

# **Projektdossier Semesterarbeit 3i**

## Projektphasen

In der SA sollen folgende Projektschritte gemäss dem Wasserfallmodell behandelt und dokumentiert werden. Nicht alle Punkte sind auf jedes Projekt anwendbar (im Zweifelsfall erkundigen).

### Analyse

Informationsbeschaffung (Tutorials, Code-Snippets), Anwenderbefragungen, Zielgruppen, Anforderungen (Must-Have's, Nice-To-Have's), Use-Cases, Use-Case Diagramm, Entity Relationship Model (ERM), HW-, SW-Umgebung inkl. deren Versionen (Kunde, Entwickler), Nutzwertanalyse verschiedener Lösungsvarianten, Installationsanleitung der Entwicklungsumgebung.

### Planung

Meilensteine, Projektrollenverteilung, Aufwandabschätzung, Zeitabschätzung (evtl. Gantt-Chart), Kostenabschätzung, Achtung: Sie werden viel mehr Zeit brauchen als sie denken! Planen Sie zusätzlich grosszügige Pufferzeiten ein!

### Design

GUI-Design (Layouts, Wireframes für alle Use Cases) mit Benutzerfeedback (Usability/ UX), Datenbankschema, Klassendiagramm, Zustandsdiagramm, Sequenzdiagramm, Tests (Listen mit detailliert beschriebenen funktionalen und destruktiven Tests).

### Realisation

Übersichtliche, gut strukturierte Implementation des Programmes inkl. Helps, Korrektes anwenden der Programmier Techniken, Implementation der Datenbank, evtl. Einrichten des Servers, Erstellen der Distribution.

Dokumentation der wichtigen Konzepte, Schlüsselstellen und Algorithmen (ergänzt mit Code-Snippets), Nennung der Pitfalls (Stolpersteine).

### Test

Testen gemäss Testliste, Unittests, Integrationstest, Deployment-Test.

### Deployment

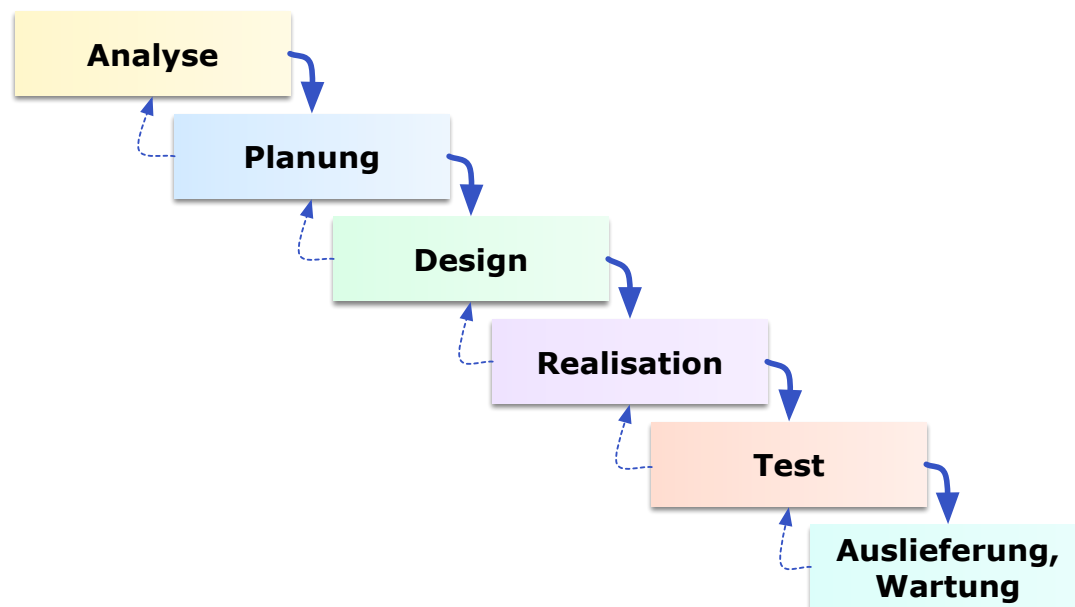
Installationsanleitung, evtl. Installer, Benutzeranleitung, Deployment-Scripts, Build-Pipes, Bulk-Dump.

# Wasserfallmodell - IPERKA - Agile Methoden

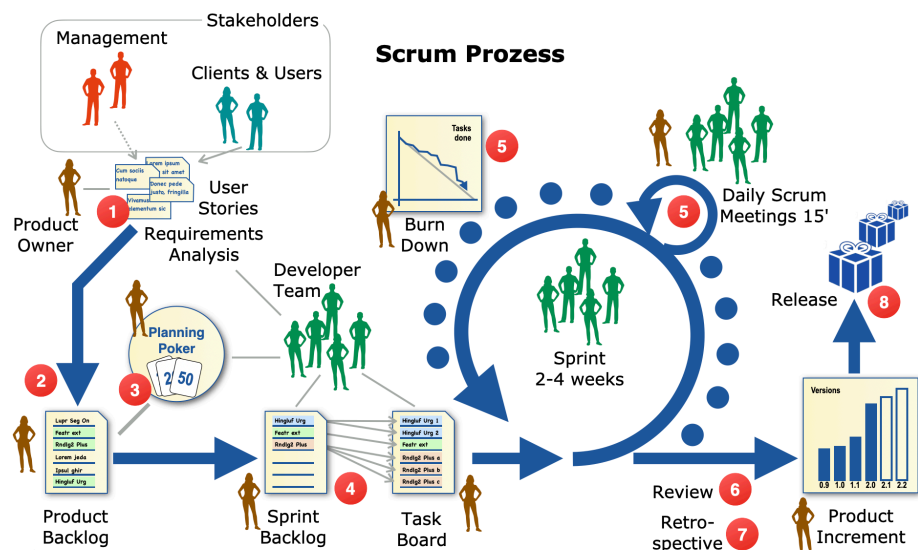
Der Klassiker bei der Softwareentwicklung ist das Wasserfallmodell. Heute werden die meisten (grösseren) Softwareprojekte allerdings mit Scrum oder Kanban agil entwickelt. Bei der Betriebsinformatik hingegen, kommt häufig IPERKA zum Einsatz.

Am Ende der IMS-Ausbildung muss die IPA (Individuelle Projektabreit) eines Applikationsentwicklers mit IPERKA geplant, durchgeführt und ausgewertet werden. Wie die folgende Übersicht zeigt, haben beide Modelle viele Gemeinsamkeiten. Sie können daher relativ einfach ineinander überführt werden. Für die SA verwenden wir das Wasserfallmodell.

## Projektphasen gemäss **Wasserfallmodell**



## Projektphasen gemäss **IPERKA**



Die Entwicklung mit **agilen Methoden** ist kaum mit dem Wasserfallmodell oder IPERKA vergleichbar. Sie enthält jedoch Elemente von ihnen.

## Analyse

Bei der Analyse geht es darum, die **Aufgabenstellung** zu **erfassen**. Es kommt immer wieder vor, dass das Projekt nicht klar beschrieben ist oder ein Auftraggeber nicht genau weiss, was er will. Während der Analyse versuchen wir deshalb, die Ziele, die **Must-Have's** und die **Nice-To-Have's** und die damit verbundenen **Anforderungen** möglichst genau herauszuarbeiten und schriftlich festzuhalten. Dazu befragen wir die Auftraggeber und die Anwender eingehend.

Dabei konzentrieren wir uns auf die Frage nach dem "**Was**". **Was** soll das Programm können? **Was** soll dargestellt, gespeichert und ausgegeben werden? **Was** wird verlangt?

Weiterhin versuchen wir möglichst viel **Informationen** (Wikipediaeinträge, Code-Stücke, Tutorials, Videos usw.) zum Thema **zusammenzutragen**. Wir schreiben auch kleine **Testprogramme** um **Codesegmente auszuprobieren**, bei denen wir unsicher sind. In der späteren Coding-Phase sollten wir das Programm nur noch "reinhacken" können. Wenn wir dann noch recherchieren müssen, haben wir die Analyse nicht gut gemacht oder zumindest nicht gut dokumentiert. Es ist wichtig, dass wir uns hier alle wichtigen **Quellen** notieren.

