# Grundlagen der Softwarearchitektur (im Softwarepraktikum)

Sommersemester 2018

# Organisatorisches

- Dienste
  - Geht alles?
  - Pool-Accounts?
- Fragen Sie! (Mail, Pool-Betreuung, …)

# DISCLAIMER

# Inhalt

- Was ist Softwarearchitektur?
- Dokumentieren mit UML
- Wie bewerte ich eine Softwarearchitektur?
- Wie plane ich eine Softwarearchitektur?
- Metriken

# WAS IST 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 Bausteinarten:
  Subsysteme, Komponenten, Frameworks, Pakete, Klassen

- Softwarearchitektur ist ein Modell eines Softwaresystems.

[Balzert, 2011, 23ff.]

# Was ist ein Modell?

**Definition**. [Glinz, 2008, 425]
Ein Modell ist ein konkretes oder mentales Abbild von etwas oder ein konkretes oder mentales Vorbild für etwas.

Drei Eigenschaften kennzeichnen ein Modell:

(i) Das Abbildungsmerkmal, d.h. es gibt eine Entität (ein Original) dessen Abbild oder Vorbild das Modell ist,

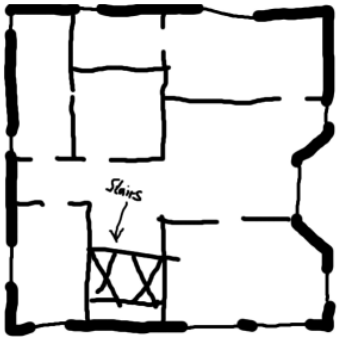
(ii) das Verkürzungsmerkmal, d.h. nur die Eigenschaften des Originals die für den Modellierungskontext relevant sind werden repräsentiert, und

(iii) die Pragmatik, d.h. das Modell wurde in einem spezifischen Kontext für einen spezifischen Zweck erstellt.

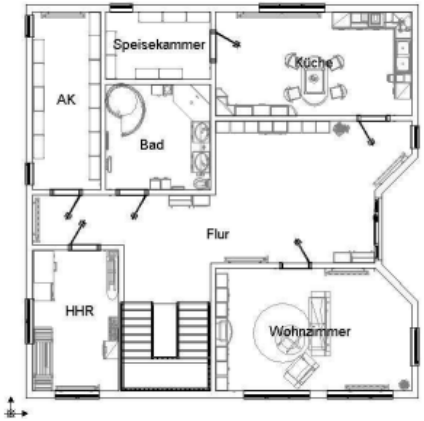
[Westphal, 2012/13, UML]

# Modell-Modi

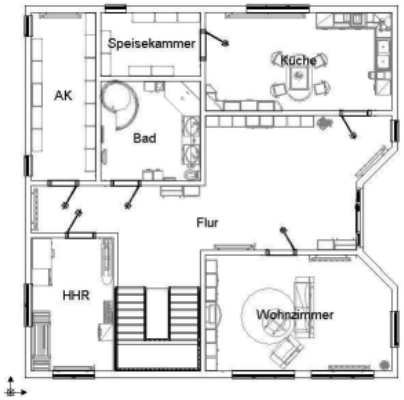
… als Skizze

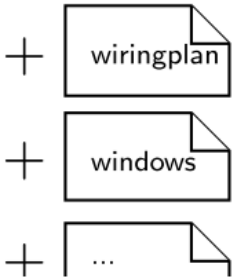



… als Blaupause




… als Programmiersprache




[Westphal, 2012/13, UML]

[http://martinfowler.com/bliki]

# Wieso Softwarearchitektur modellieren?

- Kommunikation und Dokumentation
  - …der Realisierung.
  - …der Entwurfsentscheidungen.
- Qualität planen
  - Konzeptionelle Integrität.
  - Grundlage für Bewertung anhand von Szenarien.
  - Vereinfachung der Wiederverwendung von Systembestandteilen.
- Arbeitsteilung durch Dekomposition
  - Schnittstellen identifizieren.
  - Aufgaben parallelisieren.

# WIE BEWERTE ICH EINE SOFTWAREARCHITEKTUR?

# Qualitätsmerkmale nach ISO 9126 (bzw. 25000)

- Funktionalität
  Angemessenheit, Richtigkeit, Interoperabilität, Ordnungsmäßigkeit, Sicherheit
- Zuverlässigkeit
  Reife, Fehlertoleranz, Wiederherstellbarkeit
- Benutzbarkeit
  Verständlichkeit, Erlernbarkeit, Bedienbarkeit
- Effizienz
  Zeitverhalten, Verbrauchsverhalten
- Änderbarkeit
  Analysierbarkeit, Modifizierbarkeit, Stabilität, Prüfbarkeit
- Übertragbarkeit
  Anpassbarkeit, Installierbarkeit, Konformität, Austauschbarkeit

# Qualitätsmerkmale im Softwarepraktikum

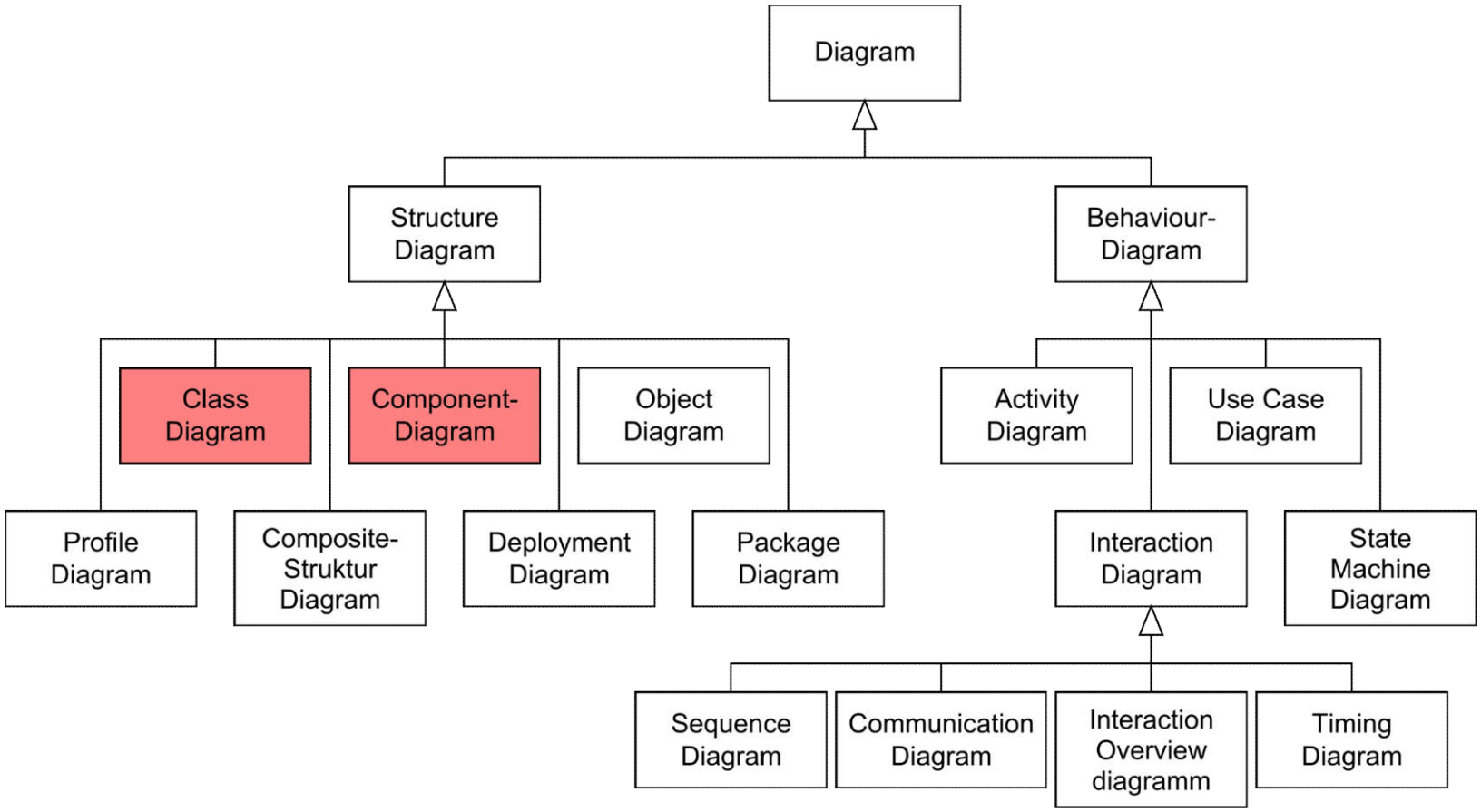
- Funktionalität
  - Testing
- Benutzbarkeit
  - Cognitive Walkthrough
  - Heuristic Evaluation
- Effizienz
  - Profiling
  - Testing (FPS mit fraps)
  - Szenarien
- Änderbarkeit
  - Szenarien
  - Abschätzung mit Metriken, z.B. Efferent / Afferent Coupling, Lack of cohesion of methods (LCOM), Type complexity

