

## SEG Bericht

# Omnidirektionaler Hörtest

Helene Lüning (939476), Lukasz Szupka (940167), Robert Rohwer (931300),  
Serdar Sahin (939871), Veronica Zylla (939683)

WiSe 2023/2024

## Inhaltsverzeichnis

<b>I. Bericht</b>	<b>3</b>
<b>1. Exposé</b>	<b>3</b>
<b>2. Lastenheft</b>	<b>4</b>
2.1. Zielbestimmung . . . . .	4
2.2. Produkteinsatz . . . . .	4
2.3. Produktübersicht . . . . .	4
2.4. Produktfunktionen . . . . .	5
2.5. Produktdaten . . . . .	6
2.6. Produktperformanz . . . . .	6
2.7. Qualitätsanforderungen . . . . .	6
<b>3. Systemmodellierung</b>	<b>7</b>
3.1. Verhaltensdiagramm . . . . .	7
3.2. Architekturmuster . . . . .	8
3.3. Klassendiagramm . . . . .	9
<b>4. Entwicklung</b>	<b>11</b>
4.1. Planung . . . . .	11

4.2. Probleme . . . . .	12
4.3. Test-Driven-Development . . . . .	12
<b>5. Teststrategie</b>	<b>15</b>
5.1. Ressourcen . . . . .	15
5.2. Genereller Ablauf . . . . .	15
5.3. Abnahmetests . . . . .	16
5.4. Zeitlicher Ablauf . . . . .	16
5.5. Dokumentation . . . . .	16
<b>6. Reflexion</b>	<b>17</b>
 <b>II. Anhang</b>	 <b>18</b>
<b>A. Testfälle</b>	<b>18</b>
<b>B. Risikoanalyse</b>	<b>19</b>
B.1. Risikomatrix . . . . .	19
B.2. Risiken . . . . .	19
<b>C. Test - Reviews</b>	<b>21</b>
<b>D. Test - Abnahmetest</b>	<b>23</b>

# Teil I.

## Bericht

### 1. Exposé

---

Phasenverantwortlich:	Helene Lüning
Erwarteter Arbeitsaufwand:	3 PT
Tatsächlicher Arbeitsaufwand:	2 PT

---

Das Konzept unseres Projektes ist ein mehrdimensionaler Hörtest mit Hilfe des Sound Domes und der Space Mouse. Ein Ton wird im Sound Dome aus einer beliebigen Richtung abgespielt und die Testperson bewegt die Space Mouse in die Richtung, aus der sie den Ton hört. Wird die richtige Richtung angezeigt, bekommt der Nutzer ein positives auditives Feedback, anderenfalls ein negatives. Die Höhe und/oder Richtung des Tons wird mit jedem Durchlauf verändert, bis der Nutzer den Ton nicht mehr hören kann.

Dies sind die möglichen Teilprobleme und Lösungsideen, die wir uns vorstellen:

- Abfangen des Inputs durch die Maus → Nutzung eines externen Plugins wenn möglich, alternativ Erstellung eines eigenen Plugins, Darstellung als OSC-Signal
- Generierung der Test Sounds → Synthese durch java.sound
- Bestimmung der Testrichtung → Umrechnung der OSC-Signale anderer Gruppen in Winkel als Zufallsgenerator
- Abgleichen des Inputs mit der Richtung des Sounds und Feedback → Entwurf und Implementierung eines Steuerungsprogrammes inklusive Kalibrierung

Als Aufwandsabschätzung für die Analysephase haben wir drei Personentage abgeschätzt. (1 PT für Exposé, 2 PT für Lastenheft inklusive Anhang)

## 2. Lastenheft

### 2.1. Zielbestimmung

Die Firma EvilCorps wird das Produkt zur Durchführung von mehrdimensionalen Tests der Hörschärfe nutzen können. Durch die Erweiterung eines klassischen Hörtests um eine spielerische Komponente wird eine jüngere Zielgruppe dazu bewegt werden ihr Hörvermögen zu testen.

### 2.2. Produkteinsatz

Das Produkt wird dafür genutzt werden Hörtests im Sound Dome durchzuführen. Zum Einsatz des Produkts ist darüberhinaus eine Spacemouse zum Angeben der Richtung, aus welcher die Töne abgespielt werden, notwendig.

Zielgruppe des Produktes sind alle an dem Testen ihres Gehörs interessierten Personen, insbesondere aber junge Menschen.

