



Softwarearchitektur im Softwarepraktikum



Inhalte

- Organisatorisches
- Softwarearchitektur
- Pause?
- Softwarearchitektur für Computerspiele



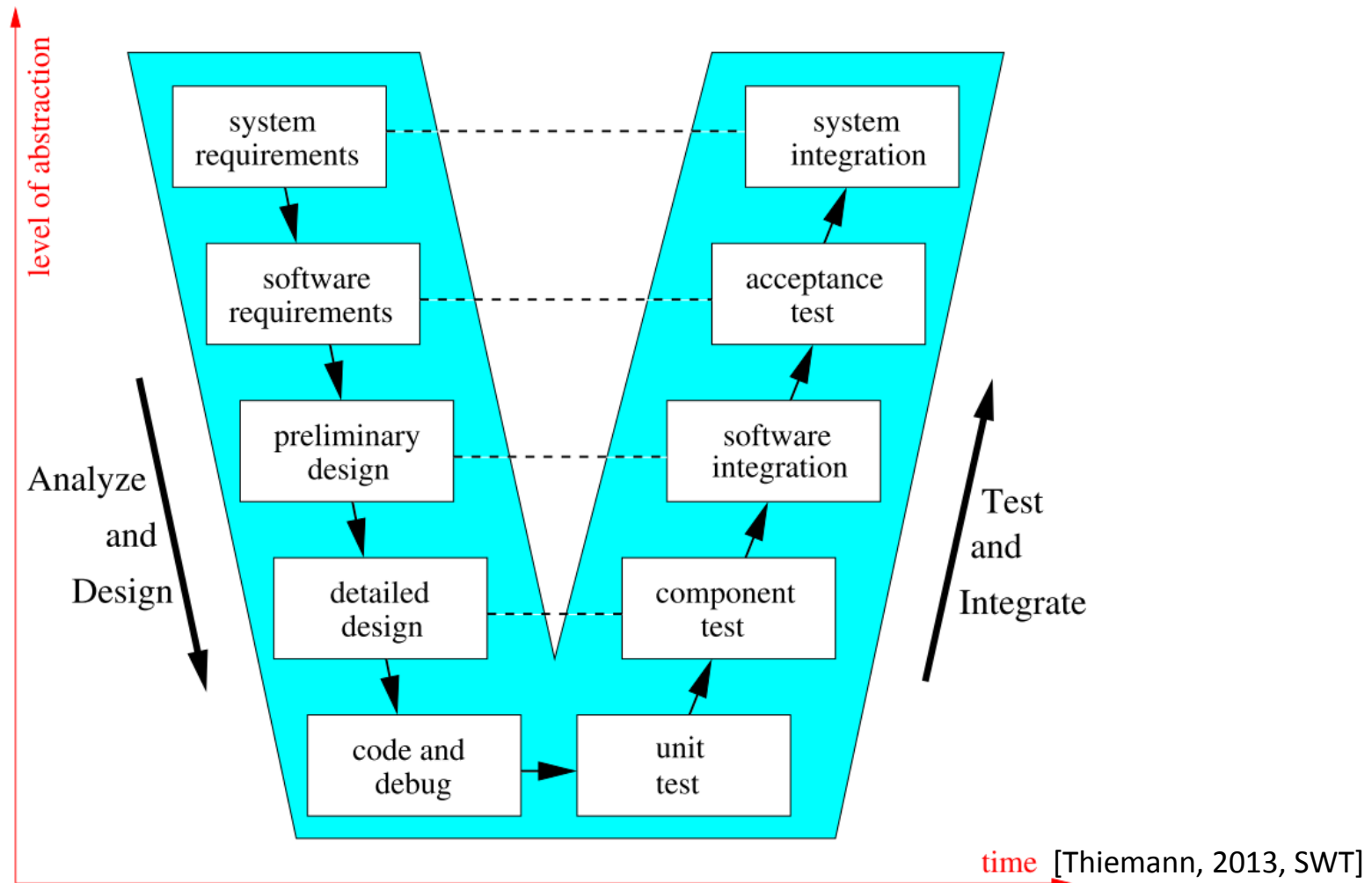
SOFTWAREARCHITEKTUR

Was ist Softwarearchitektur?

Definition. Eine **Softwarearchitektur** beschreibt die **Strukturen** eines Softwaresystems durch **Architekturbausteine** und ihre **Beziehungen** und **Interaktionen** untereinander sowie ihre physikalische Verteilung. Die **extern sichtbaren** Eigenschaften eines Architekturbausteins werden durch **Schnittstellen** spezifiziert.

- Verschiedene **Sichten**:
Statisch, **Dynamisch**, Verteilung, Kontext
- Verschiedene **Bausteinararten**:
Subsysteme, **Komponenten**, Frameworks, **Pakete**, **Klassen**
- Verschiedene **Abstraktionsebenen**

Abstraktionsebenen im V-Modell



Was ist ein Modell?

Definition. [Glinz, 2008, 425]

A **model** is a concrete or mental **image** (Abbild) of something or a concrete or mental **archetype** (Vorbild) for something.

Three properties are constituent:

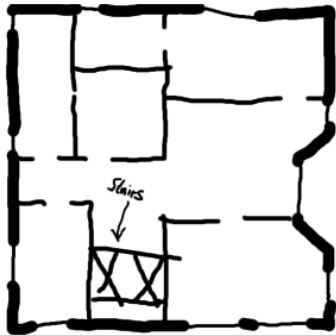
- (i) the **image attribute** (Abbildungsmerkmal), i.e. there is an entity (called original) whose image or archetype the model is,
- (ii) the **reduction attribute** (Verkürzungsmerkmal), i.e. only those attributes of the original that are relevant in the modelling context are represented,
- (iii) the **pragmatic attribute**, i.e. the model is built in a specific context for a specific purpose.

Was ist UML?

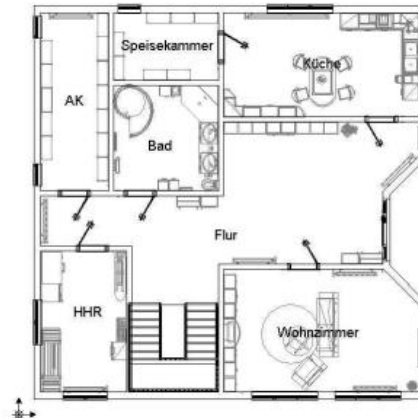
- UML ist eine **Modellierungssprache** mit graphischen Notationen.
- UML := **Unified Modeling Language**.
- UML ist standardisiert, der Standard wird von der **Object Management Group (OMG)** verwaltet.

UML-Modi

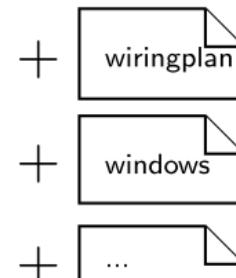
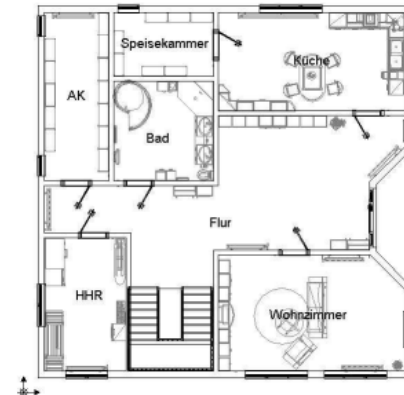
UML as sketch