# UML

# Was ist UML?

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

# Diagrammarten in UML

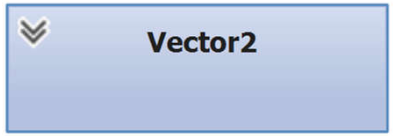


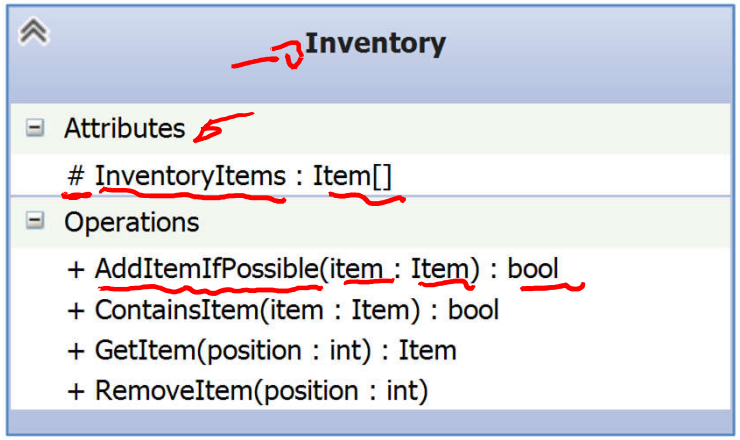
# Klassendiagramm

- Statische Sicht.
- Klassen, Interfaces, Pakete und deren statische Beziehungen als Bausteine.
- Sehr nah an der Implementierung.

- Hier: Klassendiagramm à la Microsoft Visual Studio.

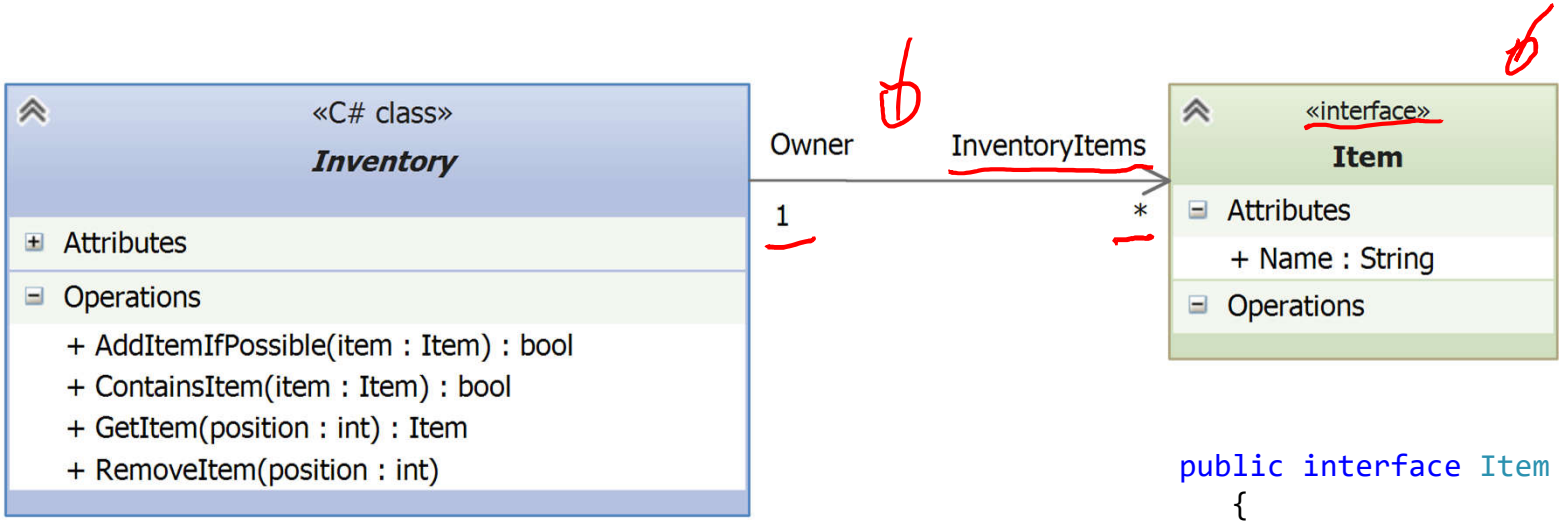
# Klassen



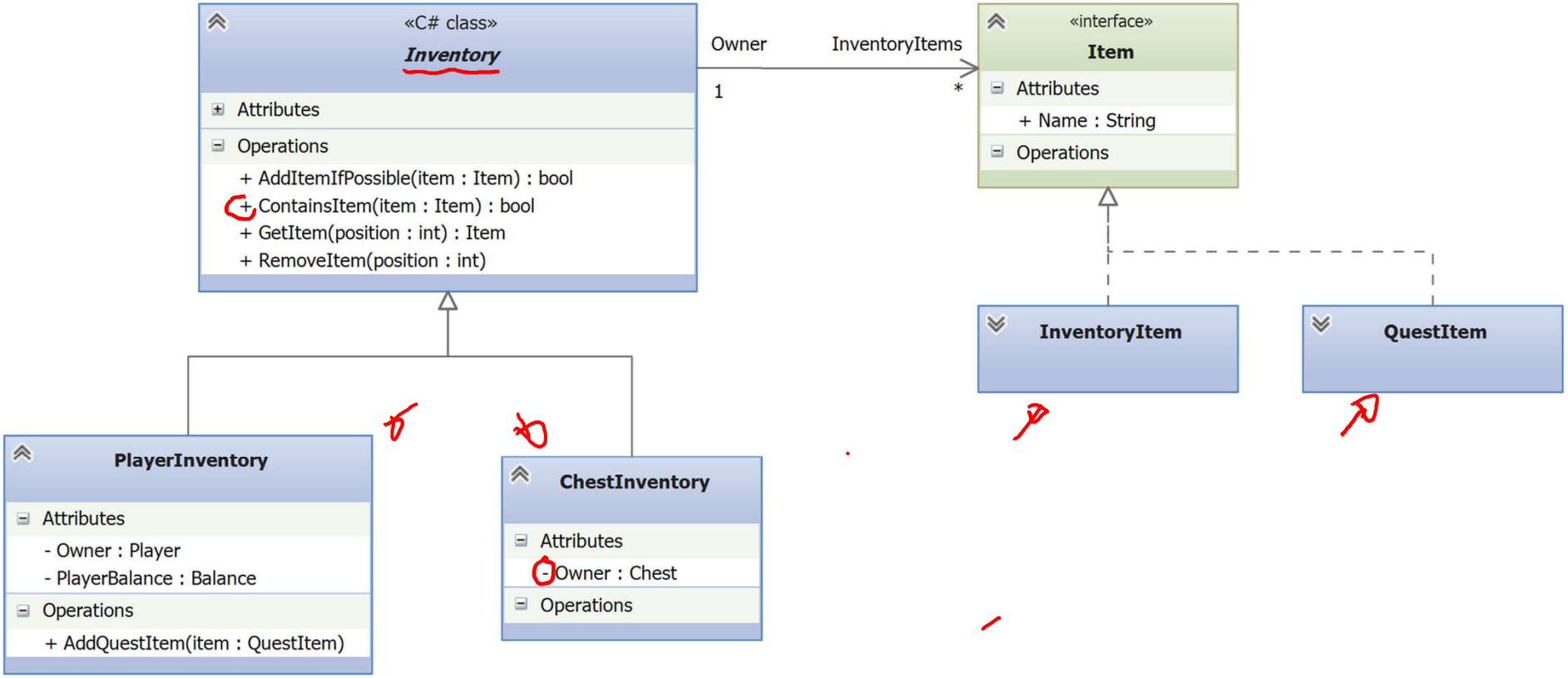

```
public class Inventory
{
    protected int[] Items;

    public Inventory(){ /*...*/ }
    public bool AddItemIfPossible(item Item) { /*...*/
    public bool ContainsItem(item Item) { /*...*/ }
    public Item GetItem(int number) { /*...*/ }
    public Item RemoveItem(int number) { /*...*/ }
    public void SortByColor () { /*...*/ }
}
```

