



Anwendungsbeispiel: Analyse



Inhalt

- Requirements Engineering Prozess
- Use-Case Modellierung
- Domänenmodell
- ERM Diagramme

Anwendungsbeispiel: Computerspieleplattform

















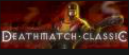


Computerspieleplattform „DAMPF“

Die Firma VAULT möchte eine Internet-Vertriebsplattform für Computerspiele erstellen bei denen Hersteller ihre Computerspiele anbieten können. Benutzer können sich die Plattform herunterladen und installieren und bequem die angebotenen Spiele durchsuchen, filtern und direkt kaufen. Gekaufte Spiele werden in einer persönlichen Bibliothek gespeichert auf die der Benutzer jederzeit Zugriff hat. Spielehersteller wie auch Benutzer müssen einen Account für die Plattform besitzen. Wenn ein Spiel von einem Benutzer über die Plattform erworben wird bekommt VAULT einen gewissen Prozentsatz des Umsatzes. Für die Bezahlung sollen verschiedene Methoden angeboten werden. Zudem soll es möglich sein das ein Benutzer ein erworbenes Spiel zurückgeben kann, wenn er dieses weniger als 10h gespielt hat. Auch werden Aktualisierungen für die Spiele ("Updates") direkt von den Herstellern der Spiele über das System verteilt.

Anwendungsbeispiel: Computerspieleplattform

Computerspieleplattform „DAMPF“

Die Firma VAULT möchte eine Internet-Vertriebsplattform für Computerspiele erstellen bei denen Hersteller ihre Computerspiele anbieten können. Benutzer können sich die Plattform herunterladen und installieren und bequem die **angebotenen Spiele durchsuchen, filtern und direkt kaufen**. Gekaufte Spiele werden in einer persönlichen Bibliothek gespeichert auf die der Benutzer jederzeit Zugriff hat. Spielehersteller wie auch Benutzer müssen einen Account für die Plattform besitzen. Wenn ein Spiel von einem Benutzer über die Plattform erworben wird bekommt **VAULT** einen gewissen **Prozentsatz des Umsatzes**. Für die Bezahlung sollen verschiedene Methoden angeboten werden. Zudem soll es möglich sein das ein Benutzer ein erworbenes **Spiel zurückgeben** kann, wenn er dieses weniger als 10h gespielt hat. Auch werden Aktualisierungen für die Spiele ("Updates") direkt von den Herstellern der Spiele über das System verteilt.

← → STORE LIBRARY		STEAMOS + LINUX		VIEW	
GAMES		★	☁	STATUS	METASCORE
					LAST PLAYED
	Amnesia: The Dark Descent			Not installed	85
	Antichamber			Not installed	82
	Aquaria			Not installed	82
	Audiosurf 2		☁	Not installed	76
	The Banner Saga		☁	Not installed	80
	Bastion		☁	Not installed	86
	BioShock Infinite		☁	Not installed	94
	Borderlands 2		☁	Not installed	89
	Braid		☁	Not installed	90
	Broforce		☁	Not installed	83
	Chaos on Deponia		☁	Not installed	78
	Counter-Strike			Not installed	88
	Counter-Strike: Global Offensive		☁	Not installed	83
	Counter-Strike: Source		☁	Not installed	88
	Darkest Dungeon		☁	Not installed	84
	Day of Defeat			Not installed	79
	Dead Island		☁	Not installed	80
	Deathmatch Classic			Not installed	
	Deponia		☁	Not installed	74

Requirements Engineering Prozess

Requirements Engineering						
Requirements Analysis					Requirements Management	
Elicitation	Interpretation	Negotiation	Documentation	Validation	Change Management	Tracing
<ul style="list-style-type: none"> ➤ Identification of stakeholders and other sources of requirements (e.g., old existing systems) ➤ Gathering of raw requirements ➤ Target analysis 	<ul style="list-style-type: none"> ➤ Identification of requirements ➤ Structuring of requirements (merging, grouping, classification) ➤ Refinement (to satisfiable criteria) 	<ul style="list-style-type: none"> ➤ Identification of dependencies ➤ Identification of inconsistencies ➤ Resolution of inconsistencies ➤ Prioritization 	<ul style="list-style-type: none"> ➤ Specification of requirements (incl. intermediate results and assumptions) 	<ul style="list-style-type: none"> ➤ Contentual validation (correctness, completeness, consistency) ➤ Formal verification 	<ul style="list-style-type: none"> ➤ Handling of change requests ➤ Management of different versions of the requirements (history) ➤ Propagation of changes 	<ul style="list-style-type: none"> ➤ Recording of assumptions and requirement sources ➤ Recording of how requirements are implemented

Requirements Elicitation (1)

- Identifizierte Stakeholder
 - Spielehersteller, Benutzer, VAULT, Tester, „Hacker“, Infrastruktur-Ingenieur, ...
- Priorisieren der Stakeholder
- Identifizierte zusätzliche Quellen
 - GOG Galaxy, Epic Games Launcher, ...

Requirements Elicitation (2)

- Identifizierte Requirements (u.a.)
 - Req-1: Benutzern sollte es möglich sein Spiele zu erwerben
 - Req-2: Der Abschluss des Kaufes eines Spiels sollte in weniger als 5 Sekunden abgeschlossen sein
 - Req-3: Das System soll eine Web Oberfläche und ein GUI bieten

Requirements Interpretation (1)

Req-1: Benutzern sollte es möglich sein Spiele zu erwerben

Req-2: Der Abschluss des Kaufes eines Spiels sollte in weniger als 5 Sekunden abgeschlossen sein

Req-3: Das System soll eine Web Oberfläche und ein GUI bieten

■ Identifizierte Beziehungen

- ☐ Req-1 wird für Req-2 benötigt

- ☐ Req-3 muss separiert werden

 - Req-3: Das System soll eine Web Oberfläche besitzen

 - Req-4: Das System soll eine Desktop GUI besitzen

Requirements Interpretation (2)

Req-1: Benutzern sollte es möglich sein Spiele zu erwerben

Req-2: Der Abschluss des Kaufes eines Spiels sollte in weniger als 5 Sekunden abgeschlossen sein

Req-3: Das System soll eine Web Oberfläche besitzen

Req-4: Das System soll eine Desktop GUI besitzen

■ Klassifikation der Requirements

- ☐ Req-1: Funktional
- ☐ Req-2: Nicht funktional
- ☐ Req-3: Funktional
- ☐ Req-4: Funktional

Requirements Interpretation (3)

Req-1: Benutzern sollte es möglich sein Spiele zu erwerben

Req-2: Der Abschluss des Kaufes eines Spiels sollte in weniger als 5 Sekunden abgeschlossen sein

Req-3: Das System soll eine Web Oberfläche besitzen

Req-4: Das System soll eine Desktop GUI besitzen

■ Gruppierung der Requirements

- ☐ Spieleerwerb: Req-1, Req-2
- ☐ User Interface: Req-3, Req-4

Requirements Interpretation (4)

Req-2: Der Abschluss des Kaufes eines Spiels sollte in weniger als 5 Sekunden abgeschlossen sein

- Konkretisierung der Requirements mit Kunden

- Req-2: Was heißt „Abschluss des Kaufes“?