UML as Blueprint



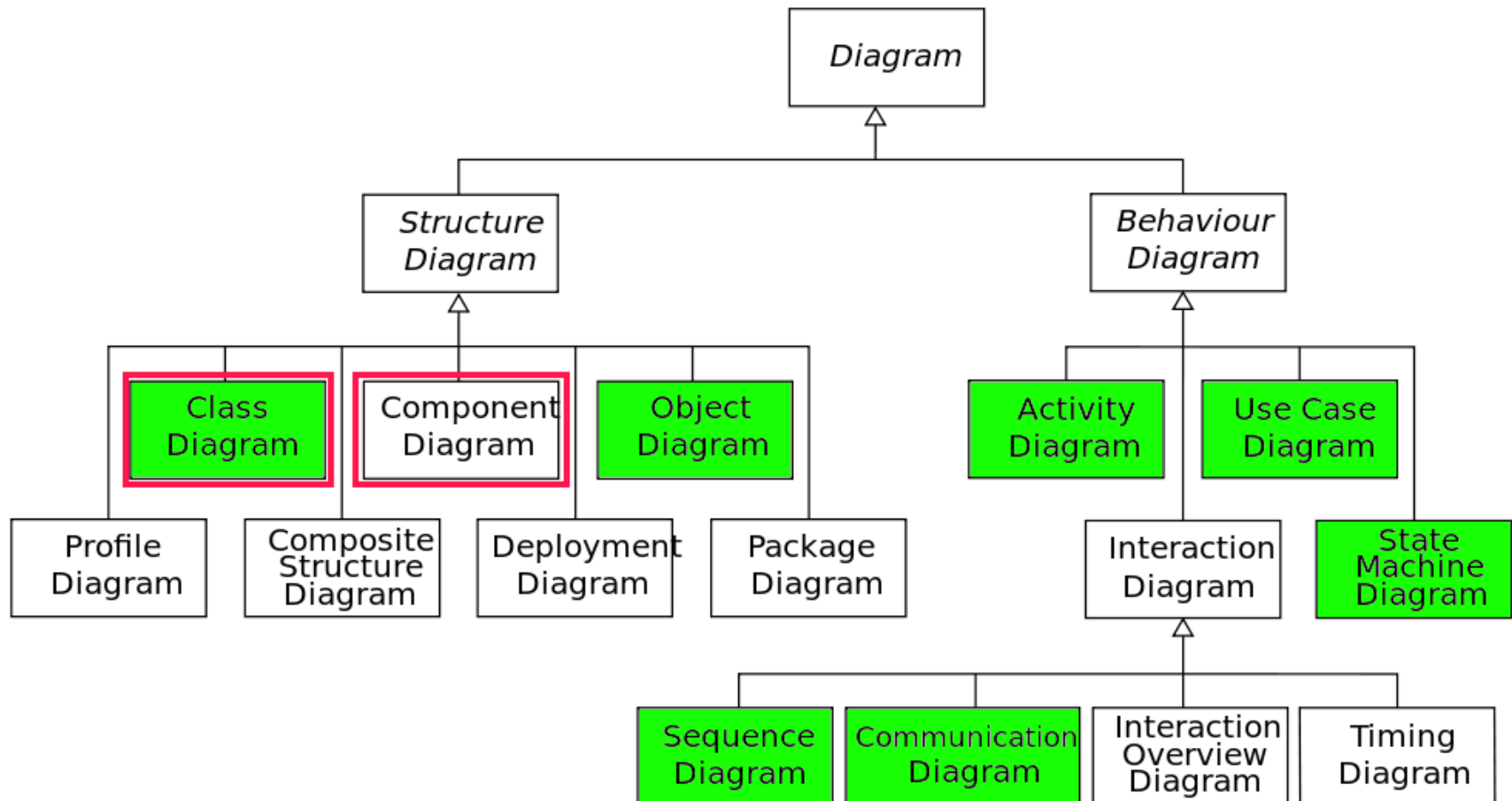
UML as programming language



[Westphal, 2012/13, UML]

[<http://martinfowler.com/bliki>]