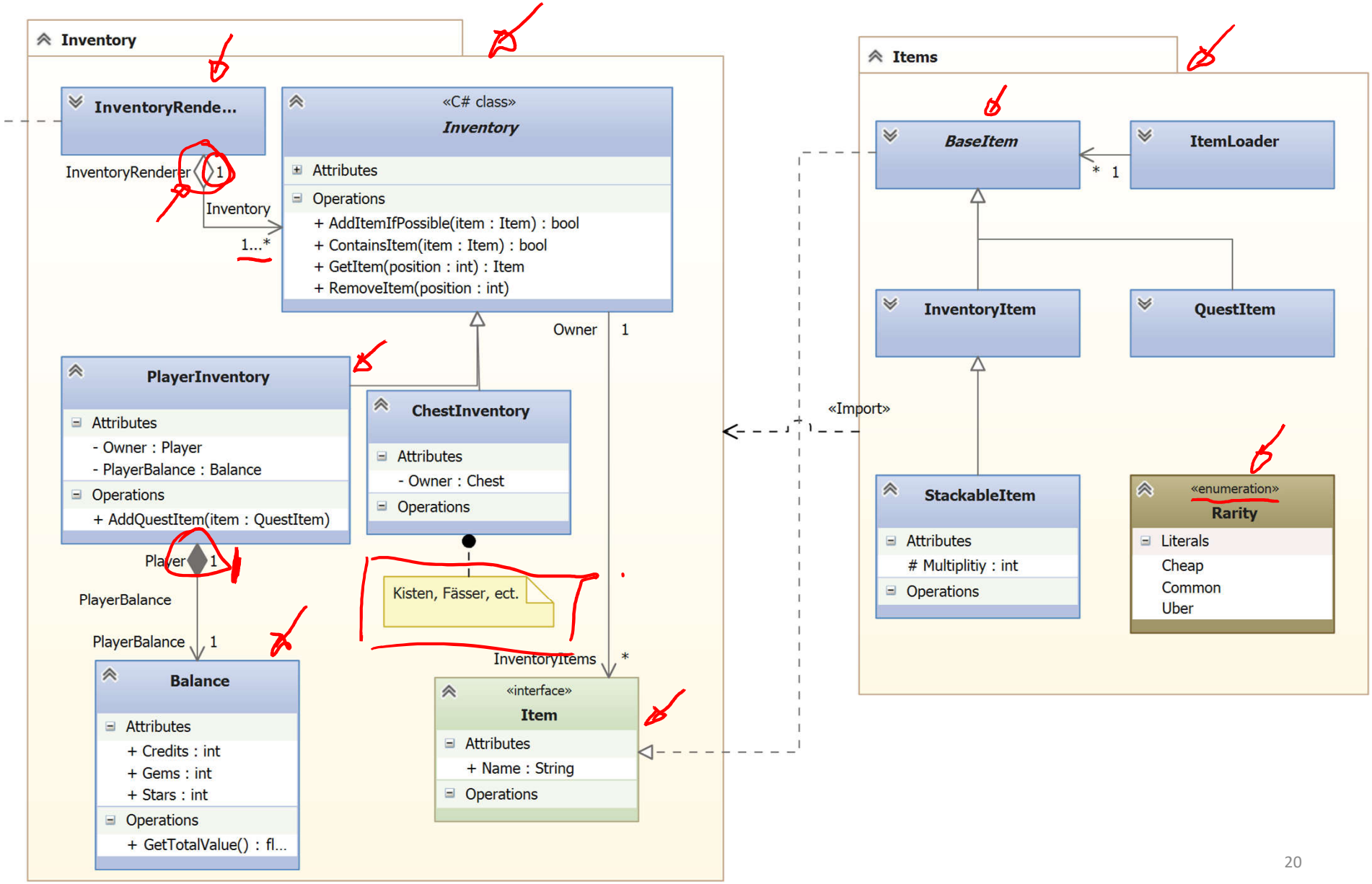
# Assoziation

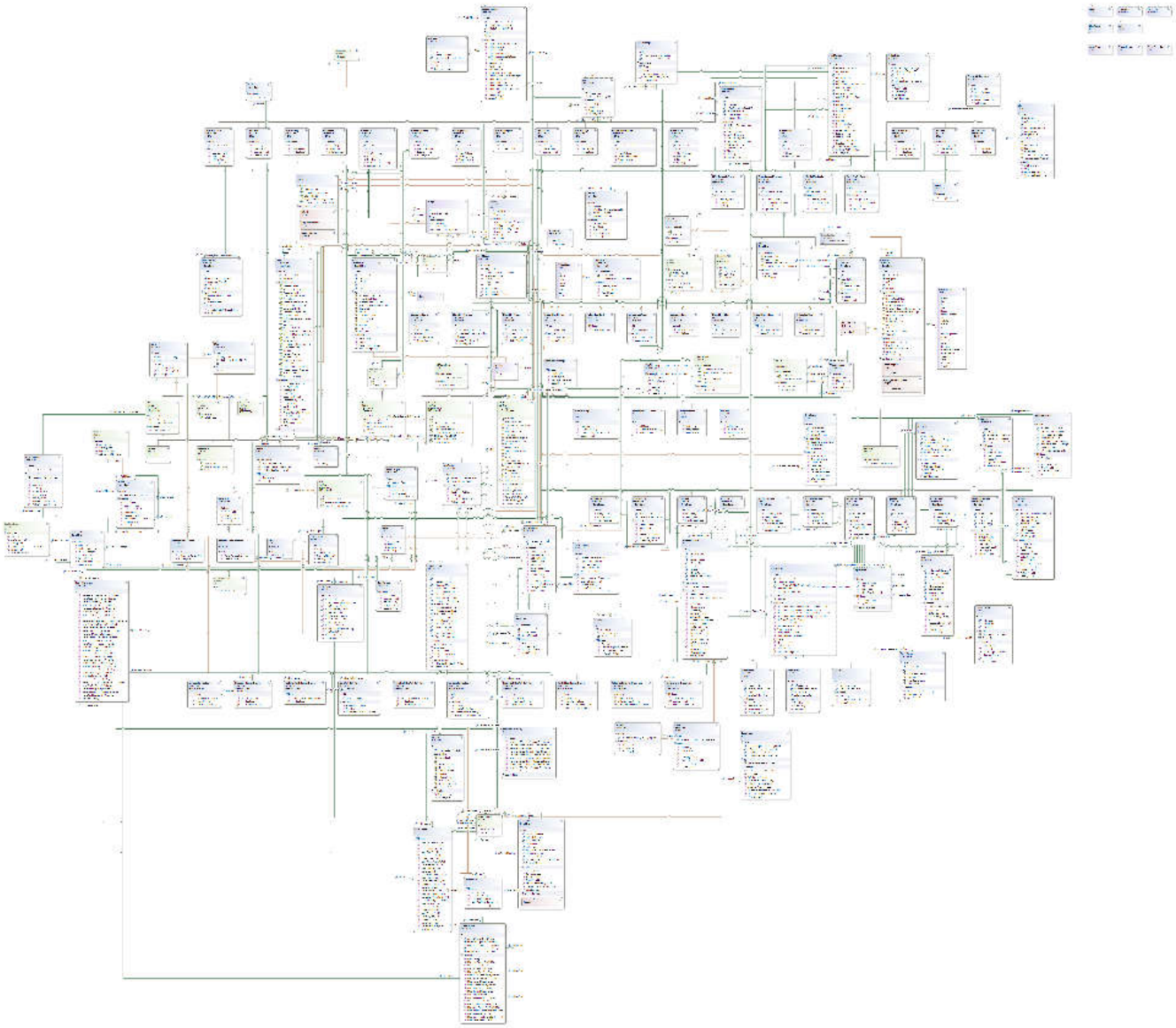


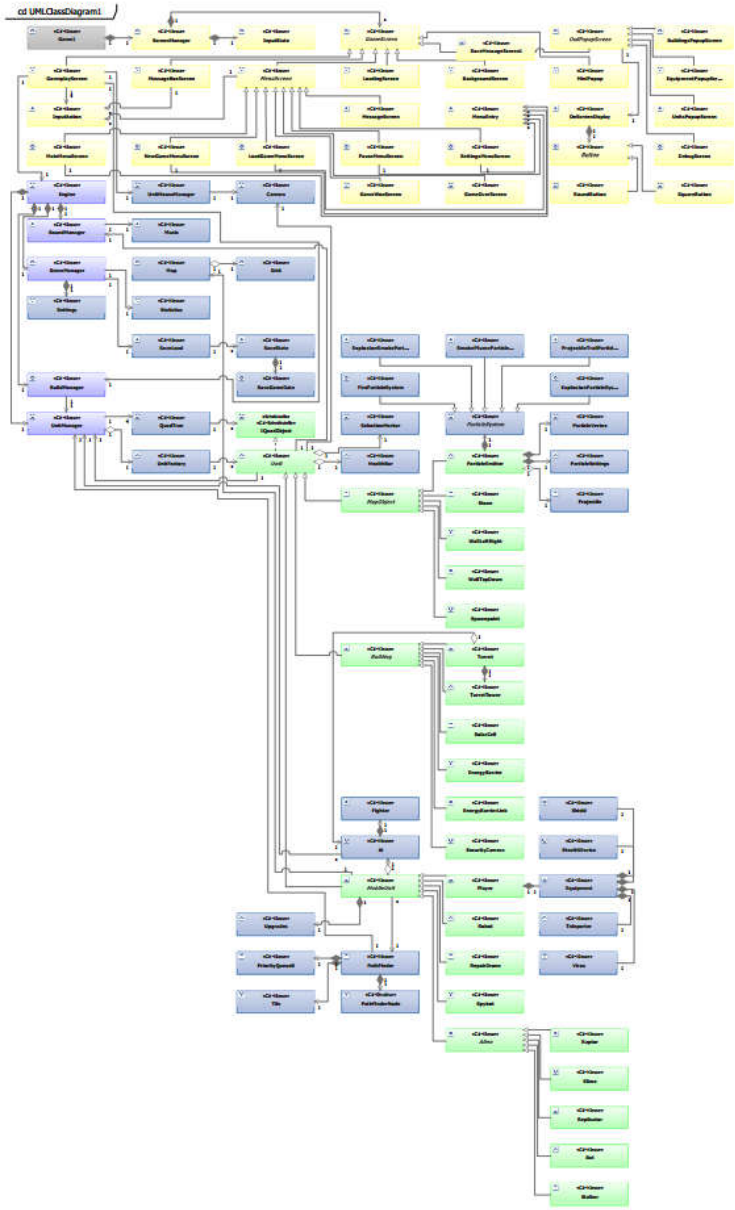

```
public interface Item
    {
        string Name
        {
            get;
            set;
        }
    }
```

# Vererbung und Realisierung




Inventory
InventoryRende...
InventoryRenderer
1
Inventory
1...*
«C# class»
Inventory
Attributes
Operations
+ AddItemIfPossible(item : Item) : bool
+ ContainsItem(item : Item) : bool
+ GetItem(position : int) : Item
+ RemoveItem(position : int)
Owner
1
PlayerInventory
Attributes
- Owner : Player
- PlayerBalance : Balance
Operations
+ AddQuestItem(item : QuestItem)
ChestInventory
