte einer Schablone müssen für jeden Vorgang (Aktion) ausgefüllt sein.

- **Übergeordneter elementarer Geschäftsprozess**
  Hinweis auf übergeordneten Geschäftsprozess, wenn der betrachtete GP in einer <<include>> bzw. <<extends>> - Beziehung steht.
- **Beschreibung**
  Der Name des UseCase.
- **Ziel**
  Was ist das Ziel des UseCase. Hier erfolgt die genaue Darstellung des UseCases.
- **Vorbedingung**
  Erwarteter Zustand, bevor der GP beginnt.
- **Nachbedingung bei erfolgreicher Ausführung**
  Erwarteter Zustand nach erfolgreicher Ausführung des Geschäftsprozesses.
- **Nachbedingung bei fehlgeschlagener Ausführung**
  Erwarteter Zustand, wenn das Ziel nicht erreicht werden kann.
- **Beteiligte Nutzer**
  Rollen von Personen oder anderen Systemen, die den GP auslösen oder daran beteiligt sind.
- **Auslösendes Ereignis**
  Wenn dieses Ereignis eintritt, dann wird der GP initiiert.

Geschäftsprozess-Schablone (Balzert)

Geschäftsprozess	Name des Anwendungsfalles
Ziel	Kurze Beschreibung des Zwecks
Kategorie	Primär, Sekundär oder Optional
Vorbedingung	Erwarteter Zustand vor Ausführung des Anwendungsfalles
Nachbedingung Erfolg	Erwarteter Zustand nach erfolgreicher Ausführung des Anwendungsfalles
Nachbedingung Fehlschlag	Erwarteter Zustand, wenn das Ziel nicht erreicht wird
Akteure	Wer ist an dem Anwendungsfall beteiligt?
Auslösendes Ereignis	Nach welchem Ereignis tritt dieser Anwendungsfall auf?
Beschreibung	Ablauf des Anwendungsfalles
Erweiterungen	Was läuft parallel, bzw. nach diesem Anwendungsfall ab und kann diesem zugerechnet werden.
Alternativen	Wie kann dieser Anwendungsfall noch ablaufen?

Zusammenfassung

Fazit

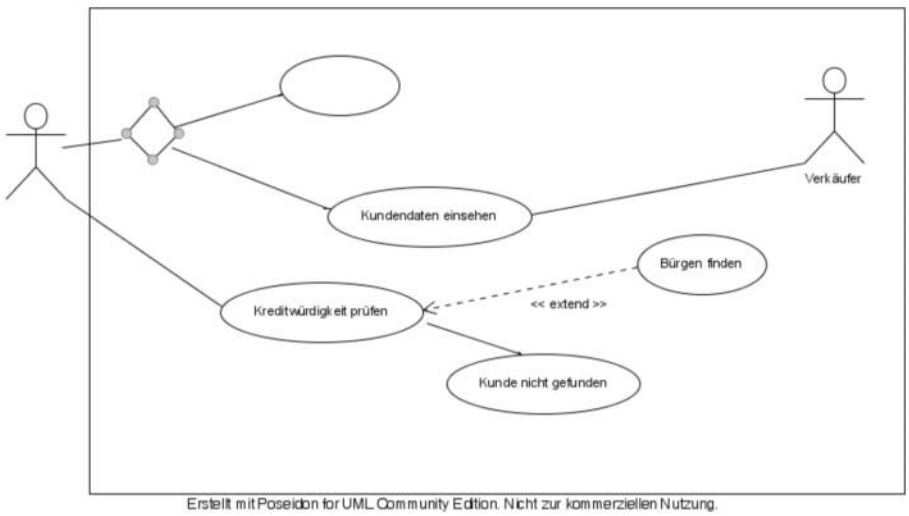
Anforderungen und ihre Zusammenhänge ohne Berücksichtigung der Reihenfolge

Use-Case-Diagramme sind Verhaltensdiagramme: Sie beschreiben bestimmte Aspekte, wie sich ein System verhält. Wie Sie nun gesehen haben, können im Use-Case-Diagramm wesentliche Funktionen des Systems hervorgehoben und zueinander in Beziehung gesetzt werden. Diesen Diagrammen fehlt jedoch eine Möglichkeit, die Reihenfolge der Ausführung festzulegen. Das geht im eingeschränkten Maße mit include- und extend-Assoziationen. Da ein Use-Case in diesen Fällen aber jeweils vom anderen Use-Case eingeschlossen wird - bei include immer, bei extend in Abhängigkeit einer Bedingung - handelt es sich nicht um eine strikte Reihenfolge: Ein Use-Case, der einen anderen mit include oder extend einschließt, läuft nach Beendigung des eingeschlossenen Use-Case weiter.

