Augmented Reality in Software-Engineering

Fortgeschrittene Softwaretechnik WS 2024/25

Lucas Kreber – kreberl@uni-trier.de

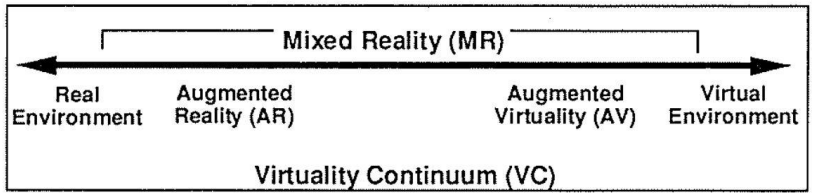
Einführung

Sensorama
(1962)

„The Ultimate Display“
Das erste HMD für immersive Umgebungen. (Ivan Sutherland, 1965)

Der Begriff Augmented Reality entsteht durch Tom Caudell (Boeing, 1990)

Milgram und Kishino definieren Taxonomie für Mixed-Reality (1994)



Apple Vision Pro

Meta Quest 3

Microsoft HoloLens2

ARCore
ARKit

Die VR/AR Geräte, wie wir sie heute kennen, kommen auf den Markt (2010-heute)

Abgrenzung von AR zu VR

„Während man unter *Virtual Reality* die Darstellung und gleichzeitige Wahrnehmung der Wirklichkeit und ihrer physikalischen Eigenschaften in einer in Echtzeit **computergenerierten, interaktiven, virtuellen Umgebung** versteht und die reale Umwelt demzufolge **ausgeschaltet** wird, zielt *Augmented Reality* auf eine **Anreicherung** der bestehenden realen Welt um **computergenerierte Zusatzobjekte** ab. Im Gegensatz zu *Virtual Reality* werden keine gänzlich neuen Welten erschaffen, sondern die **vorhandene Realität mit einer virtuellen Realität ergänzt**.“

Klein, Georg: *Visual Tracking for Augmented Reality : Edge-based Tracking Techniques for AR Applications*. Saarbrücken: VDM Publishing, 2009. ISBN 978-3-639-07891-6. S. 1

VR / AR

Virtual Reality (VR)

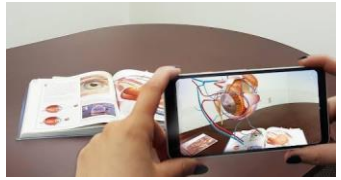
- Beschreibung: Vollständig computergenerierte, virtuelle Umgebung, die die reale Welt ersetzt.
- Ausrüstung: VR-Headsets (z. B. Meta Quest 3) und oft Controller. Umschließt gesamtes Sichtfeld, gesamte Abschirmung der Realität.

Augmented Reality (AR)

- Beschreibung: Erweiterung der realen Welt durch digitale Informationen, Bilder oder Objekte. Die reale Umgebung bleibt dabei sichtbar.
- Ausrüstung: Smartphones oder Tablets verwendet, die mit der Kamera die reale Umgebung erfassen und auf dem Bildschirm die virtuellen Inhalte einblenden. AR-Brillen, wie die Microsoft HoloLens, welche digitale Inhalte direkt ins Sichtfeld projizieren.

Klassische Anwendungsszenarien

Bildung und Schulung



Industrie und Wartung

Gaming und Unterhaltung

Gesundheitswesen

... und viele mehr!

Auch Einsatz im Kontext von
Software-Engineering
möglich?

Vergleich HMD/Smartphone

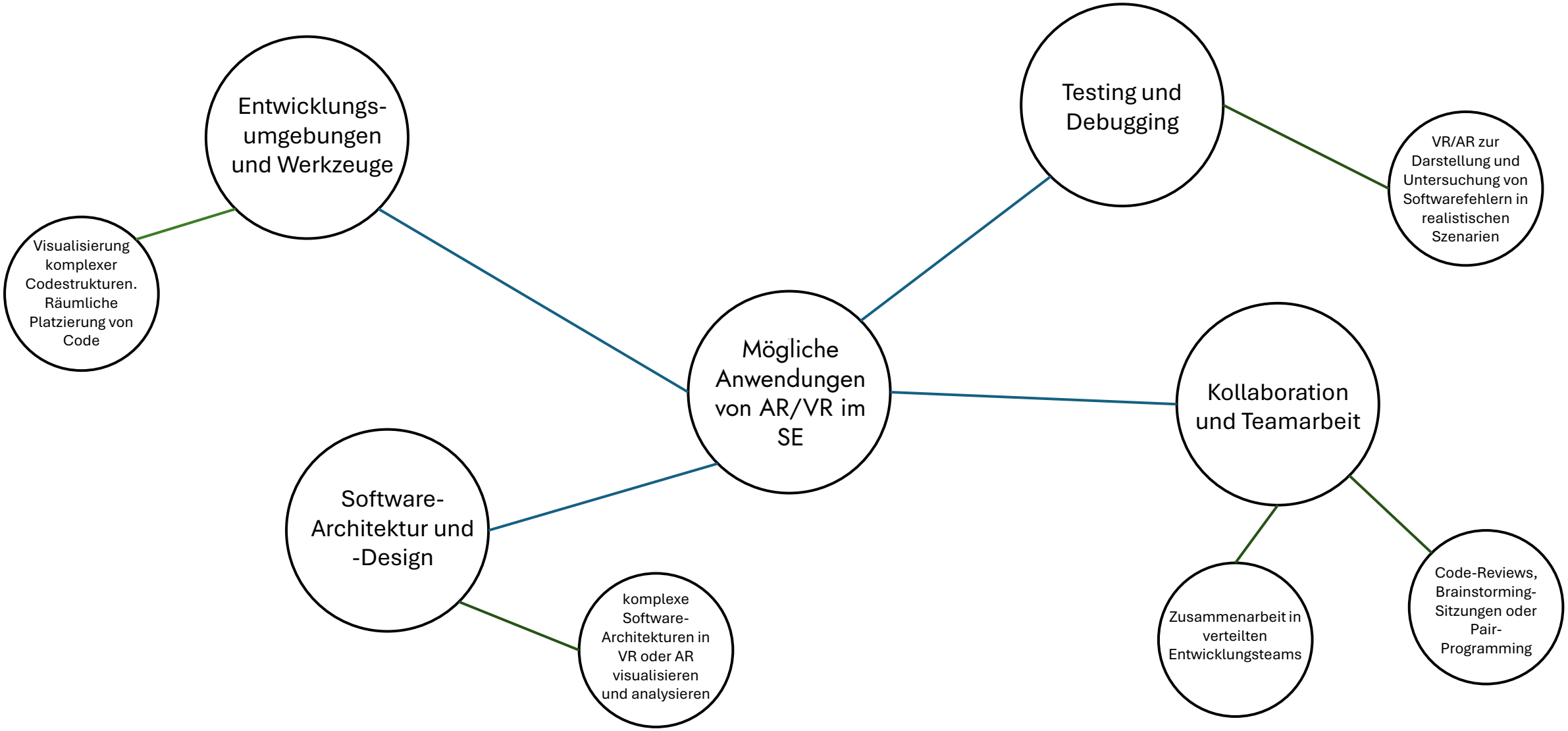
Display und Immersion
- Smartphones: AR nur auf Bildschirm, kein vollständiges Eintauchen.
- HMDs: Immersives Erlebnis, VR-HMDs mit komplett digitalem Sichtfeld, AR-HMDs projizieren Inhalte ins Sichtfeld.

Interaktionsmöglichkeiten
- Smartphones: Touchscreen und einfache Gesten, weniger natürliche Interaktion.
- HMDs: Bewegungen, Handgesten, spezielle Controller, Eye-Tracking und Hand-Tracking.

Feld der Wahrnehmung (Field of View)
- Smartphones: Eingeschränkt durch Bildschirmgröße.
- HMDs: Breiteres Sichtfeld, intensiveres VR- und AR-Erlebnis.

Tracking und Sensoren
- Smartphones: Grundlegende Sensoren.
- HMDs: Fortgeschrittene Tracking-Technik.

Eigene Ideen?



Mögliche Fragestellungen

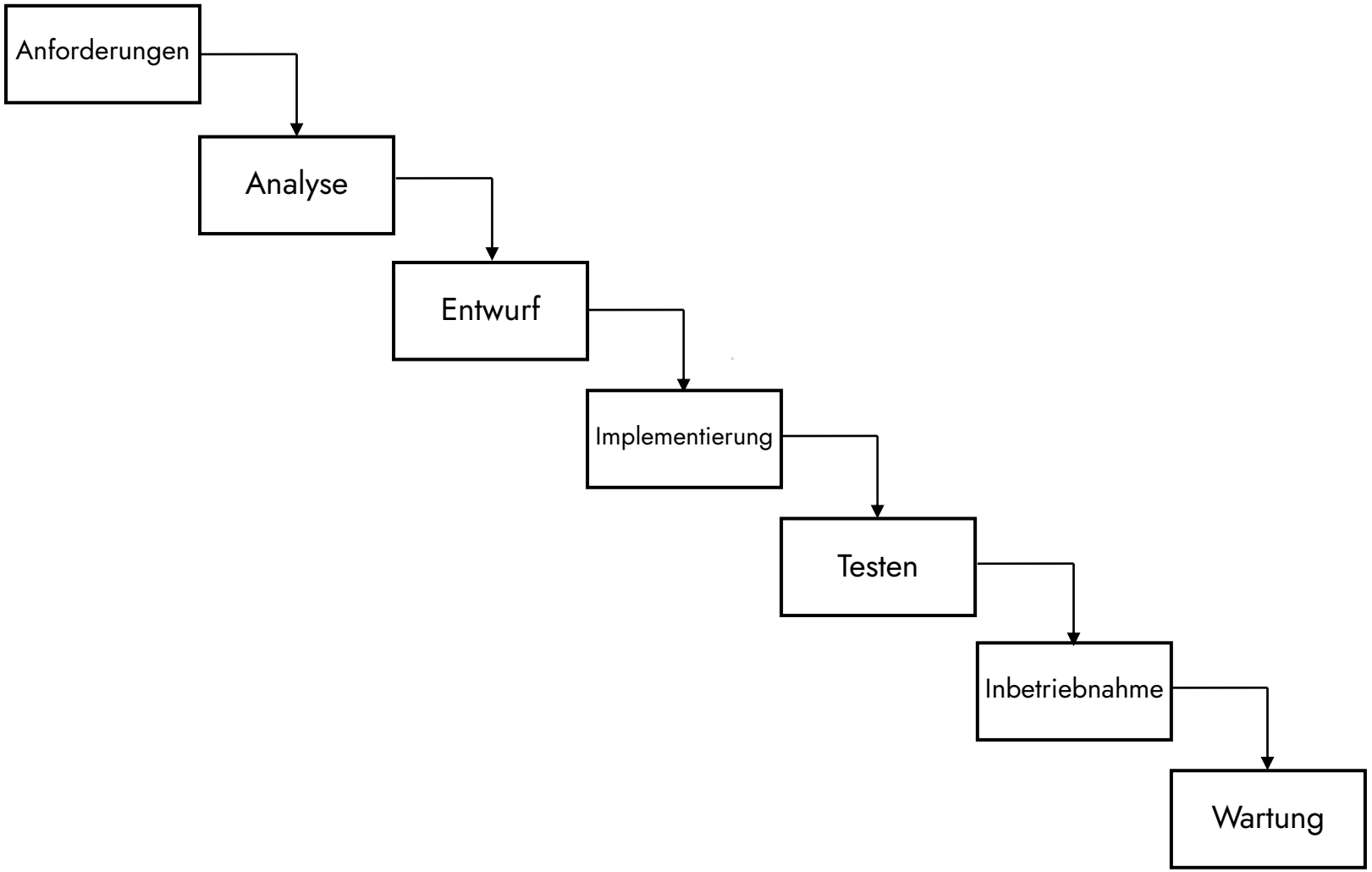
Wie lässt sich AR im Rahmen von Software Engineering sinnvoll nutzen?

Welche Anwendungen sind sinnvolle Erweiterungen?