- Der gesamte Prozess von Auswahl bis zur Eintragung in die Spielebibliothek?
 - Der letzte Schritt vom Drücken eines „Kaufen“ Buttons bis in die Eintragung in die Bibliothek?

- Glossar

Requirements Negotiation (1)

- Bei den Verhandlungen sind keine großen Streitpunkte aufgetreten
- Identifizierte Bedingungen
 - Req-1 wird für Req-2 benötigt
- Keine Inkonsistenzen in den Requirements vorhanden

Requirements Negotiation (2)

Req-1: Benutzern sollte es möglich sein Spiele zu erwerben

Req-2: Der Abschluss des Kaufes eines Spiels sollte in weniger als 5 Sekunden abgeschlossen sein

Req-3: Das System soll eine Web Oberfläche besitzen

Req-4: Das System soll eine Desktop GUI besitzen

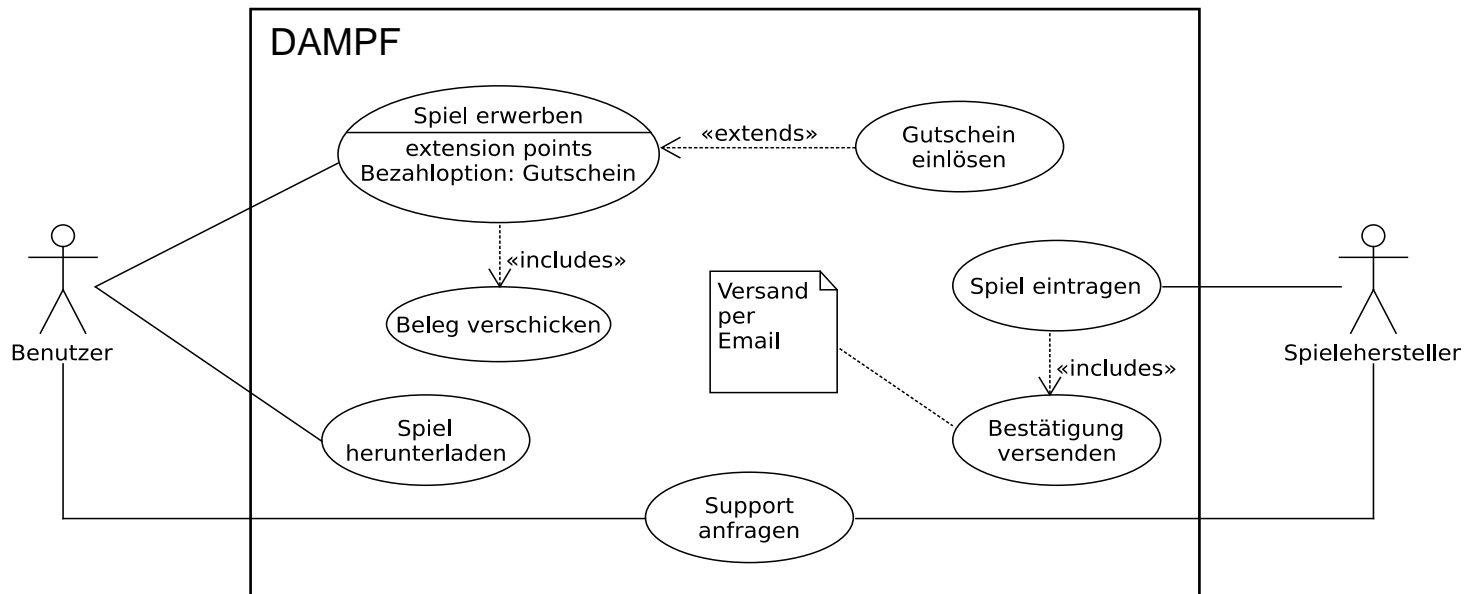
■ Priorisierung nach Kundenabsprache

- ☐ Req-1, Req-4: Must-have

- ☐ Req-2, Req-3: Optional

Requirements Dokumentation (1)

- Es wurde Entschieden einen Mix aus natürlicher Sprache und UML-Anwendungsfalldiagrammen zu benutzen



Requirements Dokumentation

(2)

Anwendungsfall:

Spiel herunterladen

Zusammenfassung:

Prozess eines Benutzers um ein Spiel herunterzuladen. Hierfür sucht sich der Benutzer ein noch nicht heruntergeladenes Spiel aus seiner Bibliothek für erworbene Spiele aus.

Akteure:

Benutzer

Auslöser:

Benutzer möchte ein erworbenes Spiel herunterladen.

Vorbedingungen:

Spiel muss in der Bibliothek für erworbene Spiele vorhanden sein.

Requirements Dokumentation

(3)

Standardablauf:

1. Benutzer startet DAMPF.
2. Benutzer loggt sich ein.
3. Benutzer navigiert zu seiner Bibliothek.
4. Benutzer wählt zu heruntergeladenes Spiel aus.
5. Benutzer betätigt herunterladen Button.
 - 5.1. Benutzer gibt Download-Verzeichnis an.
 - 5.2. Benutzer bestätigt Start des Downloads.

Nachbedingungen:

Spiel wird heruntergeladen.

Ausnahmen und Varianten:

Zu 5.1: Download-Verzeichnis bietet nicht genügend Speicherplatz: Wdh. Schritt 5.1.

Requirements Dokumentation (4)

Ergebnis:

Spiel ist heruntergeladen.

Häufigkeit:

5 mal pro Monat.

Verweise:

Glossar


Anmerkungen:

Download kann jederzeit abgebrochen werden.