### 2.3. Produktübersicht

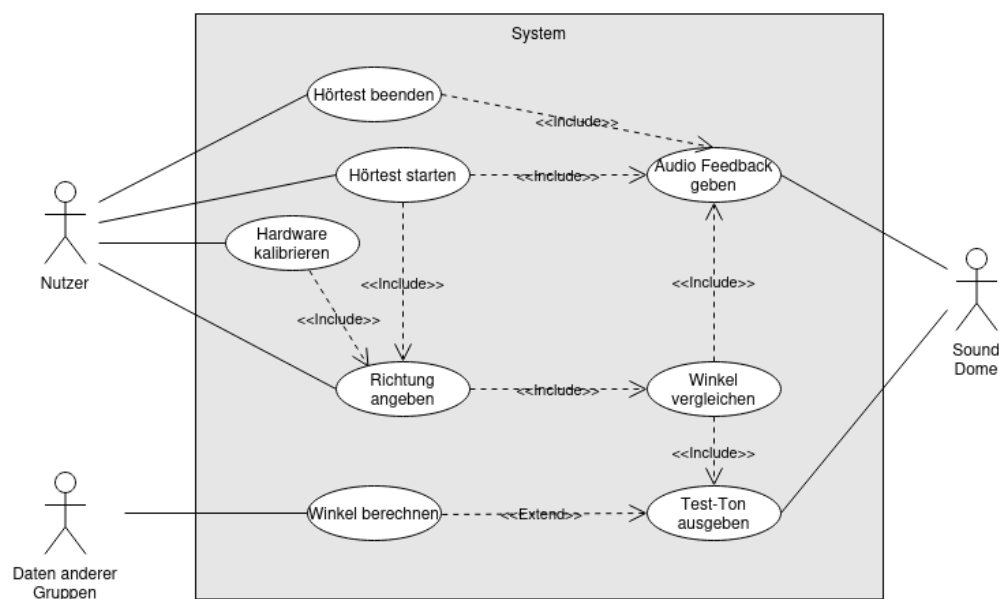


Abbildung 1: Use-Case-Diagramm

## 2.4. Produktfunktionen

/AF 10/	Name:	Hardware kalibrieren
	Akteure:	Nutzer, Sound Dome
	Beschreibung:	Um die Abstimmung der Maus und des Domes aufeinander während des Tests zu gewährleisten, wird vor Beginn das System kalibriert
/AF 20/	Name:	Hörtest starten
	Akteure:	Nutzer, Sound Dome
	Beschreibung:	Der Nutzer startet den Hörtest
/AF 30/	Name:	Richtung angeben
	Akteure:	Nutzer, Sound Dome
	Beschreibung:	Reaktion des Nutzers auf einen Testton
/AF 40/	Name:	Winkel vergleichen
	Akteur:	Sound Dome
	Beschreibung:	Der vom Nutzer angegebene Winkel wird mit dem, aus dem der Testton kam, verglichen und das Ergebnis verarbeitet
/AF 50/	Name:	Test-Ton ausgeben
	Akteure:	Sound Dome, Daten anderer Gruppen
	Beschreibung:	Der Sound Dome gibt einen Ton mit einer gewissen Frequenz aus einer gewissen Richtung aus
/AF 60/	Name:	Winkel berechnen
	Akteur:	Daten anderer Gruppen
	Beschreibung:	Aus den OSC-Signalen einer anderen Gruppe wird ein Winkel berechnet
/AF 70/	Name:	Audio Feedback geben
	Akteur:	Sound Dome
	Beschreibung:	Ein Auditives Feedback wird durch den Sound Dome gegeben
/AF 80/	Name:	Hörtest beenden
	Akteure:	Nutzer, Sound Dome
	Beschreibung:	Der Benutzer möchte den Hörtest abbrechen

Es wurden zusätzlich die folgenden optionalen Produktfunktionen spezifiziert:

/AF-O 510/	Name:	Visuelles Feedback
	Akteur:	Nutzer, Sound Dome
	Beschreibung:	Während des Tests sollen Live Ergebnisse des Tests angezeigt werden.
/AF-O 520/	Name:	Speichern von Testergebnissen
	Akteur:	Nutzer
	Beschreibung:	Die Ergebnisse des Tests werden persistent als json gespeichert

/AF-O 530/	Name:	Dreidimensionalen Test-Ton ausgeben
	Akteure:	Sound Dome, Daten anderer Gruppen
	Beschreibung:	Der Sound Dome gibt einen Ton mit einer gewissen Frequenz an einem gewissen Ort im Raum platziert aus
/AF-O 540/	Name:	Platzierung im Raum berechnen
	Akteure:	Daten anderer Gruppen
	Beschreibung:	Aus den OSC-Signalen einer anderen Gruppe wird ein Winkel berechnet
/AF-O 550/	Name:	Position vergleichen
	Akteure:	Sound Dome
	Beschreibung:	Die Position des Zeigertons wird mit der des Testtons verglichen

## 2.5. Produktdaten

Folgende Daten werden während des Programmdurchlaufs gespeichert:

/AD 10/	Fehleranzahl (maximal 100)
/AD 20/	zuletzt abgespielte Tonhöhe
/AD 30/	zuletzt verwendeter Winkel

Die persistente Speicherung dieser Daten über den Programmdurchlauf hinaus sei dabei als optional definiert.

## 2.6. Produktperformanz

/AP 10/	Die Laufzeit der Funktionen AF60 und AF40 liegt unter 10ms
/AP 20/	Das Abspielen der Funktion AF70 liegt unter 30ms
/AP 30/	Die Funktions Laufzeit von AF10 liegt unter 60ms
/AP 40/	Die Funktion AF30 liegt unter 5ms

## 2.7. Qualitätsanforderungen

Produktqualität	sehr gut	gut	normal	nicht relevant
Funktionalität		x		
Zuverlässigkeit		x		
Effizienz		x		
Änderbarkeit			x	
Übertragbarkeit				x

### 3. Systemmodellierung

---

Phasenverantwortlich:	Lukasz Szupka
Erwarteter Arbeitsaufwand:	7 PT
Tatsächlicher Arbeitsaufwand:	9 PT

---

Zu Beginn der Systemmodellierung haben wir im Hinblick auf die Entwicklungserfordernisse und zur Gewährleistung einer übersichtlichen und sinnvollen Architektur, die Anwendung in zwei Teile aufgeteilt: Die Hauptanwendung und eine Hilfsanwendung zur Ansteuerung der Spacemouse. Diese Aufteilung ist auch unter Berücksichtigung des, für das Erarbeiten des Gesamtszenarios notwendigen, Informationsaustausches mit anderen Gruppen sinnvoll.

Aufbauend auf den im Lastenheft spezifizierten Anforderungen haben wir anschließend eine objekt-orientierte Analyse durchgeführt, um eine Strukturierung des Systems zu planen und zu visualisieren. Dabei haben wir uns für eine verhaltensorientierte Vorgehensweise entschieden; zur optimalen Darstellung der Abläufe unseres Systems haben wir uns daher dem Konzept des Object-Oriented Software Engineerings (OOSE) bedient.

#### 3.1. Verhaltensdiagramm

In der Erstellung unseres Grobentwurfes haben wir zur Modellierung des Verhaltens und der Abläufe in unserem System zunächst ein Aktivitätsdiagramm (Abb. 2) erarbeitet. Dieses Aktivitätsdiagramm orientiert sich an den im Lastenheft spezifizierten Produktfunktionen. Nicht enthalten sind dabei optionale Produktfunktionen, unabhängig davon, ob diese im Verlauf des Projektes umgesetzt wurden.