Attributes
- Owner : Chest
Operations
Player
1
PlayerBalance
PlayerBalance
1
Kisten, Fässer, ect.
InventoryItems
*
Balance
Attributes
+ Credits : int
+ Gems : int
+ Stars : int
Operations
+ GetTotalValue() : fl...
«interface»
Item
Attributes
+ Name : String
Operations
«Import»
Items
BaseItem
ItemLoader
*
1
InventoryItem
QuestItem
StackableItem
Attributes
# Multiplitiy : int
Operations
«enumeration»
Rarity
Literals
Cheap
Common
Uber




cd UMLClassDiagram1

# Komponenten

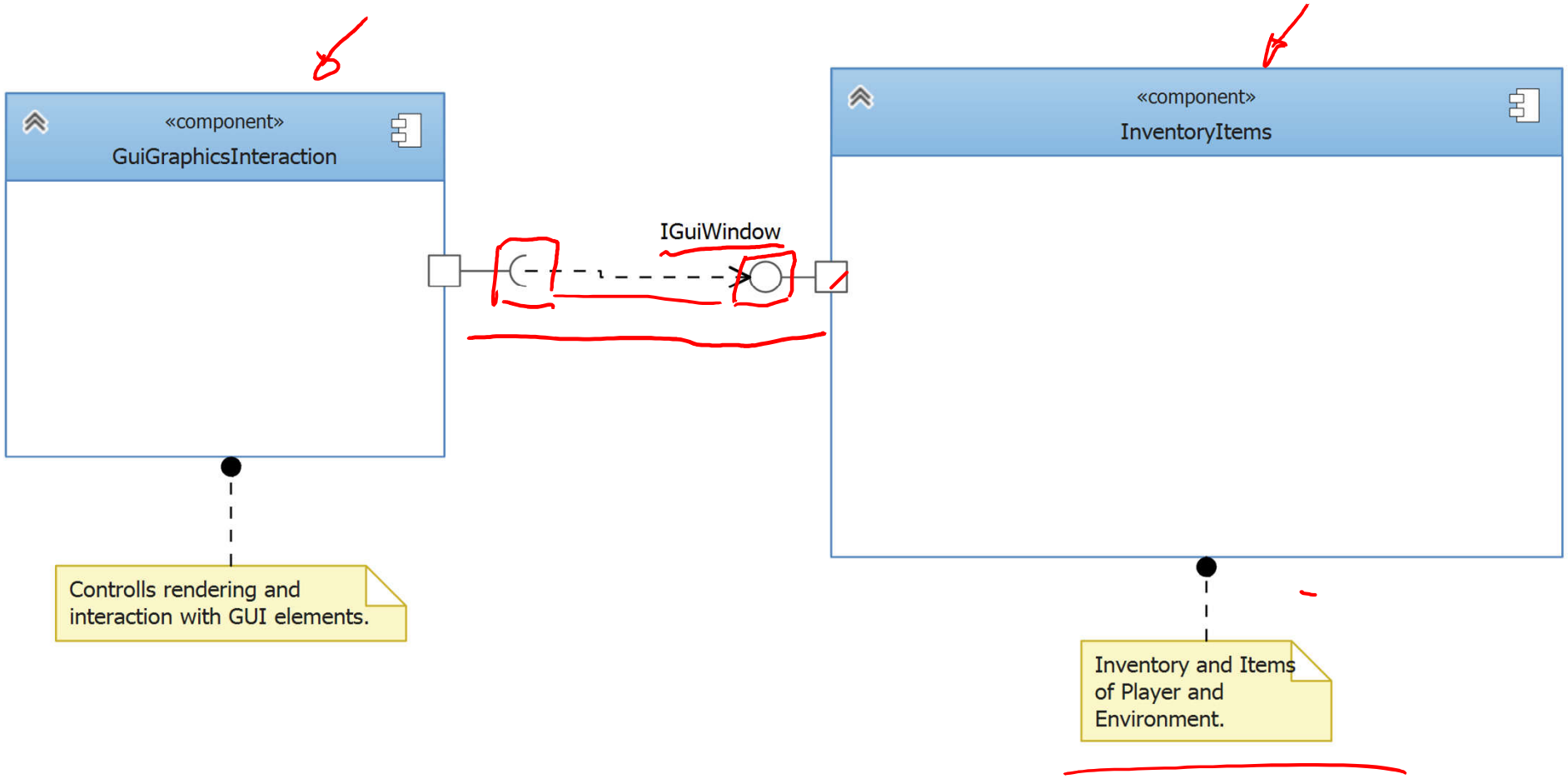
- Eine Komponente ist ein Softwarebaustein, der Dritten Funktionalität über Schnittstellen zur Verfügung stellt und nur explizite Abhängigkeiten nach außen besitzt.