Wir erstellen auch Fragenkataloge, führen **Interviews** mit Wissensträgern und machen **Umfragen** bei Anwendern.

Dann erstellen wir die **Use-Cases** und das **Entity Relationship Model** (ERM) und definieren eine **Zielgruppe**.

Damit jemand ein Projekt später weiterentwickeln und somit wieder korrekt aufsetzen kann, ist das Wissen über die komplette Entwicklungsumgebung (inkl. OS), der verwendeten Libraries und vor allem sämtlicher **Versionen** essentiell. Stark helfen kann hier auch eine **Installationsanleitung** für die **Entwicklungsumgebung**.

Mit der Installation der Entwicklungsumgebung einher geht das **Aufsetzen** eines **Git-Repositories** und deren Verknüpfung mit GitLab.

Auch der **Kunde** muss wissen auf welcher **Hardware** und welchem **OS** (Version) seine **Software laufen wird**.

Bei den meisten Projekten gibt es mehrere Varianten, wie gewisse Anforderungen erfüllt werden können (mehrere Möglichkeiten für IDE's, Programmiersprachen, Libraries, Algorithmen, Datenbanken, Services usw.). Hier bietet sich die Erstellung einer **Nutzwertanalyse** an. So kann man die für sich und das Projekt beste Variante gut evaluieren.

All dies wird im Kapitel Analyse beschrieben und festgehalten **bevor man mit dem Design** beginnt.

Aus der Analyse heraus entsteht bei realen Projekten die so genannte Spezifikation oder das Anforderungsdokument, welches vom Auftraggeber gelesen und unterschrieben werden muss und somit eine vertragliche Grundlage für das Projekt bietet. Da dies den Rahmen einer SA sprengen würde, lassen wir das Schreiben einer Spezifikation hier an der Schule bleiben und begnügen uns mit einer Analyse wie oben beschrieben.

## Planung

Bevor wir mit dem eigentlichen Design beginnen können, planen wir das Projekt. Zunächst interessiert uns der **Aufwand** in Personen-Stunden oder Personentagen. Diesen ermitteln wir, indem wir das Projekt in kleine Pakete, unterteilen und den **Aufwand für jedes Teilpaket einzeln** abzuschätzen versuchen. Zusammengezählt ergibt dies den Totalaufwand.

Nun da man den Aufwand kennt und die Hardware- bzw. Service-Kosten aus der Analyse bekannt sind, erstellt man die für den Auftraggeber sehr wichtige **Kostenabschätzung**. Wir simulieren dies in der SA indem wir für die einzelnen Rollen marktübliche Stundensätze einsetzen.

Das Verteilen der oben ermittelten Teilaufwände auf den Kalender ergibt dann die **Zeitabschätzung**. Aus der Zeitabschätzung heraus erkennen wir, wann welcher Projektschritt fertig sein wird und wann der Kunde sein Produkt erhält, bzw. ob man mit dieser Planung den Abgabetermin einhalten kann. Man sollte unbedingt darauf achten, dass man **genügend Pufferzeit** einplant. Bei der Realisation verschätzen sich oft auch geübte Informatiker! Ein hilfreicher Trick ist, den geschätzten Realisationsaufwand mit dem Faktor Pi (3.14) zu multiplizieren und vor dem Abgabetermin zusätzlich 2 Wochen Pufferzeit einzuplanen. Meist stimmt dann die Planung recht gut.

Jeder Projektschritt schliesst mit einem **Meilenstein** ab. So ist z.B. "31.10.21 Analyse fertiggestellt" ein Meilenstein. Ein Meilenstein hat keine Zeitdauer, er ist ein Zeitpunkt **wann** etwas **fertig** ist.

Auch hierhin gehört die **Projektrollenverteilung**, in der "Wer was macht" beschrieben wird.

Die Meilensteine, Aufwand- und Zeitabschätzungen sowie die Rollen können zusätzlich übersichtlich in einem **Gantt-Diagramm** festgehalten werden. Geeignete Tools findet man [hier](#).

## Design

Beim Design (Programmmentwurf) geht es darum, wie die gestellte Aufgabe umgesetzt werden kann. Dabei fragen wir nach dem "**Wie**". **Wie** soll die Applikation aussehen? **Wie** sollen die Daten strukturiert, angezeigt, gespeichert und ausgegeben werden.

Hier erstellen und beschreiben wir alle möglichen **Wireframes** (Layouts) oder ein Fast-Prototyping der App. Damit man nichts übersieht und man eine gute **Usability (UX)** gewähren kann, sollten diese die in der Analyse definierten Use-Cases enthalten und zusätzlich mit dem **Auftraggeber** oder den **Anwendern besprochen** werden.

Weiterhin helfen uns Techniken wie die Erstellung eines **Klassen-, Nassi-Shneiderman- und Sequenzdiagrammes**, die die statische und dynamische Sicht der Applikation zu veranschaulichen und definieren helfen.

Basiert unsere Applikation auf einer Datenbank, so darf ein **Datenbankschema** nicht fehlen.

Das Design sollte derart gut beschrieben sein, dass ein Programmierer genügend Informationen und Vorgaben hat, um das Programm anschliessend ohne nachfragen zu müssen realisieren kann. Idealerweise erstellt der Programmierer das Design selber und lässt es von einem Kollegen reviewen (durchdenken, prüfen).

Ebenfalls zum Design gehört die **Testvorbereitung**. Dabei wird definiert, wie während (Unittests) und was nach der Realisierung getestet wird. Auf diese Weise wird die Qualität und die korrekte Funktionsweise des Programms von Beginn weg und zu einem Zeitpunkt, wo man die Übersicht noch hat sichergestellt.

Für das sogenannte **Black-Box-Testing** nach der Realisierung werden in einer Liste die verschiedensten **funktionalen und destruktiven Testfälle** zusammen mit den erwarteten Ergebnissen zusammengestellt. Die Fälle sollten möglichst genau beschrieben werden ("1.2, 15, 'Simone' eingeben" und nicht einfach "Einen Wert eingeben") und möglichst alle Use-Cases abdecken.

## ***Realisation (Implementation, Coding)***

In dieser Phase wird der Programmcode geschrieben. Es ist darauf zu achten, dass der **Code übersichtlich** geschrieben, **gut strukturiert** und **kommentiert** wird. Programmieren ist ein Prozess, bei dem vor allem die Kenntnisse der Programmiersprache und ihrer Anwendung im Vordergrund stehen.

Hier sollten Kriterien wie Code for **Robustness, Simplicity, Effectiveness und Reliability** angewendet werden. Das Programm sollte bei Fehleingaben nicht abstürzen, einfach, schnell und zuverlässig sein.

Weiterhin ist es wichtig gelernte **Konventionen** wie Datei-Organisation (z.B. Trennung von PHP, Script, HTML oder Verwendung eines Assets-Ordners), Header, gute Bezeichner, private Membervariablen, KISS, kein Hardcoden usw. anzuwenden und einzuhalten.

Einfachheit und Übersichtlichkeit sind zwei besonders wichtige Gebote. Einfacher, gut strukturierter und übersichtlicher Code reduziert die Fehler und steigert somit die Qualität. Zudem sollte jedes Source-Code-File (ausgenommen Web) mit einem Header versehen sein, welcher den Autor, eine kurze Beschreibung und die Geschichte festhält.

Bei der Realisierung der Layouts steht die Einhaltung der **Usability-Regeln** (Softwareergonomie) an erster Stelle. Wenn möglich sollten Placeholders und Kontext-Help-Funktionen eingebaut werden.



Die Verwendung und tägliche Benutzung eines **Versionierungs-Systems** (git, GitLab) ist ein Muss!

Verwendet man eine Sprache die **Unit-Tests** unterstützt, so sollten für die wichtigsten Methoden Unit-Tests geschrieben und angewendet werden.

**Dokumentiert** werden hier Erklärungen, wieso bestimmte Dinge so gemacht wurden, sich nicht selbsterklärende Code-Blöcke, nicht offensichtliche Konfigurationen, für die Cloud nötige Registrierungen und Stolpersteine.

