Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Unterstützung der automatischen Eye Tracking Datenanalyse mit Interaktionsdaten

Supporting Automatic Analysis of Eye Tracking Data with Interaction Data

Bachelorarbeit

im Studiengang Informatik

von

Judi Arafat

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Dr. Jil Klünder
Betreuer: M. Sc. Maike Ahrens

Hannover, 18. August 2020

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 18.08.2020

_____

Unterstützung der automatischen Eye Tracking Datenanalyse mit Interaktionsdaten

Zusammenfassung

Eye Tracking ist eine immer häufiger benutzte Technologie für die Analyse der kognitiven Wahrnehmung eines Nutzers. Eye Tracker ermöglichen eine Auswertung von Eye Tracking Aufzeichnungen auf sogenannten statischen Stimuli, welche betrachtete Objekte einer Eye Tracking Aufzeichnung beschreiben. Innerhalb der Stimuli können Areas of Interest (AOIs) definiert werden, welche für die Analyse von besonderem Interesse sind. Ein Problem entsteht jedoch bei der Interaktion eines Nutzers mit dem System während einer Aufzeichnung. Scrollt Nutzer beispielsweise innerhalb eines Stimulus, so werden die Koordinaten der AOIs innerhalb des Stimulus verschoben. Moderne Eye Tracker erkennen jedoch nicht, dass ein Nutzer während einer Aufzeichnung gescrollt hat, wodurch die Positionen der AOIs nicht angepasst werden. Wenn sich der Blickpunkt eines Nutzers nun zu einem bestimmten Zeitpunkt an einer Bildschirmposition befindet, könnte es sein, dass sich die Informationen an dieser Bildschirmposition zwischenzeitlich durch Benutzerinterkationen verändert haben. Somit müssen die Zeitpunkte der Benutzerinteraktionen nach einer vollständigen Eye Tracking Aufzeichnung ermittelt und die Positionen der AOIs schließlich manuell angepasst werden. Dieser Prozess erfordert jedoch einen hohen Zeitaufwand und kann aufgrund zeitlicher Ungenauigkeit zu einer falschen Datenanalyse führen. In dieser Arbeit wird eine Anwendung vorgestellt, welche anhand der Kombination von Eye Tracking Daten und verschiedener Logging-Tools eine automatische Datenauswertung dynamischer Stimuli ermöglicht. Mit der zeitlichen Erfassung von Scrolls, Edits, Stimuluswechsel und Veränderungen an den Applikationsfenstern durch Logging-Tools, kann die Anwendung schließlich die Verschiebung der Koordinaten von AOIs berechnen, um letztendlich die einzelnen Blickpunkte eines Nutzers den AOIs zuordnen zu können.

Supporting Automatic Analysis of Eye Tracking Data with Interaction Data

Abstract

Eye Tracking is an increasingly used technology for the comprehension of a user's cognitive perception. Eye Trackers allow an analysis of eye tracking recordings on so-called static stimuli, which describe the objects of interest in an eye tracking recording. Within the stimuli, Areas of Interest (AOIs) can be defined, which are of special interest for the analysis. However, a problem arises when a user interacts with the system during a recording. For example, if a user scrolls within a stimulus, the coordinates of the AOIs within the stimulus are shifted. Modern eye trackers do not recognize that a user has scrolled during a recording, so the positions of the AOIs are not adjusted. Therefore, if a user fixates a screen position at a certain point in time, it is possible that the information at this screen position has changed in the meantime due to user interaction. Consequently, the exact times of user interactions must be determined after a complete eye tracking recording and the positions of the AOIs must be adjusted manually. However, this process requires a lot of time and can lead to incorrect data analysis due to temporal inaccuracy. In this thesis a software is presented, which enables an automatic data analysis of dynamic stimuli by combining eye tracking data and different logging tools. The logging of scrolls, edits, stimuli switches and changes to the application windows during a recording allows the software to calculate the shift of AOI-coordinates, which finally leads to a complete mapping of gaze points to AOIs.