# Komponentendiagramm

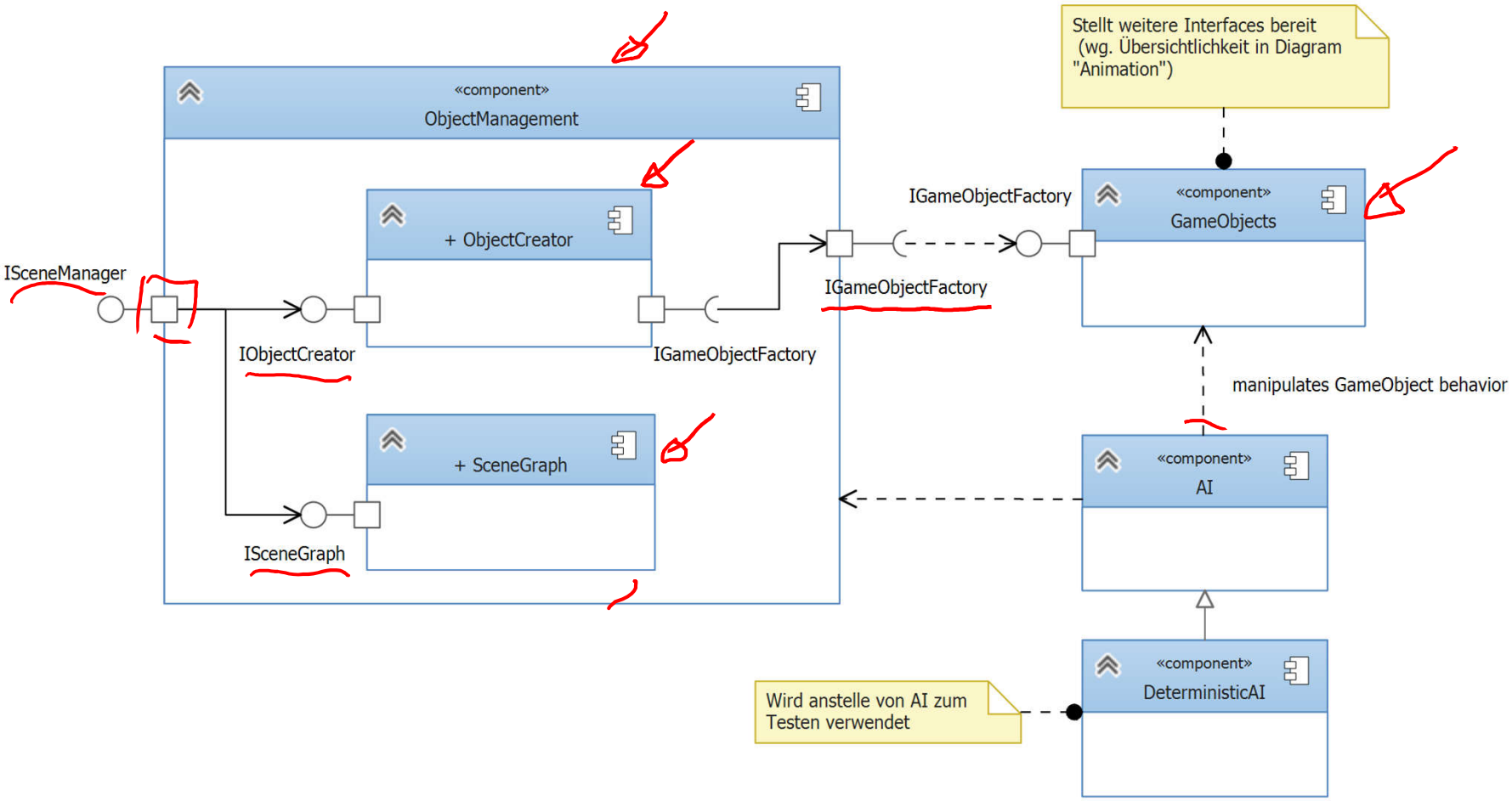
- Statische Sicht.
- Komponenten, Interfaces, Parts und deren statische Beziehungen untereinander als Bausteine.
- Abstrakter als Klassendiagram.
- Beschreibt „Verdrahtung“ von Komponenten.

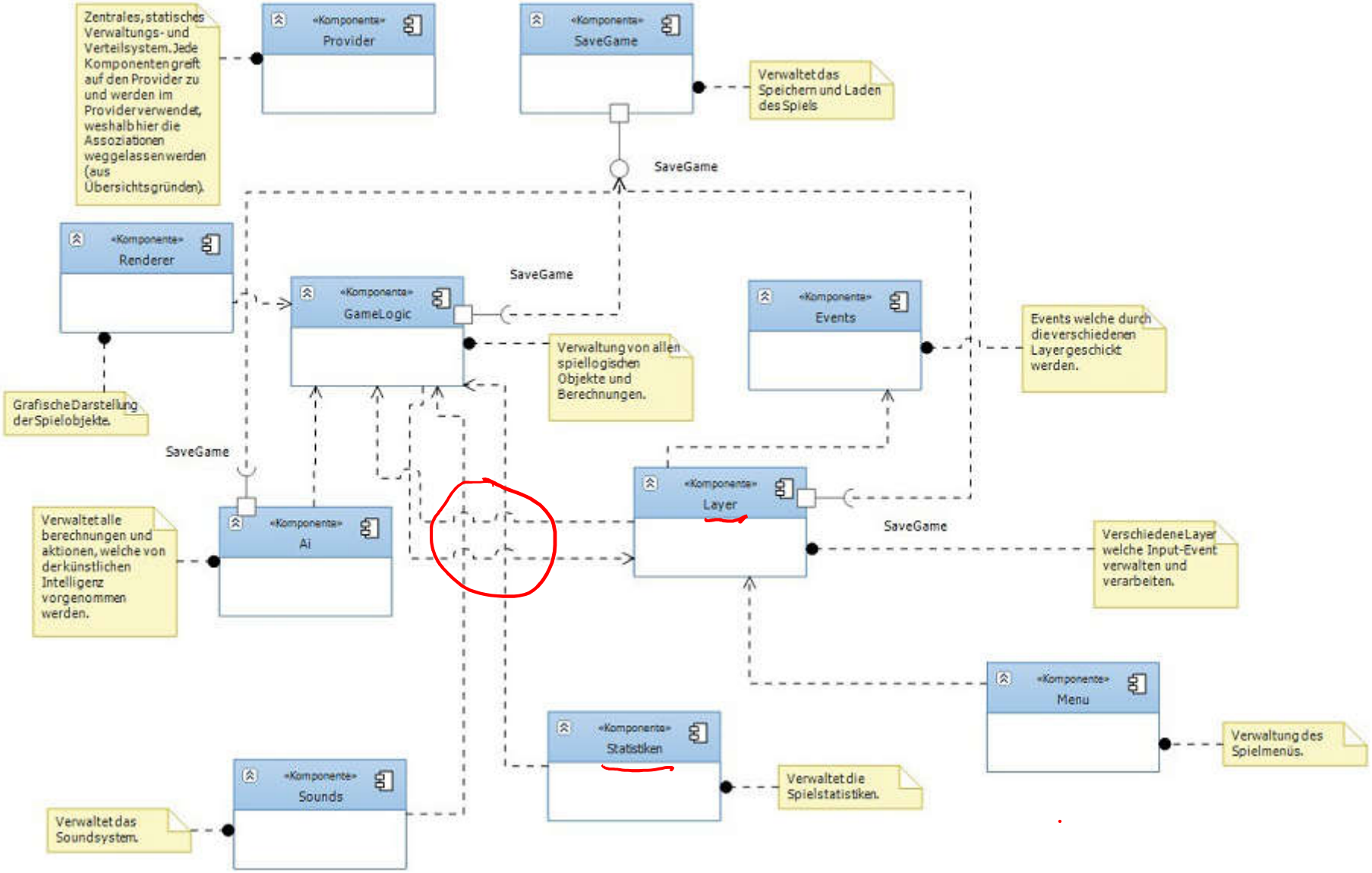
- Hier: Komponentendiagramm à la Microsoft Visual Studio.