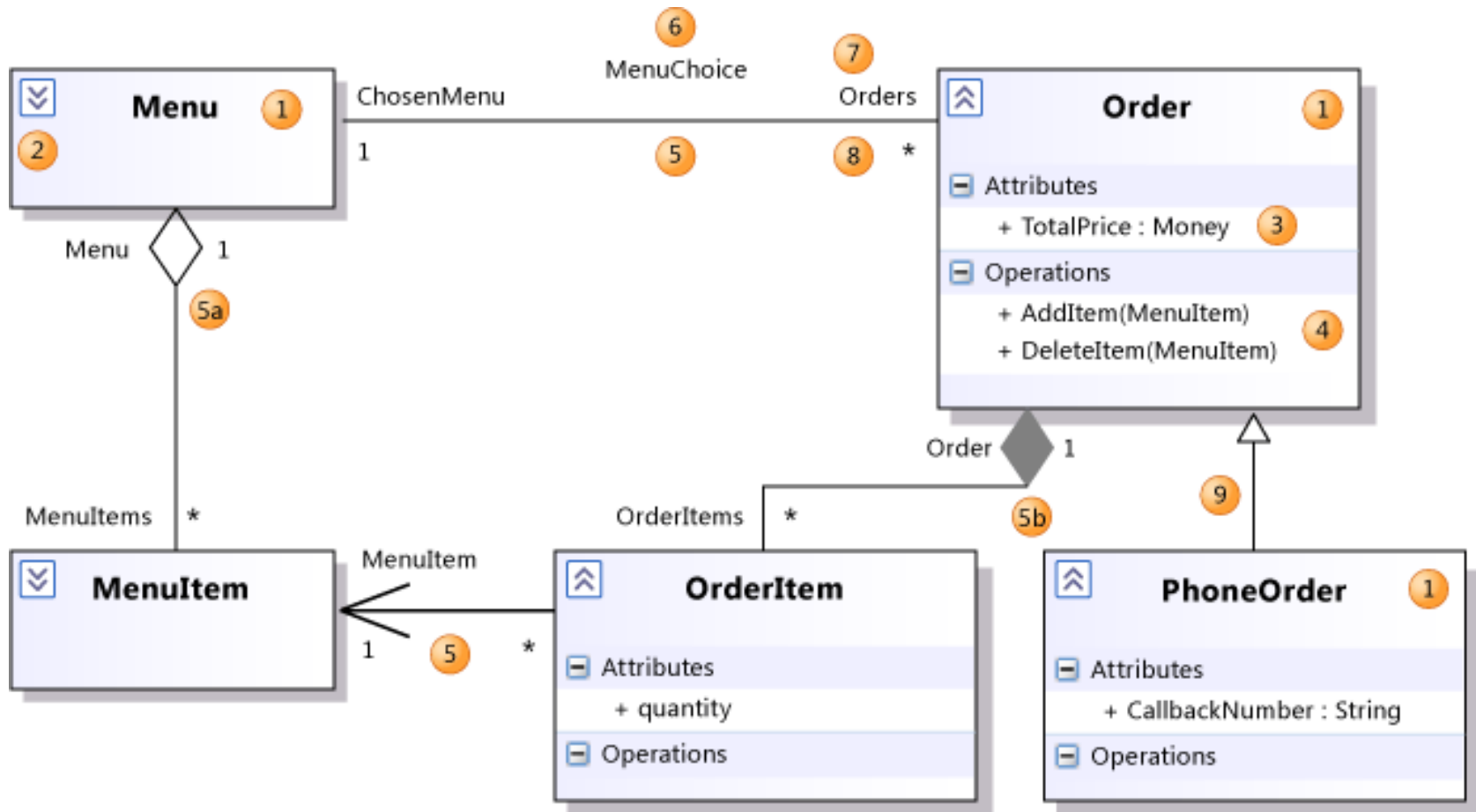
Diagrammarten in UML



Klassendiagramm

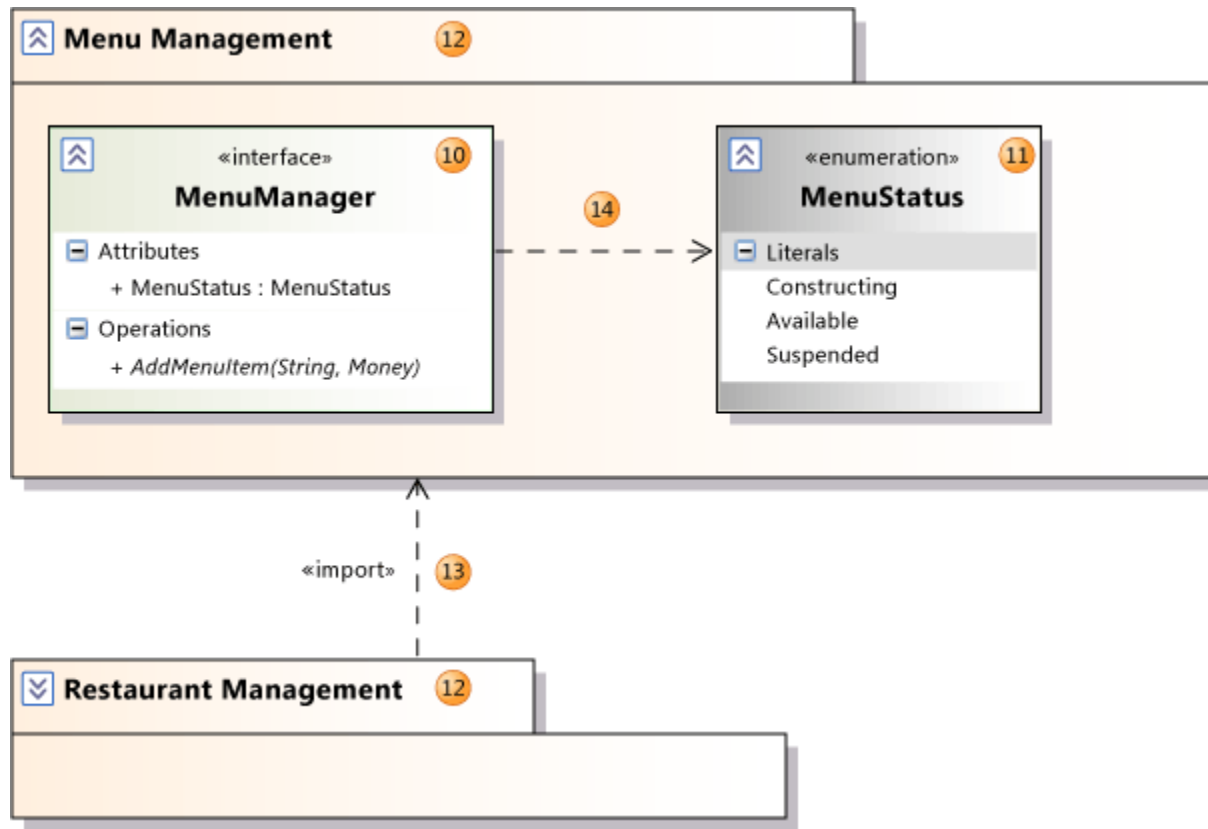
- Statische Sicht.
- Klassen, Interfaces, Pakete und deren statische Beziehungen als Bausteine.
- Sehr nah an der Implementierung.
- Hier: Klassendiagramm à la Microsoft Visual Studio.

Klassendiagramm



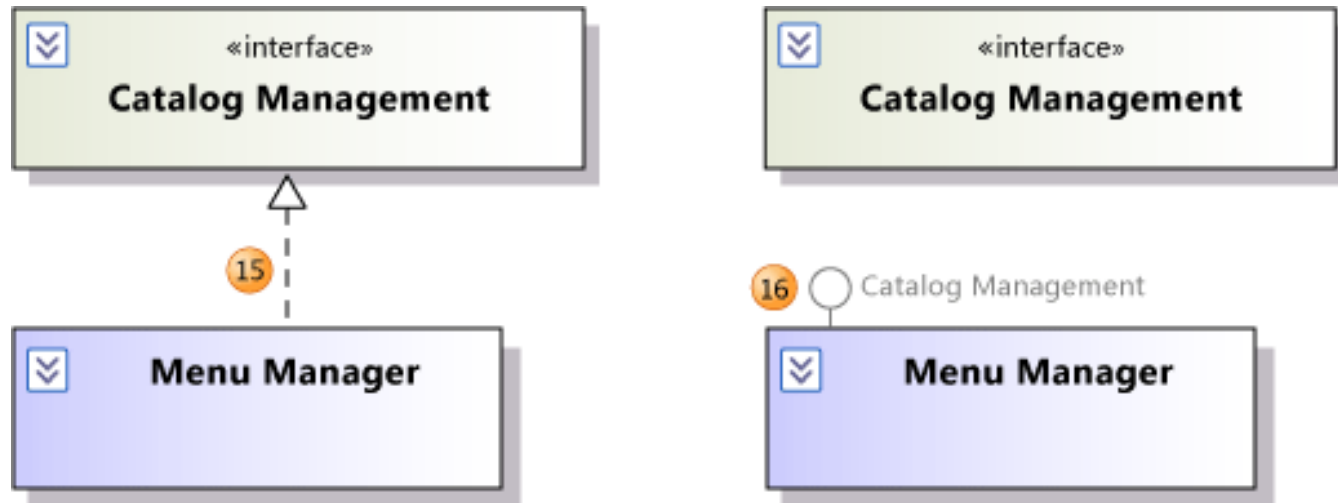
[[http://msdn.microsoft.com/en-us/library/vstudio/dd409437\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/vstudio/dd409437(v=vs.100).aspx)]