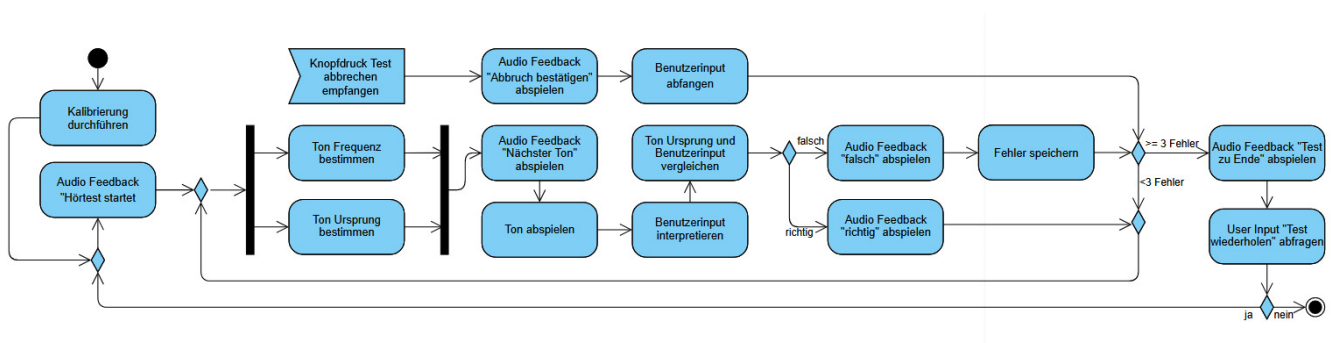


Abbildung 2: Aktivitätsdiagramm

## 3.2. Architekturmuster

Um nach der Analyse des Verhaltens in unserem System nun geeignete Klassenkandidaten zu finden, haben wir zunächst verschiedene Architekturmuster evaluiert, welche bei der Strukturierung unseres Systems assistieren.

Wir haben dabei entschieden, eine effizientere Entwicklung und bessere Skalierbarkeit sowie Wartbarkeit durch die Nutzung einer Model-View-Controller-Architektur zu gewährleisten. Konkret haben wir uns für die Nutzung der Model-View-Presenter-Variante entschieden welche im Vergleich zu einer klassischen Model-View-Controller-Architektur die Testbarkeit und Modularität weiter steigert. Nachdem im weiteren Verlauf der Objektorientierten Analyse über die Identifikation von Klassenkandidaten und den entsprechenden Klassenrollen die Klassen für unser System grob definiert wurden, haben wir ein Diagramm (Abb. 3) erstellt, welches zur Veranschaulichung der MVP-Architektur in unserem konkreten Projekt dient.

Bezüglich der Verarbeitung und Weiterleitung von OSC-Signalen haben wir uns für die Nutzung einer Event-Driven-Architektur entschieden (siehe Abb. 4 auf der nächsten Seite), welche die Reaktionsfähigkeit und die Erweiterbarkeit des Systems verbessert.

Während im Hauptprogramm sowohl die Event-Driven-Architektur als auch die Model-View-Controller-Architektur für bessere Erweiterbarkeit sorgen, wird eine gesteigerte Erweiterbarkeit aber auch eine erhöhte Wiederverwendbarkeit und Flexibilität des Systems im Programm zum Ansteuern der Space mouse durch eine Pipe-Filter-Architektur (siehe Abb. 5 auf der nächsten Seite) sichergestellt.