# Komponentendiagramm



# Komponentendiagramm




Zentrales, statisches Verwaltungs- und Verteilsystem. Jede Komponenten greift auf den Provider zu und werden im Provider verwendet, weshalb hier die Assoziationen weggelassen werden (aus Übersichtsgründen).
«Komponente» Provider
«Komponente» SaveGame
Verwaltet das Speichern und Laden des Spiels
SaveGame
«Komponente» Renderer
Grafische Darstellung der Spielobjekte.
«Komponente» GameLogic
SaveGame
Verwaltung von allen spiellogischen Objekte und Berechnungen.
«Komponente» Events
Events welche durch die verschiedenen Layer geschickt werden.
SaveGame
Verwaltet alle berechnungen und aktionen, welche von der künstlichen Intelligenz vorgenommen werden.
«Komponente» Ai
«Komponente» Layer
SaveGame
Verschiedene Layer welche Input-Event verwalten und verarbeiten.
«Komponente» Menu
Verwaltung des Spielmenüs.
«Komponente» Statistiken
Verwaltet die Spielstatistiken.
«Komponente» Sounds
Verwaltet das Soundsystem.

# Werkzeugunterstützung

- Sie können jede Art von Werkzeug verwenden. Z.B.:
  - Visual Studio
  - NClass
  - ArgoUML mit C# Importer
  - ModelMaker
  - Grafikprogramme
  - Papier, Stifte, Scanner

- Visual Studio
  - Klassen- und Komponentendiagramme können mit Visual Studio gezeichnet werden (bis VS 2015).
  - Klassendiagramme können generiert werden (sehen dann aber anders aus).
  - Aus den Diagrammen kann auch Code erzeugt werden.

# WIE PLANE ICH EINE SOFTWAREARCHITEKTUR?

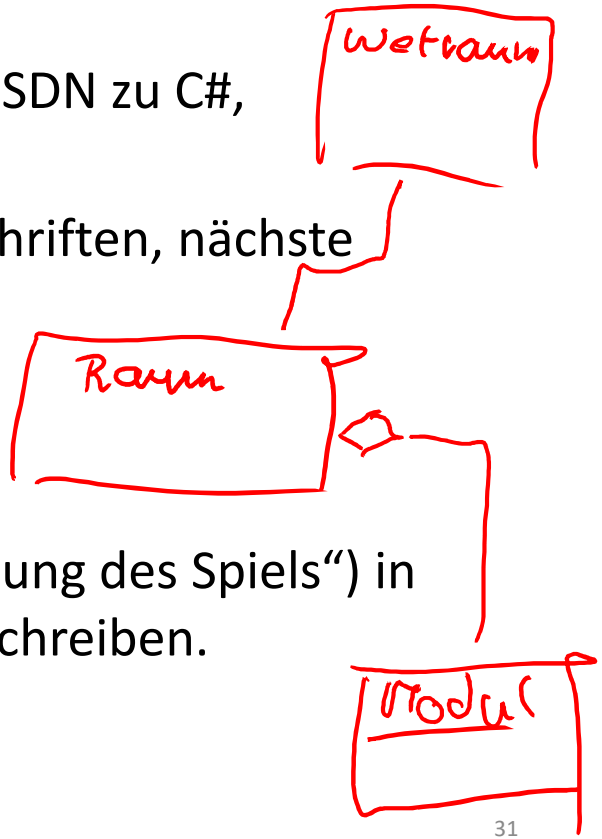
# Wie plane ich eine Softwarearchitektur?

- Schlechte Nachricht:
  - Es gibt kein deterministisches Verfahren, das in jedem Fall zu guten Softwarearchitekturen führt.

- Was nun?
  - Es gibt grundlegende Aktivitäten, Heuristiken, etc.

# Wie plane ich eine Softwarearchitektur?

- Informationen über das Problem sammeln
  - Anforderungen (GDD)
  - Randbedingungen und technischer Kontext (MSDN zu C#, MonoGame, eventuell F#)
  - Domänenwissen (Bücher, div. Websiten, Zeitschriften, nächste Vorlesung)




- Fachbegriffe sammeln und verstehen
  - Kernaufgabe des Systems (vgl. „Zusammenfassung des Spiels“) in wenigen Sätzen und mit eigenen Begriffen beschreiben.
  - Gemeinsame Sprache schaffen

# Wie plane ich eine Softwarearchitektur?

- Informationen über Lösungen sammeln
  - Wer hat eine ähnliche Aufgabe schon vorher gelöst, und wie?
  - Architekturstile und Entwurfsmuster
  - Quellen: eigene Projekte, Literatur, Internet, etc.

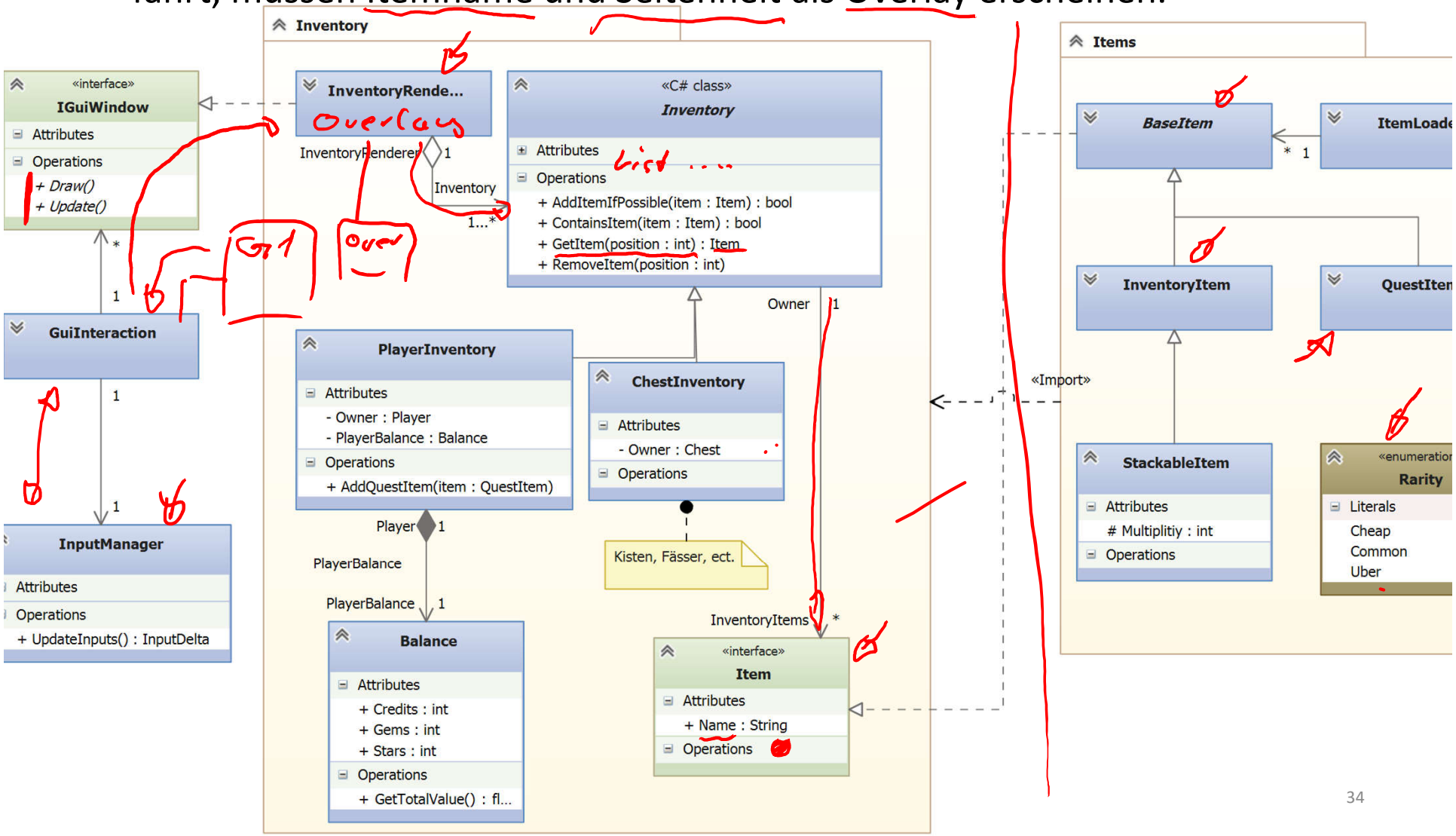
- Arbeiten Sie iterativ und inkrementell.
  - Vorsicht vor unnötiger Optimierung.
- Validieren Sie frühzeitig durch Implementierungen.
- Verwenden Sie Szenarien und User Stories zur Validierung.

