Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Projektarbeit

Andreas Müller, Claus Torben Haug, Jan-Dennis Bartels, Marjan Bachtiari

MBC-Ping-Pong

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Andreas Müller, Claus Torben Haug, Jan-Dennis Bartels, Marjan Bachtiari

MBC-Ping-Pong

Projektarbeit eingereicht im Rahmen der Wahlpflichtfach

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Martin Becke

Eingereicht am: 16. Dezember 2016

Andreas Müller, Claus Torben Haug, Jan-Dennis Bartels, Marjan Bachtiari

Thema der Arbeit
MBC-Ping-Pong

Stichworte
Ping-Pong, NodeJS, JavaScript, WebRTC

Kurzzusammenfassung
In diesem Dokument wird das Projekt im MBC-Ping-Pong, das im Rahmen des Wahlpflichtfaches Modernebrowserkommunikation an der HAW-Hamburg erstellt wird, behandelt. Hierbei handelt es sich um eine Pong Clone welcher auf einem zentralen Bildschirm spielbar ist und mnittels WebRTC gesteuert wird. Es wird auf die Architektur, JavaScript, Frontend und Backend eingegangen.
Andreas Müller, Claus Torben Haug, Jan-Dennis Bartels, Marjan Bachtiari

Title of the paper
MBC-Ping-Pong

Keywords
Ping-Pong, NodeJS, JavaScript, WebRTC

Abstract
This document is about the Project MBC-Ping-Pong, which is made for the elective course Modernebrowserkommunikation at HAW-Hamburg. Its about a Pong clone, played on a central screen nd controled via WebRTC. The architecture, JavaScript, frontend and backend are discussed.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Architektur

1.1 Arbeitsablauf zur Bearbeitung eines Issue

Um eine erfolgreiche Zusammenarbeit zu gewährleisten, sind allgemein gültige Regeln nötig. Insbesondere wird festgelegt, wie die einzelnen Arbeitsschritte ablaufen sollten, um ein Issue zu bearbeiten. Zudem werden weiterhin die Zuständigkeiten geregelt.

1.1.1 Verwaltung der zu bearbeitenden Issues

Die zu bearbeitenden Issues werden auf GitHub unter Issues (https://github.com/Transport-Protocol/MBC-Ping-Pong/issues) gepflegt. Um den Verlauf eines Issues darzustellen wird das Kanbanboard von GitHub (https://github.com/Transport-Protocol/MBC-Ping-Pong/projects) genutzt.

1.1.2 Erstellen der zu bearbeitenden Issues

Prinzipiell kann und darf jedes Projektmitglied zu jeder Zeit Issues erstellen. Gerade bei Bugs ist dies ein gewünschtes vorgehen. In der Regel sollten dies jedoch aus Gruppensitzungen hervorgehen und durch den Architekten ausformuliert werden.
Ein Issue besteht aus drei Absätzen:

- **Beschreibung**
  In der Beschreibung wird allgemein auf den Kontext des Issues eingegangen.
- **Anforderung**
  In Anforderung wird die Zielvision dargestellt.
- **Abnahmekriterien**
  In Abnahmekriterien werden alle Punkte aufgeführt, die notwendig sind, um das Issue als erfolgreich bearbeitet anzusehen.

1.1.3 Das Kanbanboard

Das Kanbanboard ist in fünf Abschnitte eingeteilt:

- **Selected for Development**
  Diese Spalte enthält alle Issues, die der Architekt zur Bearbeitung in nächster Zeit ausgewählt hat. Hier enthaltene Issues sind entweder durch den Architekten einem bestimmten Teammitglied zugeordnet. Diese sollten dann auch vorrangig bearbeitet werden. Oder (dies sollte der Normalfall sein) sie sind niemandem zugeordnet, dann kann sich jedes Teammitglied entscheiden, ob er das Issue bearbeitet. Gründe für das direkte zuweisen können unteranderem sein, dass es eine entsprechende vorhergehende Absprache gab, dass der Architekt das Issue speziell einem Bereich zugehörig sieht bzw. eine spezielle Paarung erreichen möchte, oder aber auch, weil ein Issue schon zu lange in SSelected for Development"verweilt. Hat sich ein Teammitglied für ein Issue entschieden, trägt er sich als Bearbeiter ein und zieht es in auf Ïn Development".

- **In Development**
  In dieser Spalte verweilen alle Issues, an denen gerade entwickelt wird. Wenn die Entwicklung an einem Issue abgeschlossen ist, zieht der Bearbeiter das Issue weiter auf "Needs Review".

- **Needs Review**
  Hier verweilen alle Issues, deren Entwicklung abgeschlossen ist, aber noch nicht geprüft wurde, ob die Abnahmebedingungen erfüllt sind. Normalerweise sollte die Abnahme durch den Architekten erfolgen. Issues, die der Architekt bearbeitet hat, muss das Review von einem anderen Teammitglied gemacht werden. Ein Issue bei dem das Review durchgeführt wird, wird in die Spalte Ïn Review"verschoben.

- **In Review**
  Hier sind alle Issues enthalten, die sich gerade im Review befinden. Sind alle Abnahmekriterien erfüllt, und sind durch die Bearbeitung des Issue keine neuen Probleme/Fehler hinzugekommen, wird es in die Spalte "Done"verschoben und das Issue geschlossen. Ist dies nicht der Fall, wird ein Entsprechender Kommentar mit einer möglichst detaillierten Beschreibung des Problems an das Issue angehängt, und es wieder auf Ïn Development"geschoben.

- **Done**
  Diese Spalte enthält alle abgeschlossenen Issues.

1.1.4 Git

Hier sind die Verhaltensweisen für die Nutzung von Git aufgeführt. Alles hier nicht aufgeführte kann von jedem Teammitglied nach eigenen ermessen gehandhabt werden.