Klassendiagramm

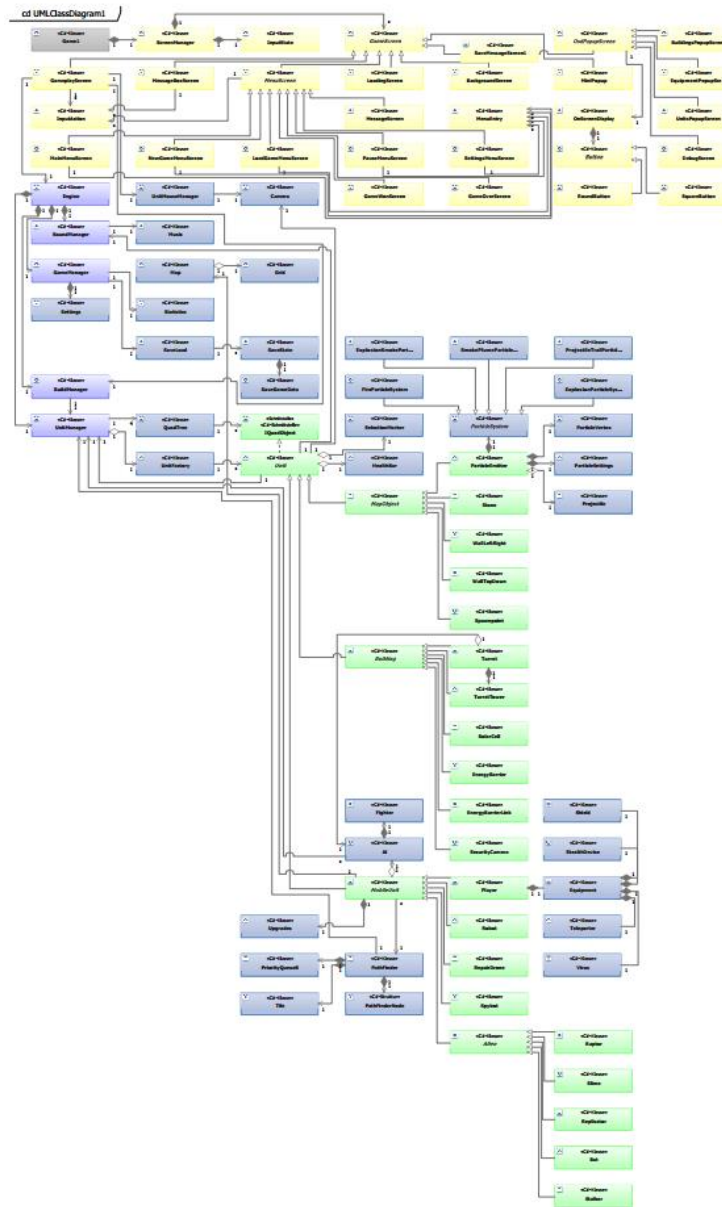


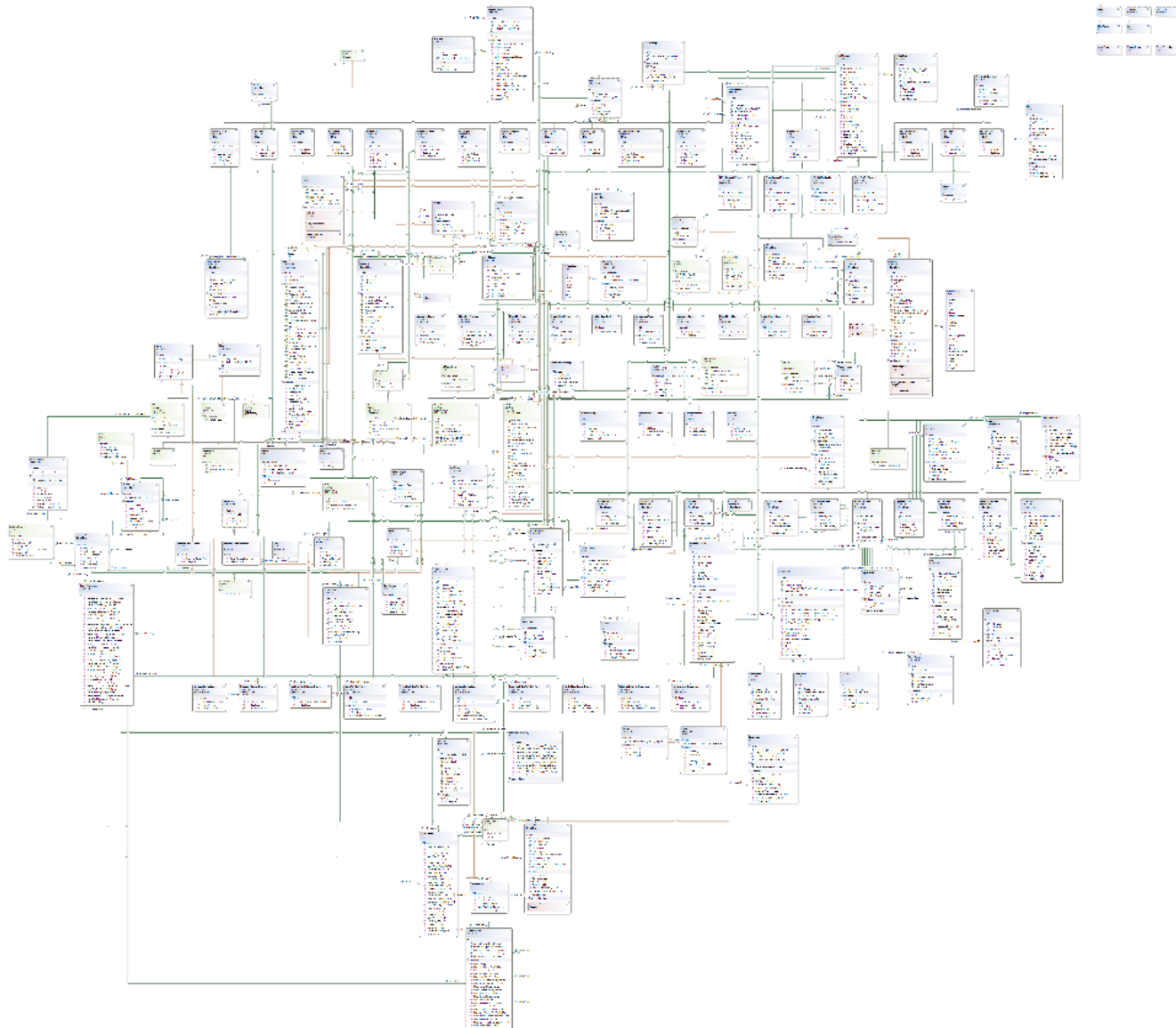
[[http://msdn.microsoft.com/en-us/library/vstudio/dd409437\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/vstudio/dd409437(v=vs.100).aspx)]

Klassendiagramm



[[http://msdn.microsoft.com/en-us/library/vstudio/dd409437\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/vstudio/dd409437(v=vs.100).aspx)]





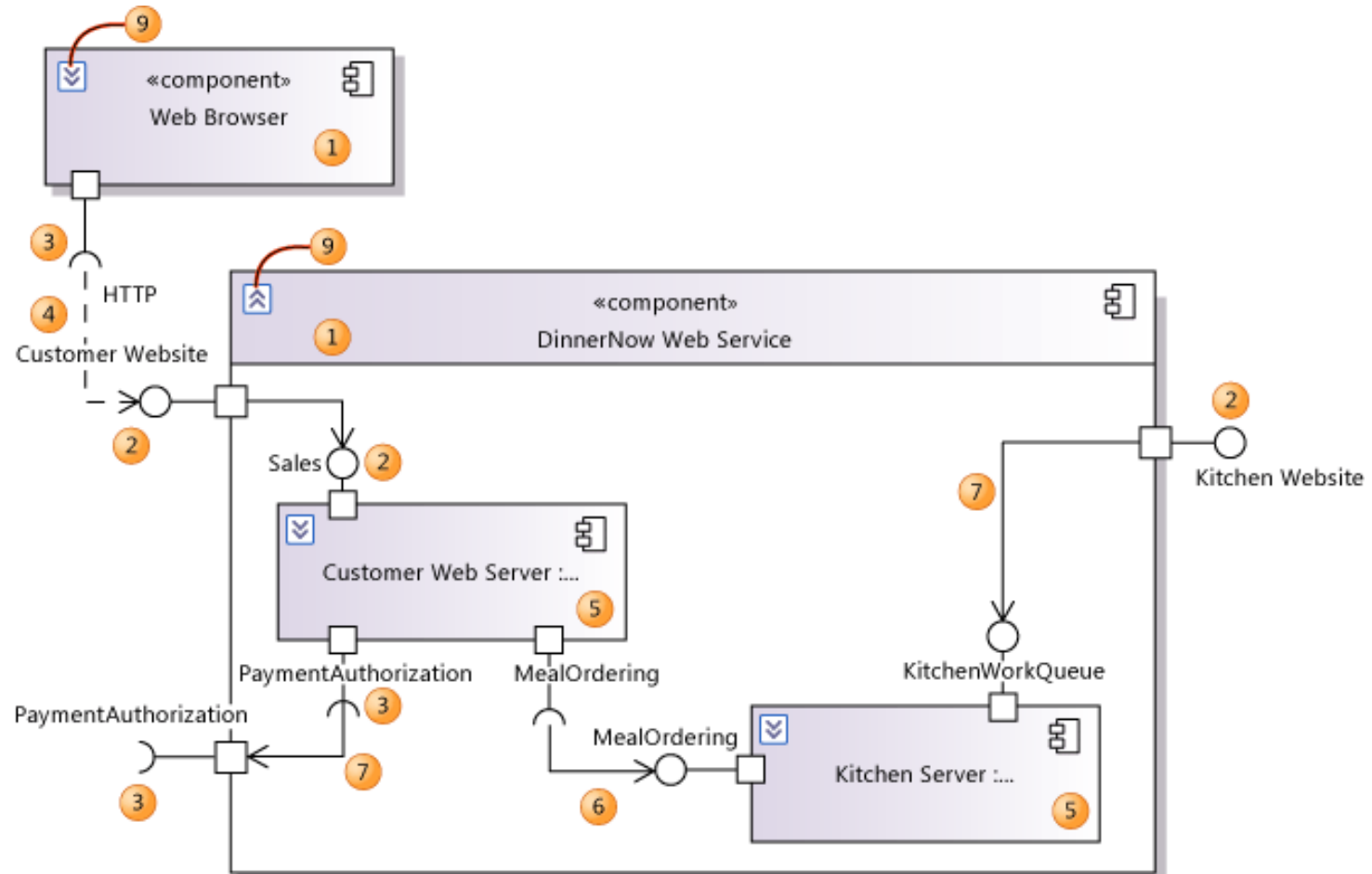
Komponenten

- Eine **Komponente** ist ein **Softwarebaustein**, der Dritten **Funktionalität** über **Schnittstellen** zur Verfügung stellt und **nur explizite Abhängigkeiten** nach außen besitzt.

Komponentendiagramm

- **Statische** Sicht.
- **Komponenten**, **Interfaces**, **Parts** und deren statische Beziehungen untereinander als Bausteine.
- **Abstrakter** als Klassendiagramm.
- Beschreibt „Verdrahtung“ von **Komponenten**.
- **Hier**: Komponentendiagramm à la Microsoft Visual Studio.