Requirements Validation

- Alle erstellten Dokumente haben einen dedizierten Reviewprozess durchlaufen
- Es wurde überprüft ob die Wünsche der Kunden berücksichtigt wurden



Requirements Change Management and Tracing

Change Managemnet

- Verwalten von angefragten Änderungen
- Sollte systematisch gehandhabt werden
- Bspw. durch Workflow der den Ablauf einer Änderungsanfrage durchläuft
- Vielzahl von existierenden Tools
 - Erlaubt erstellen von Change Request Workflows
 - Bspw: Jira

Tracing

- Nach/Rückverfolgung von Anforderungen
- Zeigt abdeckung der Anforderungen
- Verschieden Methoden:
 - Bspw: Traceability Matrix

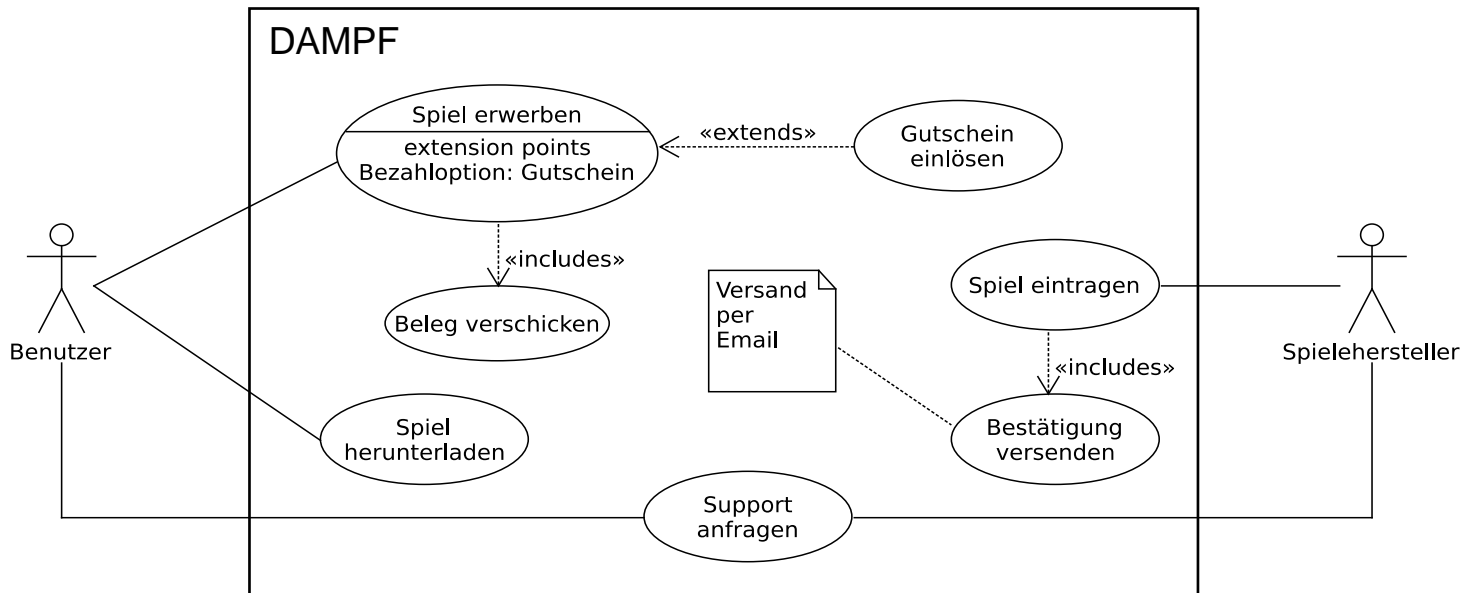
Req No.	Beschreibung	Testcase	Status
1	Benutzer sollen Spiele erwerben	TC3,TC5	TC3 pass, TC5 fail
2	Abschluss des Spielekaufs in weniger als 5 Sekunden	TC2, TC4	TC2 no run, TC4 pass
3	Das System soll eine GUI bieten	TC1	TC1 pass,



Fragen?



Use-Case Modellierung



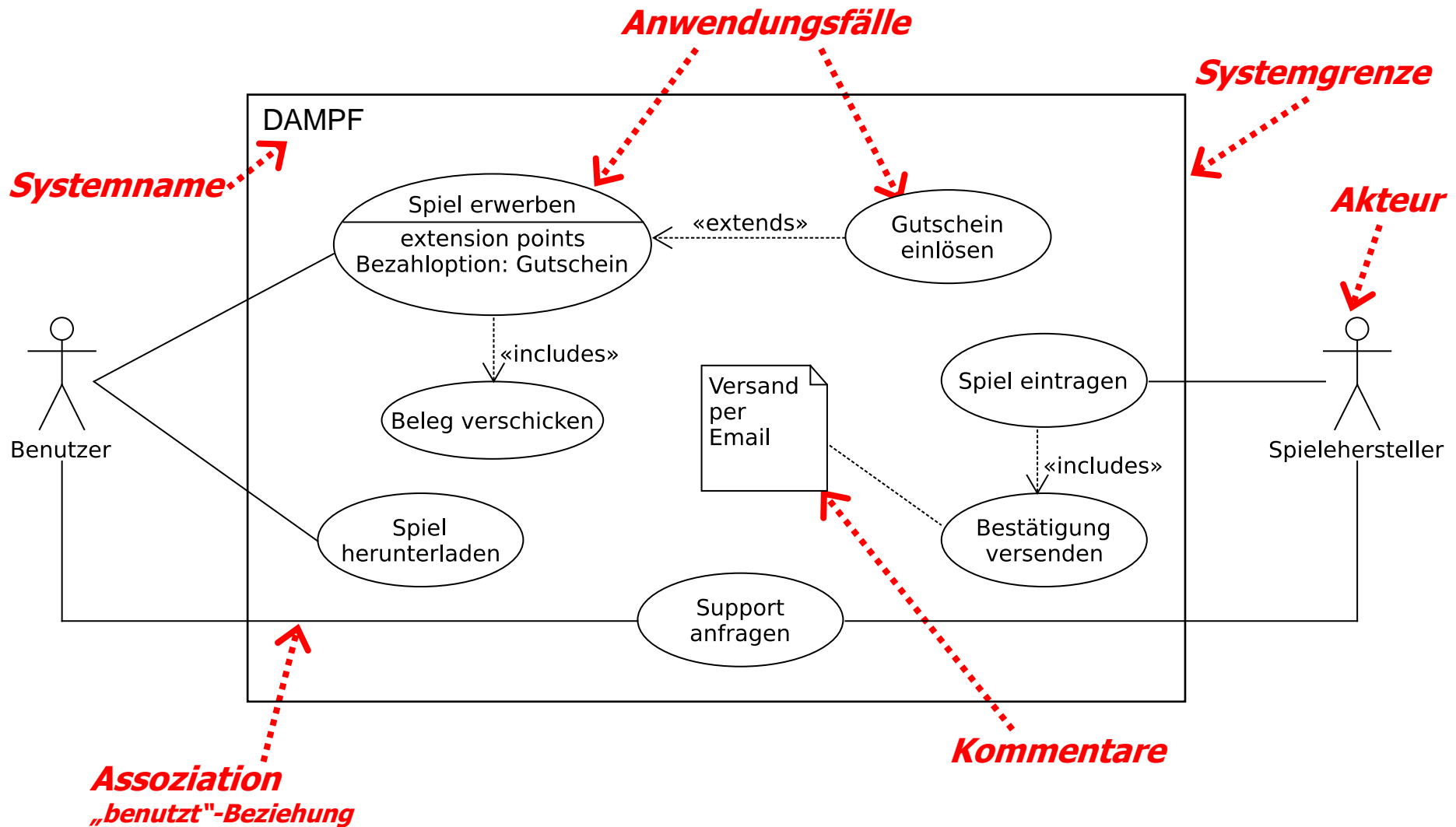
Was ist ein Use Case?