- **Branches**
  Für jedes Issue wird ein Branch erstellt, außer es handelt sich um reine Dokumentation (im Ordner Docu). Ein Branchname folgt folgendem Muster: "#<IssueNummer> <KurzerName>". Dadurch lässt sich
- **Commits**
  Commits folgen folgendem Namensschema: "#<IssueNummer> <Beschreibung>".
- **Push und Pull**
  Es sollte möglichst häufig gepusht werden, um einen eventuellen Datenverlust zu vermeiden. Beim Pull sollte mit -rebase"gearbeitet werden, um die Historie möglichst sauber zu halten.
- **Merge und Pullrequest**
  Bevor ein Issue auf "Needs Review"geschoben wird, ist der Master in den Branch zu mergen und ein Pullrequest (https://github.com/Transport-Protocol/MBC-Ping-Pong/pulls) zu erstellen. Derjenige der das Issue reviewt hat, merget den Branch dann mithilfe des Pullrequests in den Master und löscht ihn.

1.2 Meilensteine

In diesem Abschnitt werden die Meilensteine festgelegt. Hierbei wird beschrieben, was wan erreicht sein sollte.

1.2.1 Projekt Aufsetzen

- **Beschreibung**
  Die Grundlegenden für die Entwicklung notwendigen anfangs Infrastrukturen sind aufgesetzt.
- **Kriterien**
  - **NodeJS-Server aufsetzen**
    Der NodeJS Server ist aufgesetzt und stellt eine statische Website zur Verfügung

  - **Docker**
    Eine einheitliche Umgebung wird durch Docker und Docker-Compose ermöglicht.

- **Beendet:** 25.11.2016

1.2.2 Prototyp (Technik)

- **Beschreibung**
  Um die identifizierten technischen Risiken schnellst möglich in den Griff zu bekommen, werden diese möglichst früh bearbeitet. In dem Prototyp (Technik) soll gezeigt werden, das die kritische Technik funktioniert. Dies wird anhand von kleine losgelösten Beispielen, die aber nahe der Zielarchitektur sind gezeigt.

- **Kriterien**
  - **Darstellung**
    Es wird gezeigt, das im Webbrowser eine Flüssige Darstellung möglich ist.
  - **Kollisionserkennung**
    Es wird gezeigt, dass eine Kollisionserkennung erreichbar ist.
  - **Kommunikation mittels WebRTC**
    Architektur bedingt ist die Nutzung von WebRTC unumgänglich. Es ist zu zeigen, dass eine Verbindung von mehreren Handys zum Darstellungsmedium möglich ist.
  - **Steuerung**
    Die Steuerung soll über den Touchscreen geschehen. Es ist zu zeigen, dass es möglich ist, die Position des Fingers auf dem Touchscreen im Browser auszulesen.
  - **Größen der Handys/Tabletts**
    Unterschiedliche Handys und Tabletts haben verschiedene Größen und Formen. Somit ist ein Konzept zu erarbeiten, welches diesem Problem bei der Steuerung gerecht wird.

- **Beendet:** 16.12.2016

1.2.3 Release 1.0 (Zwei Spieler)

- **Beschreibung**
  In diesem ersten Release ist eine Basisversion des Spieles implementiert. Es können zwei Spieler gegeneinander Spielen, indem sie ihre Schläger mit den Handys steuern. Gleichzeitig ist diese Version die minimal Version und enthält alle "MustFeatures.

- **Kriterien**
  - **Schläger**
    Für jeden Spieler existiert ein Schläger, der mit dem Handy Steuer
  - **Ball**
    Es gibt ein Ball, der sich über das Spielfeld bewegt. Kollidiert er mit einem Schläger oder der Wand, an der sich kein Schläger befindet, prallt er davon ab. Es gilt hierbei, dass der Einfallwinkel dem Ausfallwinkel entspricht. Zudem beschleunigt der Ball, wenn er mit einem Schläger kollidiert. Wenn der Ball mit der Wand hinter einem Schläger kollidiert, wir er in die Ausgangsposition versetzt und erhält die Ausgangsgeschwindigkeit.
  - **Punkte**
    Immer wenn der Ball mit der Wand hinter einem Schläger kollidiert, erhält der andere Spieler einen Punkt.
  - **Spielende**
    Das Spiel endet automatisch nach X Spielen (wobei gilt: $X \in \mathbb{N} \wedge X \mod 2 = 1$).
    Das genaue X ist noch zu definieren.
- **Beendet:** 13.01.2017

1.2.4 Release 1.X (Diverse Features)

- **Beschreibung**
  Basierend auf der Version 1.0 wird das Spiel weiterentwickelt. Jedoch sind alle Features die hier bearbeitet werden können "CanFeatures. Daher kann es sein, dass das Release 1.X äquivalent zu dem Release 1.0 ist. Zudem sind alle hier genannten möglichen Features noch nicht näher spezifiziert und auf ihre Machbarkeit geprüft. Es gilt jedoch, dass je Umgesetztes Feature die Versionsnummer im Minorbereich um Eins steigt.
- **mögliche Kriterien**
  - **N Spieler**
    Mehr als 2 Spieler
  - **Zusätzliche Hindernisse**
    Auf dem Spielfeld sind zusätzliche Hindernisse.
  - **Highscore-Liste**
    Es wird eine Highscore-Liste geführt und angezeigt.

  - **TBD**
    To be discussed.

- **Beendet:** 24.02.2017

1.3 Highlevel View

In diesem Abschnitt wird die Grob-/Gesamtarchitektur betrachtet. Hierbei wird nicht nur auf wesentliche Schnittstellen und Komponenten eingegangen. Zur engeren Auswahl standen zwei mögliche Ansätze. Es werden beide betrachtet, und erläutert warum der Ansatz 2 umgesetzt werden wird.

1.3.1 Ansatz 1

Der Ansatz 1 verfolgt den Klassischen Client-Server-Ansatz. Hierbei dient der Server als zentrale Instanz, die alle signifikanten Logikoperationen übernimmt. Die Clients dienen lediglich zur Ein-/Ausgabe.