# Szenarien

- Beispiel:
  - „Die Erstellung und Einbindung eines neuen 3D-Modells durch einen Modellierer muss innerhalb von 6h abgeschlossen sein.“
  - „Das Spiel muss im Endlosmodus 1000 aktive, durch den Spieler steuerbare Spielobjekte gleichzeitig auf einem Screen mit mindestens 45 FPS auf der Referenzhardware darstellen können.“

- Szenarien sind Ablaufbeschreibungen mit den folgenden Eigenschaften:
  - Auslöser (z.B. Modellierer, Spieler)
  - Quelle (z.B. intern, extern, Benutzer, Betreiber, Angreifer)
  - Umgebung (z.B. Endlosmodus, Entwicklung)
  - Systembestandteil (z.B. alle, Input, KI)
  - Antwort (z.B. Modell erfolgreich eingebunden, gleichzeitige Darstellung 45FPS)
  - Antwortmetrik (z.B. mehr oder weniger als 45FPS)

- „Wenn der Spieler im Inventarfenster mit dem Mauszeiger über ein Item fährt, müssen Itemname und Seltenheit als Overlay erscheinen.“



# Wie plane ich eine Softwarearchitektur?

- Clean Code enthält viele Hinweise
  - Nahezu alle Clean Code Prinzipien beziehen sich auf Architektur.

- Grundideen
  - Abhängigkeiten zwischen Bausteinen verringern:
    Niedrige Kopplung, hohe Kohäsion
  - Open-Closed Principle:
    Offen für Erweiterungen, geschlossen gegenüber Änderungen.

# ENTWURFSMUSTER

# Entwurfsmuster

- Siehe Softwaretechnik-Vorlesung „Design Patterns“.

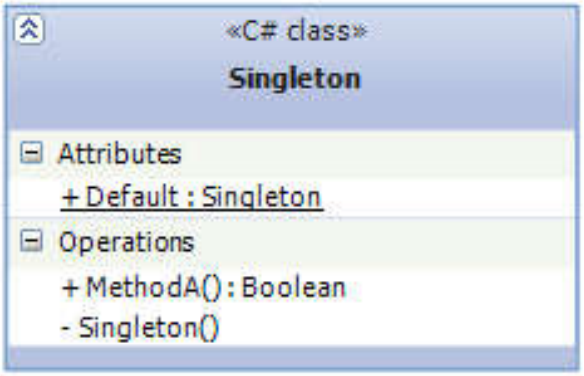
- Für Spiele:
  - Sehr nützlich:
    Composite, (Abstract) Factory, Builder, Flyweight, Observer, Visitor, Iterator
  - Vielleicht nützlich:
    Object Pool, Proxy, Prototype, Decorator, Command, Strategy, …

# Entwurfsmuster

- Vorsicht!
  - Entwurfsmuster können eine Architektur auch unnötig kompliziert machen.
  - Erst verstehen, wozu ein Muster gut ist, dann das Muster verwenden.
  - Nicht: Wie kann ich bloß dieses Muster 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.
  - Aufrufe von Methoden eines Singletons sind Aufrufe eines statischen Objekt.
  - Man kann sie überall verwenden.
- Schwierig zu testen.
  - Wie verändere ich alle Aufrufe zu einem Singleton in meinen Tests?
- 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?

# SOFTWAREMETRIKEN

# Softwaremetriken

- Metriken sind z.B. Länge, Fläche, Gewicht, Geschwindigkeit.
  - Gemessen z.B. in Einheiten wie cm, km, $m^2$, kg, km/h, …
- Eine Softwaremetrik ist eine Funktion, die eine Eigenschaft einer Software auf eine Zahl abbildet.

- Softwaremetriken werden benutzt zum
  - Vergleichen
  - Bewerten
- Verschiedene Tools berechnen Metriken
  - Sonar, NDepend, Visual Studio, …

# Lines of Code

- Anzahl an Zeilen einer Klasse bzw. des ganzen Projektes.

x = x + 2;

- Logisch
  Nur der eigentliche Code wird gezählt.
- Physikalisch
  Alle Zeilen (inkl. leere Zeilen, Kommentare, etc.).

x
=
x
+
2;

# Cyclomatic Complexity (CC)

- CC bestimmt, wie komplex eine Methode oder Klasse ist.

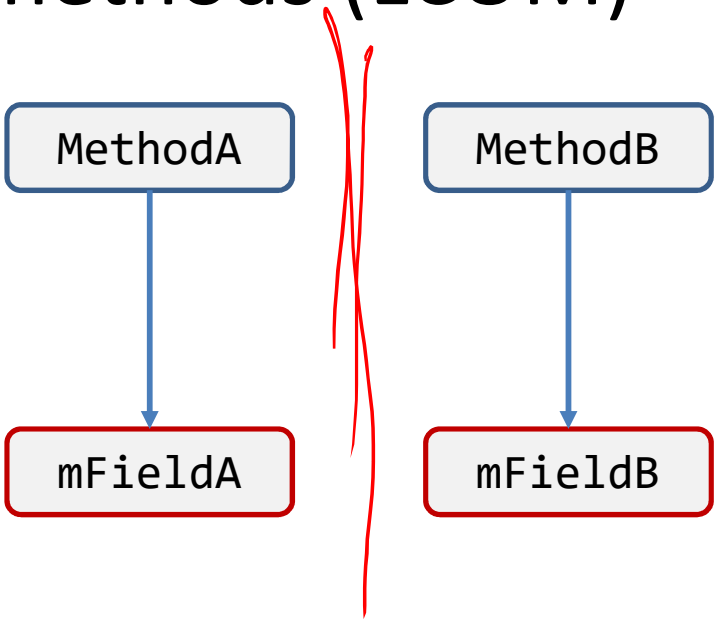
- Methode
  Anzahl der Ausführungspfade (`if`, `while`, `switch`, …)
- Klassen
  Durchschnittliche Komplexität aller Methoden oder Summe der Komplexität aller Methoden (je nach Tool).
- Schwellenwert
  Methoden mit CC > 15 sind kompliziert, bis CC < 30 aber ok

# Lack of cohesion of methods (LCOM)

- Beschreibt die Kohäsion einer Klasse
- Anzahl der unzusammenhängenden Teile einer Klasse
- Schwellenwert
  - LCOM4 = 0 bedeutet die Klasse hat keine Methoden.
  - LCOM4 = 1 bedeutet die Klasse ist zusammenhängend.
  - LCOM4 > 1 bedeutet die Klasse kann geteilt werden.

# Lack of cohesion in methods (LCOM)

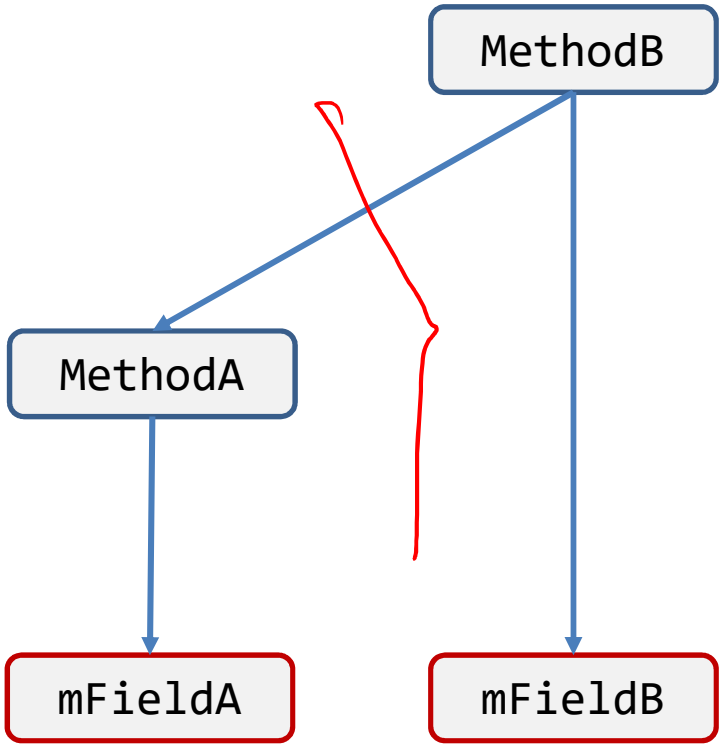
```
class Class1
{
    private int mFieldA;
    private int mFieldB;

    void MethodA()
    {
        mFieldA = 1;
    }

    void MethodB()
    {
        mFieldB = 2;
    }
}
```