## Test

Das Austesten des fertigen Programmcodes mit einem **Black-Box-Test** zeigt, ob alle geforderten Eigenschaften und Funktionen realisiert sind und ob sich das Programm tatsächlich so verhält, wie es erwartet wird.

Der Test ist eine der wichtigsten Phasen im Rahmen der Softwareentwicklung und kann aufwändig sein. Als Basis für die Tests dient die Testliste, welche während der Testvorbereitung in der Design-Phase bereits erstellt wurde.

Nun werden alle festgehaltenen funktionalen und destruktiven Testfälle von einem Tester der Reihe nach durchgespielt und genau dokumentiert, welche Tests erfolgreich waren und welche nicht. Dabei wird auch festgehalten, wer, wann, welche Version getestet hat.

In einem Zyklus wird nun solange getestet und Probleme behoben, bis alle Testfälle erfolgreich waren.

Gibt es am Schluss trotzdem noch Fehler, so werden sie in einer **Known-Bugs-Liste** festgehalten. In der Praxis werden die Known-Bugs mit einem so genannten Issue- oder Bug-Tracking-Tool festgehalten, dies kann bei uns aber auch mit einer einfachen Tabelle geschehen.

Weiterhin wird mit einem **Deployment-Test** getestet, ob sich die Applikation auf einem "Nicht-Entwicklungssystem bzw. -Rechner" erfolgreich installieren und ausführen lässt.

Weiterhin kann ein zukünftiger Anwender oder der Auftraggeber hinzugezogen werden welcher bestätigen kann, dass die Applikation seine **Absichten** die er mit der Software erfüllen will, erfüllen kann -> **User Intention fulfilled!**

## Deployment

In dieser Phase erstellen wir eine **Installations-** und eine **Benutzeranleitung**. Die Installation sollte mithilfe eines von Ihnen erstellten Installers oder eines **Installations-Scripts**, inklusive der Erstellung der Datenbank, ganz einfach vonstatten gehen. Bei einer Web-Applikation wird beschrieben, wie der Hoster die Applikation auf seinem Server zu installieren hat. Bei einer Mobile-App, wie die App auf Google Play oder dem Apple Store veröffentlicht wird.

So nun ist das Projekt fertiggestellt!

## Präsentation

Die Präsentation dient dazu, dass sie ihr Produkt "stolz" der Öffentlichkeit vorstellen. Sie ist eine Art Verkaufsveranstaltung für ihr Projekt und sollte die Zuschauer von ihrem Produkt überzeugen.



### Do's:

- Begeistern sie das Publikum! Sprechen sie frei!
- Erklären und zeigen sie ihr Produkt und wozu es gut ist, am Besten auch mit einer Live Demonstration. Erwähnen sie wie, wo und von wem es genutzt werden kann und wie es mit ihrem Produkt in Zukunft weiter geht/gehen könnte
- Zeigen sie ihr Wissen, was sie gelernt und welche Klippen sie wie gemeistert haben
- Achten sie auf und nutzen sie die Zeit die sie zur Verfügung haben
- Sprechen sie laut und deutlich, achten sie auf eine gepflegte Sprache, verwenden sie Fachbegriffe und erklären sie sie, richten sie auch mal eine Frage ans Publikum
- Bringen sie Abwechslung in ihren Vortrag mit Fotos, Videos, Comics, Sound, Demos oder auch mal einem Witz
- Halten sie den Vortrag mit gepflegtem Äusseren (Kleidung, Hygiene), halten sie Augenkontakt zum Publikum, nehmen sie eine aufrechte, wissende und souveräne Haltung ein
- Wählen sie eine Schriftart, 3-4 Grössen und Stile und eine Farbenpalette aus und verwenden sie dann nur diese
- Proben sie den Vortrag vor der Familie oder Freunden, probieren sie die Hardware einen Tag zuvor im Schulzimmer aus (Beamer, Second Screen, Sound, Laser Pointer, Demo)












### Don'ts:

- Keine Inhaltsverzeichnisse und schon gar nicht vorlesen
- Keine Videos die nicht in der Präsentation eingebettet sind
- Der Zuschauer darf nie einen voll-geklatschten Schreibtisch oder das Powerpoint- oder Keynote-Programmfenster sehen (Second Screen statt Mirroring und bei Applikationen Full Screen Modus verwenden)
- Prahlen sie nicht mit dem was alles schief lief, sondern erzählen sie, welche Klippen sie erfolgreich gemeistert haben
- Geben sie euren Kolleginnen und Kollegen keine Tipps der Form "Fangt mit eurer SA früher an als ich es getan habe"



## Projektfortschrittsbericht

| Projekt:                              |   | Arbeitspaket:   |   |                                   |
|---------------------------------------|---|---|---|-----------------------------------|
| Mitarbeiter:                          |   | Kalenderwoche:  | Datum:  |                                   |
| Projektkurzbeschreibung:              |   |   |   |                                   |
| Projektfortschritt (in Prozent):      |   |   |   |                                   |
| Arbeiten, Ziele des Projektschrittes: |   |   |   |                                   |
| Projektstand                          | Status  |   |   | Begründung (nur bei gelb und rot) |
| Termine                               |  |  |  |                                   |
| Kosten                                |  |  |  |                                   |
| Produkt                               |  |  |  |                                   |
| Störungen/Probleme/Risiken:           |   |   |   |                                   |
| Notwendige Entscheidungen:            |   |   |   |                                   |
| Anlagen:                              |   |   |   |                                   |

Jedes dieser Felder  
kann grün, gelb oder  
rot sein!

# Nützliche Programme / Werkzeuge

## Gantt-Charts

- ProjectLibre (OpenProject)
- Open Workbench
- MS Project
- OmniProject
- GanttProject (<http://ganttproject.biz>, Java basierend)
- Binfire

## Revision Control Tools

- Git (Vorgabe für SA)
- GitLab (Vorgabe für SA)
- GitHub
- Binfire
- Subversion
- MS Team Foundation Server

## Collaboration Tools

- Trello
- Scrumwise \*
- Dropbox
- OneDrive (Sharepoint)
- iCloud

## Wireframes/Mockups/Fast-Prototyping

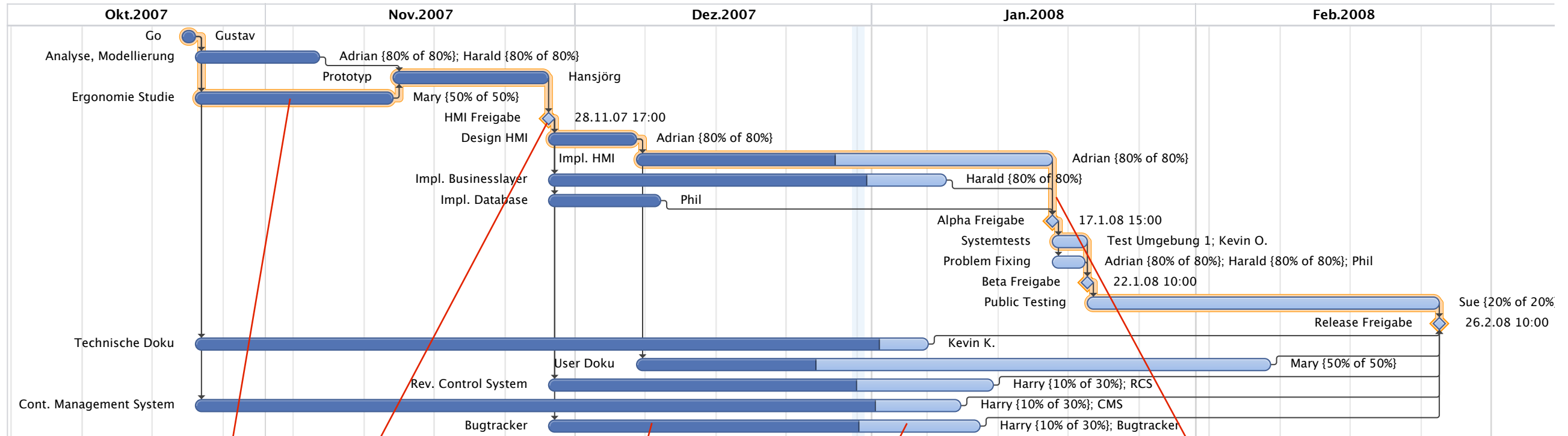
- Balsamiq Wireframes [www.balsamiq.com](http://www.balsamiq.com) \*
- [draw.io](http://draw.io)
- gliffy
- PowerPoint \*
- Keynote