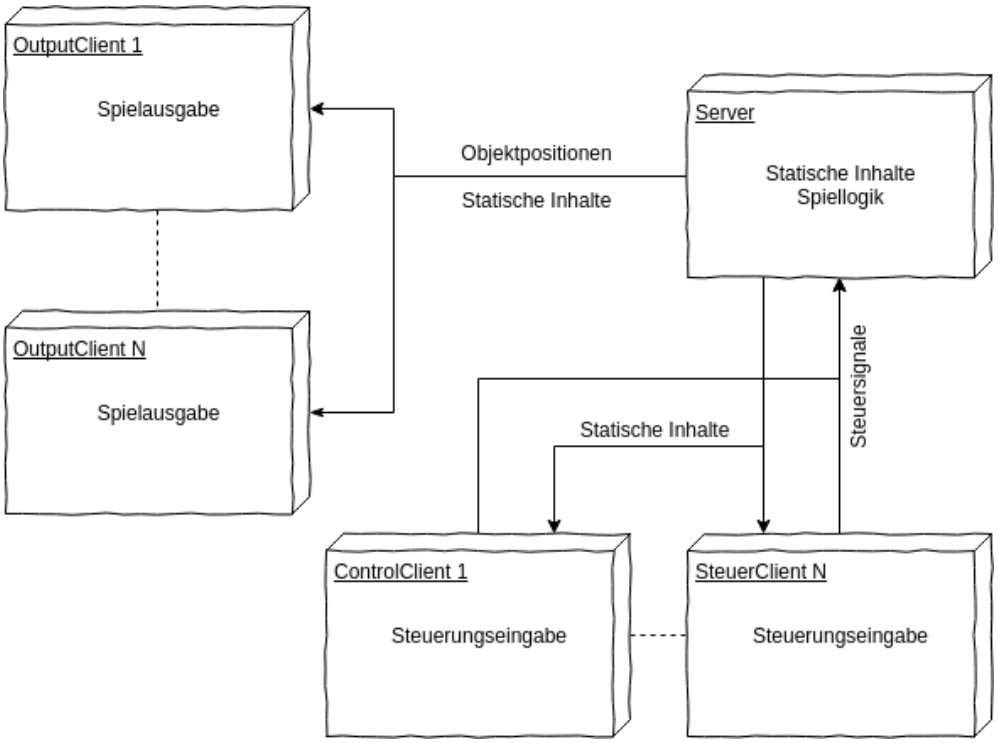

Abbildung 1.1: Highlevel Ansatz 1

Server

Der Server stellt die statischen Inhalte (HTML, CSS, JS) bereit. Zudem enthält er die gesamte Spiellogik. Der Server empfängt Steuerinformationen vom ControlClient, verarbeitet diese und sendet einen aktuellen Gamestate an den OutputClient.

ControlClient

Der ControlClient erfasst die Steuereingaben des Nutzers und sendet sie an den Server.

OutputClient

Der OutputClient empfängt den Gamestate vom Server und aktualisiert die Anzeige entsprechend.

Analyse

Einerseits ist dieser Ansatz architektonisch sehr einfach umzusetzen, da es eine zentrale Instanz gibt und Seperation of Concerns Architektur bedingt unterstützt wird. Zudem ist es möglich ein Spiel auf mehreren OutputClients darzustellen. Da sich die Clients mit dem Server verbinden, ist der Einsatz von WebSockets möglich. Mit socket.io gibt es eine sehr gute Abstraktionsschicht für WebSockets. Andererseits kann durch den Einsatz von WebSockets ein nicht zu vernachlässigendes Delay entstehen, da diese auf TCP basieren. Um diesem zu begegnen ist der Einsatz des WebRTC Protokollstacks notwendig. Zudem muss der Server die Zuordnung der ControlClients und OutputClients zu einem Spiel organisieren. Außerdem sind zwei Netzwerkübertragungen von Nöten, damit die Eingabe des Spielers auf dem OutputClient sichtbar wir. Dadurch wird das Netzwerk doppelt belastet, und es ist zweimal das Übertragungsdelay vorhanden.

1.3.2 Ansatz 2

Server

Der Server stellt die statischen Inhalte (HTML, CSS, JS) bereit. Zudem fungiert er als zentrale Instanz für den WebRTC-Protokollstack

ExternerServer

Der externe Server ist ein öffentlicher Stun-/Turn-/Ice-Server, der beispielsweise von Google bereitgestellt wird.