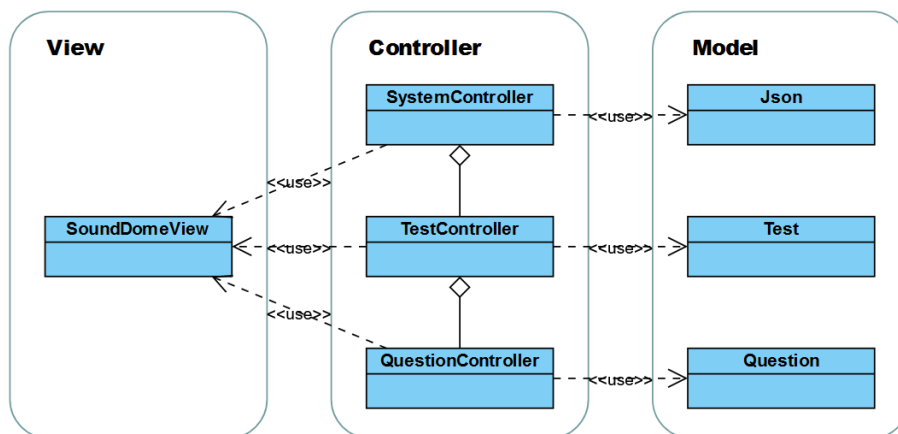


Abbildung 3: Model-View-Controller-Architektur Diagramm



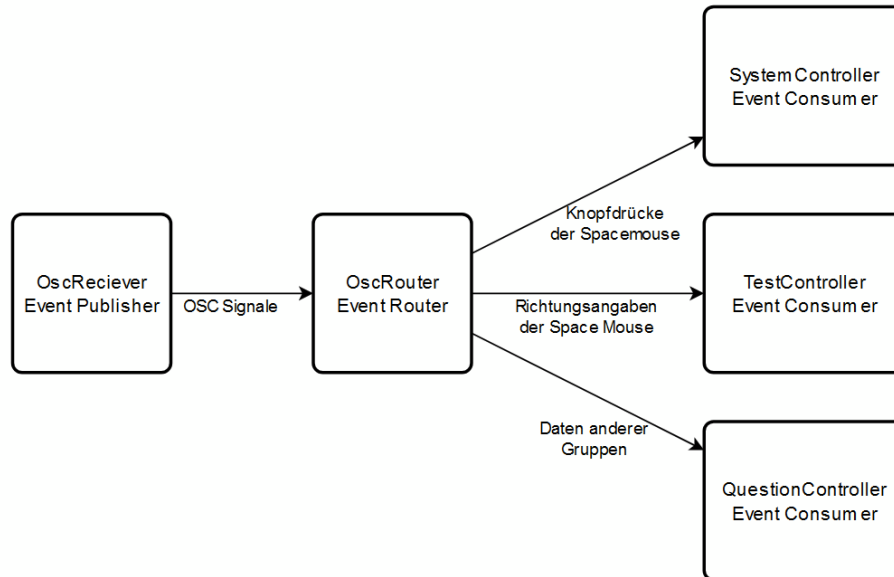


Abbildung 4: Event-Driven-Architektur Diagramm

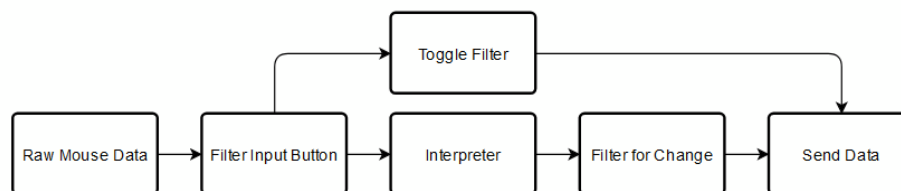


Abbildung 5: Pipe-Filter-Architektur Diagramm

### 3.3. Klassendiagramm

Im Hinblick auf die Erstellung eines ersten Grobentwurfes unseres Klassendiagramms haben wir uns an der Vorgehensweise des Object-Oriented Software Engineerings orientiert. So haben wir zunächst Klassenkandidaten aus unseren im Zuge der Analysephase erstellten Anwendungsfällen abgeleitet und anschließend Assoziationen und Aggregationen zur Verknüpfung dieser Klassenkandidaten gewählt. Nach der Fertigstellung des Grobentwurfes unseres Klassendiagramms haben wir dieses im Feinentwurf durch das Hinzufügen von Attributen und Vererbungsbeziehungen detailliert und abschließend in Pakete gekapselt. Während des gesamten Prozesses haben wir stets die bei der Erstellung des Aktivitätsdiagramms gewonnen Erkenntnisse sowie die grobe Struktur, welche durch die von uns gewählten Architekturmuster bereits festgelegt war, im Blick behalten. Im Klassendiagramm sind die im Lastenheft spezifizierten Produktfunktionen AF 10 bis AF 80, sowie die in der Implementierungsphase umgesetzte optionale Produktfunktion AF-O 520 repräsentiert.

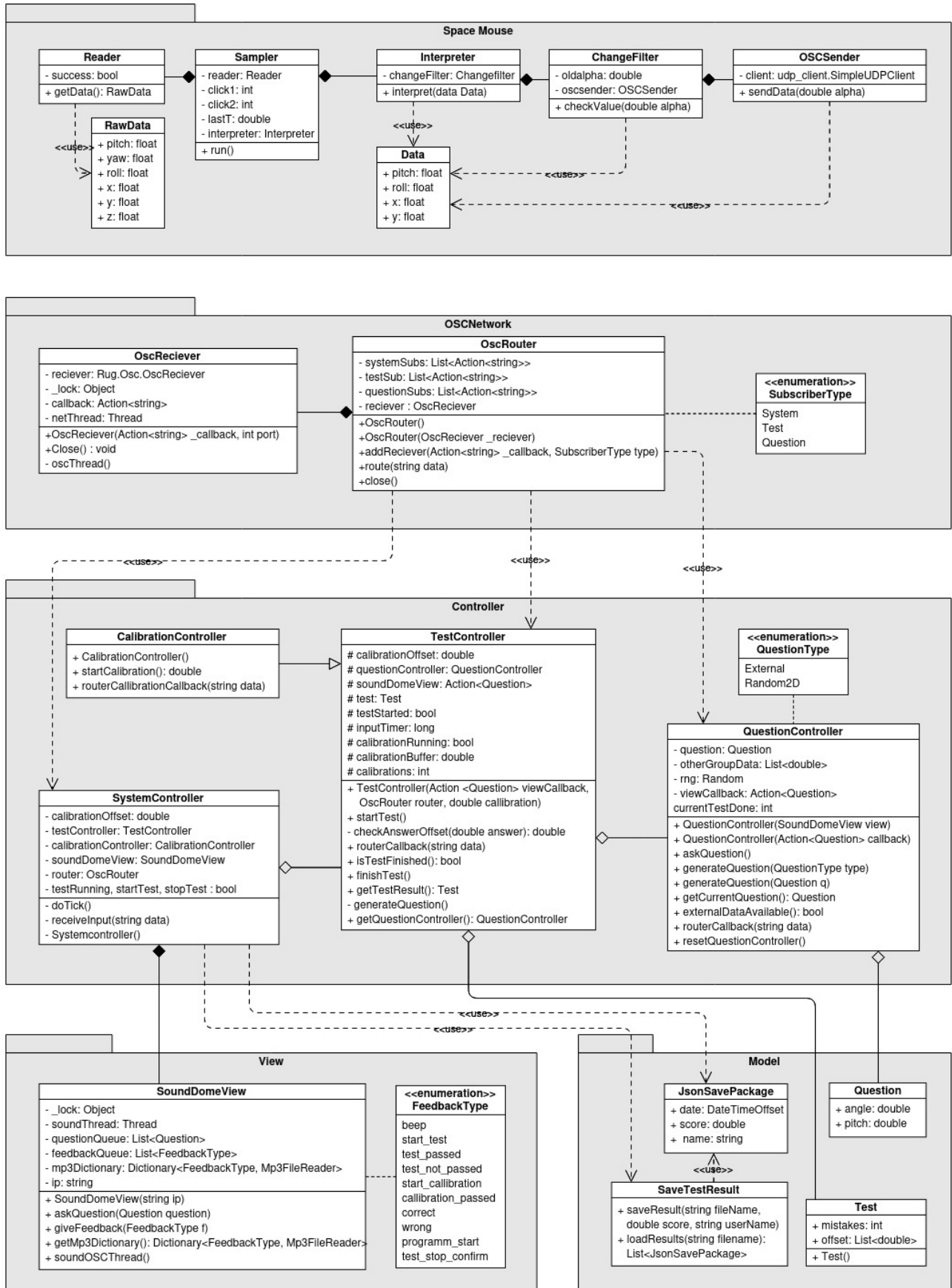


Abbildung 6: Klassendiagramm

## 4. Entwicklung

---

Phasenverantwortlich:	Robert Rohwer
Erwarteter Arbeitsaufwand:	20 PT
Tatsächlicher Arbeitsaufwand:	25 PT