## UML-Diagrams

- draw.io
- gliffy
- MySQL Workbench
- Dia
- OminGraffle
- Visio

\* Schullizenzen vorhanden

# Gantt-Beispiel



Task

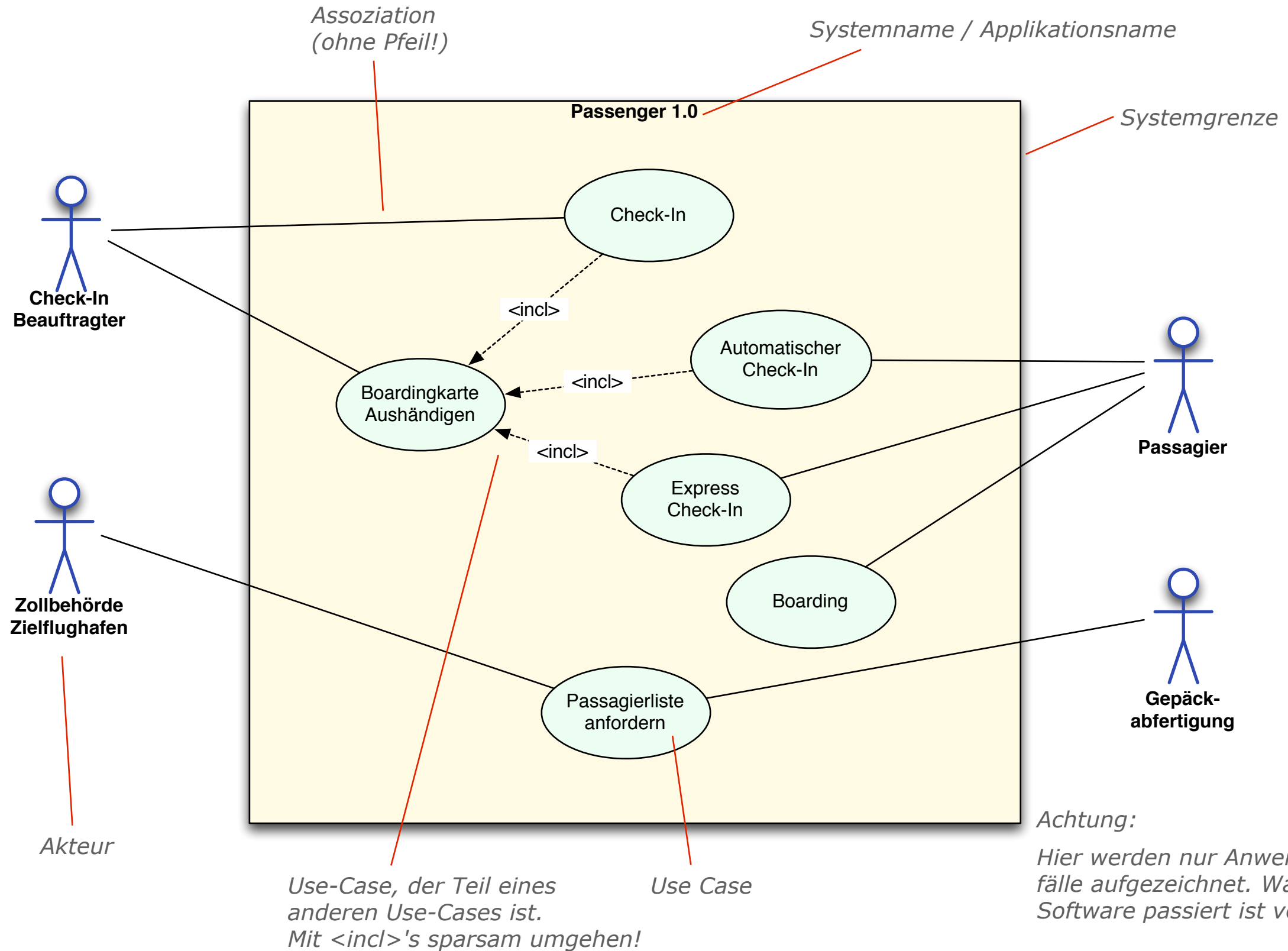
Milestone

Task: completed part

open part

critical path (orange)

# Use-Case-Beispiel



## Wireframes / Views erstellen

Typischerweise erstellt man vor dem Programm zuerst ein Wireframe (Mockup oder Fast-Prototype) mit mehreren Views, mit welchen man das GUI (Graphical User Interface) bzw. das HMI (Human Machine Interface) "trocken" durchspielen kann. Dazu stellt man jede View (Layout) schematisch, aber noch ohne Funktionalität dar. Erst wenn der Prototyp mit dem Auftraggeber besprochen wurde, beginnt man mit der Implementation.

Wir achten hierbei genau auf eine gute Software-Ergonomie. Achtung: Jeder Use-Case muss mithilfe mindestens einer View durchgeführt werden können.

Videothek

# Kunden

Kunden Videos Ausleihen

| Vorname  | Nachname   |
|----------|------------|
| Mohamad  | Al Bezier  |
| Klaus    | Biedermann |
| Hans     | Guten      |
| Denise   | Hösli      |
| Susi     | Werner     |
| Ingeborg | Laux       |
| Patrick  | Senn       |

Kundensuche:

Kundennummer:

Anrede:

Vorname:

Nachname:

Strasse:

PLZ:

Ort:

Geburtsdatum:

Telefon:

| Videonummer | Video Name                     | Ausleihdatum | Rückgabedatum |
|-------------|--------------------------------|--------------|---------------|
| 10006       | 10'000 B.C.                    | 28.4.2015    |               |
| 10005       | Bob Aziz / Der Tanz des Windes | 28.4.2015    |               |
| 10000       | American Pie 2                 | 29.4.2015    | 30.4.2015     |



Beispiele erstellt mit: Balsamiq Wireframes  
[www.balsamiq.com](http://www.balsamiq.com) , Konto bei NUE anfragen!

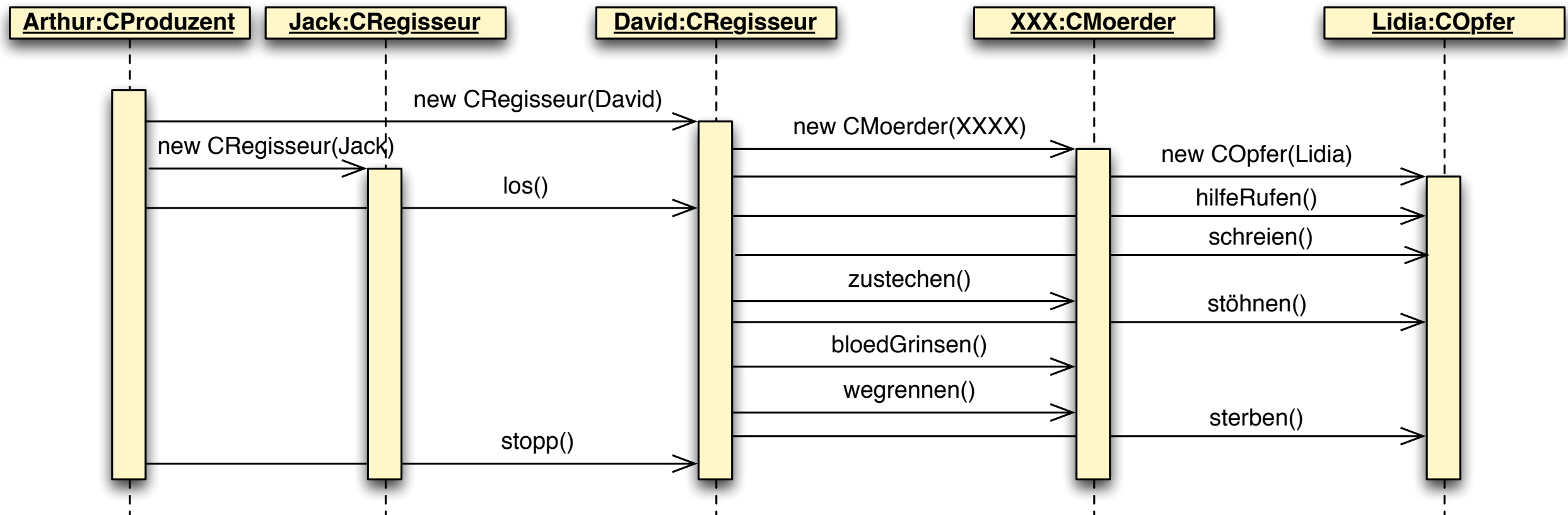
## Usability (User Experience UX)