- Etwas, was Akteure mit dem System tun können
- Akteure sind Personen oder andere Systeme
- Zu ermittelnde Aspekte
 - Systemgrenze
 - Akteure
 - Anwendungsfälle
 - Beziehungen

Informationen für ein Use-Case

- Geschäftsmodelle
 - Aus dem Umfeld des zu modellierenden Systems
- Anforderungsmodell
 - Funktionale Anforderungen (Anwendungsfälle und Akteur)
 - Nicht-funktionale Anforderungen und Constraints (Randbedingungen für die Modellierung)
- „Feature list“
 - List von Merkmalen des Systems, z.B. in Form eines Visionsdokuments

Use-Case: Elemente



Use-Case Modellierung: Vorgehen

- Finde eine mögliche Systemgrenze
- Finde Akteure
- Finde Anwendungsfälle
 - Spezifiziere Anwendungsfälle
 - Identifiziere alternative Abläufe
- Finde Beziehungen
- Wiederhole bis stabiler Zustand erreicht

Systemgrenze

■ Zentrale Fragen:

- ☐ Was gehört zu meinem System?
- ☐ Was liegt außerhalb?

■ Definiert durch

- ☐ Wer oder was benutzt das System?
 - → Akteure, die **außerhalb** des Systems stehen
- ☐ Was bietet das System den Nutzern?
 - → Anwendungsfälle, die den **Inhalt** des Systems definieren

Akteure

- Ein Akteur ist eine Rolle, die eine externe Entität bei direkter Interaktion mit dem System spielt.
 - Nutzer des Systems
 - Andere Systeme oder Hardware
 - Zeit
- Eine Person (oder ein System) kann mehrere Rollen spielen
 - Akteure sind Rollen, **nicht spezifische Personen**
 - Beispiel:
 - Akteure „Fluggast“ und „Pilot“
 - Person Hans Mustermann kann Pilot, Fluggast, Pilot und Fluggast (evtl. sogar gleichzeitig) oder weder Pilot noch Fluggast sein
- Akteure sind immer außerhalb der Systemgrenze
- Jeder Akteur braucht
 - Einen kurzen, sinnvollen Namen
 - Eine kurze Beschreibung aus Sicht des Geschäftsmodells

Akteure finden

- Wer oder was benutzt das System?
- Welche Rollen können die Benutzer spielen?
- Wer installiert das System?
- Wer oder was startet und beendet das System?
- Wer wartet das System?
- Welche Systeme interagieren mit diesem System?
- Wer oder was erhält bzw. liefert Informationen?
- Gibt es Dinge, die zu festen Zeiten passieren?

Anwendungsfälle

- Definition nach UML Reference Manual:
 - Eine Abfolge von Aktionen, inkl. Alternativen und Fehlerbehandlungen, die ein System, Untersystem oder Klasse bei Interaktion mit externen Akteuren ausführen kann
- Kurz: Etwas, was das System für einen Akteur tun soll
 - Anwendungsfälle werden **immer von einem Akteur gestartet**
 - Anwendungsfälle werden **immer aus der Sicht der Akteure beschrieben**

Anwendungsfälle finden

- Welche Funktionen erwartet ein bestimmter Akteur von dem System?
 - ☐ Liste der Akteure durchgehen
- Speichert oder Liefert das System Informationen?
 - ☐ Welcher Akteur löst dieses Verhalten aus?
- Was passiert, wenn das System den Zustand ändert (z.B. Starten und Beenden)?
 - ☐ Werden Akteure benachrichtigt?
- Wird das System durch externe Ereignisse beeinflusst?
 - ☐ Wer teilt dem System das mit?
- Interagiert das System mit externen Systemen?
- Erzeugt das System berichte?

Beziehungen

- Einfachster Fall: Beziehung zwischen Akteur und Anwendungsfall
- Spezielle Beziehungen
 - Generalisierung von Akteuren
 - Generalisierung von Anwendungsfällen
 - <<include>>
 - Ein Anwendungsfall beinhaltet Verhalten eines Anderen
 - <<extend>>
 - Ein Anwendungsfall wird durch zusätzliches Verhalten erweitert