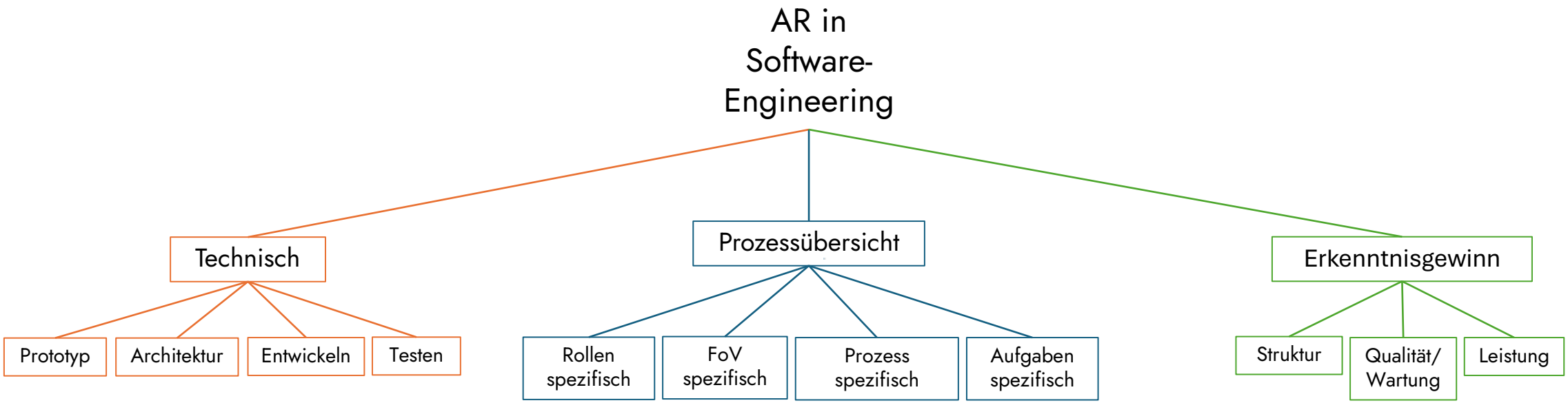
Welche Szenarien eignen sich besonders gut, welche eignen sich weniger?

Alle Phasen von SE gleich sinnvoll oder nur in ausgewählten Phasen?

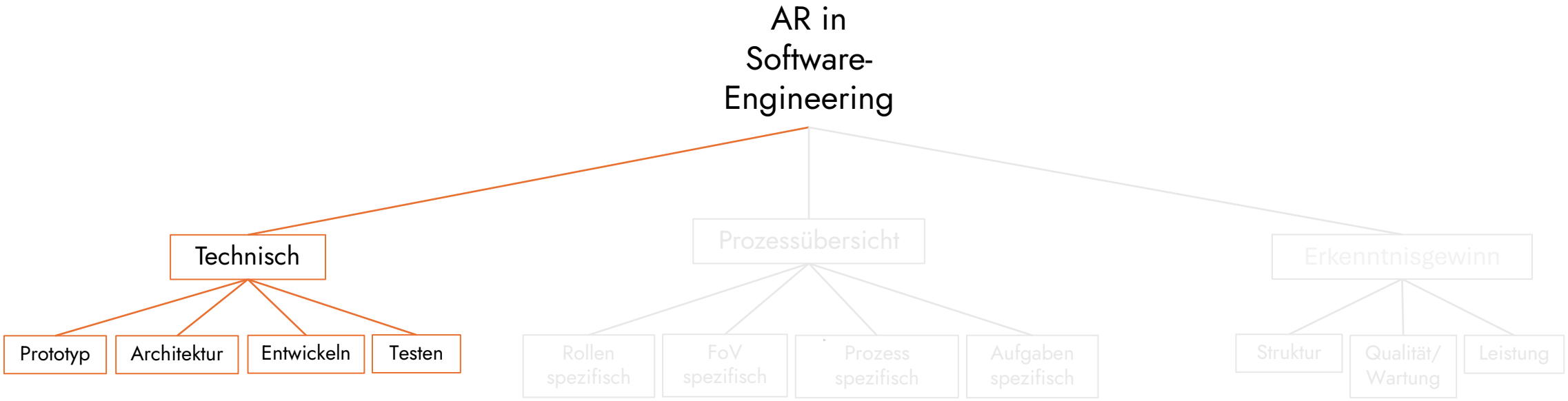
Phasen


Anforderungen
Analyse
Entwurf
Implementierung
Testen
Inbetriebnahme
Wartung

Immersive Ansätze in Software-Engineering

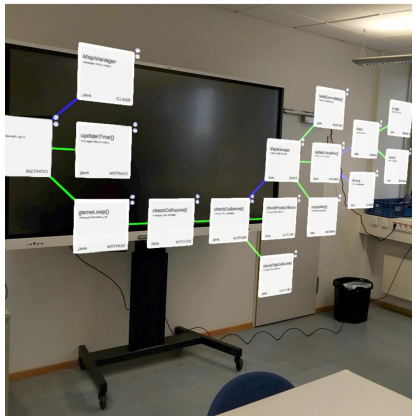


Immersive Ansätze in Software-Engineering

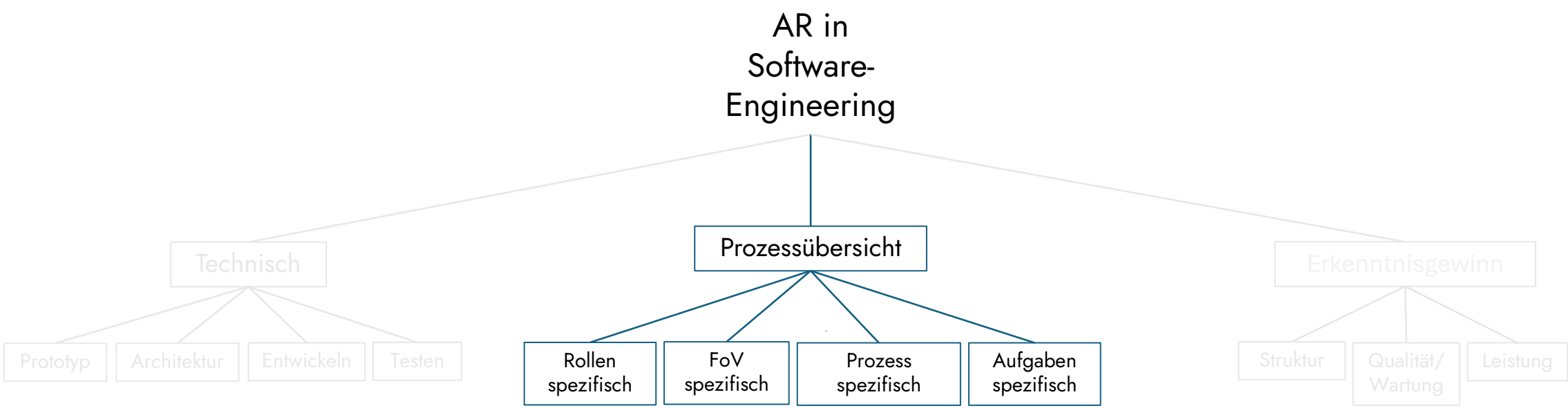


Mögliche Einsatzgebiete liegen zum Beispiel in der

- Visualisierung von Architekturen, Prototypen oder Problemen die beim Testen aufgedeckt werden,
- der Interaktion mit dem System
- und der dazugehörigen Kollaboration

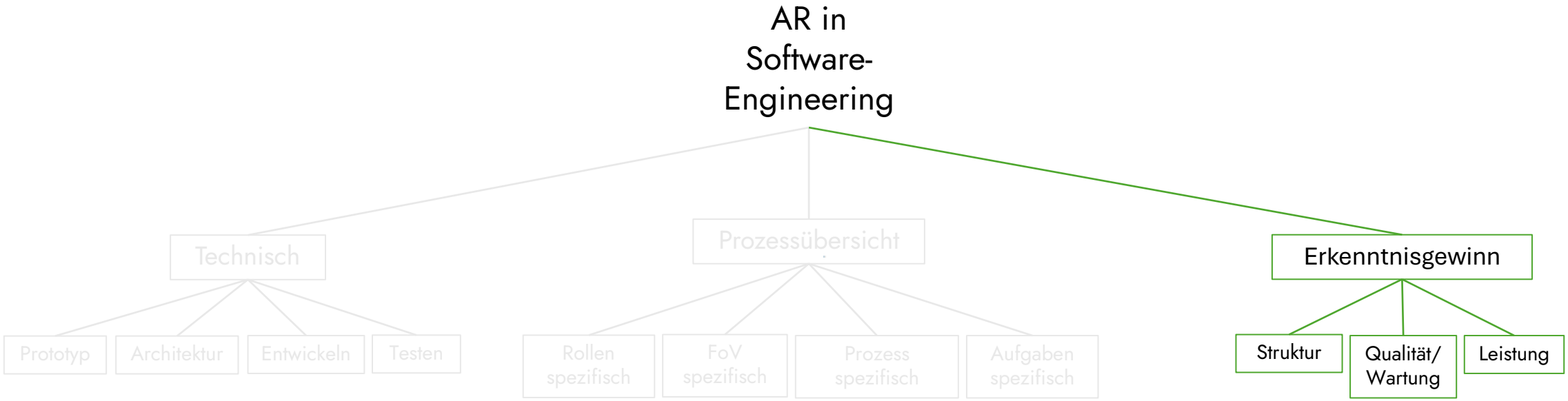


Immersive Ansätze in Software-Engineering



- Augmentieren von Daten und Aktivitäten der Entwickler und Entwicklerinnen im Entwicklungsprozess
- Anzeige zum Beispiel von:
  - Fortschritt einzelner Personen während Entwicklungsprozess
  - Monitoring einer Anwendung

Immersive Ansätze in Software-Engineering

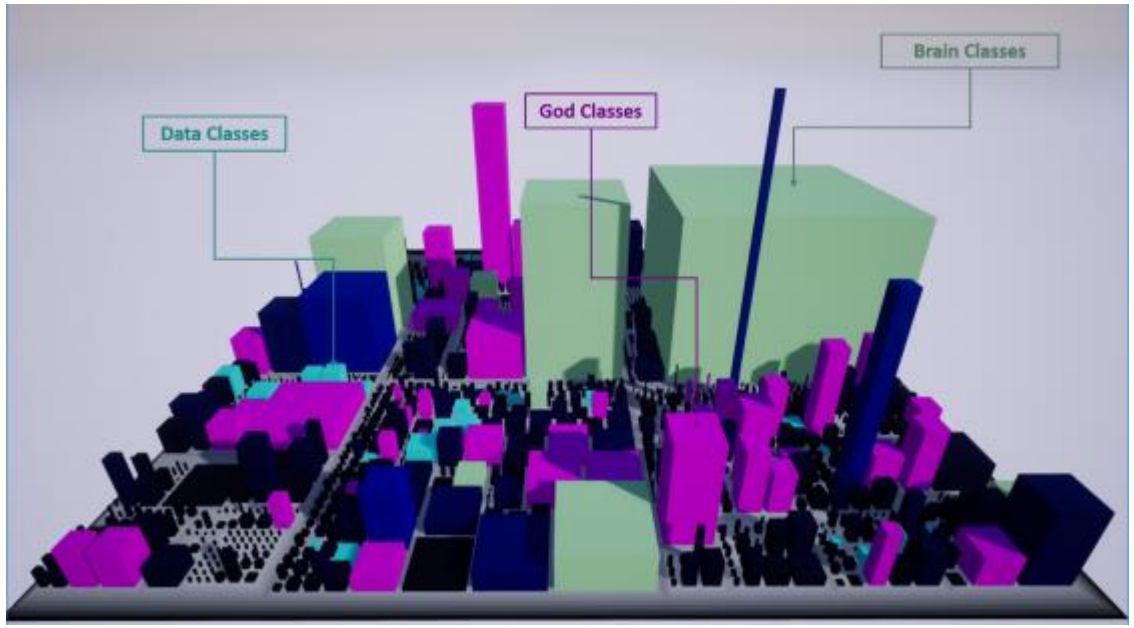


- Software wird immer komplexer und schwieriger zu verstehen
  - ⇒ Immersive Darstellungen ermöglichen ganz neue Einblicke in das System und vereinfachen das Verständnis
  - ⇒ Einblicke in die Struktur, Performance oder Qualität der Software

Werkzeuge im Kontext von Software-Engineering

City Metapher:

Eine City-Metapher als immersives Werkzeug, um Entwickler und Entwicklerinnen einen Überblick über das bestehende Software-System zu vermitteln.