Damit ein Benutzer schnell und effizient mit einer Applikation arbeiten und seine Absichten die er mit Applikation erreichen will auch erreichen kann und somit die Akzeptanz der Applikation hoch ist, gilt es einige Regeln zu beachten:

- Wo bin ich? Der Benutzer möchte jederzeit wissen wo er ist. z.B. Views mit Titel anschreiben.
- Wo kann ich hin? Der Benutzer möchte jederzeit wissen wo er hin kann? Tasten, Reiter, Menus.
- Höchstens 2-3 verschiedene Schriftarten verwenden.
- Schriftgrösse weder zu klein (ältere Leute) noch zu gross (unübersichtlich) wählen.
- Eingabemaske soll ca. 50% freie Fläche haben.
- Tasten, Buttons sollen in jeder View am gleichen Ort sein. Auch möglichst gleich wie OS.
- Zusammengehörende Felder sollen auch visuell gruppiert sein.
- Felder, wenn immer möglich, aufeinander ausrichten.
- Mit Farben äusserst sparsam umgehen. Sich möglichst ans OS (Betriebssystem) halten. Keine grellen Farben und kein Blinken verwenden. Auf guten Kontrast (ältere Leute) bei Texten achten.
- Alle Tasten und Menus sollten für "Profi-"Benutzer auch per Tastatur bedienbar sein.
- Keine unnötig grossen Banner die der Benutzer zuerst "überscrollen" muss verwenden.
- Jeder Use-Case sollte mit möglichst wenigen Klicks durchgespielt werden können.
- Formulare so gestalten, dass Fehlermeldungen nicht nötig sind und wenn doch, sollten sie den Benutzer sachlich darauf hinweisen wie er den Fehler vermeiden kann.
- Grundsätzlich soll die Applikation dem Benutzer dazu dienen, dass er seine Aufgaben einfach, korrekt und schnell erledigen kann.

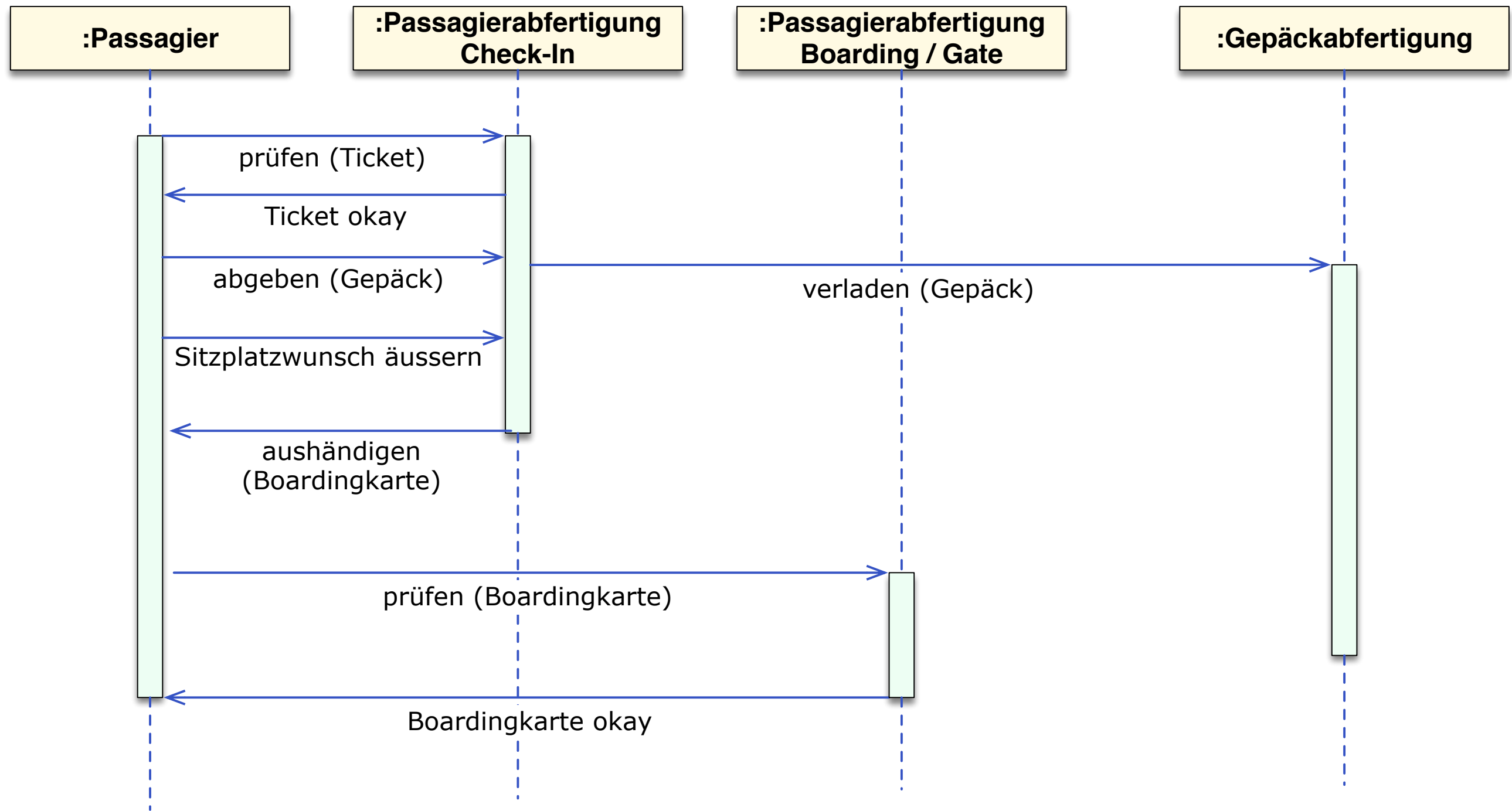


## Beispiel für ein Sequenzdiagramm für Java, C#, C++, Dart, React, JavaScript(OOP), PHP(OOP) zur Verfilmung eines Krimis