---

### 4.1. Planung

---

Datum	Task-ID	Bericht	Geschätzt [h]	Tatsächlich [h]
17.10.2023	1	Das Programm erhält Daten von der SpaceMouse	4	12
17.10.2023 – 19.10.2023	2	Die Daten werden interpretiert und umgerechnet	5	5
19.10.2023 – 21.10.2023	3	Die Daten werden in OSC umgewandelt und an der OSC-Receiver versendet	5	7
21.10.2023 – 27.10.2023	4	Das Python Programm wird fertiggestellt	14	35
14.11.2023	5	Das Python Programm wird im Sinne der Struktur und Architektur überarbeitet	7	11
23.11.2023	6	Das C# Programm wird aufgesetzt und die Entwicklung beginnt	1	1
23.11.2023	7	Der Oscrouter und Oscreceiver werden implementiert	5	7
23.11.2023 – 27.11.2023	8	Der TestController wird implementiert	8	12
27.11.2023	9	Der QuestionController wird implementiert	4	4
27.11.2023 – 02.12.2023	10	Der SoundDomeView wird implementiert	9	10
02.12.2023 – 05.12.2023	11	Der Graphische View wird implementiert	7	7
05.12.2023 – 07.12.2023	12	Die Testdaten werden als Json gespeichert	6	5
07.12.2023 – 18.12.2023	13	Merges und bugfixes	25	25

---

Die in der Tabelle dargestellten Aufgaben haben wir im Anschluss in einem Gantt-Chart visualisiert.

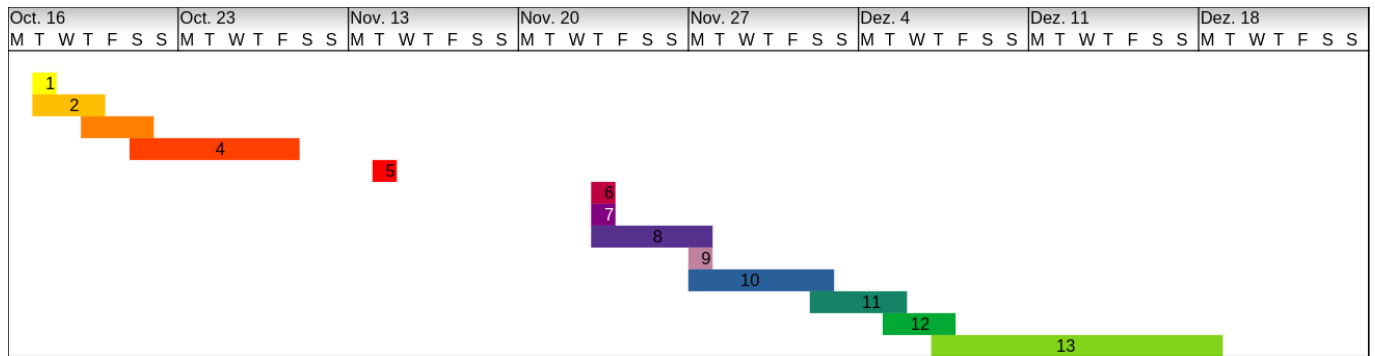


Abbildung 7: Gantt-Chart

## 4.2. Probleme

Im Verlauf der Implementierung sind wir auf vier große Hürden gestoßen:

Die erste dieser Hürden begegnete uns direkt zu Beginn, da wir es versäumten, bei der Erstellung des Repositories direkt eine vollständige gitignore einzuchecken, bevor erste Branches erstellt wurden. Dadurch entstanden im weiteren Verlauf mehrfach Merge-Konflikte durch automatisch generierte Dateien, die jeweils manuell aufgelöst werden mussten. Durch die Erstellung einer gitignore, welche alle generierten Dateien berücksichtigt, konnten wir unsere Produktivität bei der Zusammenführung verschiedener Branches spürbar steigern.

Weiterhin hatten wir Protokollprobleme mit dem ADM-Simulator, die auf Grund von mangelnder Dokumentation entstanden, sowie Komplikationen mit Race Conditions beim Multithreading.

Auch die Audioausgabe stellte zunächst eine Hürde dar. Mit der Nutzung separater Threads konnten wir hier jedoch das Problem nach einiger Zeit sinnvoll bewältigen.

## 4.3. Test-Driven-Development

Da die Anwendung nach TDD entwickelt wurde, war die Entwicklung von Unit-Tests Teil der Entwicklungsphase. Im Folgenden betrachten wir daher die Entwicklung der Unit-Tests nach TDD.

### Ablauf

Als Framework für die C# Unit Test wurde Nunit genutzt, für die Python Tests wurde PyTest genutzt. Für die Wahl der Tests haben die Entwickler die wichtigsten Testfälle und Anforderungen aus dem Lastenheft analysiert und priorisiert. Außerdem wurden neben den im Lastenheft definierten

Tests noch einige Grenzwert Testfälle berücksichtigt. Nach dem die Unit Tests erfolgreich durchgelaufen sind, wurde im Refactoring Prozess der Code noch weiter angepasst und z.B. Error Handling ergänzt.

Ein Beispiel für solch eine Anforderung und des dazu gehörigen Testes sind unten zu entnehmen:

Anforderung: "Eine Frage soll gestellt werden": Test-Ton ausgeben

## Unit-Test

Das Bestehen, der Anforderung wird einmal durch den Unittest selbst bestätigt, aber auch durch das Audio Feedback.

Test	Dauer	Merkmale	Fehlermeldung	Gruppenzusammenfassung
✓ VerarbeitungTest (21)	1 min			VerarbeitungTest.Tests
✓ VerarbeitungTest.Tests (21)	1 min			Tests in Gruppe: 21
✓ CalibrationControllerTest (1)	32,3 Sek.			🕒 Dauer gesamt: 1 min
✓ OscReceiverTest (1)	1 Sek.			Ergebnisse
✓ OscRouterAddRecieverTest (3)	1 ms			✓ 21 Bestanden
✓ QuestionControllerTest (7)	2 ms			
✓ SoundDomeViewTest (3)	13,1 Sek.			
✓ Systemcontroller (2)	10 ms			
✓ TestControllerTest (2)	16,1 Sek.			
✓ TestTest (2)	< 1 ms			

## Testabdeckung

Als Ziel für die Testabdeckung wurde ein Wert zwischen 60-80

```
namespace HelloWorld
{
    2 references
    public class Program
    {
        1 reference | 🟢 1/1 passing
        public static void Main()
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Für die Errechnung der Testabdeckung haben wir die Anzahl der getesteten Methoden durch die Gesamtanzahl der Methoden einer Klasse dividiert und mit 100 multipliziert. Daraus ergaben sich folgende Werte:

- QuestionController: 56%

- TestController: 100%
- SoundDomeView: 60%
- Test: 100%
- OSCReceiver: 100%
- OSCRouter: 75%
- SaveTestResult: 100%
- SystemController: 65%
- CalibrationController: 100%

**Durchschnittliche Abdeckung: 84%**

Unser Ziel für die Testabdeckung wurde erreicht. Dies haben wir durch das TDD erreicht, mit den geschriebenen Unittest, aber auch mit den Funktionstests.