Hier genutzt für die Darstellung von Code Smells:

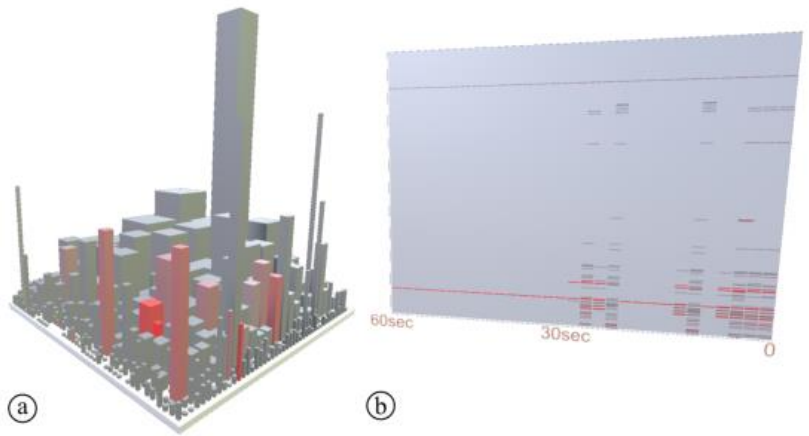
- brain classes (grün)
- data classes (türkis)
- god classes (lila)

Auch oft benutzt, um Klassen, Methoden etc. darzustellen

Werkzeuge im Kontext von Software-Engineering

PerfVis (Software Performance Monitoring):

Ein Werkzeug zum Visualisieren von Software-Performance durch Augmented Reality. Das Werkzeug zeigt statische Daten der Programmstruktur und dynamische Performance-Daten des Programms



- (a)
  - Verwendet die City-Metapher
- (b)
  - Performance-Vergleich zu unterschiedlichen Zeitpunkten
  - Suche nach Ausreißern
  - Programmverhalten über die Zeit beobachten

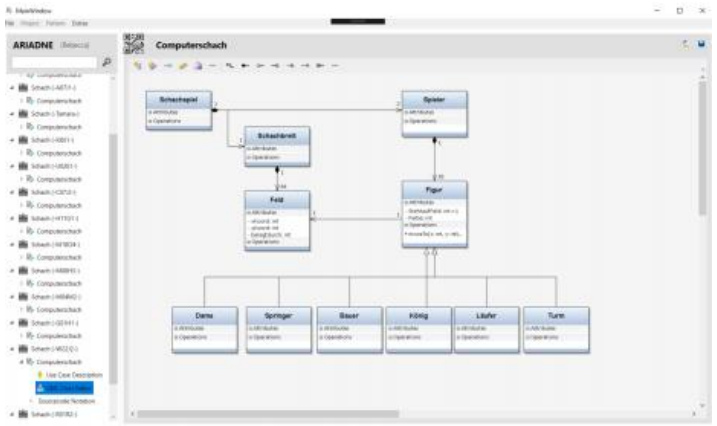
PerfVis: Pervasive Visualization in Immersive Augmented Reality for Performance Awareness

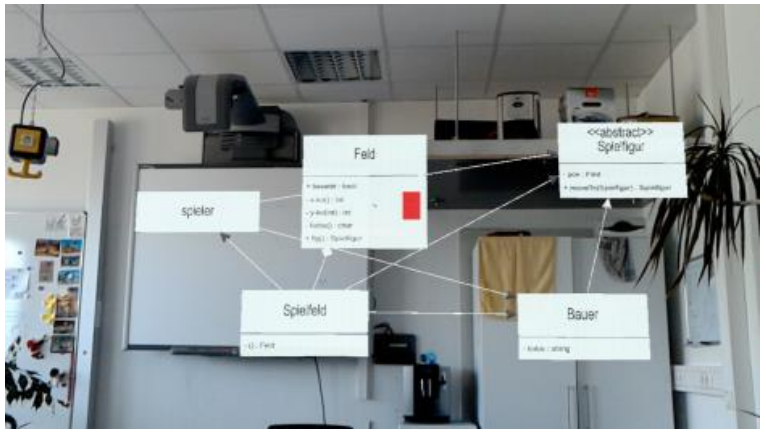
Werkzeuge im Kontext von Software-Engineering

Ariadne:

Werkzeug zum Entwickeln und Darstellen von UML-Diagrammen in einer 3D-Umgebung

- Entwicklung noch am Anfang
- Evaluation zeigte keine Tendenz, ob besser oder schlechter

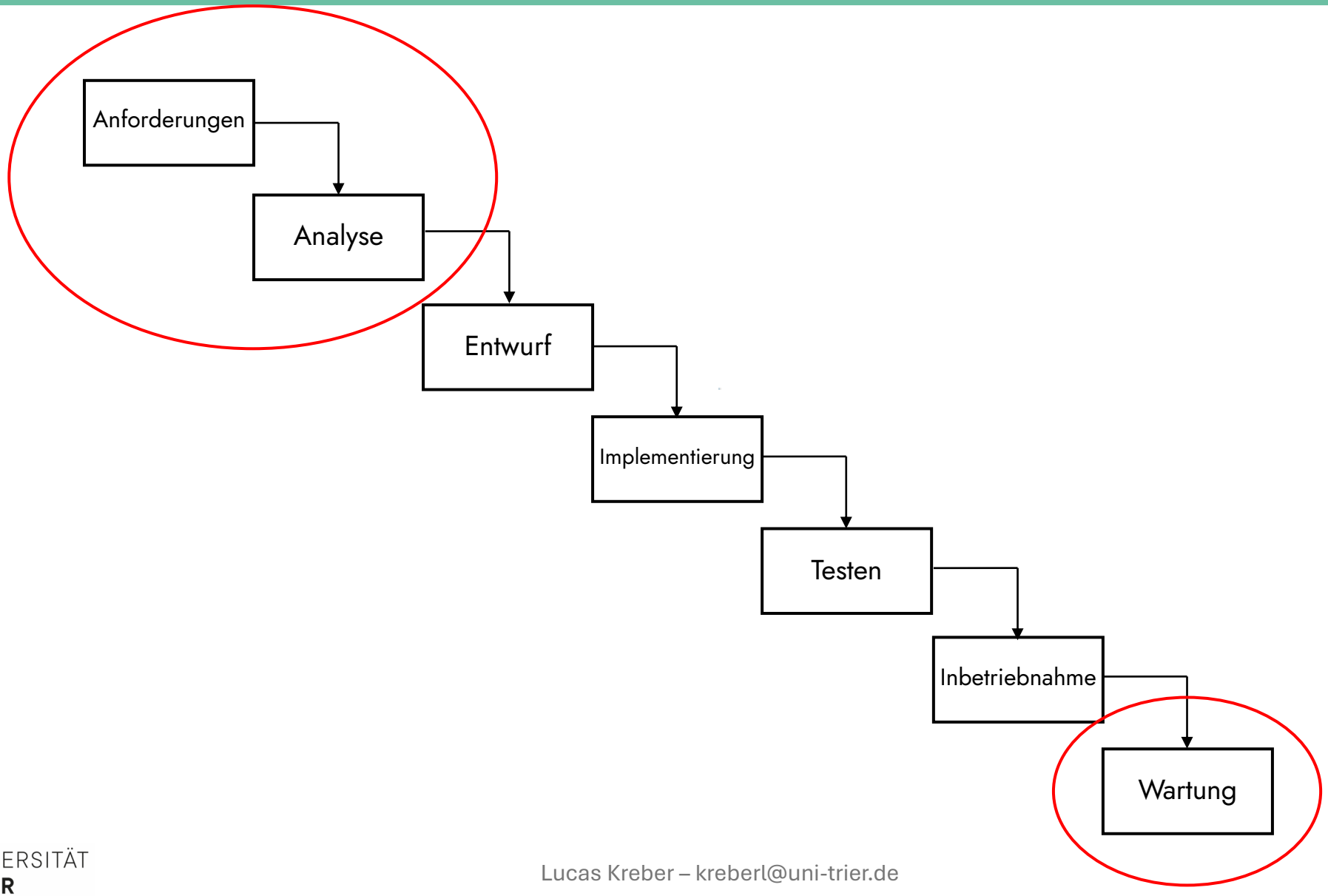




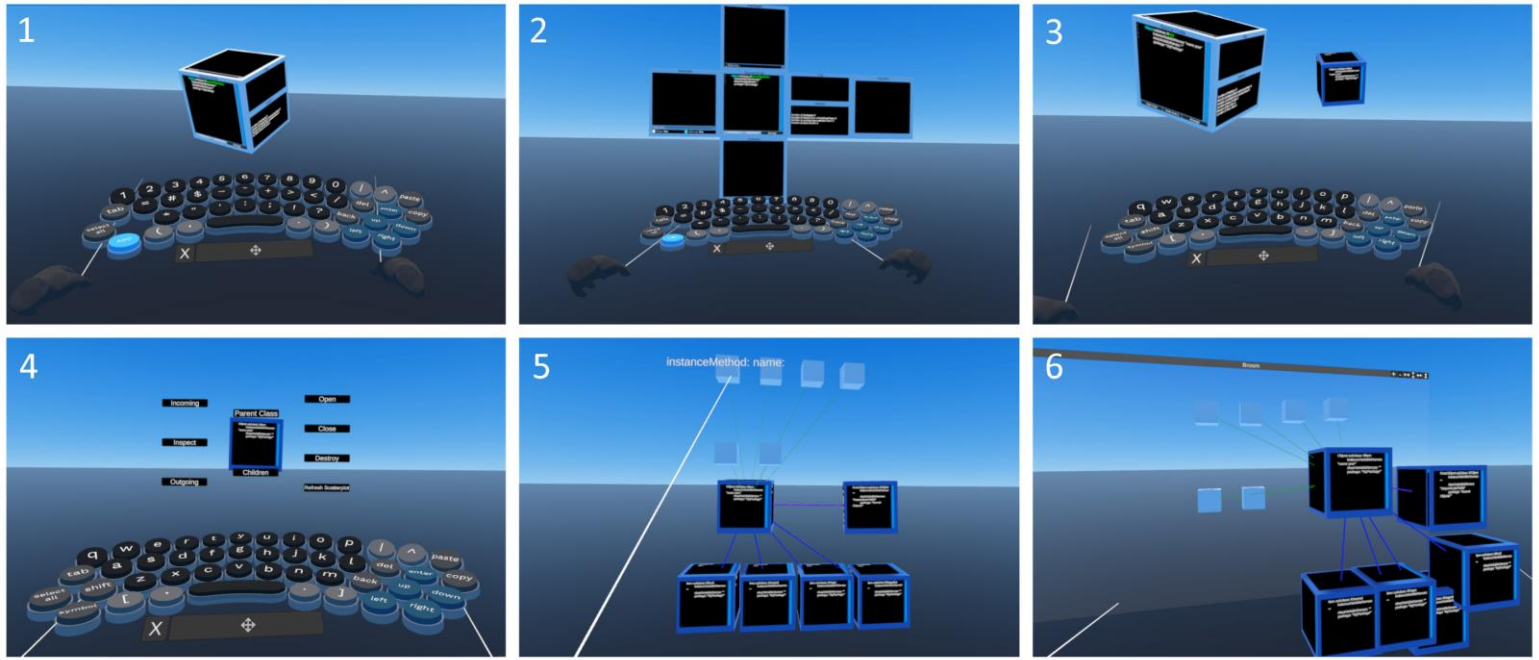
Werkzeug bietet sowohl 2D-Sicht, als auch AR-Sicht

Using Augmented Reality in Software Engineering Education?