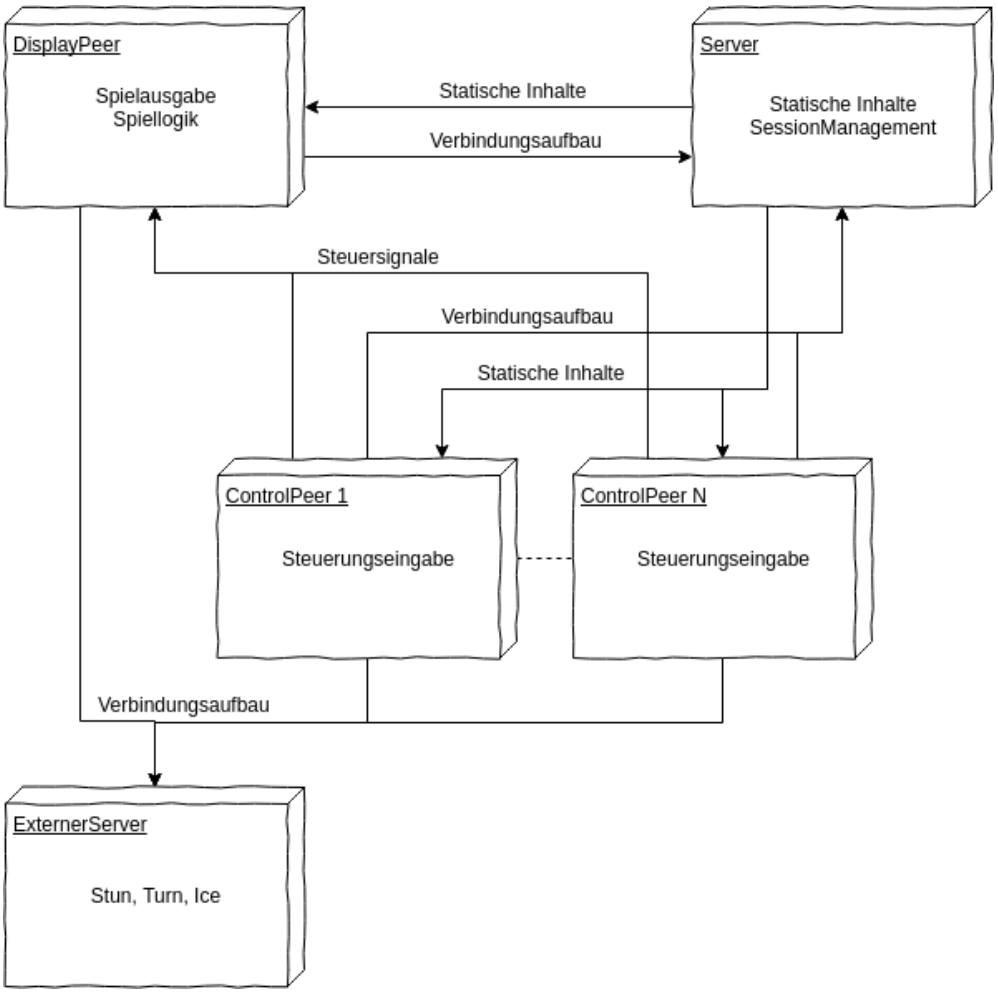

Abbildung 1.2: Highlevel Ansatz 2

ControlPeer

Der ControlPeer erfasst die Steuereingaben des Nutzers und sendet sie an den DisplayPeer.

DisplayPeer

Der DisplayPeer ist ein Fat-Client. Er enthält die gesamte Spiellogik und empfängt über den WebRTC-Protokollstack direkt die Steuereingaben des Spielers. Zudem zeigt er das Spiel an.

Analyse

Der Ansatz 2 ist architektonisch recht Komplex, da die Aufteilung auf Client und Server wegfällt, und somit die native Unterstützung von Seperation of Concern nicht gegeben ist. Es muss während der Entwicklung verstärkt darauf geachtet werden, das die Trennung von Spiellogik und Ausgabe eingehalten wird. Zu dem ist man auf den WebRTC-Protokollstack angewiesen, da die Peers direkt miteinander Kommunizieren müssen. Außerdem ist man auf einen einzelnen DisplayPeer beschränkt. Andererseits erfolgt die Übertragung der Steuerung direkt von dem ControlPeer an den DisplayPeer, dadurch ist das kleinst mögliche Delay zwischen Eingabe, Verarbeitung und Ausgabe gewährleistet. Außerdem ermöglicht der Einsatz von WebRTC den Einsatz von UDP als Transportprotokoll, wodurch das Delay weiter verringer werden kann, da UDP Verbindungslos ist.

1.3.3 Fazit

Auch wenn der Ansatz 2 zunächst komplexer erscheint und einen geringeren Funktionsumfang bietet, da nur ein AusgabeClient pro Spiel unterstützt wird und WebRTC eingesetzt werden muss, überwiegen doch die Vorteile dieses Ansatzes. Durch die fehlende zweite Netzwerkübertragung und der mögliche Einsatz von UDP, werden Delays minimiert. Gleichzeitig wird mehr Rechenleistung auf die Clients ausgelagert und eine aufwändige Verwaltung, welcher Spieler zu welchem Spiel gehört ist auch nicht notwendig.

1.4 Risiken

In diesem Kapitel wird auf die Risiken eingegangen, die zu Schwierigkeiten bei der Projektdurchführung führen können. Die Auswirkungen und Eintrittswahrscheinlichkeit werden in 3 Kategorien eingeteilt: "1:Gering, 2:Mittel, 3: Hoch". Das potenzielle Risiko ist das Produkt aus Auswirkungen und Eintrittswahrscheinlichkeit.

1.4.1 Technische Risiken

Zunächst wird auf die technischen Risiken eingegangen

Darstellung ist nicht möglich

- **Beschreibung:**
  Gerade bei Ping-Pong wird die Bewegung des Balls irgendwann sehr schnell. Dies muss trotzdem im Browser darstellbar sein, ohne das es ruckelt.

- **Eintrittsgründe;**
  - Das gewählte Grafikframework ist nicht leistungsfähig genug.
  - Das gewählte Grafikframework wurde nicht richtig genutzt.
  - Das Zielsystem ist nicht Leistungsfähig genug.
- **Folgen:**
  - Das Spielen ist nicht möglich => Projektfehlschlag.
- **Eintrittswahrscheinlichkeit:** 2
- **Auswirkungen:** 3
- **Risiko:** 6
- **Maßnahmen:**
  - Bereits im Prototyp eine Beispielimplementierung durchführen.
  - Möglichst früh einen Test auf der Zielplattform absolvieren.

Kollisionserkennung funktioniert nicht

- **Beschreibung:**
  Durch die schnelle Ballbewegung bei Ping-Pong ist eine gute Kollisionserkennung notwendig.
- **Eintrittsgründe;**
  - Die gewählte Physicsengine ist nicht leistungsfähig genug.
  - Die gewählte Physicsengine wurde nicht richtig genutzt.
  - Das Zielsystem ist nicht Leistungsfähig genug.
- **Folgen:**
  - Das Spielen ist nicht möglich => Projektfehlschlag.
- **Eintrittswahrscheinlichkeit:** 2
- **Auswirkungen:** 3
- **Risiko:** 6
- **Maßnahmen:**

  - Bereits im Prototyp eine Beispielimplementierung durchführen.
  - Möglichst früh einen Test auf der Zielplattform absolvieren.

Kommunikation mittels WebRTC funktioniert nicht

- **Beschreibung:**
  Um die Übertragung in akzeptabler Geschwindigkeit zu gewährleisten, ist der Einsatz von WebRTC unausweichlich. WebRTC ist jedoch absolutes Neuland für das gesamte Team.
- **Eintrittsgründe;**
  - WebRTC wird nicht korrekt genutzt.
  - Im Zielnetzwerk wird die Verwendung durch Firewalls behindert.
- **Folgen:**
  - Die Architektur muss von Fat-Client Ansatz auf einen Serverzentrierten Ansatz umgestellt werden => Projektverzögerung.
- **Eintrittswahrscheinlichkeit:** 2
- **Auswirkungen:** 2
- **Risiko:** 4
- **Maßnahmen:**
  - Bereits im Prototyp eine Beispielimplementierung durchführen.
  - Möglichst früh einen Test auf der Zielplattform absolvieren.