Generalisierung

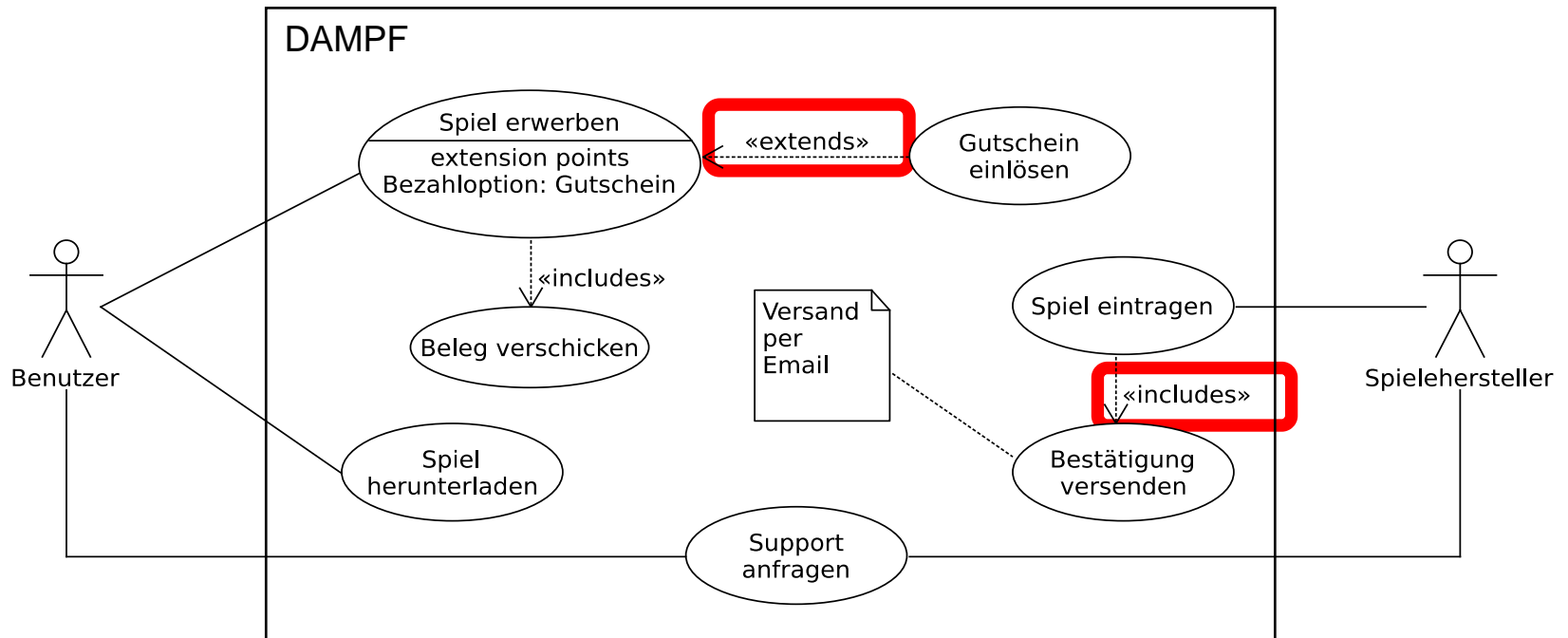
■ Generalisierung von Akteuren

- Akteure interagieren ähnlich mit dem System
 - Gemeinsames Verhalten durch Generalisierung ausgliedern
 - Spezialisierungen erben alle Beziehungen und Rollen

■ Generalisierung von Anwendungsfällen

- Ein oder mehrere Anwendungsfälle sind echte Spezialisierungen eines allgemeinen Falls
- Nur sinnvoll, wenn das Modell einfacher wird
- Spezialisierung erben automatisch alle Eigenschaften des allgemeinen Falls

Includes und Extends



- **<<includes>>** (use): besitzen mehrere Anwendungsfälle gleiche Teilabläufe wie das Verschicken einer Benachrichtigung, so können diese Teilabläufe separat beschrieben werden; es wird also Funktionalität ausgelagert und wiederverwendet.
- **<<extends>>**: Sonderfälle oder Varianten eines Anwendungsfalles werden oft als separate Anwendungsfälle beschrieben; es wird also Funktionalität zu einem Anwendungsfall hinzugefügt. **Extension point** beschreibt wann die erweiterte Funktionalität ausgeführt wird.

Use-case Modellierung

- Im Requirements Engineering wird die **Sprache der Anwender** benutzt
- Konsequenz für Anwendungsfalldiagramme:
 - Nur so komplex, dass Kunden sie noch verstehen können
 - Erfassen der gewünschten Systemfunktionen
- Zusätzlich zu den Diagrammen wird für jeden Use-Case eine **detaillierte Beschreibung** angefertigt
- Faustregel
 - Kurz und einfach
 - Was, nicht wie

Best Practices

■ Akteure

- ☐ einfach abgrenzbare Rollen
- ☐ nicht zu feingranular

■ Use Cases

- ☐ keine Abläufe
- ☐ meistens „Objekt Verb“
 - (z.B. „Pizza backen“)
- ☐ keine eierlegenden Wollmilchsäue
 - („XXX managen“ oder „XXX verwalten“)



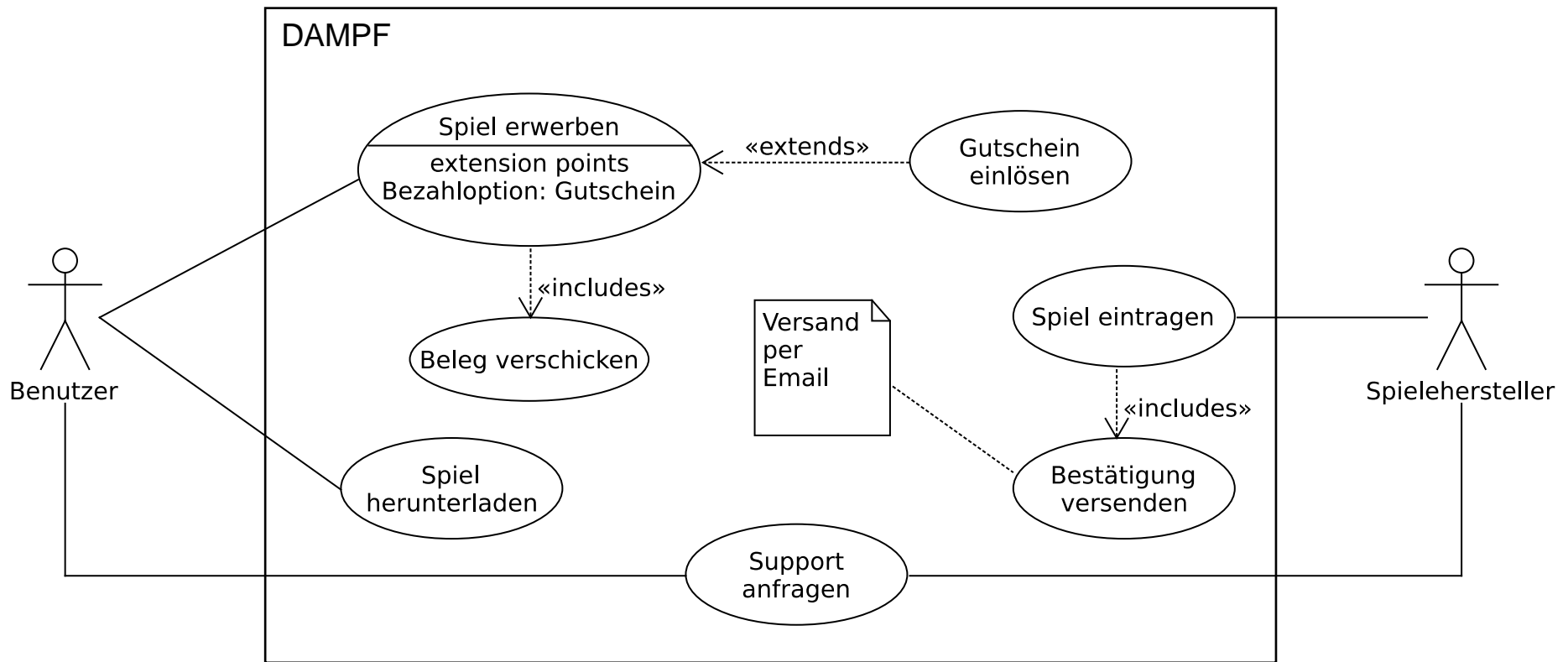
Fragen?