Phasen


Anforderungen
Analyse
Entwurf
Implementierung
Testen
Inbetriebnahme
Wartung

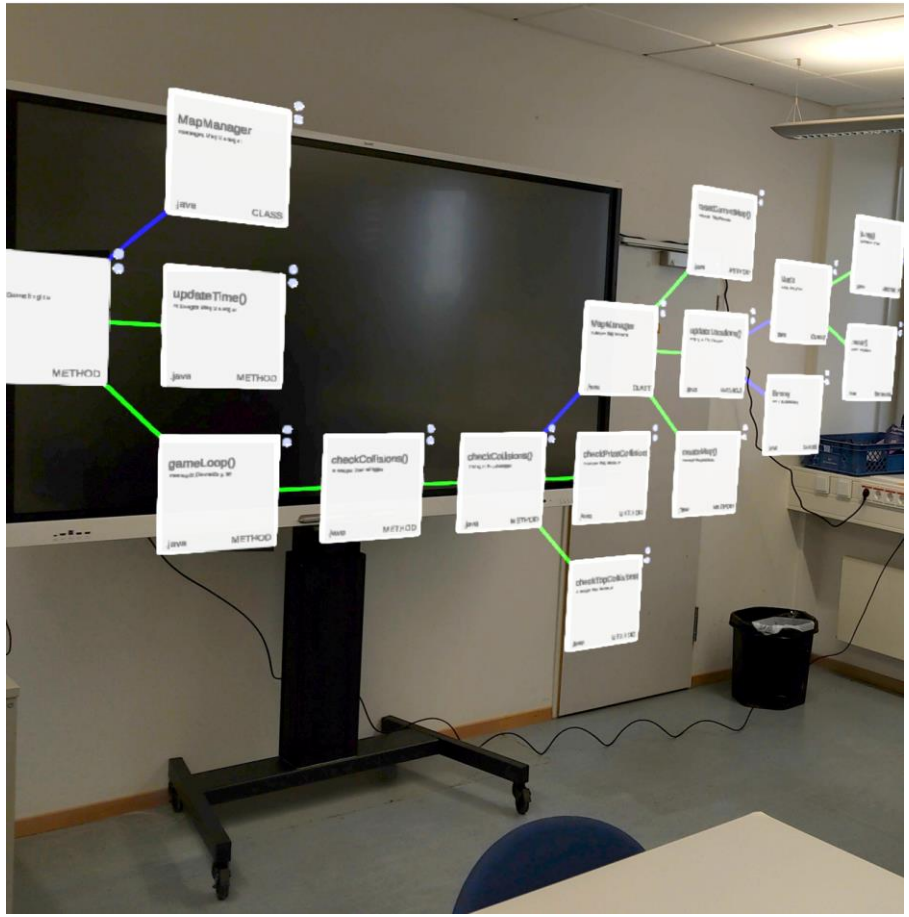
Werkzeuge im Kontext von Software-Engineering



VRIDE (2021)

- VR basierte Entwicklungsumgebung
- Code Cube zeigt Klasse
- Jede Fläche zeigt andere Aspekte der Klasse
  - Klassendefinition, Methodenliste, Code von Methoden, eingehende und ausgehende Kanten,…

Werkzeuge im Kontext von Software-Engineering



IDEVELOPAR (2022)

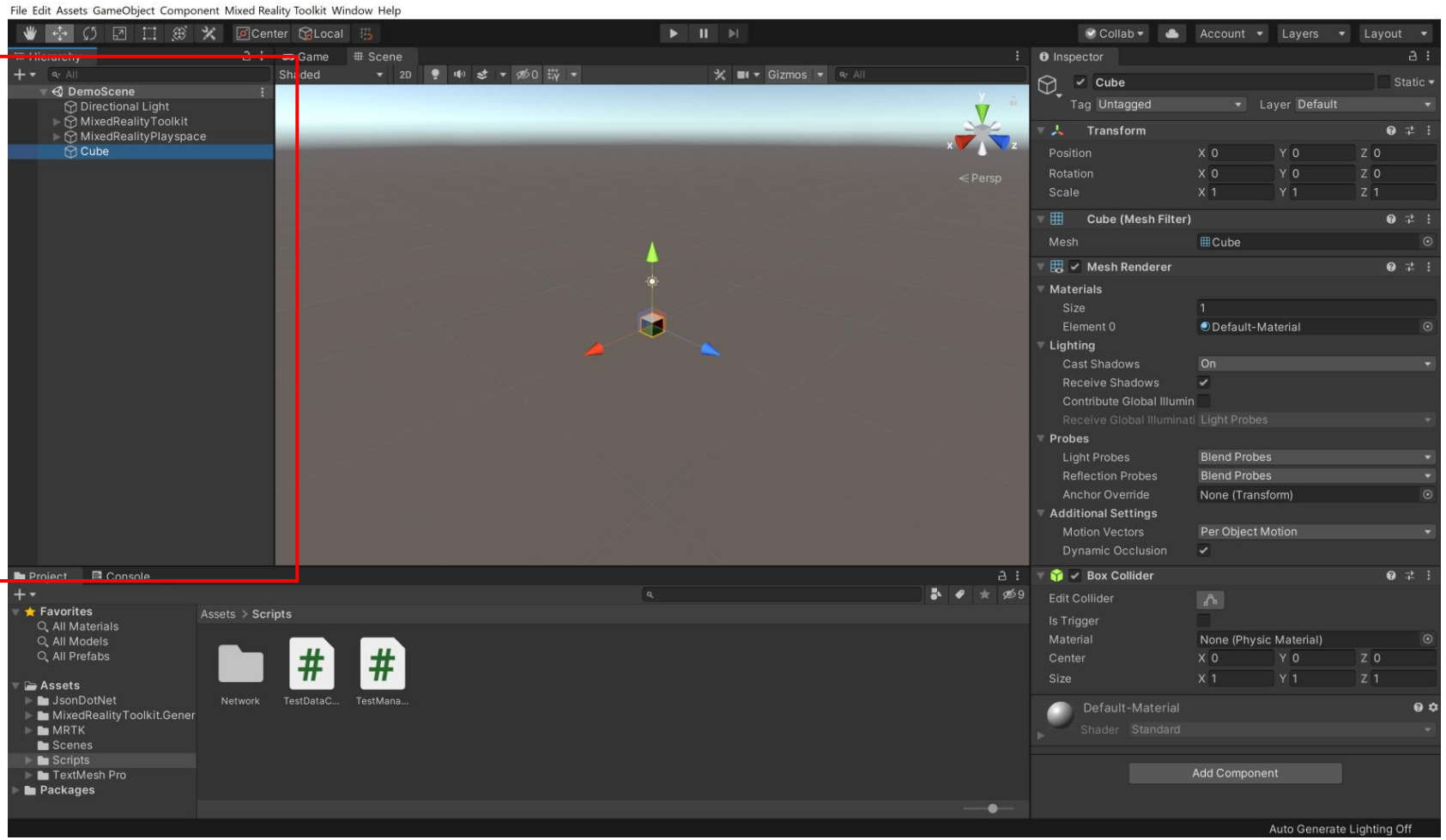
- AR basierte Entwicklungsumgebung (HoloLens 2 und Quest 3)
- Jedes Codefragment wird in eigenem Panel angezeigt
- Verbindungen zwischen den einzelnen Panels (Navigationsbaum)
- Panels können frei im Raum platziert werden
- Es besteht eine Verbindung zu einer IDE, Sourcecode in Panels kann direkt in AR-Umgebung bearbeitet werden

Unity

Erste Schritte in Unity

Hierarchy View:

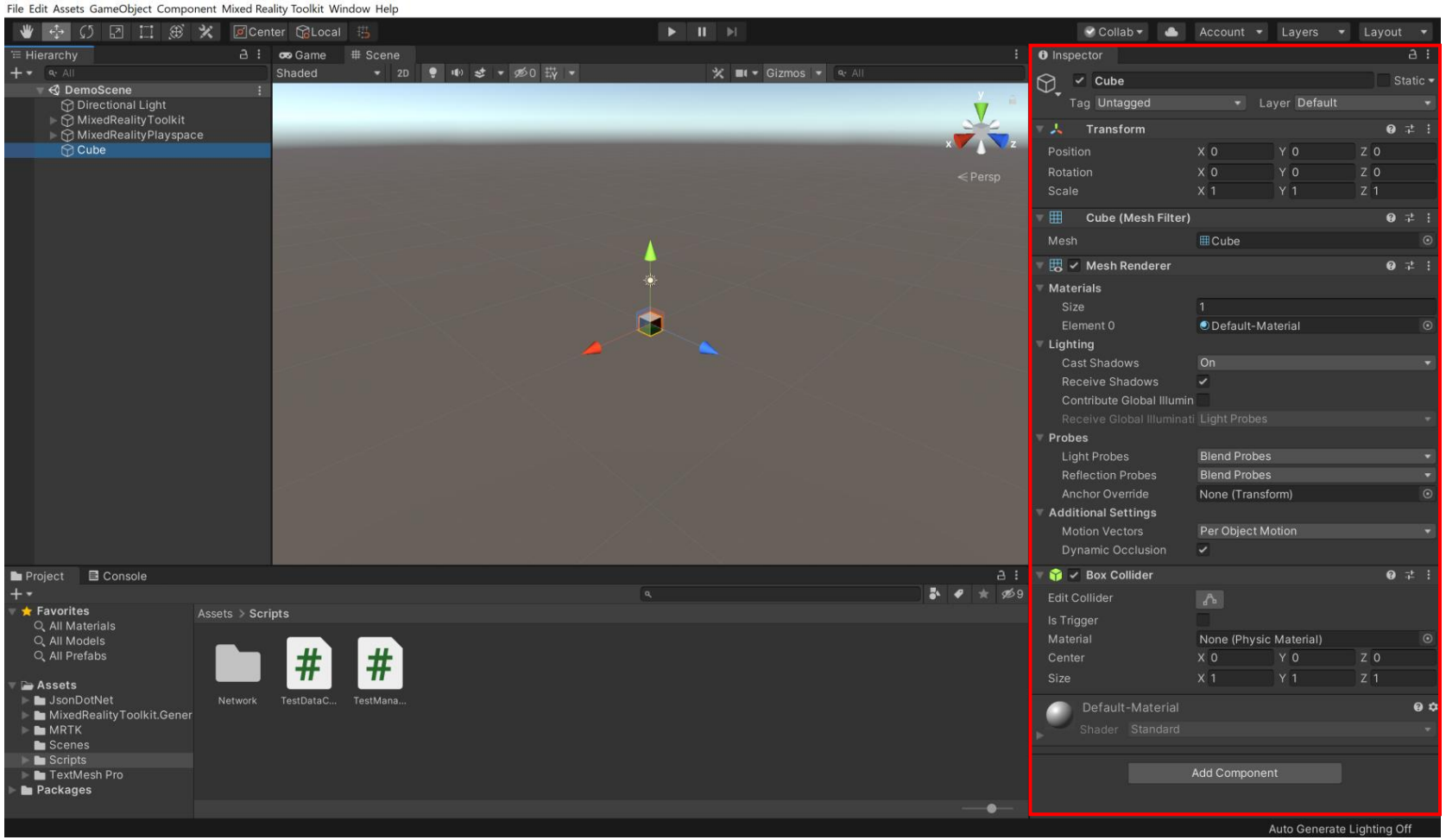
- Auflistung aller GameObjects der Szene (z.B. Kamera, Beleuchtung, Primitive, …)
- Auswahl von Objekten zur Bearbeitung/ Konfiguration



Erste Schritte in Unity

Inspector View:

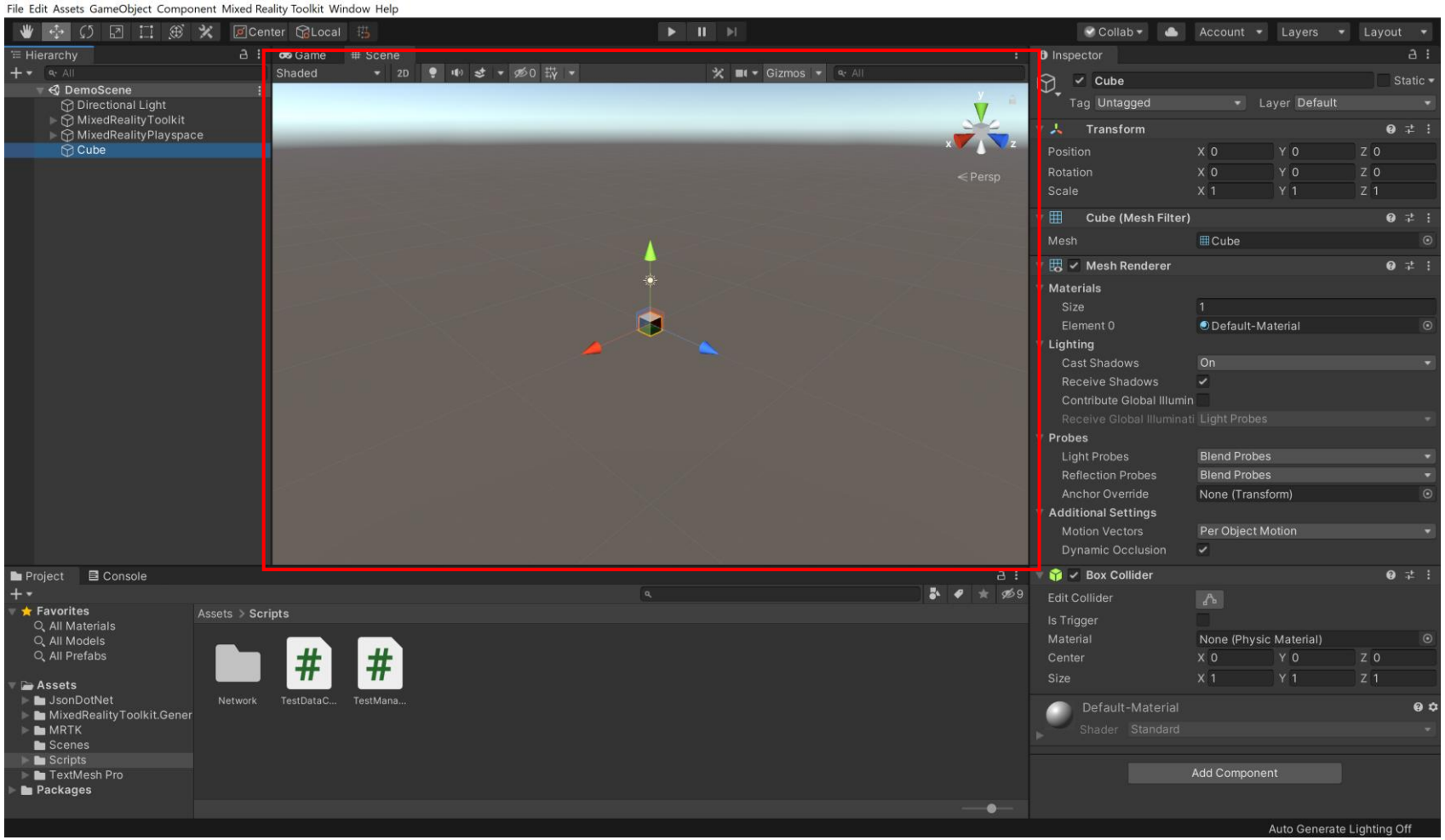
- Alle Eigenschaften des ausgewählten Objekts (Transform, Skripte etc.)
- Änderungen am Objekt über Inspector oder Skript
- Hinzufügen von Skripten über das Inspector-Fenster (Add Component)



Erste Schritte in Unity

Scene/Game View:

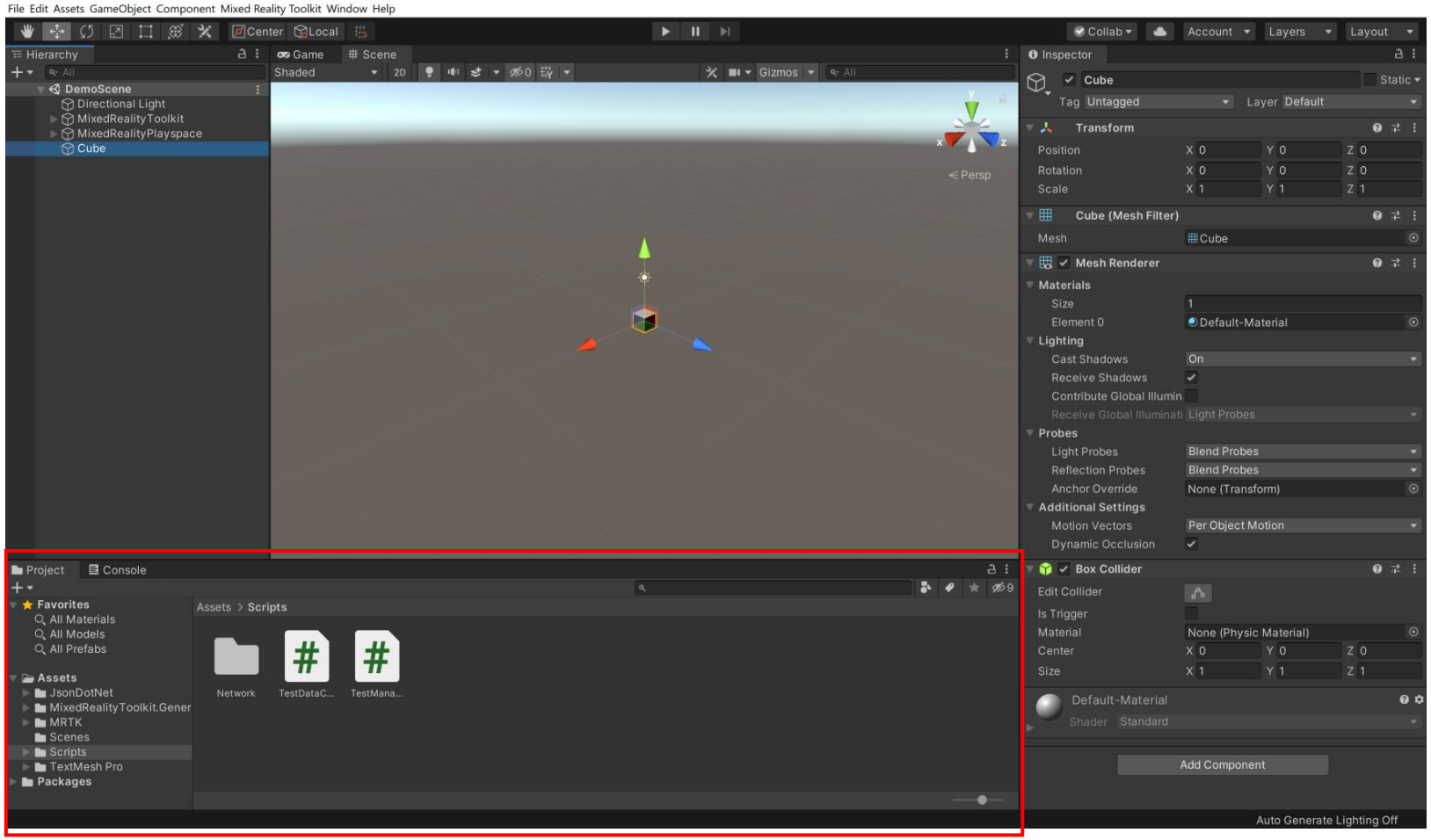
- Vorschau der Szene
- Objekte können im Scene-View transformiert werden
-
- Über „Play“ kann in den Playmode gewechselt werden
- Änderungen, die während aktiviertem Game-Mode durchgeführt werden sind temporär



Erste Schritte in Unity

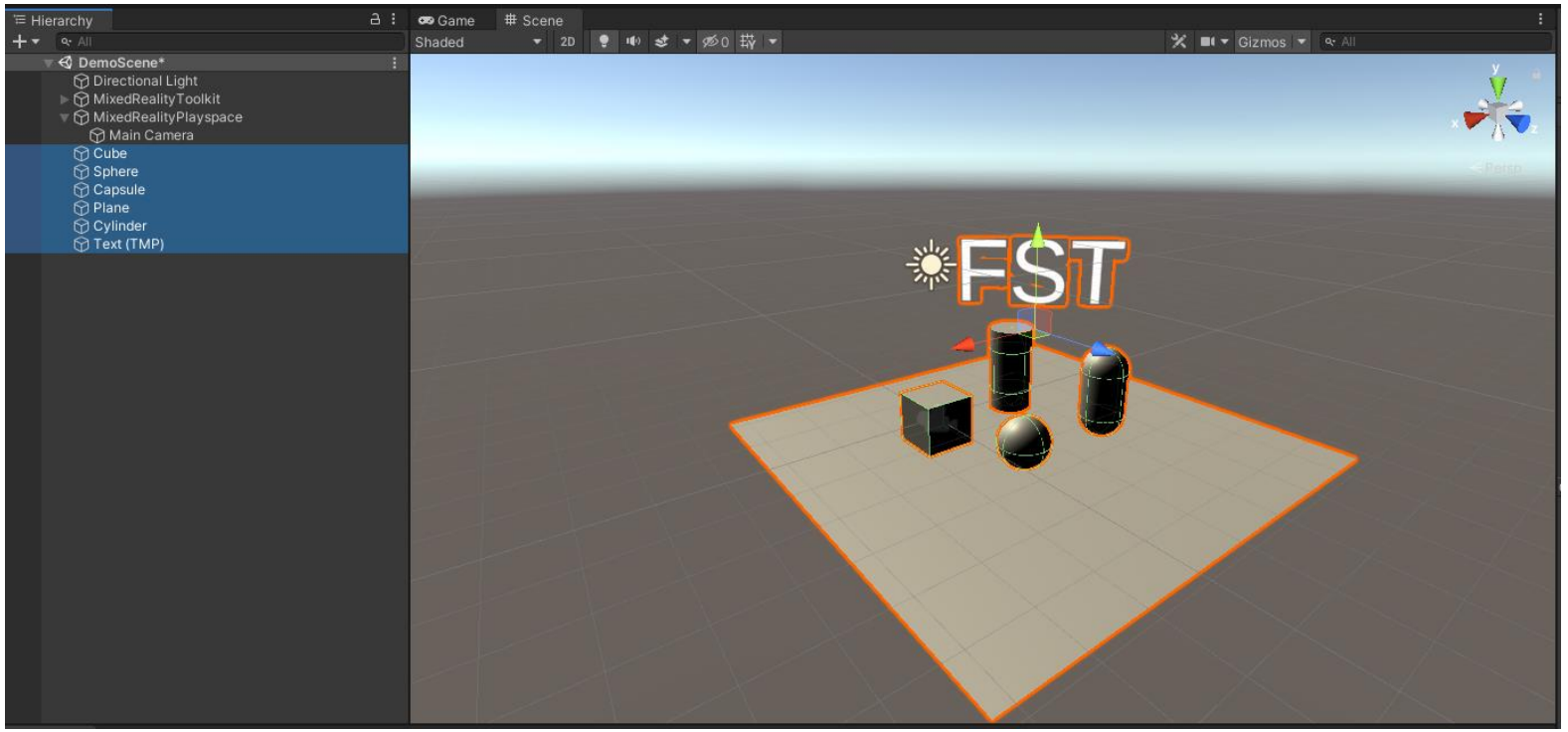
Project/Console View:

- Der Project-View zeigt die Projektstruktur an
- Hier können z.B. neue Skripte angelegt werden
- Der Console-View zeigt die Konsolenausgabe
- Debug.log() gibt Inhalte auf der Konsole aus



Erste Schritte in Unity – GameObjects

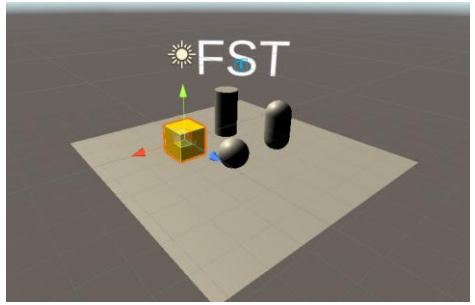
- Jedes Objekt ist ein GameObject (Primitive, 3D-Modelle, Text, Camera, …)
- Standardmäßig wenig Funktionalität, aber Container für Komponenten (z.B. Skripte)
- Eine Einheit von 1 entspricht etwa 1m in der Realität