Steuerung ist nicht möglich

- **Beschreibung:**
  Das auslesen der Position des Fingers auf dem Bildschirm über den Browser funktioniert nicht, bzw. nicht schnell genug.
- **Eintrittsgründe;**
  - Zu altes Handy verwendet (Browser unterstützt es nicht).
  - Schnittstelle nicht korrekt verwendet

- **Folgen:**
  - Das Spiel ist nicht Steuerbar => Projektfehlschlag.
- **Eintrittswahrscheinlichkeit:** 1
- **Auswirkungen:** 3
- **Risiko:** 3
- **Maßnahmen:**
  - Bereits im Prototyp eine Beispielimplementierung durchführen.

1.4.2 Konzeptuelle Risiken

Steuerung ist nicht Fair

- **Beschreibung:**
  Gerade mit dem Prinzip des Browser basierten Ansatzes werden sehr viele unterschiedliche Endgerätetypen angesprochen. Jedes Endgerät hat jedoch eine andere Bildschirmgröße und Pixeldichte.
- **Eintrittsgründe;**
  - Es möchten Personen mit unterschiedlichen Endgeräten gegeneinander antreten
- **Folgen:**
  - Das Spiel ist unfair => geringere Akzeptanz.
- **Eintrittswahrscheinlichkeit:** 3
- **Auswirkungen:** 1
- **Risiko:** 3
- **Maßnahmen:**
  - Das Risiko wird durch die weitergehende Analyse, und dem Entwurf eines möglichst Fairen Steuerungskonzeptes verringert.
  - Ggf. zum Teil ignorieren

1.4.3 Organisatorische Risiken

Temporärer Personalausfall

- **Beschreibung:**
  Es kann jederzeit zu einem temporären Personalausfall kommen.
- **Eintrittsgründe;**
  - Krankheit.
  - Klausuren.
  - Unmotiviertheit.
- **Folgen:**
  - Arbeitskraftmangel => Projektverzögerung.
  - Fachwissensmangel => Projektverzögerung.
- **Eintrittswahrscheinlichkeit:** 3
- **Auswirkungen:** 1
- **Risiko:** 3
- **Maßnahmen:**
  - Puffer einplanen.
  - Möglichst wenig Must-Anforderungen definieren.

Kompletter Personalausfall

- **Beschreibung:**
  Es kann jederzeit zu einem temporären Personalausfall kommen.
- **Eintrittsgründe;**
  - Krankheit.
  - Unmotiviertheit.
- **Folgen:**
  - Arbeitskraftverlust => Projektverzögerung.
  - Fachwissensverlust => Projektverzögerung.

- **Eintrittswahrscheinlichkeit:** 2
- **Auswirkungen:** 2
- **Risiko:** 4
- **Maßnahmen:**
  - Puffer einplanen.
  - Möglichst wenig Must-Anforderungen definieren.

1.5 Reviews

1.5.1 Auswahl einer geeigneten 2D-/Physiksengine

- **Datum:** 05.12.16
- **Prüfung der Anforderungen:**
  - ✓ Aufruf der Seite http://<IpDesNodeJsServers>:<HttpPortDesNodeJsServers>/test2DEngine.html zeigt die Testseite.
  - ✓ Ein quadratischer Bereich mit einer Form in der Mitte auf dem Bildschirm wird dargestellt.
  - ✓ In diesem Bereich bewegt sich eine Form (am besten eine Kugel) waagerecht von Links nach Rechts.
  - ✓ Kollidiert die Form mit der rechten Wand, bewegt sich die Form von Rechts nach Links.
  - ✓ Kollidiert die Form mit der linken Wand, bewegt sich die Form von Links nach Rechts.
- **Prüfung des Codes:**
  - Der Code ist strukturiert und lesbar.
  - Bis auf 2 Zeilen ist der Code verständlich:
    - this.ball.body.immovable = true;
      Ein besserer Kommentar, warum wäre gut gewesen. Nach einigem suchen, gehe ich davon aus, dass nur Objekte, die mit dem Ball kollidieren, beeinflusst werden, aber nicht der Ball selbst.

Dieses Verhalten ist zunächst OK, sollte jedoch bei späteren Features noch einmal geprüft werden.

* this.ball.body.bounce.set(1);
  Es funktioniert in der momentan verwendeten Version, daher ist es OK, sollte jedoch geprüft werden. Laut Dokumentation (http://phaser.io/docs/2.6.2/Phaser.Physics.Ninja.Body
  Sollte der Wert zwischen 0 und 1 liegen, also nicht 1.0 sein. Empfohlen wird ein wert von 0.999. Eine Prüfung ergab, das auch Werte oberhalb von 1.0 akzeptiert werden. Dies führt dann zu einer Beschleunigung des Balls.

1.6 Prototyp

In diesem Kapitel wird der Prototyp behandelt.

1.6.1 Schnittstellen

Dieses Diagramm (Abbildung 1.3) dient dem gemeinsamen Verständnis, welche Klasse welche Schnittstelle bietet.

1.6.2 Fachlicher Ablauf

Im fachlichen Ablauf (Abbildung 1.4) wird dargestellt, wie sich das Programm verhalten soll. Es wird jedoch nicht auf genaue Implementierungsdetails wert gelegt. Er dient dem allgemeine Verständnis, wie der Programmablauf sein soll. Auch die Funktionsnamen bei fremd-APIs dienen lediglich der Beschreibung, welches Verhalten von der API erwartet wird. **Allerdings sollte der Ablauf später im Programmcode ersichtlich sein.**