Komponentendiagramm



[[http://msdn.microsoft.com/en-us/library/vstudio/dd409390\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/vstudio/dd409390(v=vs.100).aspx)]

Werkzeugunterstützung

- Sie können **jede Art von Werkzeug** verwenden. Z.B.:
 - Visual Studio Ultimate
 - ArgoUML mit C# Importer
 - ModelMaker
 - Grafikprogramme
 - Papier, Stifte, Scanner
- **Visual Studio Ultimate**
 - Klassen- und Komponentendiagramme können mit Visual Studio **gezeichnet** werden.
 - Klassendiagramme können **generiert** werden (sehen dann aber anders aus).
 - Aus den Diagrammen kann auch **Code erzeugt** werden (ist aber nicht bzw. nur mit hohem Aufwand nützlich).



Wie finde ich eine Architektur?

- Siehe Softwaretechnik-Vorlesung „Object-Oriented Analysis / UML Diagrams“.

Wie bewertet man eine SW-Architektur?

- Das kommt darauf an!
- Will man...
 - Architekturen **vergleichen**?
 - auf bestimmte **Qualitätsmerkmale** prüfen?
 - auf **Risiken** prüfen?
 - nur „**klassifizieren**“?
- **Qualitätsmerkmale** orientieren sich an **Produktqualität**.
Erweiterbarkeit, Zuverlässigkeit, Wartbarkeit, Erreichbarkeit, Benutzbarkeit, Skalierbarkeit, Sicherheit, Fehlertoleranz, Abwärtskompatibilität, Performanz,

Qualitätsmerkmale im Softwarepraktikum

- Es gibt viel mehr Methoden!
- Erweiterbarkeit
 - Abschätzung mit Metriken, z.B. Efferent / Afferent **Coupling**, Lack of **cohesion** in methods (**LCOM**), Type **complexity**, ...
- Zuverlässigkeit
 - Testing
- Benutzbarkeit
 - Cognitive Walkthrough
 - Heuristic Evaluation
- Performanz
 - Profiling
 - FPS

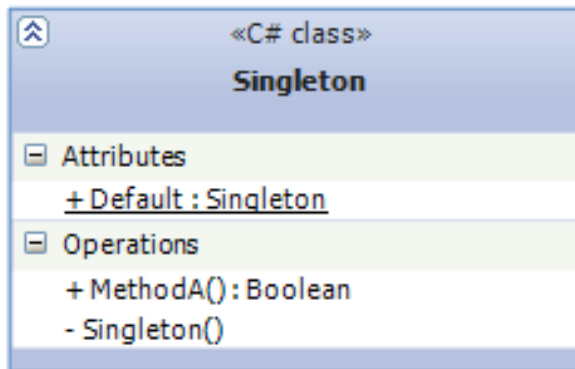
Design Patterns

- Siehe Softwaretechnik-Vorlesung „Design Patterns“.
 - Für Spiele nützlich:
(Abstract) Factory, Builder, Flyweight, Observer, Visitor, Iterator
- Viele weitere nützliche Patterns.
 - Z.B. Object Pool, Proxy, Prototype, Composite, Decorator, Command, Strategy, ...

Design Patterns

- **Vorsicht!**
 - Design Patterns können eine Architektur auch **unnötig kompliziert** machen.
 - Erst verstehen, wozu ein Pattern gut ist, dann das Pattern verwenden.
 - **Nicht:** Wie kann ich bloß dieses Pattern verwenden?
- Siehe auch Anti-Pattern **Cargo Cult Programming**.
- Beliebtes Beispiel: **Singleton**.

Singleton



```

public class Singleton {
    private static Singleton sInstance;

    public static Singleton Default {
        get {
            if (sInstance == null) {
                sInstance = new Singleton();
            }
            return sInstance;
        }
    }

    private Singleton() { /* ... */ }

    public bool MethodA() { /* ... */ }
}
  
```

Nachteile des Singleton-Patterns

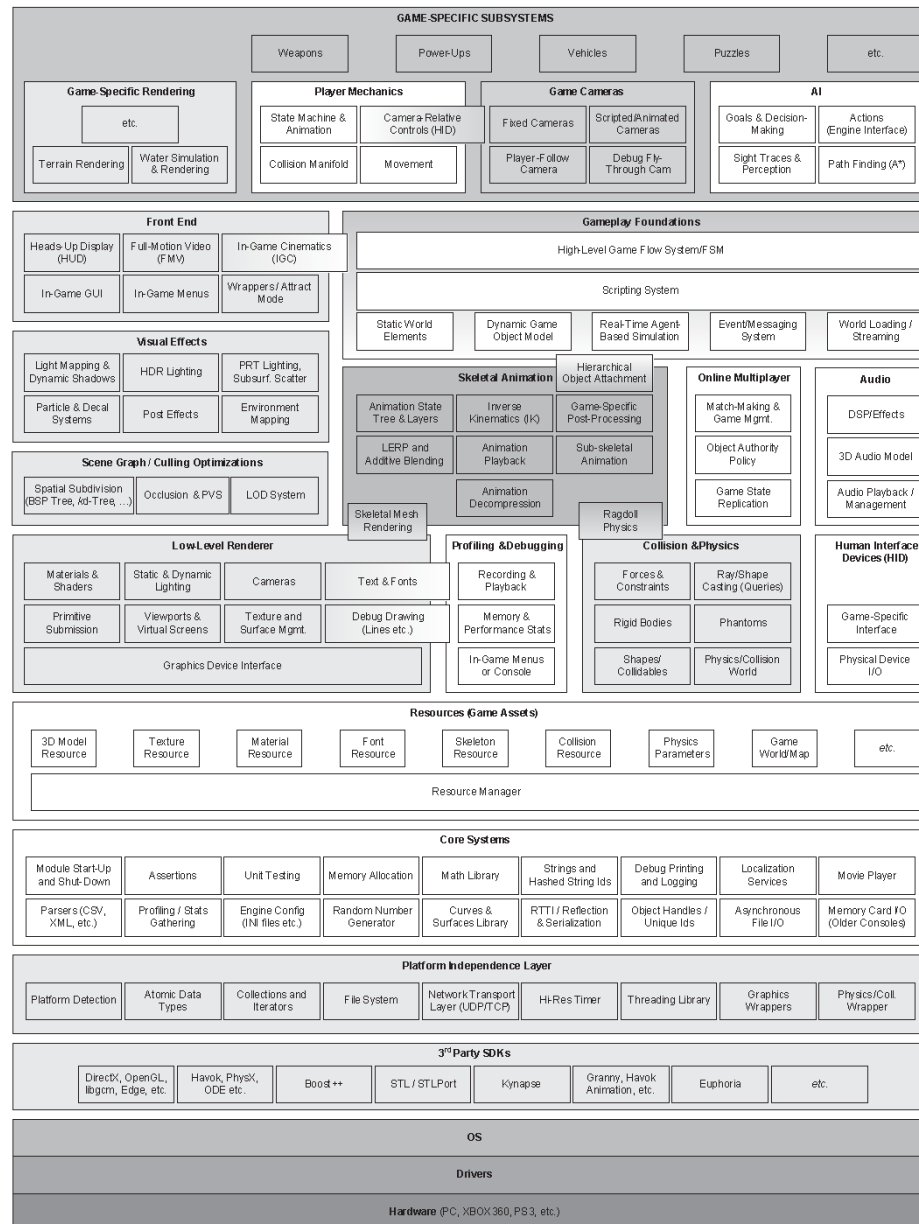
- **Versteckt Abhängigkeiten.**
 - Calls zu Singletons sind Calls zu einem statischen Objekt.
 - Man kann sie überall verwenden.
- **Schwierig zu testen.**
 - Es ist sehr schwierig, alle Calls zu einem Singleton zu Testzwecken zu verändern.
- **Schwierig abzuleiten.**
 - Da die Initialisierung statisch geschieht, kann man sie in abgeleiteten Klassen nicht einfach überschreiben.
- **Sprachabhängig.**
 - Z.B. in Java gibt es nicht einen statischen Kontext pro VM, sondern pro Classloader.
- **Schwierig zu verändern.**
 - Was ist, wenn man doch plötzlich zwei Objekte braucht?