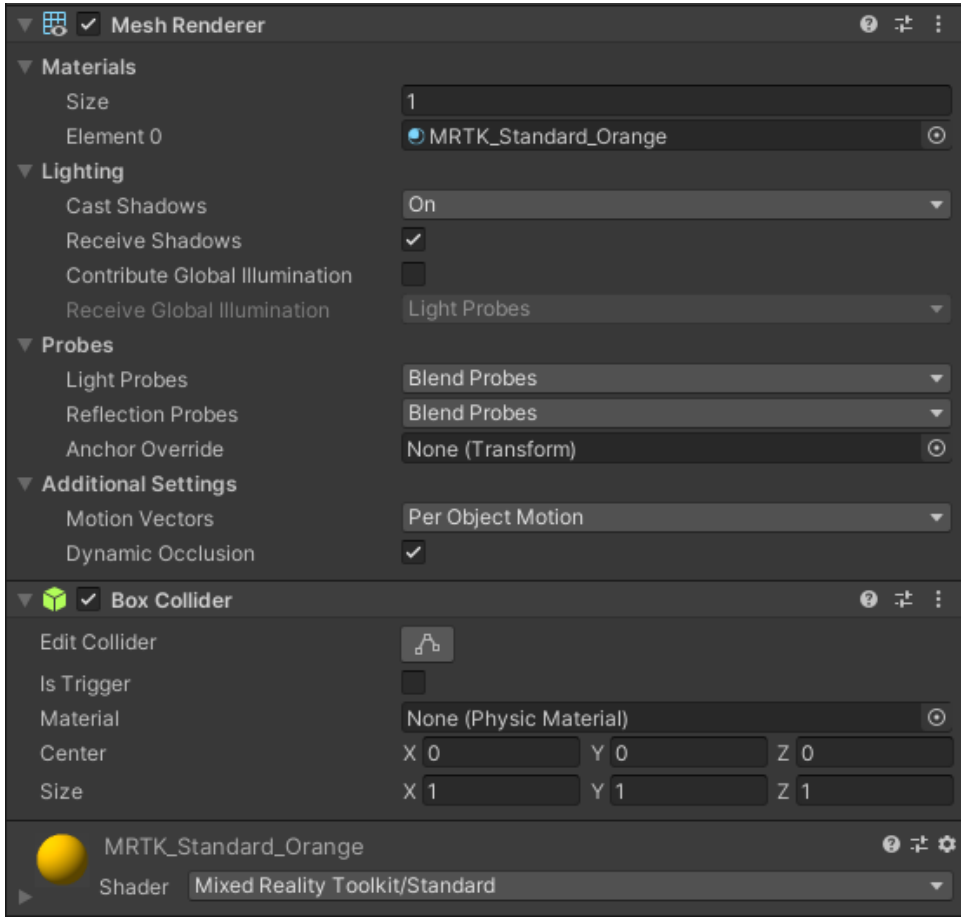


Erste Schritte in Unity – GameObjects

Mesh Renderer und Box Collider:

- Darstellung von Primitiven über Mesh-Renderer
- Hier kann z.B. das Material geändert werden
  =>Das MRTK-Toolkit bringt Standard-Materials mit (MRTK_Standard_<color>)
- Primitive können Box Collider enthalten Dieser definiert die Grenzen des Objekts (wichtig für Interaktionen und Kollision)

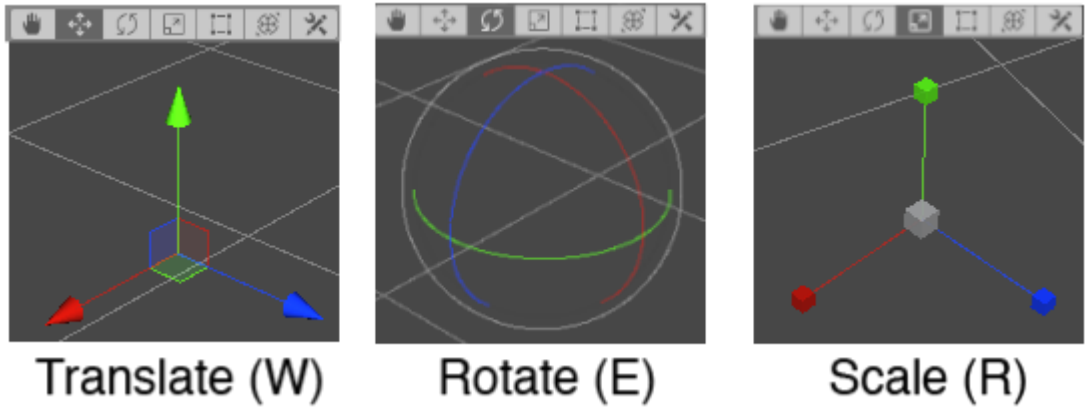


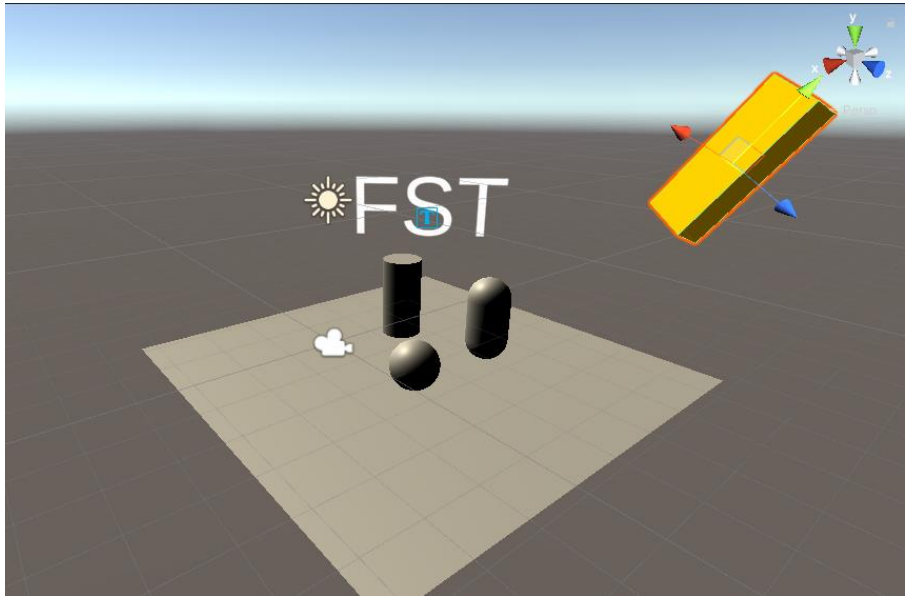


Erste Schritte in Unity – GameObjects

Transform:

- Position: Position im Raum nach X, Y und Z Koordinaten
- Rotation: Rotation um X, Y und Z-Achse in Grad
- Scale: Skalierung entlang der X, Y und Z-Achse (Wert 1 = original Größe)





Erste Schritte in Unity – Skripte

Objekte bewegen:

```
public class CubeMove : MonoBehaviour
{
    // Start is called before the first frame update
    Event function
    void Start()
    {
        Debug.Log( message: "Start rotating");
    }

    // Update is called once per frame
    Event function
    void Update()
    {
        gameObject.transform.Rotate( xAngle: 0, yAngle: 0.2f, zAngle: 0);
    }
}
```

Rotieren

oder

gameObject.transform.eulerAngles(x,y,z);
(Rotate addiert Winkel immer auf den aktuellen Stand, eulerAngles nicht)

```
public class CylinderScale : MonoBehaviour
{

    private bool scaleBack;
    private float scale;

    // Start is called before the first frame update
    Event function
    void Start()
    {
        Debug.Log( message: "Start scaling");
    }

    // Update is called once per frame
    Event function
    void Update()
    {
        if (scaleBack)
        {
            scale -= 0.005f;
        }
        else
        {
            scale += 0.005f;
        }

        if (scale > 4)
        {
            scaleBack = true;
        } else if (scale < 0)
        {
            scaleBack = false;
        }

        gameObject.transform.localScale = new Vector3( x: 1f, y: scale, z: 1f);
    }
}
```

Skalieren

```
public class CubeMove : MonoBehaviour
{

    public GameObject cube;  Unchanged
    private bool moveBack;
    private float move;

    // Start is called before the first frame update
    Event function
    void Start()
    {
        Debug.Log( message: "Start Moving");
    }

    // Update is called once per frame
    Event function
    void Update()
    {
        if (moveBack)
        {
            move -= 0.005f;
        }
        else
        {
            move += 0.005f;
        }

        if (move > 3)
        {
            moveBack = true;
        } else if (move < 0)
        {
            moveBack = false;
        }

        cube.transform.position = new Vector3( x: move, y: 0, z: 0);
    }
}
```

Bewegen

Erste Schritte in Unity – Skripte

Programmatischer Zugriff auf GameObjects:

Laden des GameObjects „Receiver“:

Aus beliebigen Skripten:

```
GameObject.Find(“Receiver”);
```

Mit GameObject „Network“ als parent:

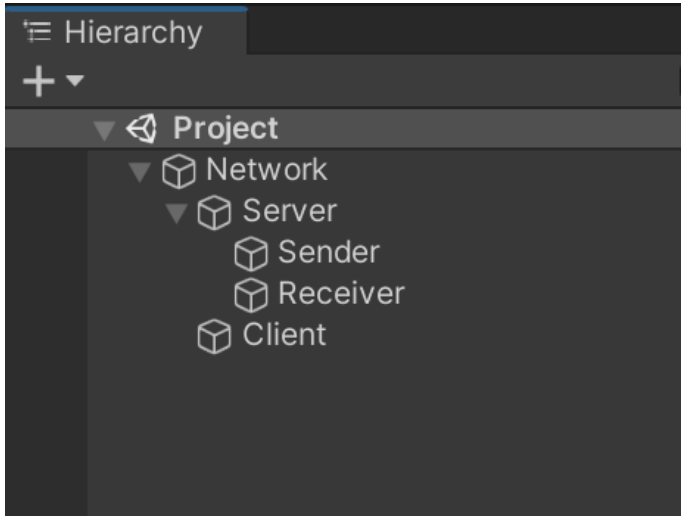
```
Network.transform.GetChild(0).transform.GetChild(1)
```

Server Receiver

oder

```
Network.Find(“Server”).transform.GetChild(1)
```

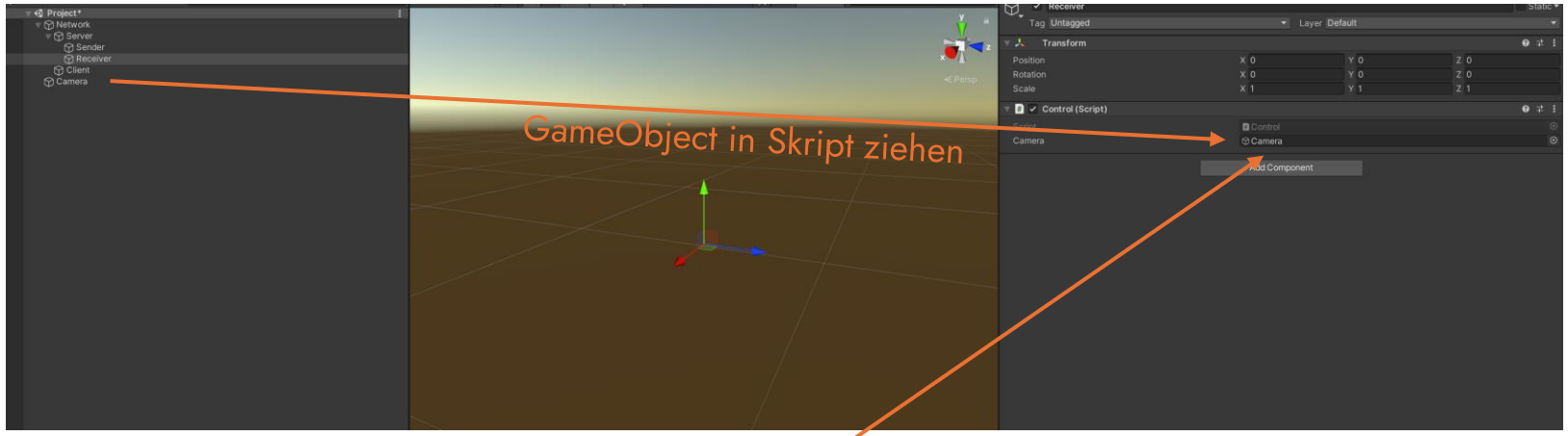
Server Receiver



Parent setzten:

```
Receiver.transform.parent = Server.transform;
```

GameObject Drag & Drop




```
No asset usages
public class Control : MonoBehaviour
{
    public GameObject camera;  Unchanged

    Event function
    private void Start()
    {
        Transform cameraTransform = camera.transform;
    }
}
```