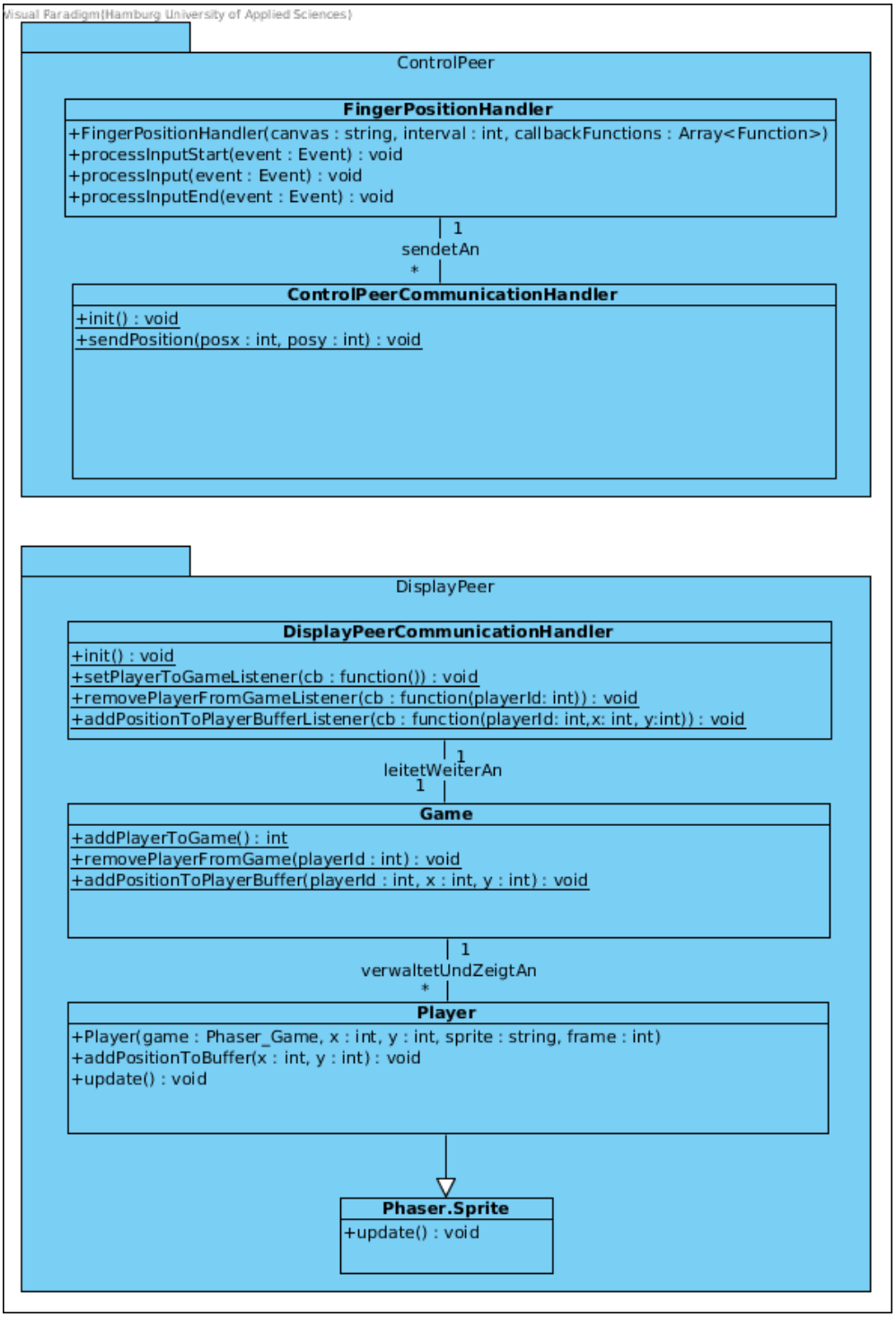

Abbildung 1.3: Prototyp Schnittstellen - Klassendiagram

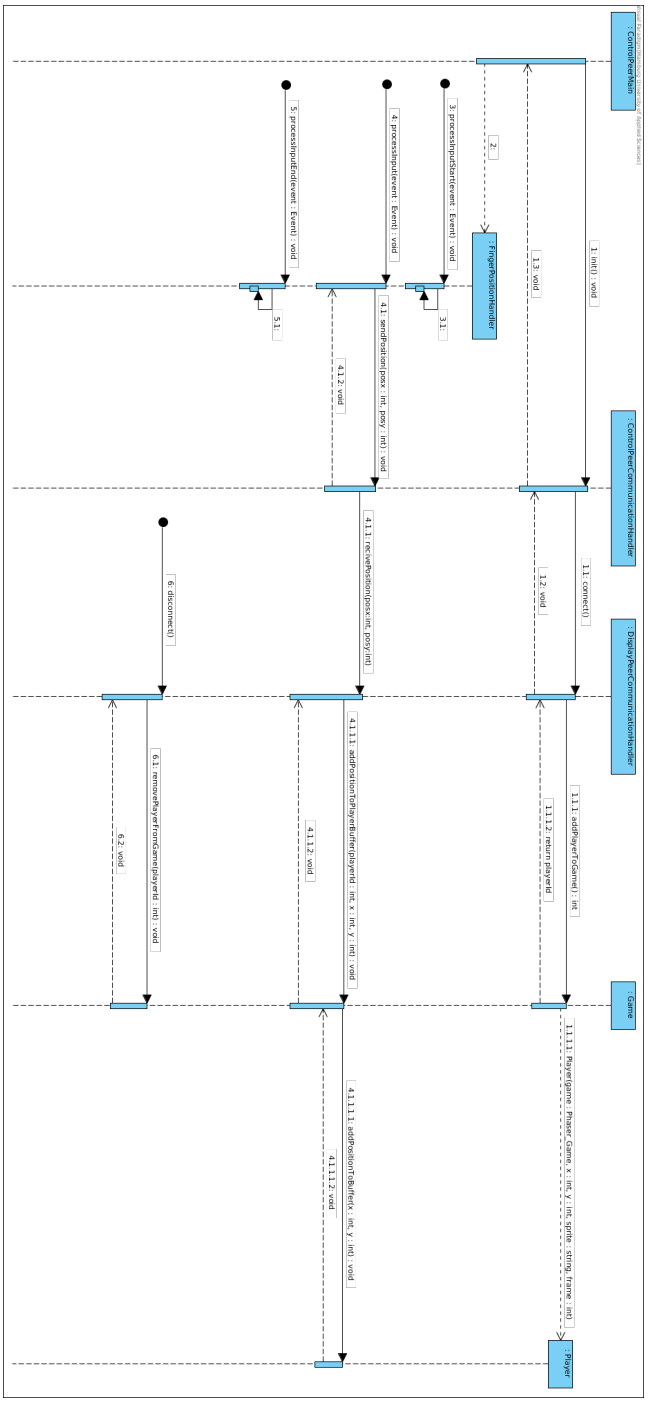

Abbildung 1.4: Prototyp fachlicher Ablauf - Sequenzdiagramm

2 JavaScript

3 Backend

3.1 Kommunikation

3.1.1 Zeitkritische Informationen

Als zeitkritisch werden Informationen eingestuft, sofern sie die direkten Eingaben der ControlClients und eventuelles Feedback des OutputClient betreffen. Da es sich um ein Reaktionsspiel handel,t müssen diese Daten zeitnah von Sender zu Empfänger gelangen. Solch eine Relation ist nur über eine Peer-to-Peer Verbindung zu realisieren.