SOFTWAREARCHITEKTUR FÜR COMPUTERSPIELE

Viele Fragen

- Um was muss ich mich alles kümmern?
 - Während das Spiel läuft (**Runtime**):
Eingabe/Ausgabe, Menüs, Einstellungen, Speichern/Laden, Rendern, Kamera, Eigenschaften von Spielobjekten, Interaktion zwischen Spielobjekten, Sound, Musik, KI, Netzwerk, Pathfinding, Kollisionserkennung, Debug-Helfer, Effekte,...
 - Während der Entwicklung (**Tools**):
 - Wie bekomme ich Assets (Modelle, Texturen, Sound, etc.) ins Spiel?
 - Wie erstelle ich Assets (z.B. Level)?
 - Debug-Helfer.
- Was wird mir bereits zur Verfügung gestellt?
- Wie verbinde ich alles?

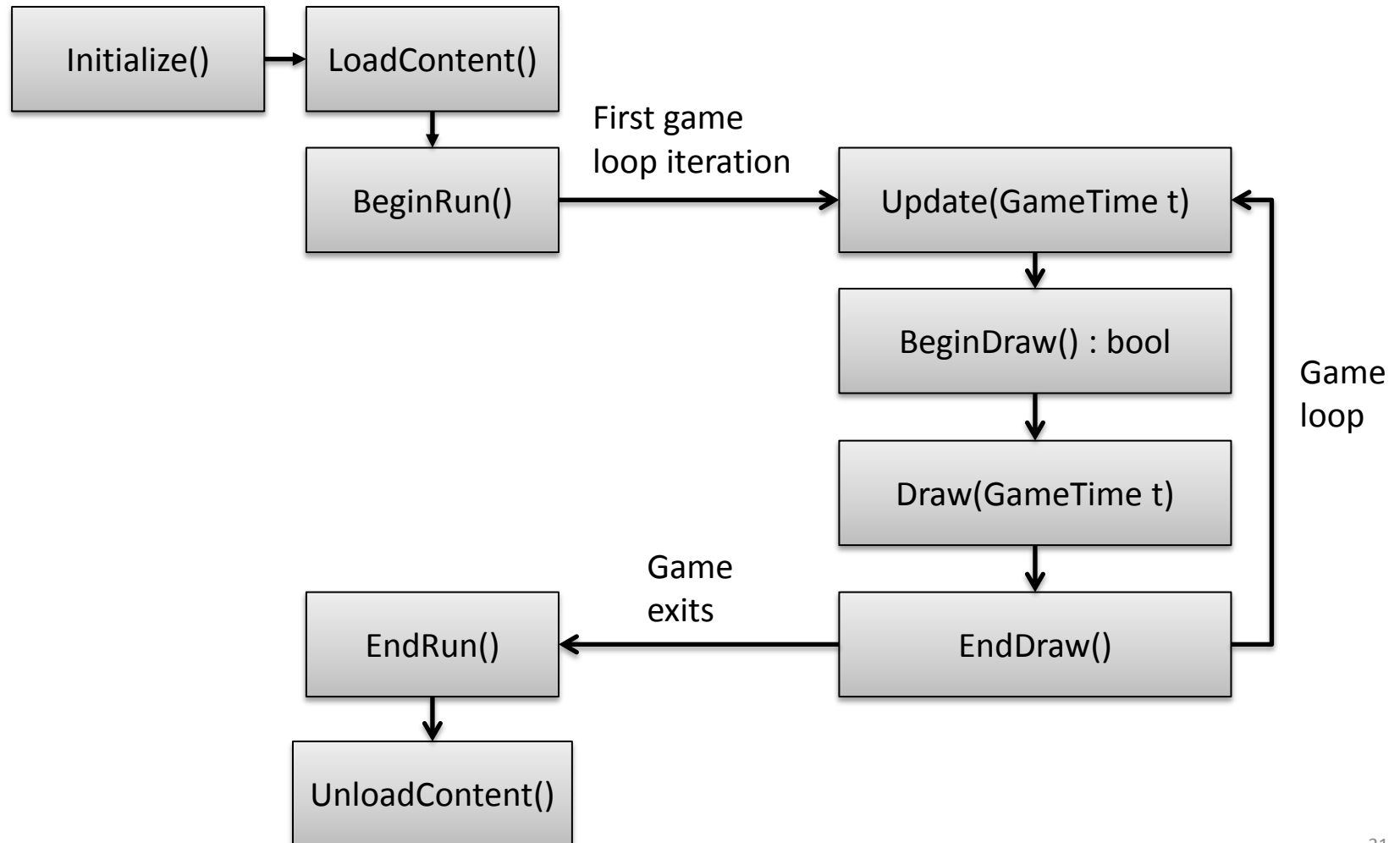


[Gregory, 2009, 29]

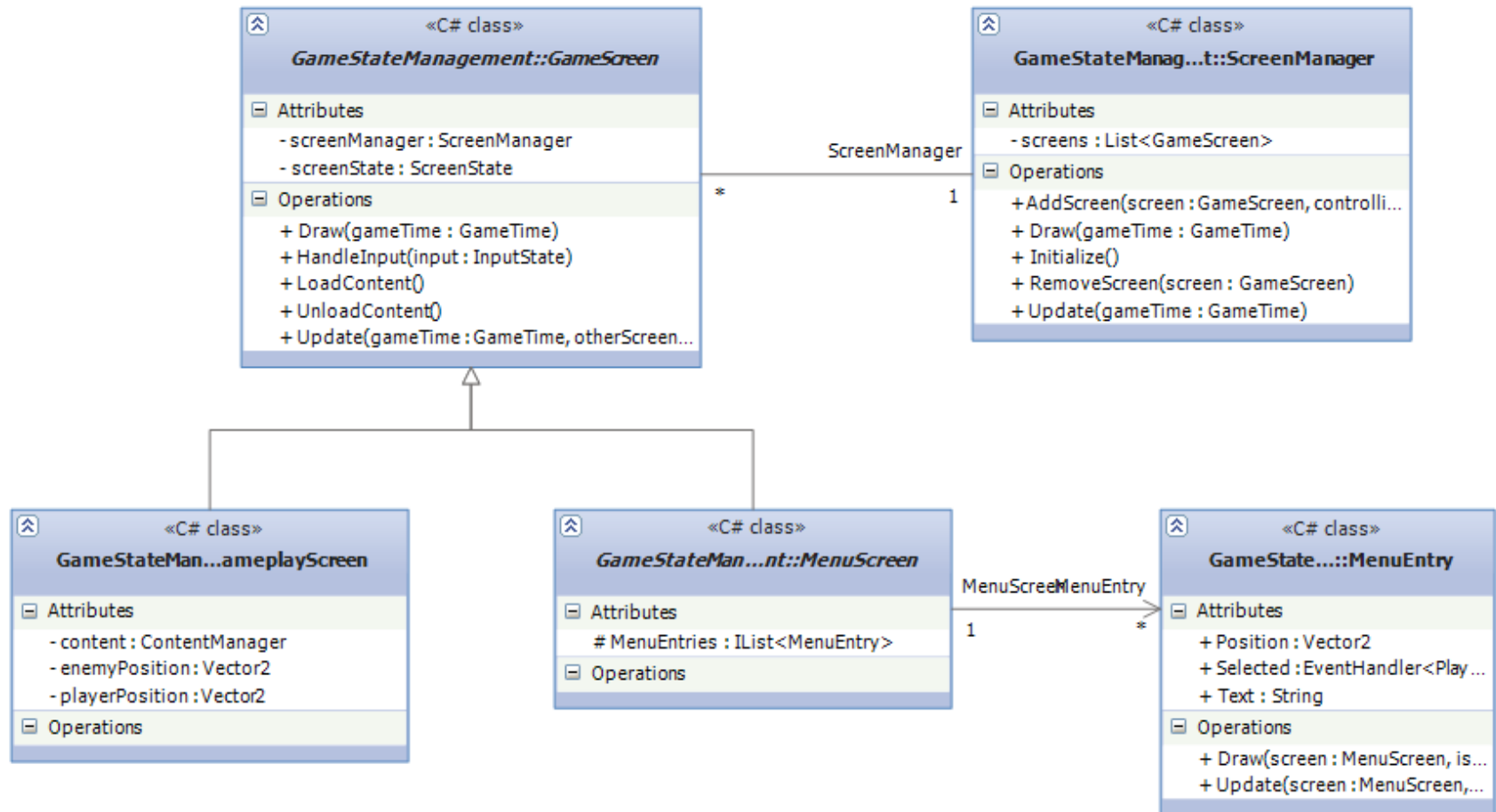
XNA und .NET

- **XNA** übernimmt bereits viele Aufgaben:
 - Abstraktion von Grafik, Sound und Input.
 - Aufbereiten von Assets (Content Pipeline).
 - Einfache Anzeigemethoden (BasicEffect, SpriteBatch).
 - Typen (Vector2D, 3D, Matrix, Rectangle, ...).
- **.NET** hilft ebenfalls:
 - Mathematik und Zufallszahlen.
 - Serialisierung / Deserialisierung von XML und binären Daten.
 - Diverse Datenstrukturen.
 - Debugging.