```
[TestCase("input:180", 180)]
[TestCase("input:0.5", 180)]
[TestCase("input:0,5", 180)]
0 | 0 Verweise
public void TestOscQuestionGeneration(String osc, int angle)
{
    controller.routerCallback(osc);
    controller.generateQuestion(QuestionController.QuestionType.External);

    Assert.Multiple(() => {
        Assert.That(controller.getCurrentQuestion(), Is.Not.Null);
        Assert.That(Math.Round(controller.getCurrentQuestion().angle), Is.EqualTo(angle));
        Assert.That(controller.getCurrentQuestion().pitch, Is.GreaterThan(0));
    });
}

[Test]
0 | 0 Verweise
public void TestOscQuestionGeneration()
{
    controller.routerCallback("input:1aa8s0");
    controller.generateQuestion(QuestionController.QuestionType.External);

    Assert.Multiple(() => {
        Assert.That(controller.getCurrentQuestion(), Is.Not.Null);
        Assert.That(Math.Round(controller.getCurrentQuestion().angle), Is.EqualTo(0));
        Assert.That(controller.getCurrentQuestion().pitch, Is.Zero);
    });
}
```

## Schlusswort

Zusammenfassend lässt sich feststellen, dass die Tests eine effizientere Entwicklung ermöglichen, indem wir rasch überprüfen können, ob neue Änderungen keine Probleme verursachen. Insgesamt haben die Unit-Tests unsere Software robuster und zuverlässiger gemacht.

## 5. Teststrategie

---

Phasenverantwortlich:	Serdar Sahin
Erwarteter Arbeitsaufwand:	8 PT
Tatsächlicher Arbeitsaufwand:	7 PT

---

In diesem Kapitel betrachten wir unsere Teststrategie. Die zugehörigen Testfälle sind in Anhang **A** auf Seite **18** und die Risikoanalyse in Anhang **B** auf Seite **19** zu finden.

### 5.1. Ressourcen

Menschliche Ressourcen:

**Testmanager:** Serdar Sahin

**Testanalyst 1:** Lukasz Szupka

**Testanalyst 2:** Robert Rohwer

Hardwareressourcen:

- IDE, die Python ausführen kann
- Space Mouse
- Spezielle Testgeräte sind nicht erforderlich

Softwareressourcen:

- Python und notwendige Dependencies
- PyTest/JUnit für Unittests
- Selenium für mögliche Automatisierung
- Pylint für statische Codeanalyse

### 5.2. Genereller Ablauf

Die Testphase sieht vor, dass während der Entwicklung UnitTests im Rahmen von TDD geschrieben werden. Diese Tests werden im Laufe der Zeit angepasst und erweitert. Sobald die Entwicklung vorerst abgeschlossen ist, finden Reviews statt. Diese Reviews beinhalten eine Zusammenarbeit von zwei Reviewern mit einem Entwickler, welche eine Checkliste des Testverantwortlichen abarbeiten. Nach den Reviews sind Abnahmetests geplant, an denen sich alle Test- und Entwicklungsverantwortlichen beteiligen.

### 5.3. Abnahmetests

In unserem Abnahmetest legen wir großen Wert darauf sicherzustellen, dass unsere Software alle Anforderungen erfüllt und zuverlässig funktioniert. Dabei haben wir zwei Ansätze in Betracht gezogen.

Der erste Ansatz wäre die Einbindung von Testkunden, um sicherzustellen, dass die Software ihren praktischen Anforderungen entspricht. Leider ist dies aus organisatorischen Gründen nicht umsetzbar, daher haben wir uns für einen zweiten Ansatz entschieden.

Beim zweiten Ansatz überprüfen wir, ob unsere Software die festgelegten Anforderungen erfüllt, indem wir die Testfälle/Anforderungen aus dem Lastenheft mit dem geschriebenen Code abgleichen. Das bedeutet, wir überprüfen systematisch, ob die Programmierung genau das umsetzt, was wir uns vorgestellt haben. So stellen wir sicher, dass die Software nicht nur auf dem Papier, sondern auch in der Umsetzung unseren Erwartungen entspricht.

Diese zweite Methode ermöglicht uns eine effiziente Überprüfung, ohne zusätzlichen organisatorischen Aufwand. Damit gewährleisten wir, dass die Software stabil und den Anforderungen entsprechend funktioniert.

### 5.4. Zeitlicher Ablauf

Die UnitTests werden im Laufe des Entwicklungsprozesses geschrieben. Die Reviews und Abnahmetests sind für die Kalenderwochen 49 und 50 geplant.

### 5.5. Dokumentation

Die Testdokumentation besteht in erster Linie aus exportierten Testfällen, die detailliert die verschiedenen Test-Szenarien und -fälle beschreiben. Jeder Testfall enthält klare Anweisungen, Bedingungen, Abläufe und erwartete Ergebnisse. Dies ermöglicht eine umfassende Überprüfung der Kernfunktionalität des Systems.

Im Rahmen der Dokumentation werden auch Reviews durchgeführt. Diese Reviews beinhalten das Ausfüllen spezifischer Checklisten durch Teams von Testmanagern, Testanalysten und Entwicklern. Diese Checklisten dienen als strukturierte Richtlinien für die Überprüfung von Codequalität, Funktionalität und Erfüllung der Anforderungen.



## 6. Reflexion

Im Rahmen des Projekts haben wir erste praktische Erfahrungen im Software Engineering gesammelt. Im Folgenden wollen wir die zentralen Erkenntnisse, welche wir durch das Projekt erworben haben, wiedergeben:

- Die starke Orientierung an den Entwicklungsphasen und die Benennung von Phasenverantwortlichen hat den Entwicklungsfluss stark behindert. Aufgrund von vielfältigen Abhängigkeiten zu den jeweils vorangehenden Schritten war es oftmals nicht möglich, mit mehreren Personen parallel an verschiedenen Aufgaben zu arbeiten.
- Die Erstellung von Verhaltensdiagrammen hat sehr zu einem gemeinsamen Verständnis der eigentlich abstrakten Anforderungen innerhalb des Teams geführt. Es wurde so eine gemeinsame Vision vereinbart, welche im weiteren Projektverlauf anderenfalls notwendige, langwierige Diskussionen über die Ausrichtung des Projekts erspart hat.
- Die Abschätzung der Arbeitsaufwände und die Bestimmung der tatsächlichen Arbeitsaufwände hat sich aufgrund der Verteilung über das Semester als schwierig herausgestellt. Durch die Verteilung auf zwölf Vorlesungswochen mit häufigen Unterbrechungen durch andere Module war die auf das Modul entfallende Zeit nicht immer klar zu bemessen.

Insgesamt haben wir durch das Projekt einen interessanten Einblick in das Software Engineering erhalten und dabei die Vielfältigkeit der verschiedenen Teildisziplinen kennengelernt.

# Teil II.

## Anhang

### A. Testfälle

#### Hohe Priorität

- TF-01 Maus Kalibration starten: Dieser Testfall überprüft das grundlegende Funktionieren der Maus-Kalibration, eine Schlüsselkomponente des Systems
- TF-03 Hörtest durchführen: Die korrekte Durchführung des Hörtests ist entscheidend für die Hauptfunktionalität des Systems
- TF-04 Hörtest abbrechen: Die Fähigkeit, den Hörtest erfolgreich abzubrechen, ist wichtig für die Benutzerfreundlichkeit

#### Mittlere Priorität

- TF-02 Maus Kalibration mit Abweichung: Dieser Testfall überprüft eine spezifische Bedingung während der Maus-Kalibration und ist wichtig, aber nicht so kritisch wie das Grundverfahren.
- TF-05 Richtung angeben: Die korrekte Angabe der Richtung ist wichtig für die Validierung der Nutzerantworten auf den Testton.

#### Niedrige Priorität (Optionale Ziele)

- TF-06 Ergebnisse anzeigen: Dieser Testfall überprüft die Anzeige der Ergebnisse auf dem Laptop und ist wichtig, aber möglicherweise weniger kritisch als die Hauptfunktionalität.
- TF-07 Testergebnisse speichern und anzeigen: Die Funktion zum Speichern und Anzeigen von Testergebnissen über verschiedene Sitzungen hinweg ist nützlich, aber nicht unbedingt für die Kernfunktionalität entscheidend.

## B. Risikoanalyse

### B.1. Risikomatrix

Um unseren Hörtest mit dem Sound Dome und der Space Mouse erfolgreich umzusetzen, haben wir eine klare Teststrategie entwickelt. Diese Risikomatrix ist ein wichtiger Bestandteil dieser Strategie und hilft uns dabei, mögliche Probleme frühzeitig zu erkennen und zu bewerten.

Unser Projekt beinhaltet technische Herausforderungen wie die Integration von Plugins für die Space Mouse und die Synthese von Testtönen mit NAudio. Auch organisatorische Aspekte wie die Verfügbarkeit von Ressourcen und externe Faktoren wie mögliche Änderungen in den Anforderungen spielen eine Rolle.

Die Risikomatrix zeigt in einfacher Form, welche Risiken auftreten könnten, wie wahrscheinlich sie sind und welche Auswirkungen sie haben. Das ermöglicht es uns, unsere Aufmerksamkeit gezielt auf die wichtigsten Risiken zu richten und Maßnahmen zu ergreifen. Durch regelmäßige Aktualisierungen stellen wir sicher, dass unsere Teststrategie flexibel bleibt und wir das Projekt erfolgreich abschließen können.

### B.2. Risiken

#### 1. Plugin-Integration für die Space Mouse:

**Beschreibung:** Es könnte Schwierigkeiten bei der Integration eines externen Plugins für die Space Mouse auftreten.

**Maßnahmen:** Vorabrecherche für geeignete Plugins, Entwicklung eines eigenen Plugins als Backup.

#### 2. Generierung der Test Sounds:

**Beschreibung:** Die Synthese von Testtönen könnte zu unerwarteten Schwierigkeiten führen

**Maßnahmen:** Test der Sound-Synthese im Vorfeld, Implementierung von Alternativen.

#### 3. Umrechnung der OSC Signale:

**Beschreibung:** Die Umrechnung der OSC-Signale in Winkel könnte ungenau sein.

**Maßnahmen:** Sorgfältige Kalibrierung und Testläufe, Implementierung von Fehlerkorrekturen.

#### 4. Entwurf und Implementierung des Steuerungsprogrammes:

**Beschreibung:** Probleme bei der Implementierung des Steuerungsprogrammes könnten zu unzureichendem Feedback führen.

**Maßnahmen:** Durchdachte Architektur, regelmäßige Code-Reviews, um potenzielle Fehler zu identifizieren.

5. Kompatibilität mit Hardware:

**Beschreibung:** Inkompatibilitäten mit bestimmten Hardwarekomponenten können auftreten.

**Maßnahmen:** Testläufe auf verschiedenen Hardwarekonfigurationen, frühzeitige Identifizierung von Kompatibilitätsproblemen.

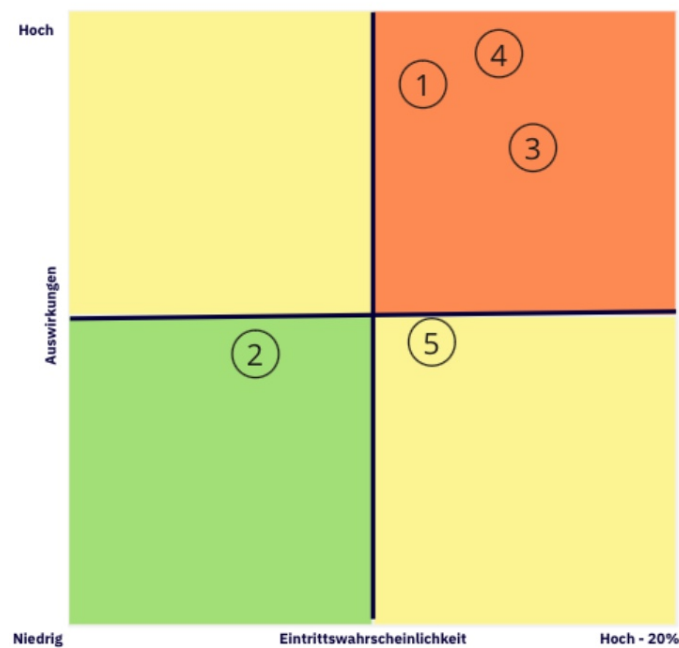


Abbildung 8: Risikomatrix

## OH - SEG - Reviews

### Python Review => [🔗](#)








**Testanalysten:** Serdar Sahin, Lukasz Szupka

**Entwickler:** Robert Rohwer

**Datum:** 09.12.2023

#### SEG - Checkliste Reviews

Erfüllt?

ID	Beschreibung	Ja	Nein	Teilweise
1	Linten durchgelaufen (Pylint / Stylecop)			
2	Ist der Code lesbar und gut strukturiert?			
3	Erfüllt der Code die Anforderungen?			
4	Wurde ausgiebig kommentiert?			
5	Wurde Exception Handling genutzt?			
6	Sind die Unit Tests ausgiebig genug?			
7	Sind die Commit-Nachrichten Aussagekräftig?			

#### Review Ergebnisse =>

Aufgefallen, während des Reviews ist, dass kein Linter genutzt wurde, dies wurde von den Testanalysten noch während des Reviews nachgeholt.

Neben dem haben die Kommentare für den Code gefehlt, die Ergänzung dafür ist während der KW 50 geplant. Ebenso wie das Nachtragen von Exception Handling.

Alles in allem ist der Code jedoch zufriedenstellend.

### C# Review => [🔗](#)








**Testanalysten:** Serdar Sahin, Lukasz Szupka

**Entwickler:** Robert Rohwer

**Datum:** 09.12.2023

## SEG - Checkliste Reviews

Erfüllt?

ID	Beschreibung	Ja	Nein	Teilweise
1	Linten durchgelaufen (Pylint / Stylecop)			
2	Ist der Code lesbar und gut strukturiert?			
3	Erfüllt der Code die Anforderungen?			
4	Wurde ausgiebig kommentiert?			
5	Wurde Exception Handling genutzt?			
6	Sind die Unit Tests ausgiebig genug?			
7	Sind die Commit-Nachrichten Aussagekräftig?			

### Review Ergebnisse =>

Während des Überprüfungsprozesses ist aufgefallen, dass kein Linter verwendet wurde. Dies wurde von den Testanalysten während der Überprüfung nachgeholt. Es fehlen außerdem einige Kommentare im Code. Die Ergänzung dafür ist für die Kalenderwoche 50 geplant. Darüber hinaus sind die vorhandenen Unit Tests zu knapp bemessen und sollten erweitert werden. Diese Punkte sind essentiell für die Codequalität und sollten daher in den kommenden Tagen priorisiert werden.

# OH - SEG Abnahmetest

17.12.2023 — 20:30 - 21:15

Testmanager: Helene Lüning

Testkunde: Mika Schick

Testumgebung: Virtual / SoundDome Simulator

---

## Kundenporträt:

Name: **Mika Schick**

Alter: **25**

Erfahrung bezüglich Hörtests: **Hat in seinem Leben 5 Hörtests gemacht**

---

## Ziel des Tests:

Das Hauptziel dieses Tests besteht darin, sicherzustellen, dass der Kunde in der Lage ist, den Hörtest durchzuführen.

---

## Ablauf des Tests:

### 1. Vorbereitung

#### 2. Erklärung des Hörtest-Ablaufs:

- Dem Testkunden wurden folgende Dinge erklärt:
  - Das Konzept des Hörtests
  - Die Funktionalität der Space Mouse
  - Die Funktionalität des Sound Domes und Sound Dome Simulators

### 2. Durchführung des Hörtests:

- Der Testverantwortliche stellte die Hardware bereit und startete die beiden nötigen Programme und den Simulator
- Der Kunde hörte den Anweisungen des Programmes zu und folgte diesen.
- Nach dem Ersten beendeten Test, bekam der Kunde vom Testverantwortlichen nach einander die Aufgaben
  - einen weiteren Test zu starten
  - einen Test abubrechen
  - den Abbruch eines Testes abubrechen
- Bei der Durchführung des Tests war der Kunde kurz aufgrund der Darstellung des Simulators verwirrt. Nach erneuter Erklärung, wie der eigentliche Sound Dome funktioniert, konnte er jedoch den Test problemlos durchführen.

### 3. Testumgebung:

- Der Test fand in einem ruhigen Raum mit Hilfe der Space Mouse und eines Laptops mit allen benötigten Programmen statt.

### 4. Feedback des Testkunden:

- Der Kunde gab zu bemerken, dass die wiederholte Erklärung bei wiederholten Tests nervig ist.

- Der Kunde erwähnte die Verwirrung aufgrund des Simulators, war jedoch der Meinung, dass dies im tatsächlichen Sound Dome kein Problem sein sollte.
  - Der Kunde bemerkte positiv, dass das Programm sehr genau beschrieb, wie es zu bedienen ist und er immer wusste, was er tun muss.
- 

#### **Zusätzliche Bemerkungen:**

- Während des Tests traten keine unvorhergesehenen Probleme auf.
  - Die Testdurchführung verlief reibungslos, und der Testkunde konnte die erforderlichen Aufgaben erfolgreich bewältigen.
- 

#### **Empfohlene Maßnahmen:**

- Eine alternative Tonspur für den Start weiterer Tests nach dem ersten.
- 

#### **Abschluss:**

Der Abnahmetest wurde erfolgreich durchgeführt, und die Ergebnisse zeigen, dass der Hörtest den Anforderungen entspricht. Es wurde sich gegen die Umsetzung der vorgeschlagenen Maßnahme entschieden, da der aktuelle Zustand vertretbar ist und wir lieber zu viel als zu wenig Erklärung haben.