3.1.2 Möglichkeit 1: Predefined Packages 'rtc.io'

Der NodeJS Server wird als Verbindungsserver für die Etablierung einer Peer-to-Peer Verbindung zwischen ControlClients und OutputClient genutzt. Als Technik wird konret WebRTC eingesetzt. Auf dem NodeJS-Server wir das Package "rtc-switchboardëingesetzt, welches eine Grundlage für einen Signalisierungsserver ist. Die Clients nutzen "rtc-quickconnectüm neues Channels anzufordern und eine Peer-to-Peer Verbindung zu etablieren.

Vorteile:

- Leichtere Implementierung, da alle Ebenen der Kommunikation bereits abgedeckt wurden und nur noch semantisch auf Informationen eingegangen werden muss.

Nachteile:

- Durch Generalisierung ein deutlicher Overhead.
- Status ist offiziell noch 'unstable'.

3.1.3 Möglichkeit 2: Abstrakte WebRTC implementation 'WebRTC'

Wie auch bei Möglichkeit 1 wird der NodeJS-Server als Verbindungsserver genutzt. Jedoch muss auf Client-Seite, also für OutputClient und ControlClient, eine eigene Signalling Implementation stattfinden. Es wird eine viel abtraktere Schicht genutzt.

Vorteile:

- Da die genutzen Packages nur eine Grundlage bilden, ist eine spezialisierte Implementierung möglich.

Nachteile:

- Implementierungsaufwand deutlich höher.
- Spezialisierung für dieses Projekt eventuell nicht nötig.
- Status ist offiziell noch 'unstable'.

3.1.4 Möglichkeit 3: Socket.io P2P

Das Package Socket.io bietet auch selber eine Peer-to-Peer lösung auf WebRTC Basis. Hierbei wird eine Spezielle Art eines Sockets genutzt, welche wie ein normaler WebSocket agiert, bis ein Upgrade durchgeführt wird und die Clients nun direkt miteinander Kommunizieren.

Vorteile:

- Das Signalling und die P2P Kommunikation können mit einem Package geregelt werden.
- Variabel kann die Kommunikation über den Server, oder P2P ablaufen.
- Signalling events werden von Client-Server Kommunikation direkt zu P2P übernommen.

Nachteile:

- Wenig Einfluss auf die Upgrade-Mechanismen.

3.1.5 Statusinformationen

Für den Austausch nicht zeitkritischer Statusinformationen werden zwischen den Clienten und dem Server einfache WebSockets verwendet. Der Austausch von Zeitkritischen Statusinformationen muss über das Signalling des WebRTC durchgeführt werden.

3.1.6 Fazit

Die eigene Implementation auf Basis des WebRTC bietet die meisten Möglichkeiten, birgt jedoch auch enorme Risiken. RTC.io bietet eine Implementation die sehr gut für Prototypen geeignet ist, aber neben dem Zeitunkritischen Signalling eine eigene Einheit bietet, welche eine Generalisierung des WebRTC darstellt und somit viel Overhead besitzt. Die P2P Imeplementierung von Socket.io steht von der Implementierungs Komplexität in der Mitte. Es bietet viele

bereits bekannte Mechanismen des Client-Server Signalling über Events, welche mit einem Upgrade nun auch von Client zu Client funktionieren.

Socket.io P2P entspricht den Anforderungen und wird für den Prototypen verwendet.

4 Frontend

4.1 Auswahl einer Physics-Engine

> Auswahl einer geeigneten 2D-/Physiksengine #3
> Die Darstellung auf dem Anzeigegerät ist über den DOM nicht möglich, da die Bewegung des Balls und der Schläger zu schnell werden. Zudem wird auch die Physik auf dem Anzeigegeräts berechnet. Hierfür ist eine Kollisionserkennung erforderlich.

Für die physikalisch korrekte Kollision des Balles mit der Spielwelt und den Schlägern haben wir uns entschieden eine Physics-Engine zu verwenden.

4.1.1 Matter.JS

Matter js ist eine sehr mächtige Engine. Sie unterstützt viele Formen und Physikalische Eigenschaften wie zum Beispiel Masse und wirkende Kräfte auf die jeweiligen Objekte. Es besteht die Möglichkeit physikalische Objekte zusammen zusetzen und sogar diese Elastisch erscheinen zu lassen, so ist es beispielsweise möglich Stoff oder schwingende Seile zu erstellen. Nach mehrstündiger Einarbeitung kam ich zu dem Schluss das diese Engine für unser Projekt nicht geeignet ist. Gründe hierfür sind:

- Zu Komplex: Die Einrichtung des Spielfeldes erwies sich als überaus schwierig. Gute Anleitungen für einfache Szenarien fehlten, die verfügbaren Anleitungen sind zu grundlegend beschrieben. Ebenfalls half die Anleitung der Engine nicht bei der Verwendung der einzelnen Komponenten.
- Die Positionierung der Objekte bezog sich immer auf den Mittelpunkt des Objektes, man kann beispielsweise kein Rechteck von (x,y) nach (x1,y1) erstellen sondern muss den Mittelpunkt und die Abmaße des Objektes angeben.
- Physik nicht immer korrekt. Die Engine sollte den Ball richtig von einer Ebene abprallen lassen, diese Engine allerdings lies den Ball teilweise an Plattformen abrollen obwohl

keine Schwerkraft vorhanden war. Ich schließe darauf das die Engine Reibungskräfte und vielleicht sogar Anziehungskräfte zwischen den Objekten herstellt. Für unser Projekt ist dies aber nicht zu gebrauchen.

- Schwer zu debuggen. Während meiner Versuche bin ich immer wieder auf Probleme gestoßen. Einige der Debugausgaben ließen sich gut ableiten und waren hilfreich. Allerdings bin ich auch auf einige Probleme gestoßen welche nicht in den Debugausgaben behandelt wurde. Meine letzten Versuche endeten alle darin das der Browser gecrasht ist aufgrund eines Memory-leaks.