Erste Schritte in Unity – Skripte

MonoBehaviour:

- Basis-Klasse von der jeder Skript mit Unity Funktionalität erben muss
- Stellt standardmäßig u.a. zwei Methoden zur Verfügung
  - ⇒Start(): Wird beim ersten Erzeugen der Klasse aufgerufen (ähnlich wie Konstruktor)
  - ⇒Update(): Wird mit der Berechnung jedes neuen Frames aufgerufen (GameLoop)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CubeMove : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

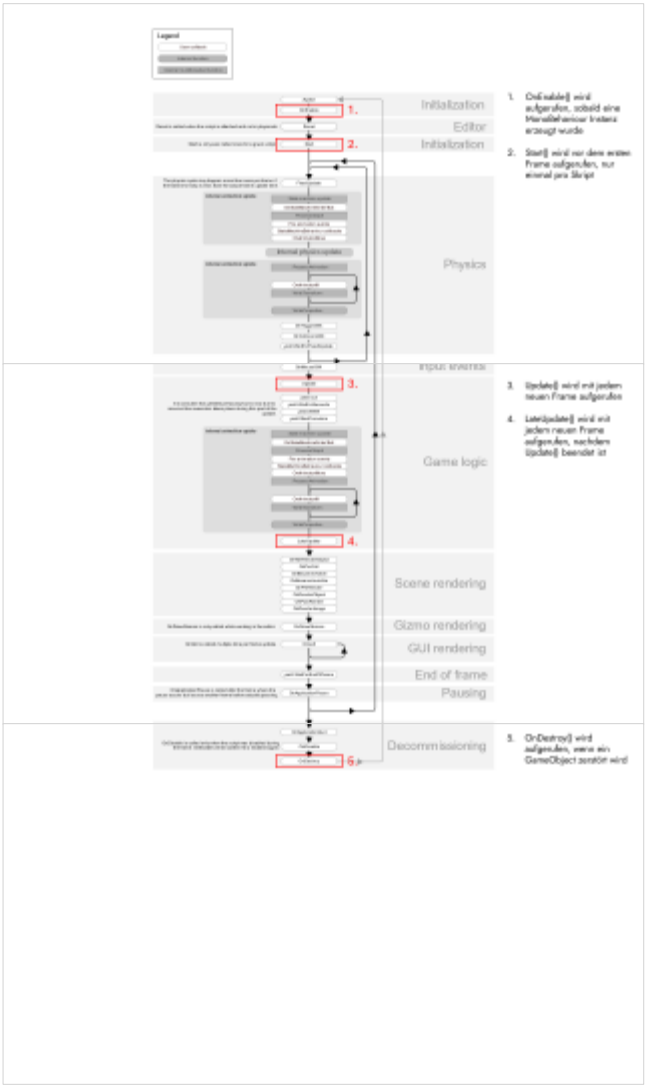
Skripte programmatisch laden, hinzufügen und entfernen:

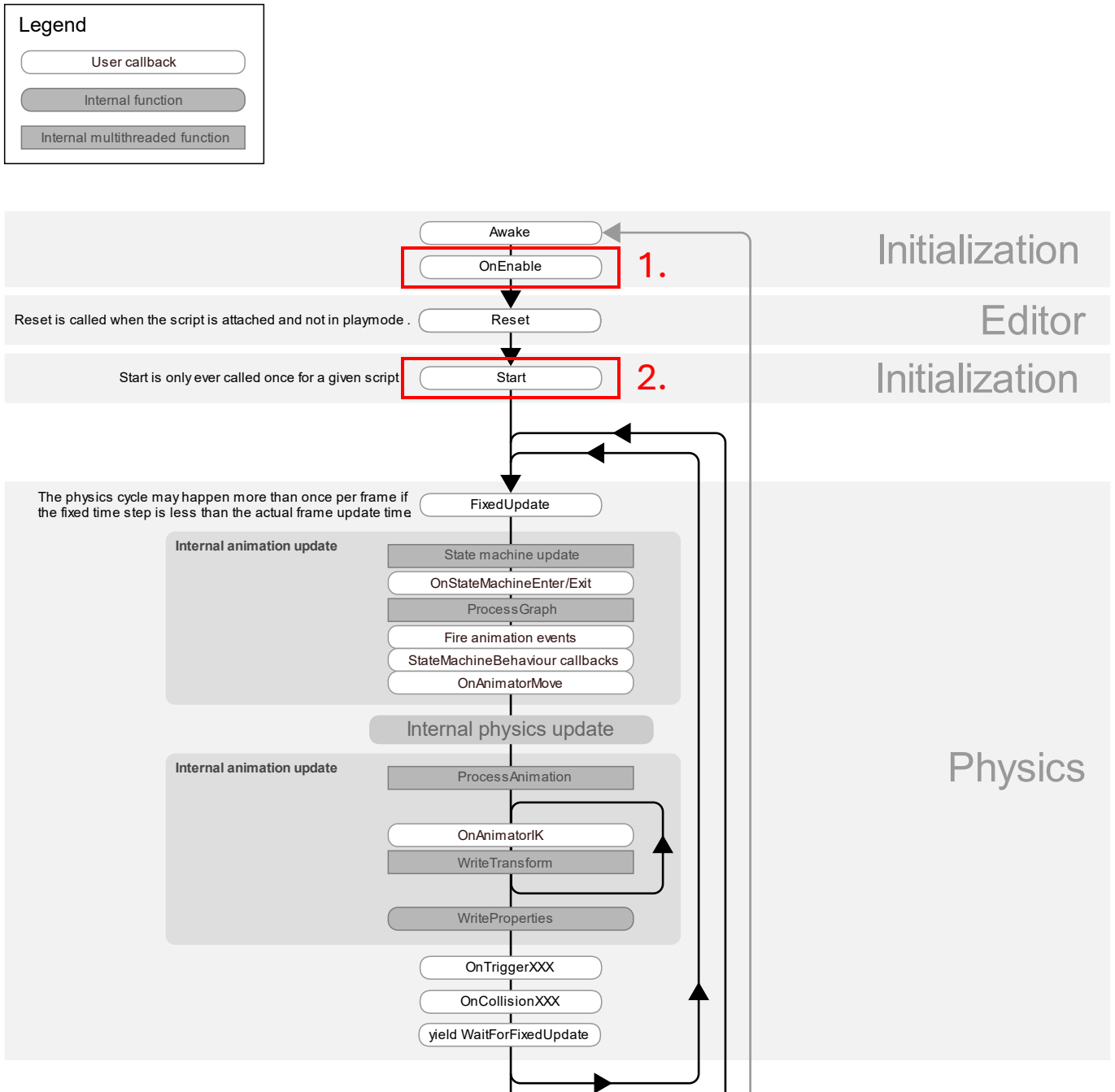
Laden: **`Type component = gameobject.GetComponent<Type>()`**

Hinzufügen: **`Type component = gameobject.AddComponent<Type>()`**

Entfernen: **`Destroy(component)`**

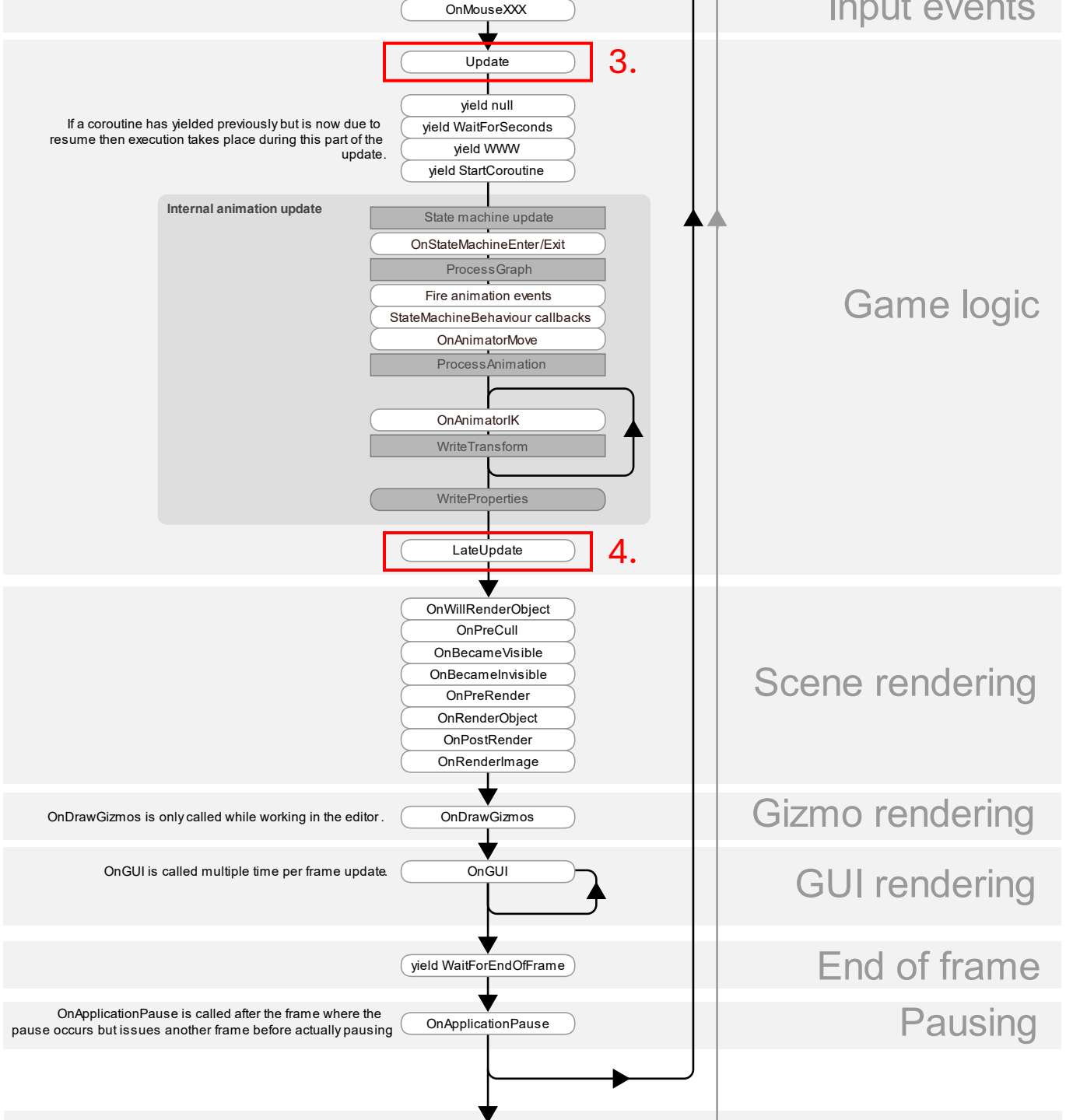
Erste Schritte in Unity – Script Lifecycle





1. OnEnable() wird aufgerufen, sobald eine MonoBehaviour Instanz erzeugt wurde

2. Start() wird vor dem ersten Frame aufgerufen, nur einmal pro Skript



3. Update() wird mit jedem neuen Frame aufgerufen

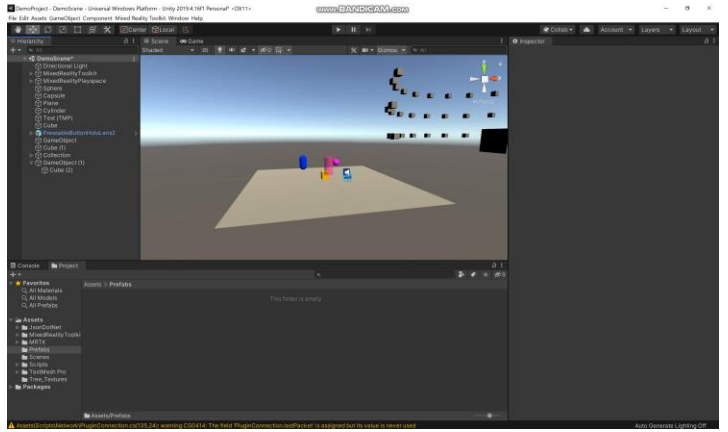
4. LateUpdate() wird mit jedem neuen Frame aufgerufen, nachdem Update() beendet ist