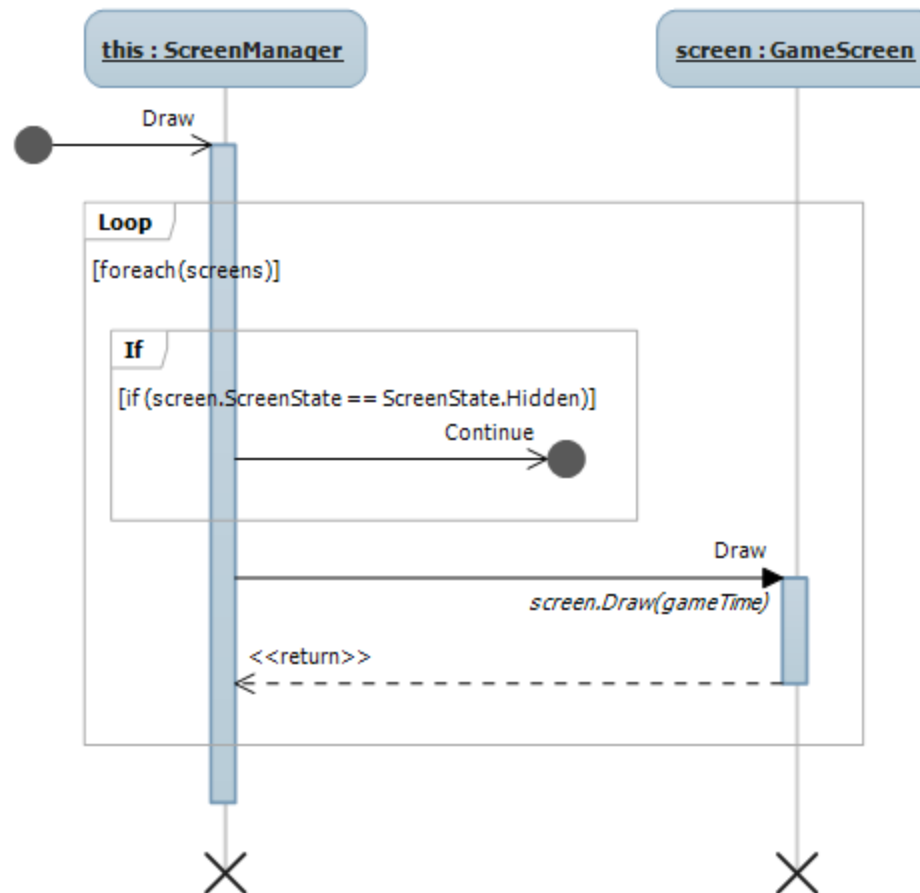
XNA Game Lifecycle

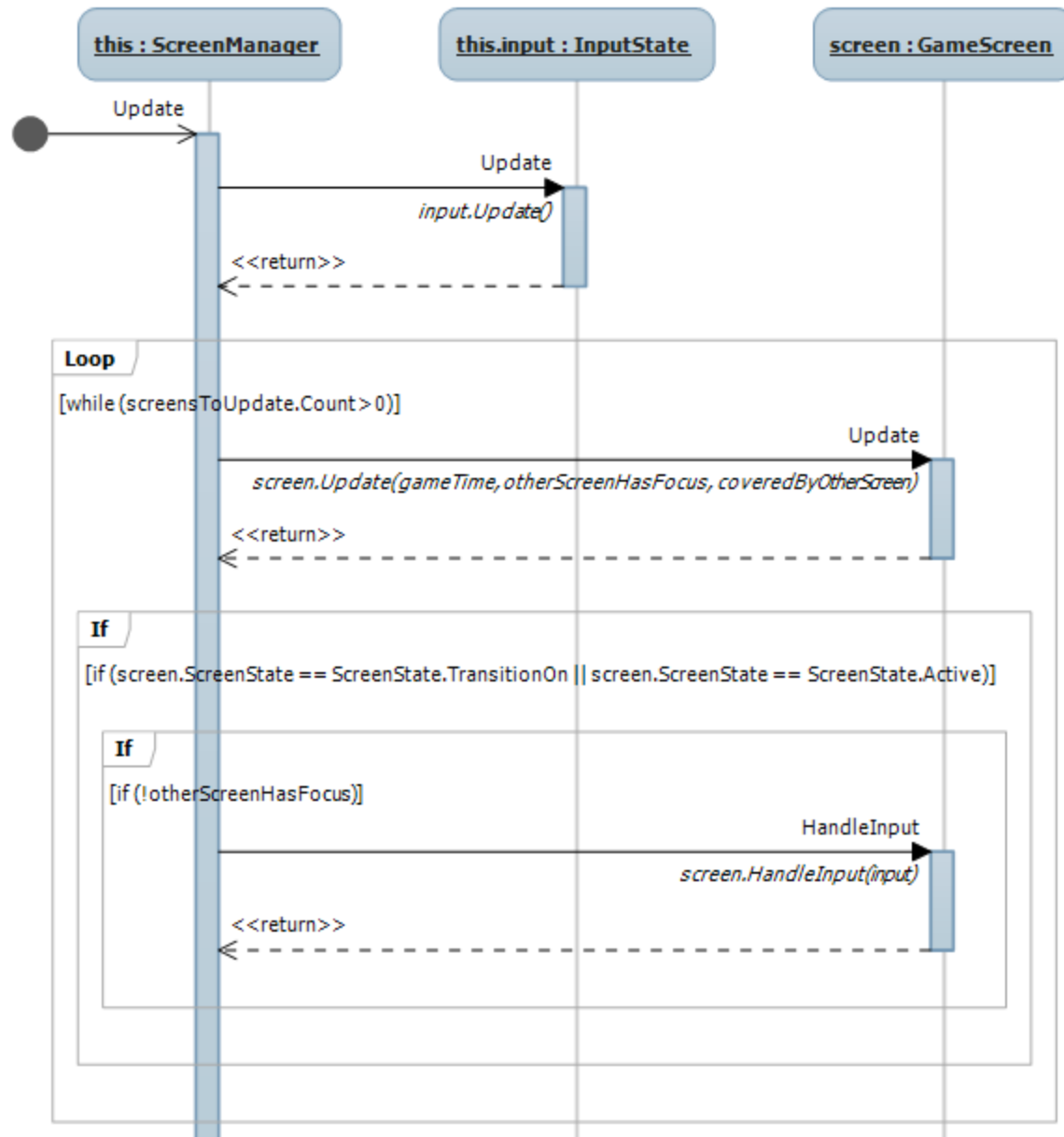


Einstieg: GameStateManagement Sample



GameStateManagement



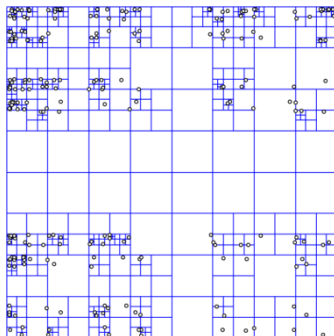


Spielobjekte

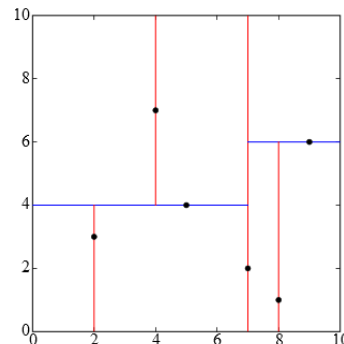
- **Spielobjekte** sind all die Dinge, die eine **Repräsentation** in der **Spielwelt** haben.
 - Charaktere, Fahrzeuge, Bäume, Raketen, Gras, Steine, Trigger, Lichtquellen, ...
- Spielobjekte müssen manchmal...
 - **gezeichnet** werden.
 - sich **bewegen**.
 - **zerstörbar** sein.
 - miteinander **kollidieren**.
 - etc.
- Wie kann ich diese vielen Objektarten und ihre Operationen verwalten?