Inhaltsverzeichnis

1. Einleitung

1.1 Motivation

„Die kognitiven Prozesse eines Menschen ähneln einer ‚Blackbox', da sie nicht direkt messbar sind. Die Eye-Tracking-Technologie bietet die Möglichkeit, über die Augenbewegungen an diese Prozesse heranzukommen und gilt seit langer Zeit als vielversprechendes Erkenntnismittel der Wissenschaft." [1]

Die Eye Tracking Technologie ermöglicht es, die Augenbewegungen eines Menschen zu erfassen, aufzuzeichnen und hinsichtlich verschiedener Zwecke auszuwerten. Ein Anwendungsfall ist die Nutzung der Eye Tracking Technologie in dem Bereich des Software Engineerings: Einem Probanden werden eine Software und diverse Anleitungen (im Browser oder als PDF-Dateien) bezüglich der richtigen Benutzung der Software vorgelegt. Die Entwickler der Software möchten die Benutzbarkeit der entwickelten Software optimieren und verwenden aufgrund dessen einen Eye Tracker, um die Augenbewegung des Probanden aufzuzeichnen. Mittels der erhobenen Daten kann die kognitive Wahrnehmung des Probanden nachvollzogen werden und für eine ausführliche Analyse der Blickbewegungen genutzt werden, wodurch die Entwickler beispielsweise erkennen können, welche Bereiche der Software und welche Teile der Anleitung besonders lange von dem Probanden angeschaut wurden. Falls der Proband besonders lange einen bestimmten Teil der Anleitung betrachtet hat, spricht diese Erkenntnis für die Notwendigkeit der besseren Gestaltung der Funktionalität, welche in ebendiesem Teil der Anleitung beschrieben wurde, da der Proband möglicherweise Schwierigkeiten bei der Benutzung dieser Funktionalität hatte. Analog zu vielen anderen Anwendungsfällen bietet die Eye Tracking Technologie viele Möglichkeiten zur wissenschaftlichen Analyse in verschiedensten Branchen.

1.2 Problemstellung

Bei der Analyse von erhobenen Eye Tracking Daten werden häufig sogenannte *Areas of Interest* (AOIs) betrachtet. AOIs sind vordefinierte Areale innerhalb eines sogenannten Stimulus (z.B. einer Webseite oder eines PDF-Dokuments), die für den Analysierenden von besonderem Interesse sind. Für diese Bereiche auf dem Bildschirm sind Metriken,

wie beispielsweise die Blickdauer oder Blickhäufigkeit auf die jeweiligen AOIs von Interesse. AOIs können innerhalb einer Webseite Objekte wie ein Logo, eine Navigationsleiste, ein Video oder Ähnliches sein.

Das Problem, welches hierbei häufig vorkommt, ist, dass sich die AOIs an festen Bildschirmpositionen eines Computers befinden müssen, damit die x- und y- Koordinaten der Blickpunkte eines Probanden zu den jeweiligen AOIs automatisch zugeordnet werden können. Falls der Proband während der Testdurchführung scrollt oder zwischen verschiedenen Anwendungen wechselt, ist es der bereits existierenden Software nicht mehr möglich, die Blickpunkte eines Probanden automatisch den jeweiligen AOIs zuzuordnen, da sich die Positionen der AOIs durch die Benutzerinteraktion verschoben haben. Bei dem betrachteten Eye Tracker handelt es sich um einen stationären Tobii Pro Eye Tracker. Die Software Tobii Pro Lab bietet einige Analysetools für die Nutzung des Eye Trackers an.

Das Ziel dieser Arbeit ist es, eine Anwendung zu entwickeln, die eine automatische Datenauswertung von dynamischen Inhalten ermöglicht, indem Benutzerinterkationen während einer Eye Tracking Aufzeichnung automatisch erkannt und verarbeitet werden. Die zu entwickelnde Anwendung dient dem Zweck, den manuellen Analyseaufwand zu reduzieren, den Prozess der Verarbeitung von Benutzerinteraktionen zu automatisieren und zusätzlich eine bessere Qualität der Ausgabedateien im Vergleich zu einer manuellen Auswertung zu gewährleisten.