Domänenmodell

Erstellen eines Domänenmodells

Ausgangslage



Erstellen eines Domänenmodells

CRC-Karten Prozess

- Identifizieren von potentiellen Klassen (Entitätstypen)
 - Erstellen von „Blanko“ CRC-Karten
- Durchspielen von Anwendungsfällen
 - Verteilen der CRC-Karten an Personen
 - Wenn Person im Anwendungsfall „dran“ ist hebt er seine Karte
 - Trägt dann Responsibilities und ggfs. Collaborations ein

Erstellen eines Domänenmodells

Entitätstypen Identifizieren

Computerspieleplattform „DAMPF“

Die Firma VAULT möchte eine Internet-Vertriebsplattform für Computerspiele erstellen bei denen Hersteller ihre Computerspiele anbieten können. Benutzer können sich die Plattform herunterladen und installieren und bequem die angebotenen Spiele durchsuchen, filtern und direkt kaufen. Gekaufte Spiele werden in einer persönlichen Bibliothek gespeichert auf die der Benutzer jederzeit Zugriff hat. Spielehersteller wie auch Benutzer müssen einen Account für die Plattform besitzen. Wenn ein Spiel von einem Benutzer über die Plattform erworben wird bekommt VAULT einen gewissen Prozentsatz des Umsatzes. Für die Bezahlung sollen verschiedene Methoden angeboten werden. Zudem soll es möglich sein das ein Benutzer ein erworbenes Spiel zurückgeben kann, wenn er dieses weniger als 10h gespielt hat. Auch werden Aktualisierungen für die Spiele ("Updates") direkt von den Herstellern der Spiele über das System verteilt.

Erstellen eines Domänenmodells

Entitätstypen Identifizieren

Computerspieleplattform „DAMPF“

Die Firma VAULT möchte eine **Internet-Vertriebsplattform** für **Computerspiele** erstellen bei denen **Hersteller** ihre Computerspiele anbieten können. **Benutzer** können sich die Plattform herunterladen und installieren und bequem die angebotenen **Spiele** durchsuchen, filtern und direkt kaufen. Gekaufte Spiele werden in einer persönlichen **Bibliothek** gespeichert auf die der Benutzer jederzeit Zugriff hat. Spielehersteller wie auch Benutzer müssen einen **Account** für die Plattform besitzen. Wenn ein Spiel von einem Benutzer über die Plattform erworben wird bekommt VAULT einen gewissen Prozentsatz des Umsatzes. Für die **Bezahlung** sollen verschiedene **Methoden** angeboten werden. Zudem soll es möglich sein das ein Benutzer ein erworbenes Spiel zurückgeben kann, wenn er dieses weniger als 10h gespielt hat. Auch werden Aktualisierungen für die Spiele ("Updates") direkt von den Herstellern der Spiele über das System verteilt.

Erstellen eines Domänenmodells

Durchspielen eines Use-cases

■ Spiel herunterladen

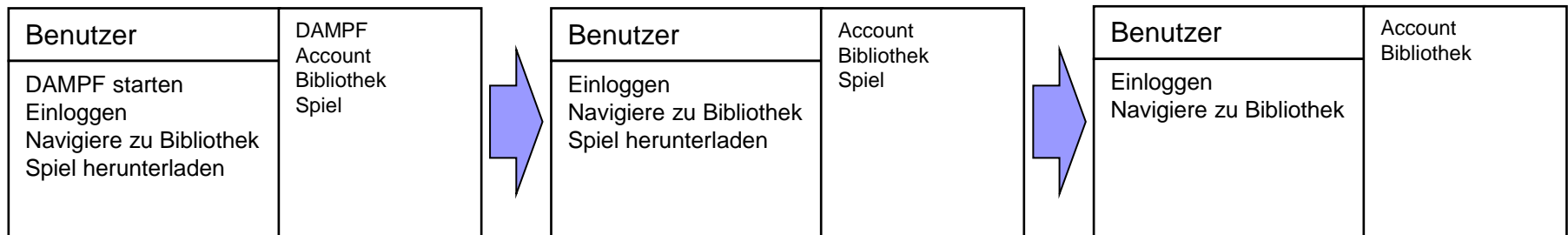
- ☐ Benutzer startet DAMPF
- ☐ Benutzer loggt sich ein
- ☐ Benutzer navigiert zur eigenen Bibliothek
- ☐ Benutzer wählt zu heruntergeladenes Spiel aus
- ☐ Benutzer gibt Download-Verzeichnis an
- ☐ Benutzer bestätigt Start des Downloads

Benutzer	DAMPF Account Bibliothek Spiel
DAMPF starten Einloggen Navigiere zu Bibliothek Spiel herunterladen	

Erstellen eines Domänenmodells

Probleme

- Es gibt nicht immer das eine richtige Modell
- Filtern von Information (relevant oder nicht) essentiell
- **Verfeinern** der Modelle ist oft schwierig



Erstellen eines Domänenmodells

Erster CRC-Karten Entwurf

Bezahloption	

Benutzer	Bibliothek Bezahloption
Lädt Spiele in Bibliothek Bezahlt Spiele mit Bezahloption	