$$LCOM4 = 2$$

# Lack of cohesion in methods (LCOM)

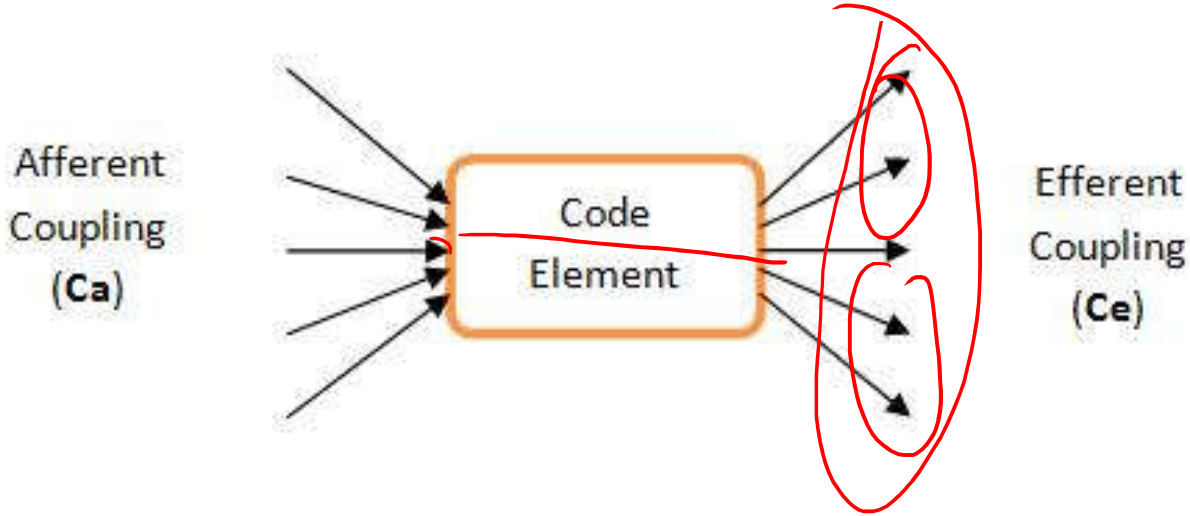
```
class Class1
{
    private int mFieldA;
    private int mFieldB;

    void MethodA()
    {
        mFieldA = 1;
    }

    void MethodB()
    {
        mFieldB = 2;
        MethodA();
    }
}
```




$LCOM4 = 1$

# Coupling

- Beschreibt wie viele Typen von einer Klasse (oder Modul, Namespace, Assembly) abhängen oder umgekehrt.



# Visual Studio Code Metrics

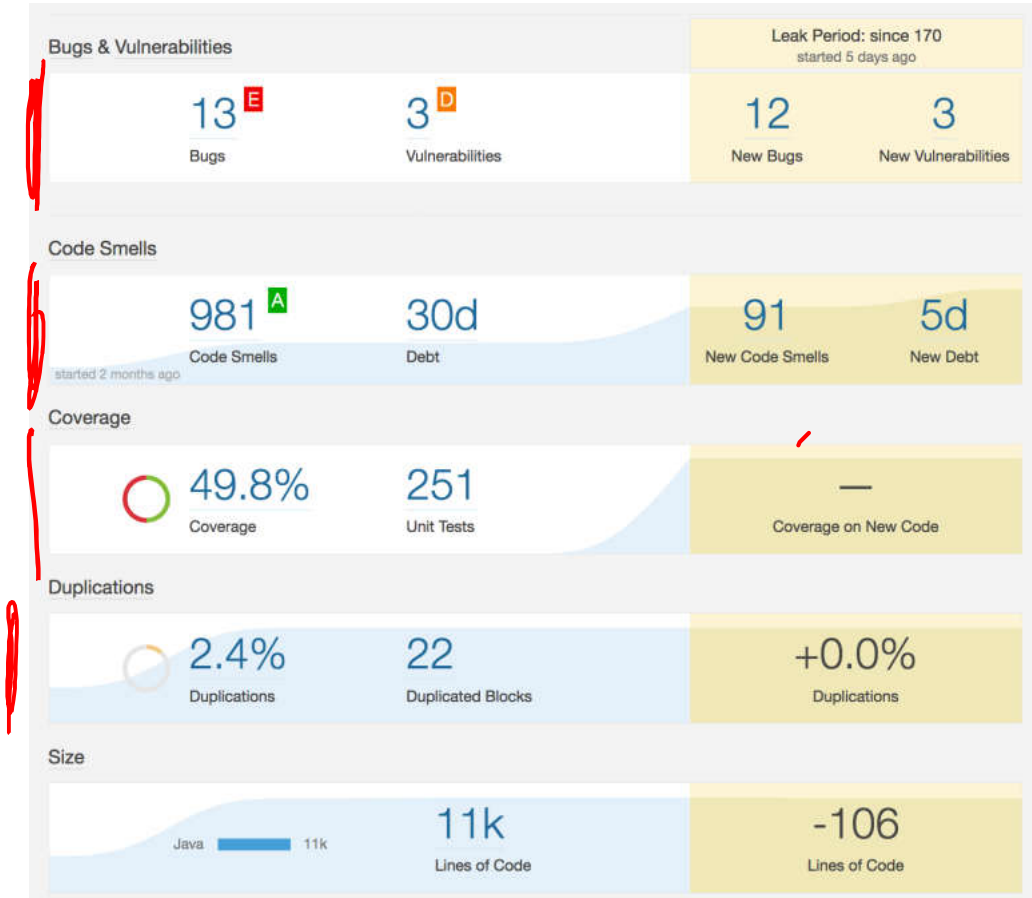
- Analyze -> Calculate Metrics

Code Metrics Results

Filter: None  Min:  Max:

| Hierarchy | Maintainability Index | Cyclomatic Complexity | Depth of Inheritance | Class Coupling | Lines of Code |
|---|---|---|---|---|---|
| TryGameTry (Debug) | 81 | 44 | 3 | 45 | 122 |
| TryGameTry | 81 | 44 | 3 | 45 | 122 |
| MeshedDrawableG | 77 | 6 | 3 | 14 | 15 |
| Game1 | 81 | 8 | 2 | 14 | 13 |
| RunningGame | 64 | 6 | 2 | 20 | 36 |
| DrawableGameObj | 100 | 3 | 2 | 1 | 1 |
| Draw() : void | 100 | 1 | | 0 | 0 |
| DrawableGame | 100 | 1 | | 1 | 1 |
| Update() : void | 100 | 1 | | 0 | 0 |
| InputWrapper | 65 | 9 | 1 | 12 | 37 |
| ScreenManager | 83 | 5 | 1 | 6 | 13 |
| Screen | 98 | 4 | 1 | 2 | 2 |
| Program | 80 | 2 | 1 | 3 | 3 |
| GameObject | 85 | 1 | 1 | 2 | 2 |

# Sonar


Bugs & Vulnerabilities
Leak Period: since 170
started 5 days ago
13 E
Bugs
3 D
Vulnerabilities
12
New Bugs
3
New Vulnerabilities
Code Smells
981 A
Code Smells
30d
Debt
91
New Code Smells
5d
New Debt
started 2 months ago
Coverage
49.8%
Coverage
251
Unit Tests
—
Coverage on New Code
Duplications
2.4%
Duplications
22
Duplicated Blocks
+0.0%
Duplications
Size
Java
11k
11k
Lines of Code
-106
Lines of Code

# Softwaremetriken

- Metriken geben Hinweise.
- Metriken sagen nichts über Funktionalität und Qualität zur Laufzeit aus.
- Metriken benötigen Kontext.
- Metriken können Details verschleiern.

- Erreichen einer bestimmten Metrik ist kein Ziel!

# 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.
- [Starke, 2011] Starke, G. (2011). Effektive Softwarearchitekturen: Ein praktischer Leitfaden. Hanser Fachbuch. ISBN 3446427287, 9783446427280.