Szenengraph

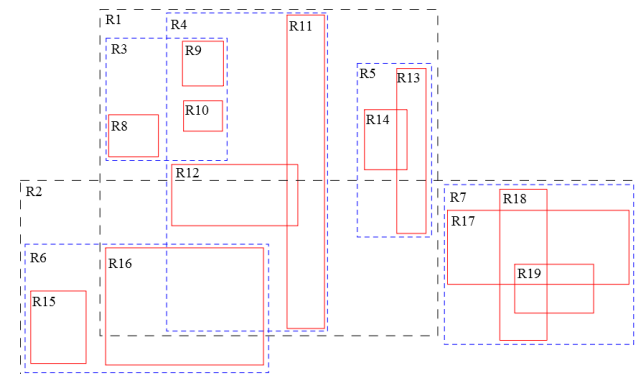
- Ein **Szenengraph** ist eine zentrale Datenstruktur, die logische und/oder räumliche Repräsentation einer Szene verwaltet.
- Wird z.B. verwendet um
 - Antworten auf **räumliche** Fragen zu beschleunigen.
 - Update- und Draw-Aufrufe an alle Spielobjekte weiterzureichen.
- **Beispiele**
Liste, Heap, Quad- / Octree, KD-Tree, R-Tree, ...



Quadtree



KD-Tree



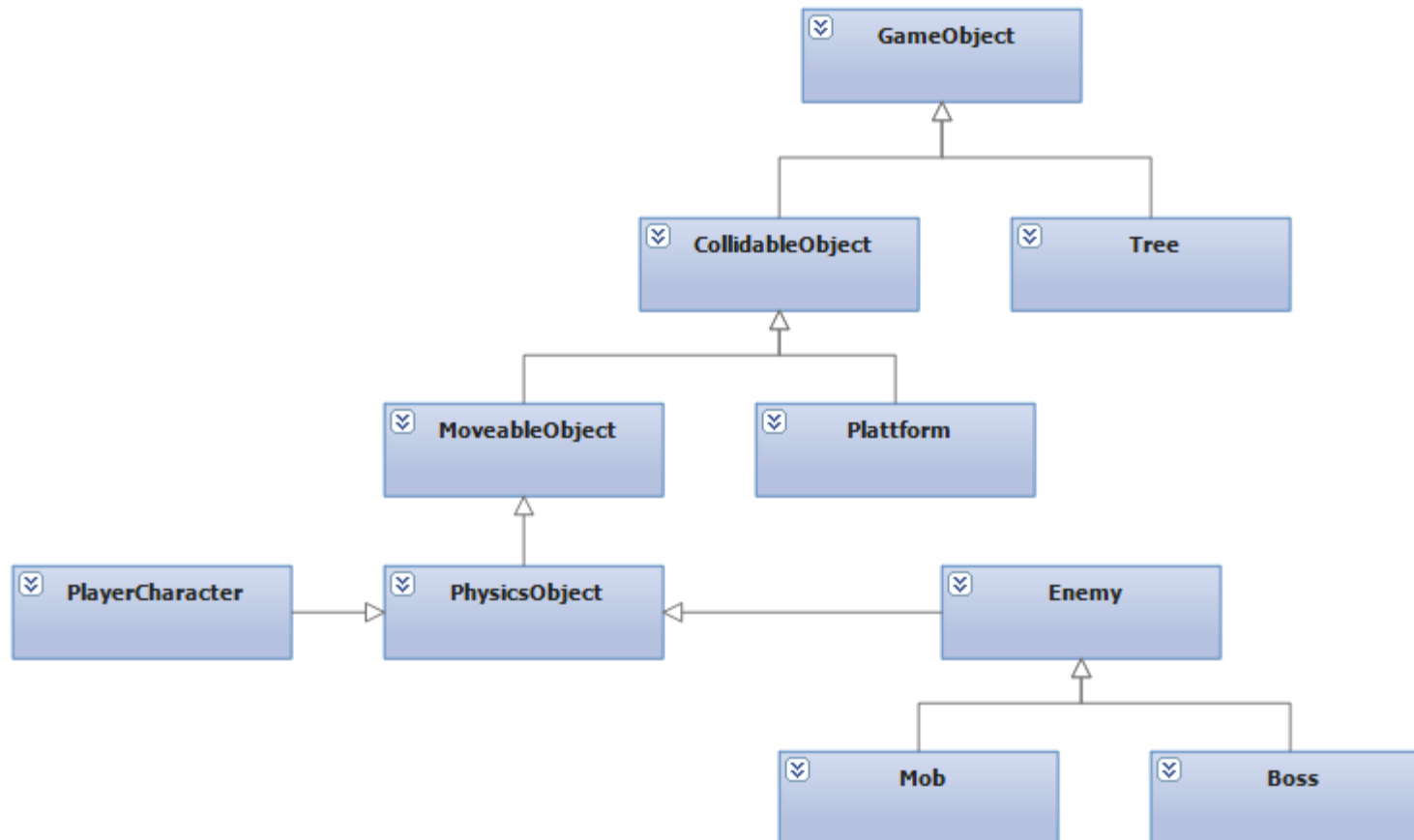
R-Tree

Spielobjekt-Architekturen

- **Objekt-zentriert**
 - Spielobjekte werden als Klassen implementiert und als Instanzen repräsentiert.
 - Eigenschaften und Verhalten wird durch die Klasse(n) gekapselt.
 - Spielwelt ist eine Ansammlung von Spielobjekt-Instanzen.
- **Eigenschaften-zentriert**
 - Eigenschaften der Spielobjekte werden als Tabellen gespeichert, eine pro Eigenschaft.
 - Spielobjekte sind nur eine ID.
 - Ähnlichkeit zu relationalen Datenbanken.
- ...

Deep Hierarchy

- Eine große Vererbungsstruktur für Spielobjekte.





Wunsch

	Tree	Plattform	PlayerCharacter	Mob	Boss	Camera
GameObject	X	X	X	X	X	X
CollidableObject		X	X	X	X	
MoveableObject			X	X	X	X
PhysicsObject			X	X	X	
Enemy				X	X	



Composition

- Spielobjekte werden aus **universellen** Komponenten zusammengesetzt.

Ein weites Feld...

- Sehr viele **Fragen bleiben offen**.
 - Netzwerk-Multiplayer?
 - Sound?
 - Grafikeffekte?
 - Pathfinding?
 - KI?
 - ...
- Kommen Sie in die **Poolsprechstunde**, um sich speziell für Ihr Spiel beraten zu lassen.



FRAGEN?

Quellen

- [Thiemann, 2013, SWT] Thiemann, P. (18.4.2013). Softwaretechnik. Vorlesung. Prozessmodelle. Universität Freiburg.
- [Westphal, 2012/13, UML] Westphal, B. (23.10.2012). Software Design, Modelling, and Analysis in UML. Vorlesung. Introduction. Universität Freiburg.
- [Balzert, 2011] Balzert, H. (2011). Lehrbuch Der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb. Springer. ISBN 3827422469, 9783827422460.
- [Glinz, 2008] Glinz, M. (2008). Modellierung in der Lehre an Hochschulen: Thesen und Erfahrungen. Informatik Spektrum, 31(5):425–434.
- [Gregory, 2009] Gregory, J. (2009). Game Engine Architecture. A K Peters Limited. ISBN 1568814135, 9781568814131 .