5. OnDestroy() wird aufgerufen, wenn ein GameObject zerstört wird

Erste Schritte in Unity – Prefabs

Prefabs erstellen:

1. Einen Ordner Prefabs im Ordner Assets erstellen (Im Project-View Rechtsklick auf Assets -> Create -> Folder)
2. Im Hierarchy-View ein leeres GameObject erstellen (Rechtklick -> Create Empty)
3. Erstelltes GameObject in den Ordner Prefabs ziehen
4. Doppelklick auf das GameObject
5. Prefab-View öffnet sich
6. Gewünschtes Prefab erstellen



Prefabs nutzen (In Editor):

1. Prefab aus dem Ordner Prefabs in den Hierarchy-View ziehen

Prefabs nutzen (Über Skript):

1. Dem Skript das Prefab übergeben
2. Im Skript mit Instantiate(gameobject) konkrete Instanz erzeugen

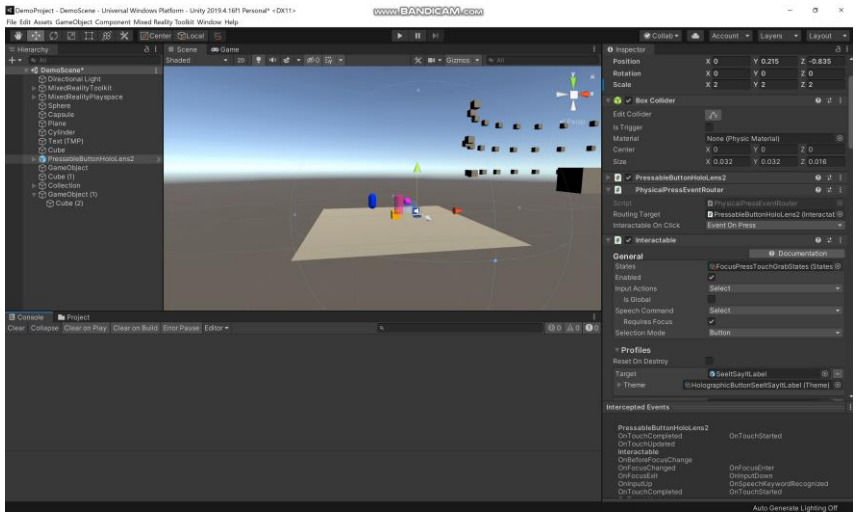
Erste Schritte in Unity – Text

Text erstellen:

1. Im Hierarchy-View Rechtklick -> 3D Object -> Text – TextMeshPro
2. Im Inspector-View Unter „Text“ Text eingeben

Über Skript ändern:

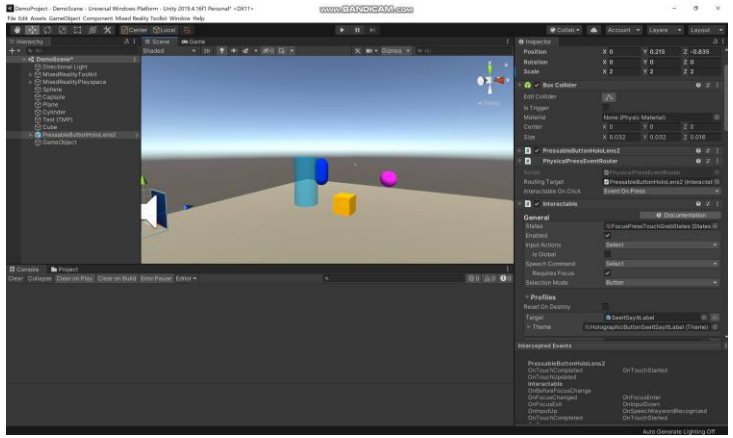
```
public TextMeshPro tmp;

…

tmp.text = „Text“
```



Erste Schritte in Unity – Farben ändern

Farben von Objekten in Unity ändern:

1. GameObject auswählen
2. Im Inspector-View Material ausklappen
3. Albedo-Wert beliebig ändern



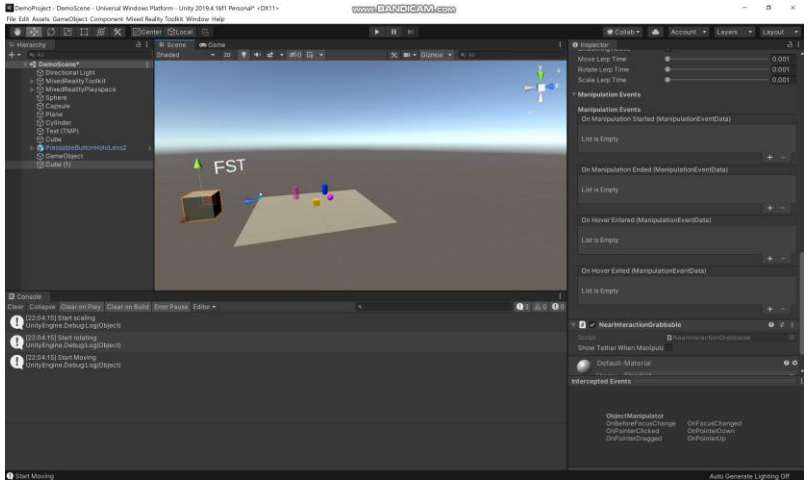
Farben von Objekten über Skripte ändern:

1. Aus dem entsprechenden GameObject die Renderer Komponente laden
2. Auf das Material zugreifen und neue Farbe setzen
   ```
   this.gameObject.GetComponent<Renderer>.material.color = new Color(r, g, b);
   ```
   (jeweils Werte zwischen 0-1)

Erste Schritte in Unity – ObjectCollection

Objekte in Gruppen platzieren:

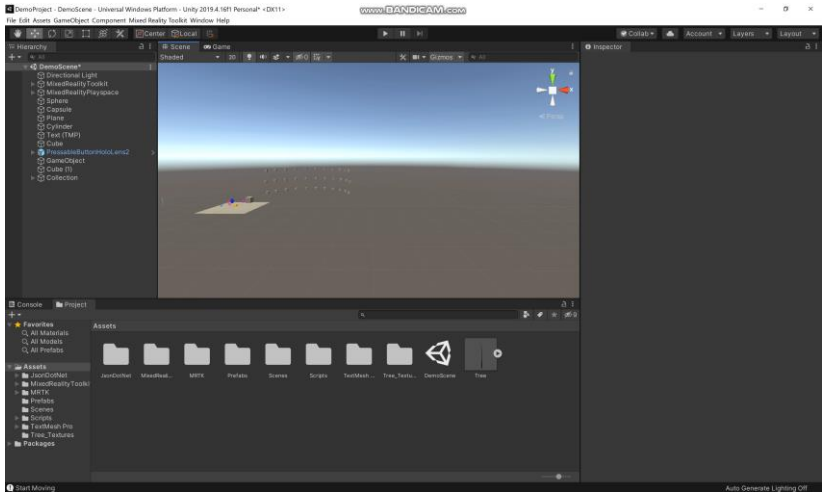
1. Leeres GameObject erstellen und die Komponente „GridObjectCollection“ zuweisen
2. Alle zu gruppierenden Objekte als Kinder dieses GameObjects zuweisen
3. Auf leeres GameObject drücken und im Inspector-View Einstellungen anpassen
4. Im Inspector-View auf „Update Collection“ drücken (Funktioniert auch über Aufruf von Update() im Skript)



Skalierung in eine Richtung

GameObject nur in eine Richtung skalieren:

1. Leeres GameObject erstellen
2. Zu skalierendes GameObject in als Kind des leeren GameObjects setzten
3. Dieses GameObject um 0.5 an der gewünschten Achse verschieben
4. Leeres GameObject skalieren



Erste Schritte in Unity – LineRenderer

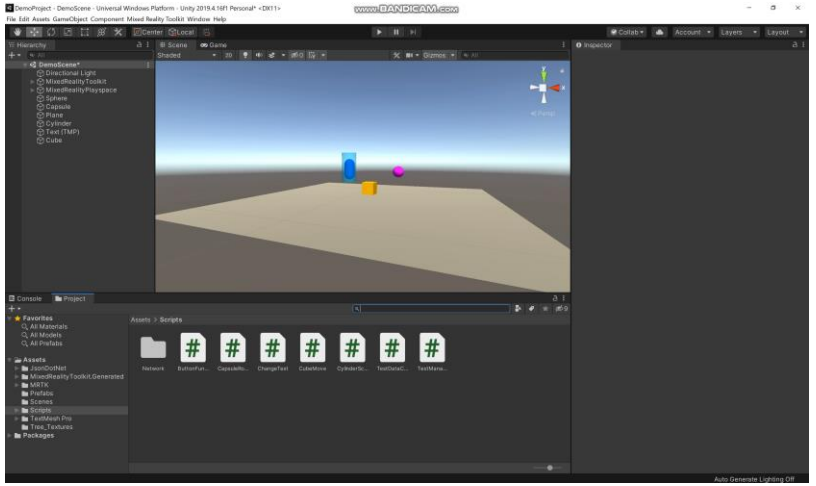
Linien zeichnen:

```
private LineRenderer lineRenderer;

void Start() {
    lineRenderer = gameobject.AddComponent<LineRenderer>();
    lineRenderer.startWidth = 0.1f;
    lineRenderer.endWidth = 0.1f;
    lineRenderer.startColor = Color.red;
    lineRenderer.endColor = Color.red;
    lineRenderer.material = new Material(Shader.Find(„Sprites/Default“));
    lineRenderer.positionCount = 2; //Anzahl an Knoten
}

void Update() {
    lineRenderer.SetPosition(0, Vector3);
    lineRenderer.SetPosition(1, Vector3);
}
```

Knöpfe und Funktionen

Knöpfe erstellen und beliebige Funktionen aufrufen:

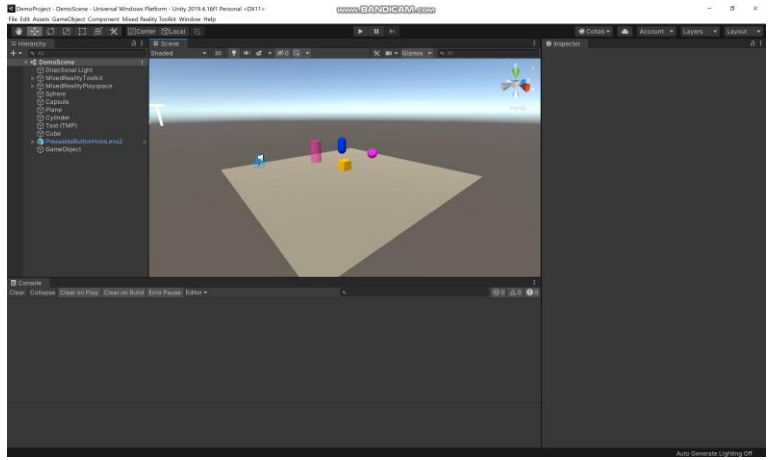
1. Suche im Project-View nach „PressableButtonHoloLens2“
2. Ziehe das Prefab in den Hierarchy-View
3. Skaliere Button je nach Szene auf richtige Größe
4. Im Inspector View unter „Events“ auf + bei onPressed() drücken
5. GameObject, welches entsprechende Funktion beinhaltet, in das Feld Object ziehen
6. Bei „No Function“ gewünschte Funktion auswählen



Objekte greifen und manipulieren

Objekte greifen und manipulieren:

1. Füge eine BoxCollider Komponente hinzu
2. Füge einen Object-Manipulator Komponente hinzu
3. Füge eine NearInteractionGrabbable Komponente hinzu



Erste Schritte in Unity – Steuerung Playmode

- Umschauen: rechte Maustaste
- Bewegen: rechte Maustaste halten, mit W,A,S,D bewegen
- Greifen: Virtuelle Hand anzeigen mit Leertaste, mit Mausrad vor und zurück bewegen, mit linker Maustaste greifen

Hinweis: Alle Änderungen, die während des aktiven Playmodus gemacht werden sind temporär. Für dauerhafte Änderungen den Playmodus immer beenden (nicht pausieren)