Dieser Nachteil des Use-Case-Diagramms ist an sich kein Nachteil, da Use-Case-Diagramme nicht für die Beschreibung einer Reihenfolge von Abläufen vorgesehen sind. Es ist also wichtig zu verstehen, dass jedes Diagramm bestimmte Aspekte eines Systems hervorhebt und beschreibt, aber nie alle. Es ist nicht die Aufgabe des Use-Case-Diagramms, die Reihenfolge einer Ausführung zu beschreiben. Zu diesem Zweck kann ein anderer Diagrammtyp verwendet werden, den Sie im nächsten Kapitel kennenlernen werden.

Fragen zu UseCase

- Was ist falsch an folgender Darstellung



- Die folgenden Anforderungen beschreiben eine einfache Artikelverwaltung, die objektorientiert zu modellieren ist.

1. Eine Firma will ihre Artikel und Bestellungen verwalten.
2. Für jeden Artikel werden die Artikelnummer, die Artikelbezeichnung und der Verkaufspreis festgehalten. Jeder Artikel gehört zu einer oder mehreren Artikelgruppen.
3. Jeder Artikel kann an mehreren Orten gelagert werden. Beispielsweise lagert die Firma Artikel in ihren Filialen Truchtelfingen, Adolzfurt und Ahrenviöllfeld. Pro Lagerort werden für jeden Artikel dessen Maximalbestand, Mindestbestand, aktueller Bestand und Name des Lagers gespeichert. Eine Lagerliste soll Auskunft über die Bestände geben.
4. Jeder Artikel kann von verschiedenen Lieferanten bezogen werden. Dabei gibt es je nach Lieferant unterschiedliche Einkaufspreise und Verpackungseinheiten. Bei jedem Lieferanten besitzt der Artikel eine eigene Bestellnummer.
5. Für jeden Lieferanten müssen dessen Name, Adresse, Telefonnummer und der Ansprechpartner gespeichert werden. Jeder Lieferant kann mehrere Artikel liefern
6. Jeden Tag werden die Lagerbestände kontrolliert. Wird der Mindestbestand unterschritten, so wird ein Bestellvorschlag erstellt. Er enthält außer den Daten eines Artikels auch die verschiedenen Lieferanten mit ihren Lieferkonditionen. Für jedes Lager wird errechnet, wie viele Artikel nachbestellt werden müssen. Die vorgeschlagene Anzahl errechnet sich aus dem Maximalbestand und dem aktuellen Bestand. Die Anzahl ist auf ein n-faches der Verpackungseinheit eines jeden Lieferanten abzurunden.
7. Jeder Artikel kann in beliebiger Anzahl bestellt werden. Dabei wird vom System automatisch der günstigste Lieferant gewählt.

8. Jeder Artikel wird einzeln bei einem Lieferanten bestellt. Für jede Lieferantenbestellung werden das Bestelldatum und das Lieferdatum festgehalten. Der Einfachheit halber werden Teillieferungen ausgeschlossen.
9. Ein Kunde kann eine oder mehrere Bestellungen erteilen, die erfasst werden müssen. Jede Bestellung erhält eine eindeutige Bestellnummer. Außerdem werden das Bestelldatum, das Lieferdatum und die Portokosten festgehalten.
10. Jede Kundenbestellung kann sich auf mehrere Artikel beziehen. Für jeden bestellten Artikel ist die gewünschte Anzahl festzuhalten. Außerdem kann der Gesamtpreis für mehrere Artikel ungleich Anzahl*Verkaufspreis sein.
11. Jede Kundenbestellung wird von einem Sachbearbeiter bearbeitet. Für jeden Sachbearbeiter sollen dessen Personalnummer, ein Kürzel, der Name und die Telefonnummer gespeichert werden.
12. Außerdem sollen folgende Statistiken erstellt werden:
    - Welchen Umsatz hat ein Kunde X im aktuellen Jahr erzielt?
    - Welchen Umsatz hat ein Sachbearbeiter X mit seinen verschiedenen Kunden erzielt.