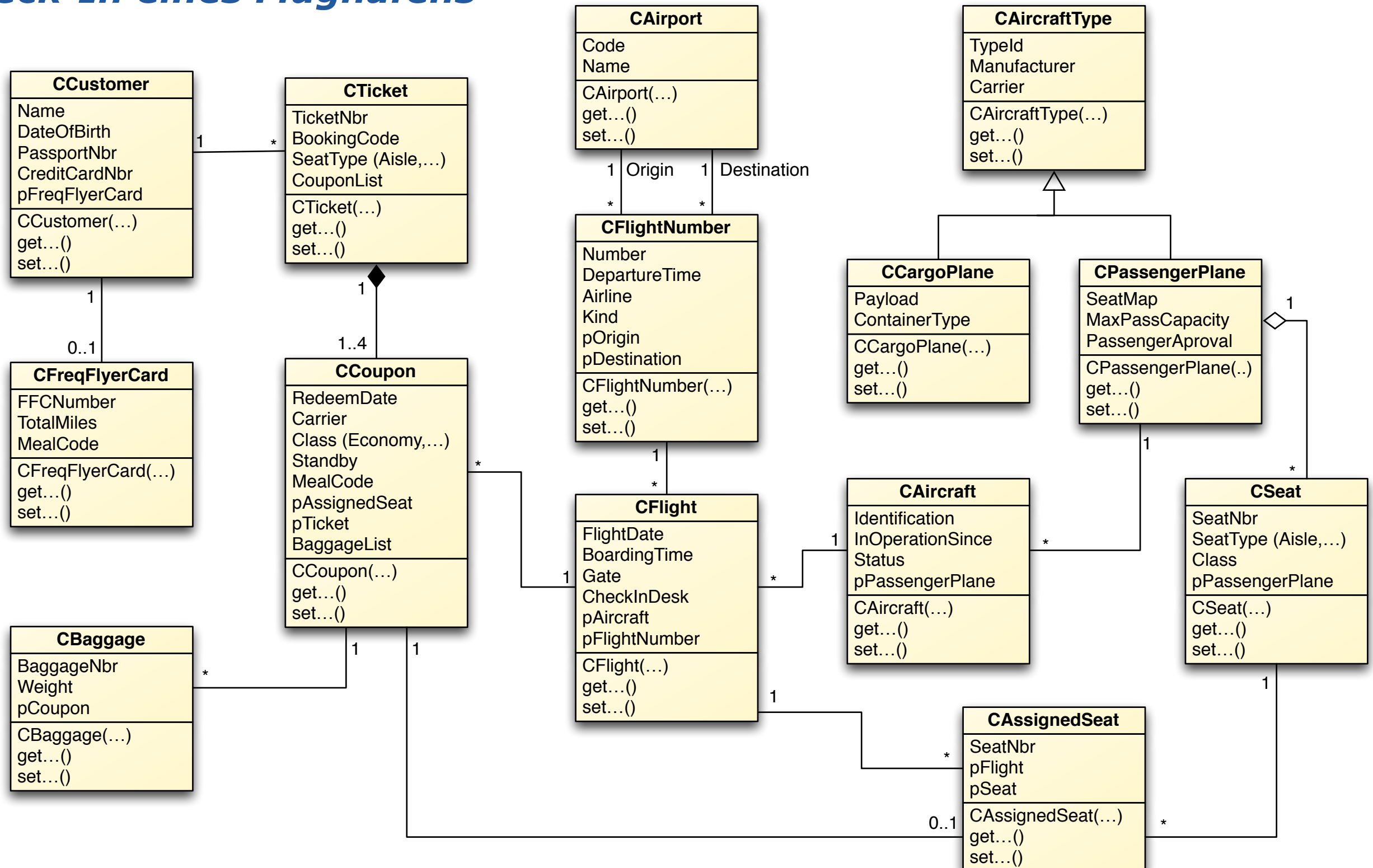


## Beispiel für ein Betriebsablauf-Sequenzdiagramm

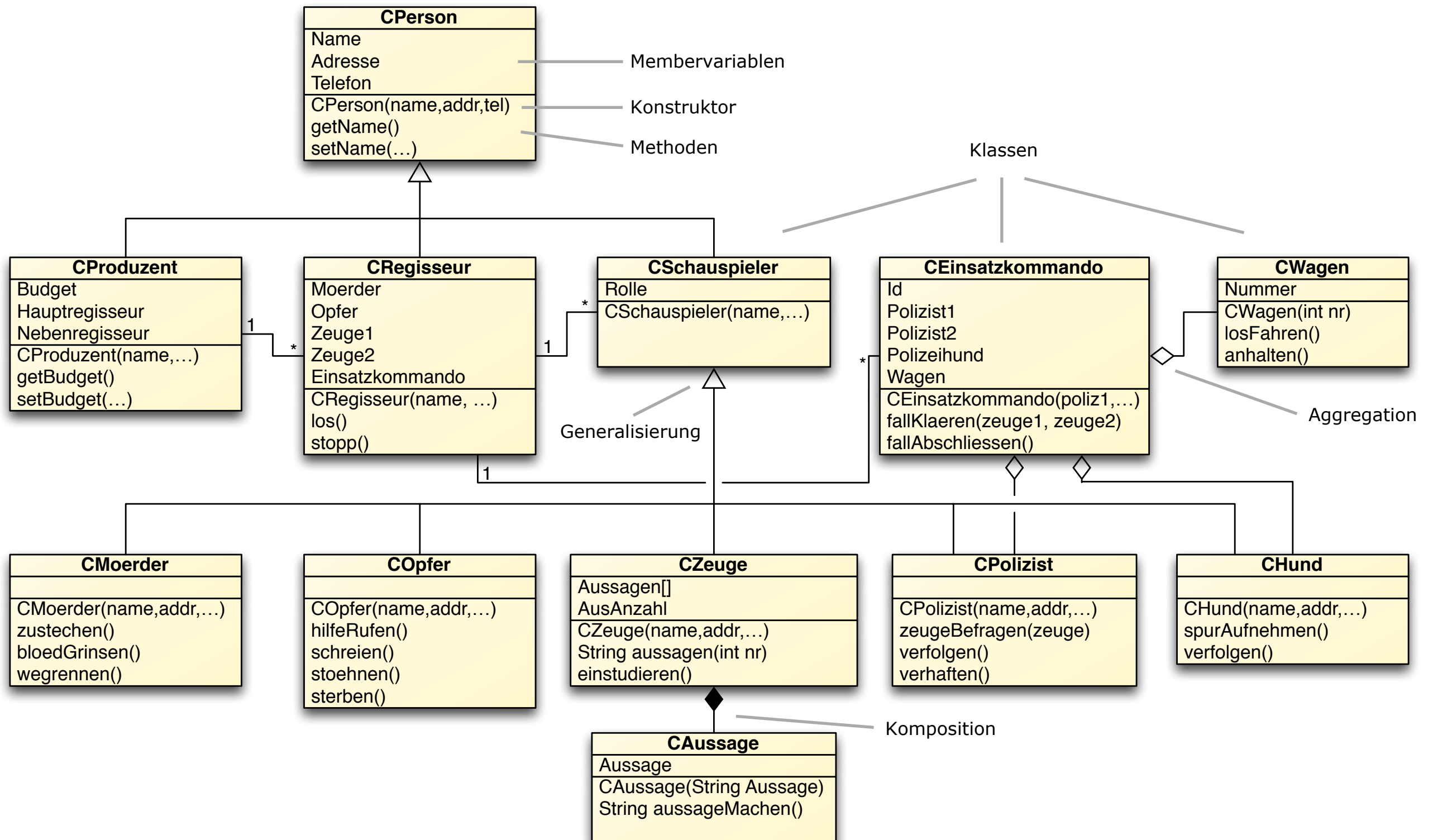
### Passagierabfertigung Flughafen (nicht OOP!)