Bibliothek	Spiel
Verwaltet Spiele	

DAMPF	Account
Verwaltet Accounts	

Account	Benutzer Spiele- hersteller
Verwaltet Benutzerdaten	

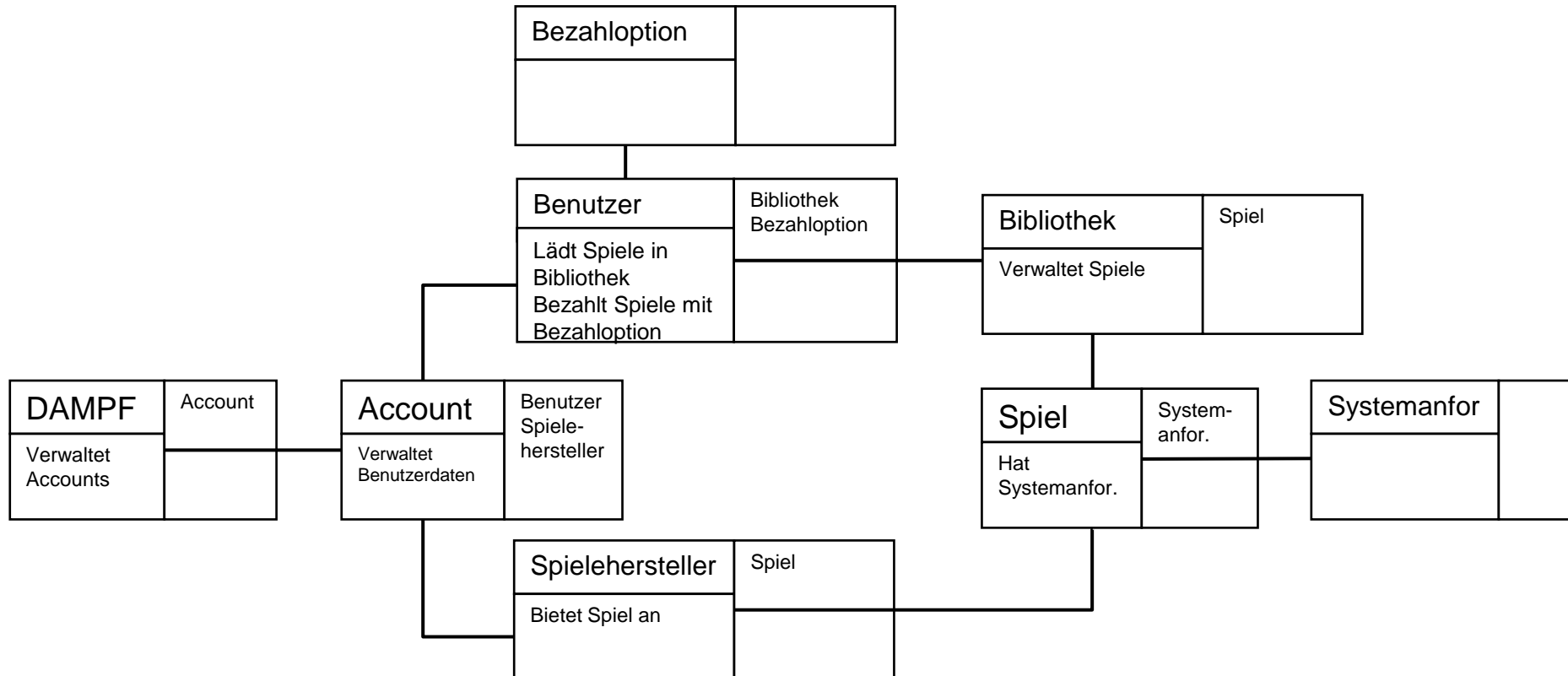
Spiel	System- anfor.
Hat Systemanfor.	

Systemanfor	

Spielehersteller	Spiel
Bietet Spiel an	

Erstellen eines Domänenmodells

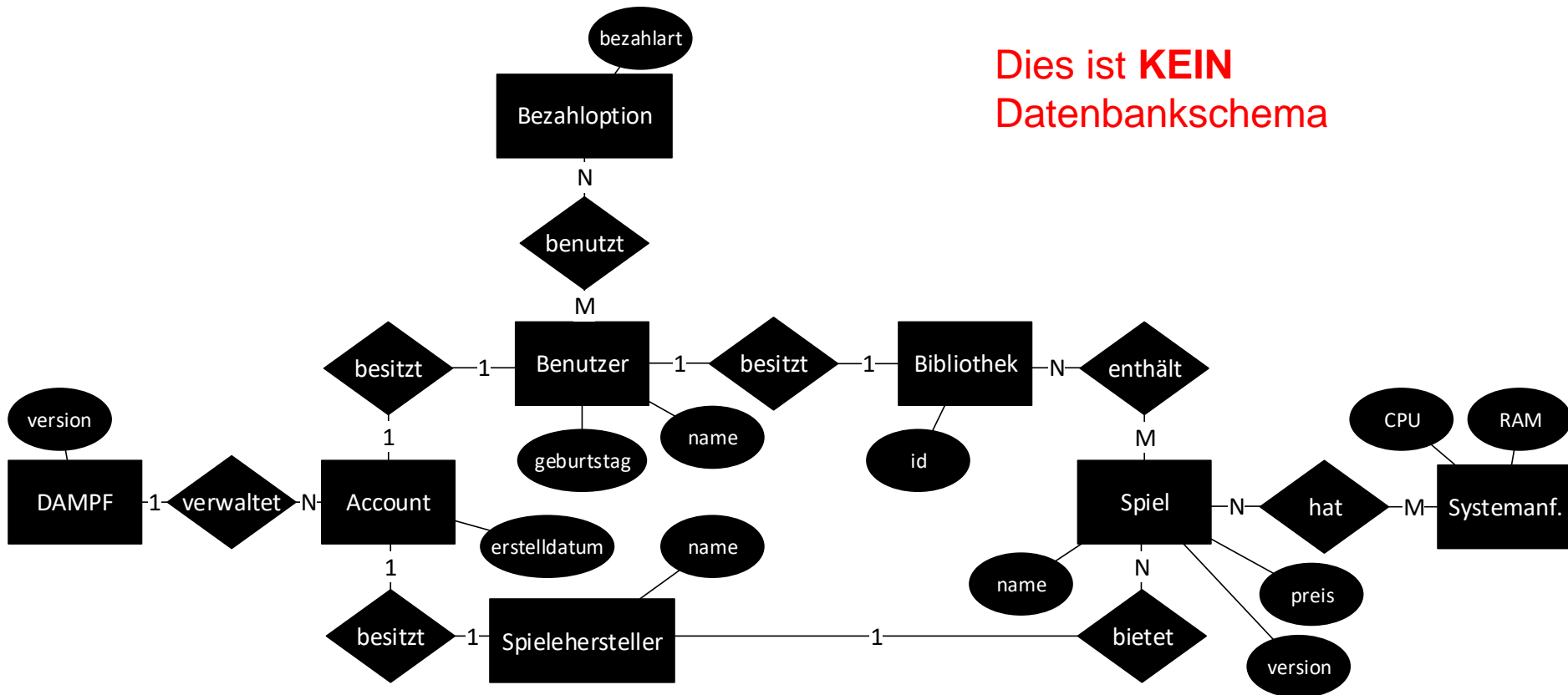
Erster CRC-Karten Entwurf



Erstellen eines Analysemodells

Erster ERM-Diagramm-Entwurf

Dies ist **KEIN**
Datenbankschema





Fragen?