4.1.2 Phaser.io

Phaser.io beschreibt sich selber als html5 Game Framework. Es wurde nach dem mobile-first Prinzip entwickelt und ist opensource. Die Entwicklung des gewünschten Prototypen erwies sich als sehr leicht, da es viele gute Beispiele gibt. Die Engine unterstützt von Haus aus eine Arcade-Physic, diese ist perfekt für unser Projekt. Sie beinhaltet Kollisions und Bewegungsfunktionen für den 2-Dimensionalen Raum

4.1.3 Erstellung eines Prototypen

In Phaser erstellt man ein Spiel über den Aufruf 'new Phaser.Game(...) die ersten Beiden Argumente geben die Dimensionen des Spielfeldes an, also die Weite und Höhe in Pixeln.
Der Nächste Parameter bestimmt die Render-Engine. Mögliche Werte sind 'Phaser.CANVAS', 'Phaser.WEBGL' oder 'Phaser.AUTO', wenn 'Phaser.AUTO' verwendet wird so probiert die Engine erst WebGL aus und für den Fall das der Browser WebGL nicht unterstützt wird Canvas verwendet.
Das nächste Argument gibt das Ziel im DOM an, wenn man diesen Parameter nicht setzt wird das Spiel einfach im Body angehängt.
Man kann als 5. Argument ein Startzustand angeben. Zustände kann man sich wie Spielszenen vorstellen. Ich habe mich dafür entschieden die Spielszene erst später hinzuzufügen, das bietet mir den Vorteil das ich diesem Zustand einen Namen geben kann und diesen Später erneut verwenden könnte.

Eine Szene bzw. einen Spielzustand kann man per game.state.add('name',State) vobei 'game' die Instanz des Spiels darstellt. Gestartet wird der Zustand per: 'game.state.start('name)' Ein Zustand ist wie folgt aufgebaut:

```
{
    preload: function () {

    },

    create: function () {

    },

    update: function () {

    },
};
```

Zu den einzelnen Funktionen:

- **preload:** Diese Funktion ist für das Vorladen von Assets gedacht. Beispielsweise Sprites oder Sounds werden hier vorgeladen damit während das Spiel läuft ohne Ladezeit zur Verfügung stehen.
- **create:** Hier werden alle Objekte erstellt die mit dem Beginn des Spielzustandes vorhanden sein sollen. Hier kann man auch Starteigenschafen wie Geschwindigkeit, Schwerkraft oder Ausrichtung setzen.
- **update:** Die update Funktion wird für die Berechnung jedes Frames aufgerufen. In dieser Funktion werden beispielsweise Kollisionen überprüft und darauf reagiert.

Für den Ball des Ping Pong Prototypen habe ich diese Funktionen wie folgt erstellt:

- **preload:**
  Laden des Sprites in den Namen 'ball':

```
game.load.image('ball','assets/testBall.png');
```

  Bekanntmachung der Ball Variable: this.ball

- **create:**
  Erstellen des Balls und einstellen des Ausrichtungspunktes in die Mitte des Sprites:

```
this.ball =
 game.add.sprite(game.world.centerX, game.world.centerY, 'ball');
this.ball.anchor.set(0.5, 0.5);
```

  Aktivieren der Physik für den Ball:

```
game.physics.startSystem(Phaser.Physics.ARCADE);
game.physics.enable(this.ball, Phaser.Physics.ARCADE);
```

  Als nächstes hab ich den Ball so konfiguriert das er mit den Spielfeldrändern kollidiert:

```
this.ball.checkWorldBounds = true;
this.ball.body.collideWorldBounds = true;
```

  Damit der Ball keine zusätzliche Geschwindigkeit beim Kollidieren mit einem anderem sich bewegendem Objekt erhält habe ich ihn 'unbeweglich' gemacht, damit erhält der Ball keine zusätzlichen Impulse von anderen Objekten.

```
this.ball.body.immovable = true;
```

  Und das er keine Geschwindigkeit verliert beim Abprallen:

```
this.ball.body.bounce.set(1);
```

  Als letztes musste ich dem Ball nur noch einen Startimpuls geben:

```
this.ball.body.velocity.setTo(200,0);
```

- **update:**
  Eine Update Funktion war nicht notwendig da der Ball im Prototypen nur mit dem Spielfeld kollidieren soll. Für die späteren Versionen muss hier die Kollision mit den Schlägern und anderen Objekten definiert werden.

Der fertige Prototyp sieht so aus:

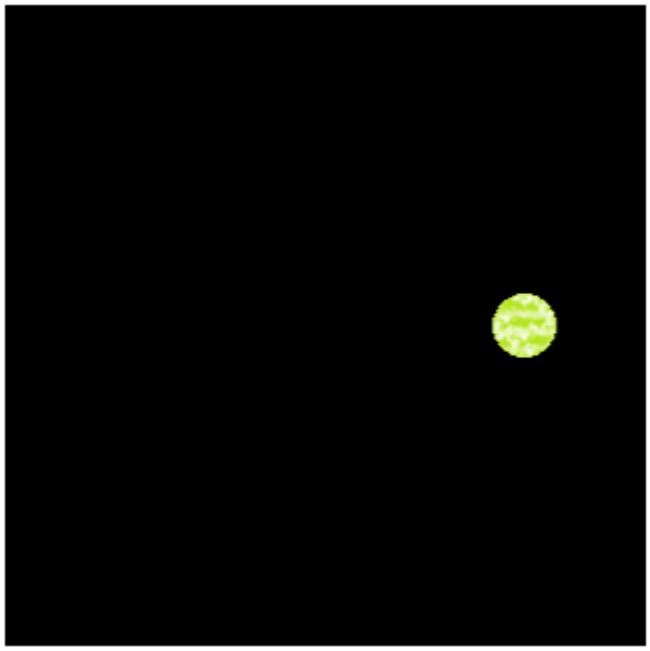

Abbildung 4.1: Erster Prototyp

See also ?.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 16. Dezember 2016 Andreas Müller, Claus Torben Haug, Jan-Dennis Bartels, Marjan Bachtiari