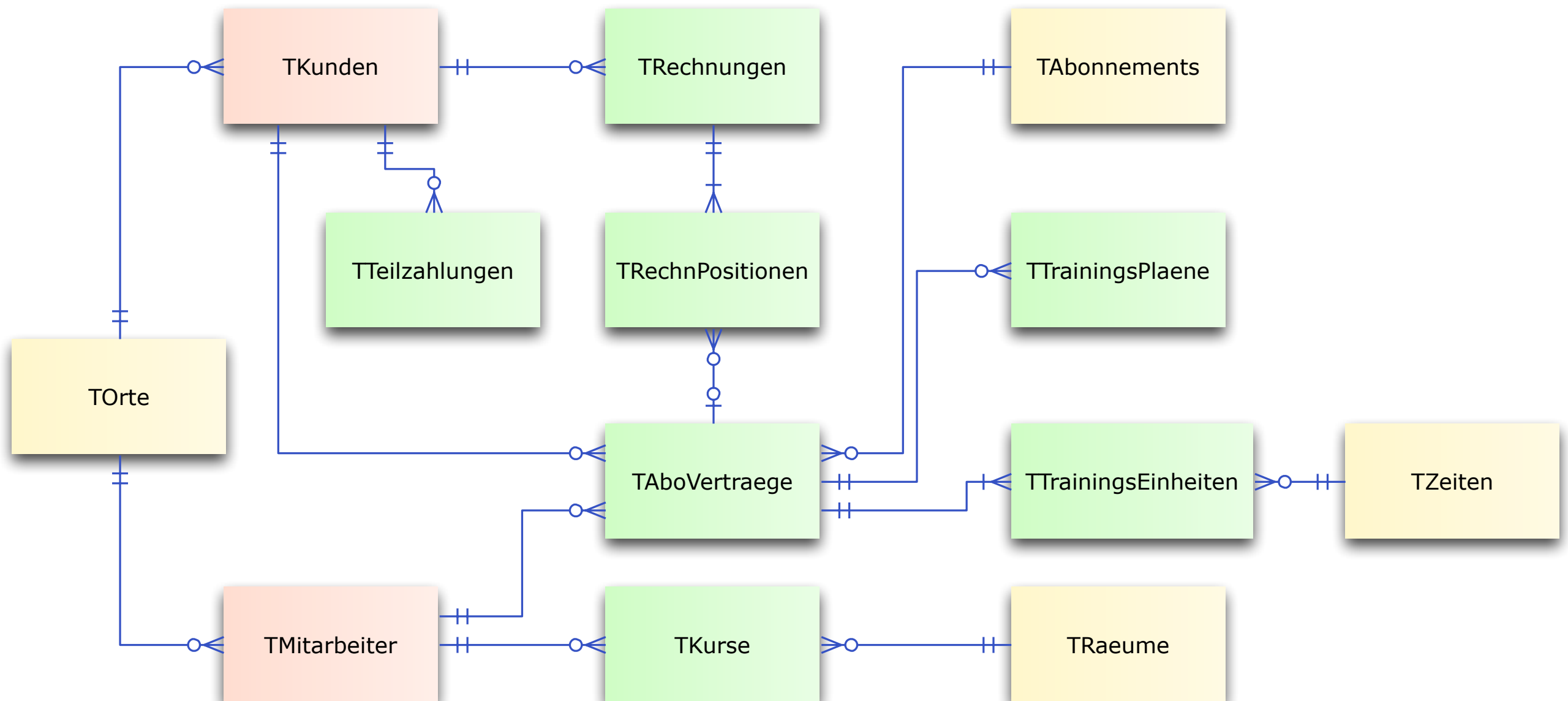
# Beispiel für ein Klassendiagramm für den Check-In eines Flughafens



# Beispiel für ein Klassendiagramm für die Verfilmung eines Krimis



## Beispiel für ein ERM für ein Fitness-Studio



## Testing

Um sicher zu gehen, dass sich eine Applikation auch so verhält, wie sie sich verhalten soll, müssen wir Tests durchführen. Bei den Tests ist es äusserst wichtig methodisch und strukturiert vorzugehen und somit wirklich alle Teilverhalten, Aspekte und Bedingungen genau zu prüfen und die Ergebnisse gut zu dokumentieren. Man muss sich jedoch bewusst sein, dass man nie alles testen kann ( $2^{\text{Anzahl if's}}$ !).

Man notiert sich dazu bereits in der Designphase so genannte Testfälle, die die wichtigsten Funktionalitäten abdecken sollten. Dabei unterscheidet man zwischen funktionalen und destruktiven Testfällen.

**Funktionale Testfälle** demonstrieren, dass die Applikation, wenn der Benutzer vernünftige, "gute" und wohlwollende Eingaben macht und korrekte Daten vorhanden sind, korrekte Ergebnisse liefert und gemäss Spezifikation funktioniert.

**Destruktive Tests** hingegen zeigen, dass die Applikation auch dann robust und vernünftig reagiert, wenn der Benutzer falsche und unmögliche Eingaben macht oder importierte Daten fehlerbehaftet sind.

Die Testfälle sollten dann möglichst von Anwendern, mindestens aber von "Nicht-Entwicklern" durchgespielt werden.

Wann werden Testfälle formuliert?

- Tests dann überlegen, wenn man noch die Übersicht hat, also in der Designphase.

Wie werden Testfälle formuliert?

- Als Grundlage verwendet man die Use-Cases und überlegt sich für jeden Use-Case mehrere Testfälle.
- Die Tests, die erwarteten Resultate und das Ergebnis müssen sehr präzise formuliert werden. Nur so können sie reproduzierbar immer gleich und bei jedem Release erneut getestet werden.
- Wir erstellen zwei Testtabellen, eine mit **funktionalen** Tests und eine mit **destruktiven** Tests.
- Die Tests sollen so formuliert sein, dass sie von jedermann durchgeführt werden können und absolut reproduzierbare Ergebnisse liefern.



## Beispiele für Testfälle

Getestet von: **David Steger**Getestete Version: **1.12**Test-Datum: **4.2.2021**

### Funktionale Testfälle

| Test   | Ergebnis  | Bestanden |
|--|---|-----------|
| Layout Kunde: Button "Kunde Neu" anklicken.  | Neuer Datensatz mit neuer Kundennummer wird angelegt, alle anderen Kunden-Eingabefelder (Anrede, ..., Telefon) sind leer, Cursor steht im Feld Anrede.      | ✓         |
| Layout Kunde: Button "Kunde Neu" anklicken, Feld Geburtsdatum anklicken, Eingabe von 14.8.01 in das Eingabefeld Geburtsdatum, Tabulator Taste drücken. | Geburtsdatum 14.08.01 erscheint im Eingabefeld. Cursor steht im Feld Telefonnummer.   | ✗         |
| Layout Kunde: Button "Videos" anklicken, 2. Video anklicken, Button "Kunde" anklicken, Button "Videos" anklicken.                                      | Videos Layout wird mit dem gleichen 2. Datensatz angezeigt wie dieses Layout das letzte Mal verlassen wurde.  | ✓         |
| Layout Kunde: In Kunden-Liste "Susanne Lèlange-Rüdisühli" anklicken.   | Der komplette Datensatz von Susanne Lèlange-Rüdisühli mit den Feldern Anrede, Knd. Nr. bis Telefon (siehe Anhang) alle korrekt ausgefüllt werden angezeigt. | ✓         |

### Destruktive Testfälle

| Test  | Ergebnis   | Bestanden |
|---|--|-----------|
| 3 Benutzer gleichzeitig: Layout Kunde: Button "Kunde Neu" anklicken, neuen Kunden Herr Klaus Biedermann, Hauptstrasse 17, 8521 Pfyn, 12.3.92, 071 432 56 47 eintragen, in den weissen Rand klicken. | Bei einem Benutzer wird der Datensatz übernommen, bei den anderen erscheint die Meldung Kunde bereits vorhanden. | ✓         |
| Layout Kunde: Button "Kunde Neu" anklicken, Feld Geburtsdatum anklicken, Eingabe von 14.8.21 in das Eingabefeld Geburtsdatum, Tabulator Taste drücken.  | Es erscheint die Meldung: Geburtsdatum liegt in der Zukunft. Nach OK steht Cursor auf Feld Geburtsdatum.         | ✗         |
| Layout Kunde: Button "Kunde Löschen" so viele Male drücken bis kein Kunde mehr in der linken Liste. Dann noch ein weiteres Mal drücken.   | Button nicht mehr aktiv. Es passiert nichts.   | ✗         |