ERM Diagramme

Entity Relationship Model (ERM)

- Konzeptuelles Model

- Abstrakte Beschreibung einer Domäne in einem grafischen Framework, welches später in ein logisches Modell transferiert werden kann

- Wir benutzen die originale “Chen Notation”, welche von Peter Pin-Shan Chen in 1976 veröffentlicht wurde

ERM: Konzepte

- Die Hauptkonzepte der Chen-Notation sind:
 - Entitätstypen
 - Beziehungen

ERM: Entitätstypen

■ Entitätstypen

- ☐ Repräsentieren ein Konzept der realen Welt
- ☐ Besitzen einen Namen und Attribute
- ☐ Beispiel: Spiele

Spiel

■ Attribute

- ☐ Stellen relevante Eigenschaften eines Entitätstypen dar
- ☐ Jedes Attribute kann bestimmte Werte einer Domäne halten
- ☐ Beispiel: Name oder Mindestalter des Spiels

Name

ERM: Entitäten

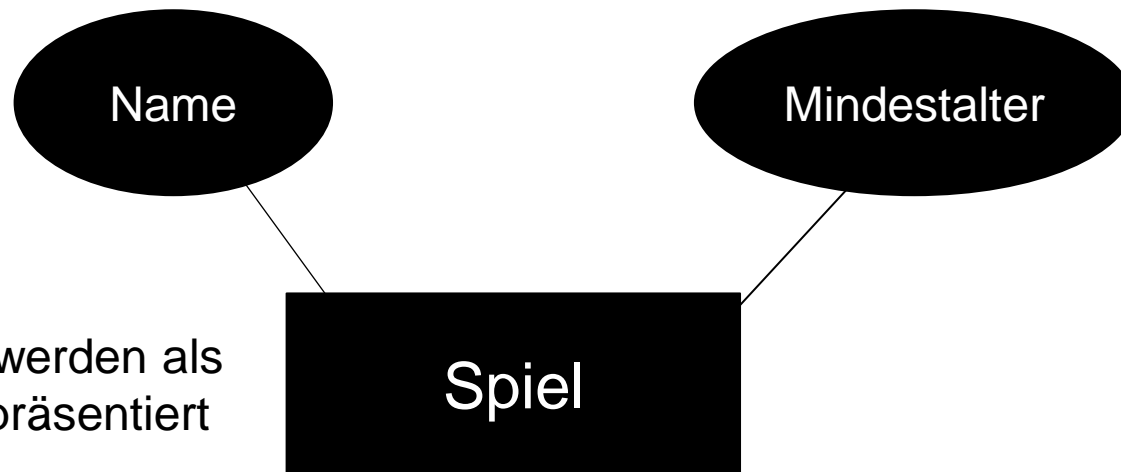
■ Entitäten

- ☐ Repräsentieren reale Objekte
- ☐ Sind Instanziierungen eines Entitätstypen
- ☐ Beispiel:
 - Spiel, Name: Super Mario Bros., Mindestalter: 0

ERM: Grafische Darstellung

Attribute werden als
Ellipse repräsentiert

Entitätstypen werden als
Rechtecke repräsentiert

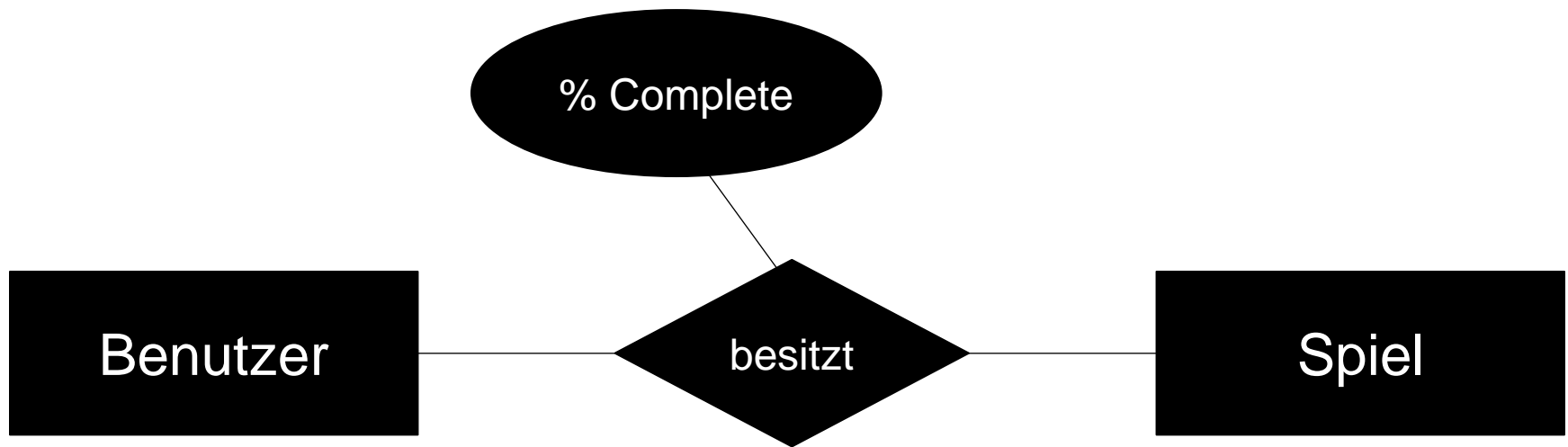


ERM: Beziehungstypen

■ Beziehungstypen

- Repräsentieren Beziehungen zwischen Entitäten
- Beziehungen zwischen zwei Entitäten ist eine binäre Beziehung
- Allerdings können auch Beziehungen zwischen **mehr als zwei** Entitäten bestehen
- Beziehungen können auch Attribute besitzen, welche relevante Eigenschaften beschreiben
- Beispiel:
 - Spiele befinden sich in einer Spielebibliothek

ERM: Grafische Darstellung



ERM: Kardinalitäten

■ Kardinalitäten

- ☐ Jedem Beziehungstyp werden Kardinalitäten zugewiesen
- ☐ Diese beschreiben die Anzahl der Beziehungen welche eine Entität eines bestimmten Typs mit einer anderen Entität haben kann

ERM: Kardinalitäten

- Es gibt drei verschiedene Arten von Beziehungskardinalitäten
 - **1:1**: Jede Entität des ersten Entitätstypen kann mit **höchstens einer** Entität des zweiten Entitätstypen in Beziehung stehen (und umgekehrt)
 - **1:N**: Jede Entität des ersten Entitätstypen kann mit **beliebig vielen** Entitäten des zweiten Entitätstypen in Beziehung stehen. Aber: Jede Entität des zweiten Entitätstypen kann nur mit **höchstens einer** Entität des ersten Entitätstypen in Beziehung stehen
 - **M:N**: Jede Entität des ersten Entitätstypen kann mit **beliebig vielen** Entitäten des zweiten Entitätstypen in Beziehung stehen (und umgekehrt)

ERM: Grafische Darstellung