## Beispiel Testfälle für die App SuperCalc

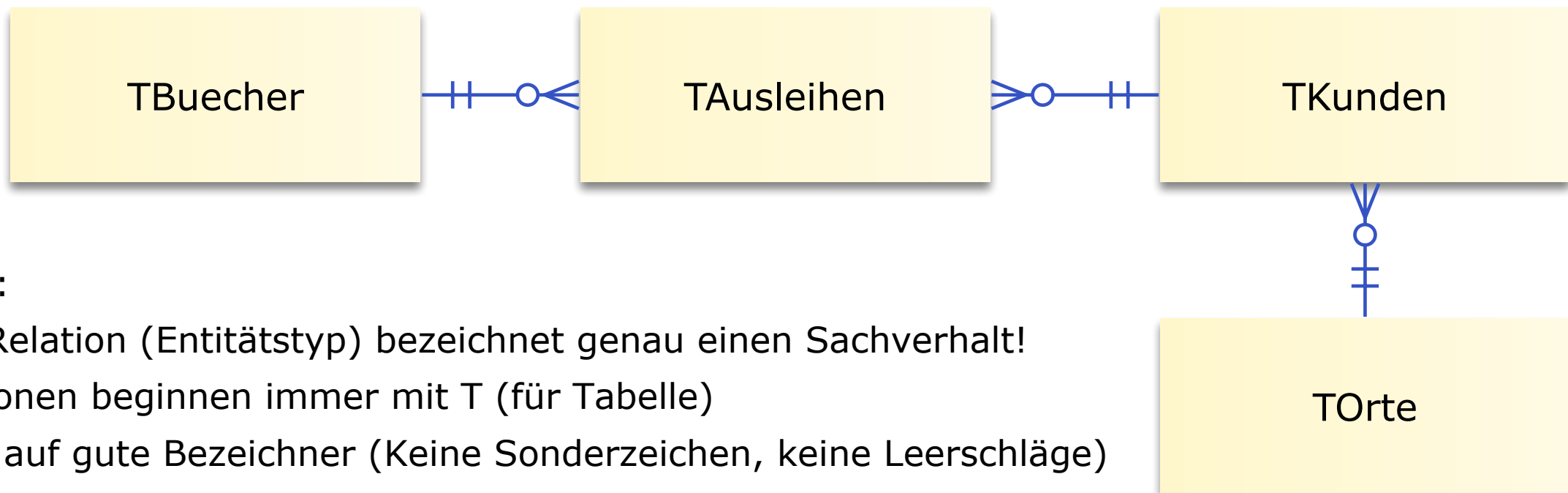
| SuperCalc            |   | Vers. 0.85   |      |        |       |
|----------------------|---|--|------|--------|-------|
| Nr.                  | Testfall  | Resultat   | Test |        | Visum |
|                      |   |  | OK   | Failed |       |
|                      | Konstruktive Tests:   |  |      |        |       |
| 1                    | Das Fenster wird durch click & drag an der rechten unteren Ecke aufs Maximum vergrößert.      | Der Inhalt ist ansprechend angeordnet und gut über das Fenster verteilt.   |      |        |       |
| 2                    | Das Fenster soll durch click & drag an der rechten unteren Ecke aufs Minimum verkleinert.     | Der Inhalt ist ansprechend angeordnet. Nichts ist verdeckt oder abgeschnitten. Alles kann gelesen werden.          |      |        |       |
| 3                    | Das Datum 7.2.1824 wird gefolgt von der Enter Taste per Tastatur ins Datumfeld eingegeben.    | Im Ausgabefeld wird: "Dies ist ein Samstag." ausgegeben.   |      |        |       |
| 4                    | Die Applikation WeekDayCalc wird mit Doppelclick auf WeekDayCalc.jar geöffnet.                | Die Applikation öffnet sich. Man kann wie gewohnt damit arbeiten.  |      |        |       |
|                      | Destruktive Tests:  |  |      |        |       |
| 5                    | Ins Ausgabefeld per Tastatur "Hallo" schreiben.   | Das Ausgabefeld lässt sich nicht anklicken und auch nicht beschreiben.   |      |        |       |
| 6                    | Das Datum 7.2.1824 wird gefolgt von 6 x Tabulatortaste per Tastatur ins Datumfeld eingegeben. | Fokus springt zwischen dem Berechnen-Knopf und dem Eingabefeld hin und her und ist am Schluss auf dem Eingabefeld. |      |        |       |
|                      |   |  |      |        |       |
| Tests ausgeführt am: |   | Unterschrift:  |      |        |       |

# Entwicklung einer relationalen Datenbankapplikation

## 1. ERM Entity Relationship Model

Als erstes benötigen wir ein normalisiertes bereinigtes ERM.

Hier am Beispiel einer Bibliothek.



Wichtig:

- Jede Relation (Entitätstyp) bezeichnet genau einen Sachverhalt!
- Relationen beginnen immer mit T (für Tabelle)
- Achte auf gute Bezeichner (Keine Sonderzeichen, keine Leerschläge)

Jede Beziehung besteht aus 2 Assoziationen, welche wie folgt gelesen werden:

- **Ein** Buch kann **nie bis mehrmals** ausgeliehen werden.
- **Eine** Ausleihe bezieht sich auf **genau ein** Buch.

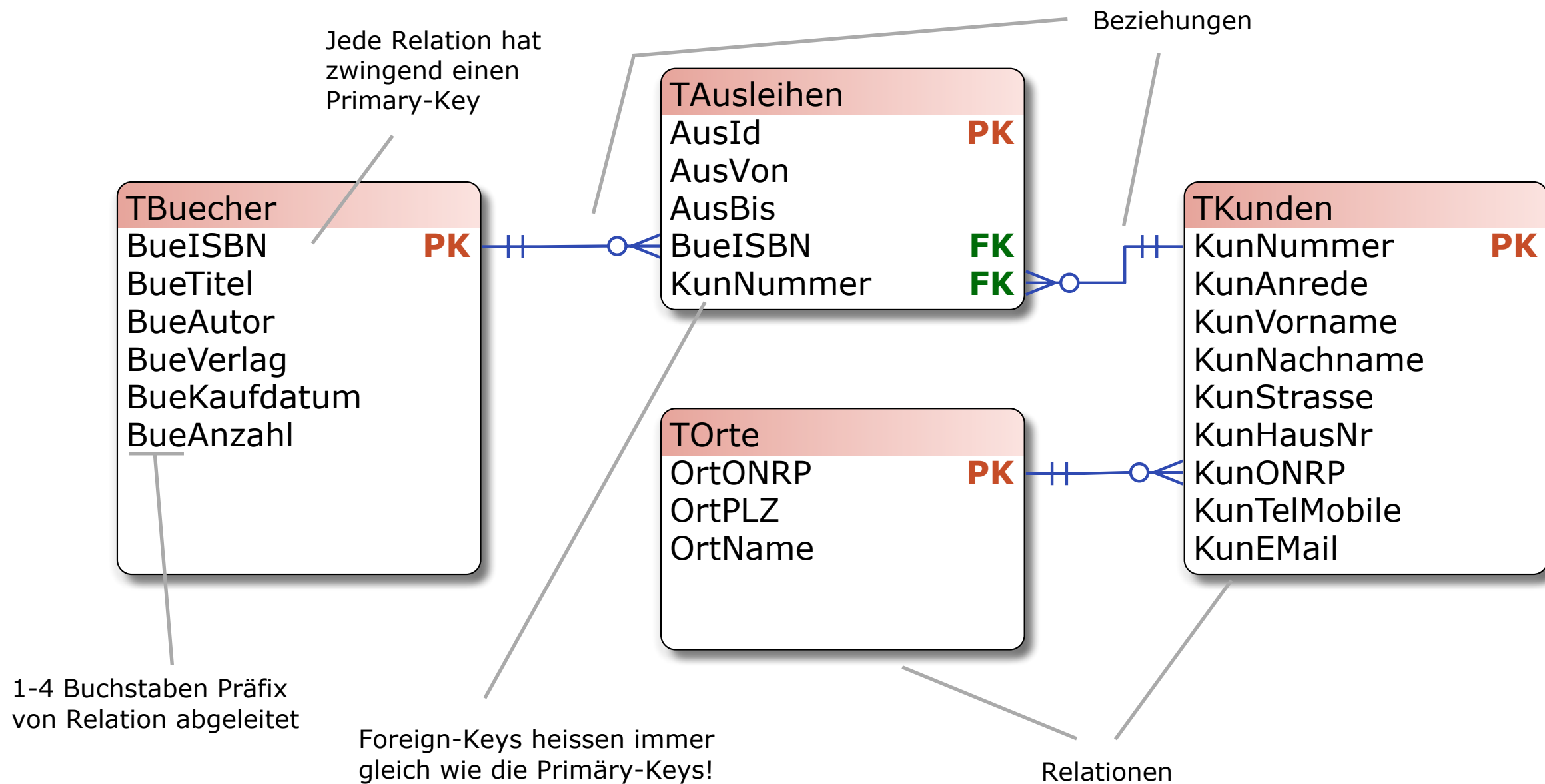
Immer EINS!

## 2. Datenbank-Schema

Dieses ERM bauen wir zu einem Datenbank-Schema aus.

Ein Datenbank-Schema ist ein ERM ergänzt mit Attributen, Primär- und Fremdschlüsseln.

Auch hier achten wir auf gute Bezeichner (Präfixe verwenden, Kamelnotation, keine Sonderzeichen, keine Leerschläge, siehe Grafik)!



## DB-Fachbegriffe