Zur Veranschaulichung der Problemstellung kann folgender Anwendungsfall betrachtet werden: Während einer Eye Tracking Aufzeichnung scrollt ein Proband innerhalb einer Applikation vertikal um 50 Pixel. Die Anwendung, welche zur Lösung dieser Problemstellung entwickelt werden soll, erkennt diese Benutzerinteraktion und verschiebt alle AOIs, welche sich innerhalb der betroffenen Anwendung befinden, um 50 Pixel auf der y-Achse. Würde die Anwendung diesen Prozess nicht automatisieren, müsste diese Benutzerinteraktion manuell erkannt und alle Positionen der verschobenen AOIs zum genauen Zeitpunkt der Interaktion angepasst werden. Bei einer längeren Eye Tracking Aufzeichnung entsteht somit ein sehr hoher manueller Aufwand und es könnte zu verfälschten Ergebnissen kommen, falls Benutzerinteraktionen bei der manuellen Auswertung übersehen werden.

1.3 Lösungsansatz

Um dieses Ziel zu erreichen, werden die Benutzerinteraktionen während eines Eye Tracking Experimentes aufgezeichnet. Dafür sind Logdateien für diese Arbeit als gegeben zu betrachten. Für die Erstellung dieser Logdateien können bereits vorhandene Logging-Tools genutzt oder eigene entwickelt werden. Diese erfüllen den Zweck, verschiedene Benutzerinteraktionen während einer Aufzeichnung zu dokumentieren. Zusätzlich als gegeben zu betrachten sind die Ausgabedateien der Eye Tracking Software, welche nach einer Eye Tracking Aufzeichnung ausgegeben werden können. Diese definieren die Dauer und Koordinaten der Blickpunkte und versehen jede Fixation mit einem Zeitstempel.

Mit den vorhandenen Informationen aus den Eingabedateien soll somit die Software für diese Arbeit entwickelt werden. Anhand der Koordinaten der Fixationspunkte und den Applikationszuständen aus den Logdateien soll die Software zuordnen können, in welcher AOI und Anwendung sich die jeweilige Fixation befindet. Die Software soll die gewonnenen Daten aufbereiten und diverse Metriken (z.B. Fixationsanzahl, Fixationsdauer, Verweildauer usw.), welche während der Programmausführung erfasst werden sollen, ausgeben. Zusätzlich werden die Ergebnisse in Form eines Gantt-Diagramms visualisiert. Das Gantt-Diagramm soll visualisieren, welche AOI zu welchem Zeitpunkt der Aufzeichnung angeschaut wurde.

1.4 Struktur der Arbeit

Die Arbeit ist wie folgt strukturiert: Kapitel 2 definiert und erläutert grundlegende Begriffe, welche für das Verständnis dieser Arbeit nötig sind. In Kapitel 3 werden funktionale, nicht-funktionale und qualitative Anforderungen an die Software erhoben und dokumentiert. Kapitel 4 befasst sich mit der Konzeptentwicklung einer Software unter Berücksichtigung der im vorherigen Kapitel definierten Anforderungen. In Kapitel 5 wird die Architektur der Anwendung beschrieben und zusätzlich die Umsetzung des Konzepts erläutert. Kapitel 6 evaluiert die entwickelte Anwendung und beschreibt die genutzten Testverfahren. In Kapitel 7 werden verwandte Arbeiten vorgestellt. In Kapitel 8 wird letztendlich ein Fazit gezogen und ein Ausblick auf mögliche und sinnvolle Erweiterungen an der entwickelten Software gegeben.

2. Grundlagen

In dem folgenden Kapitel werden einige Begriffe, Konzepte und nötiges Vorwissen zum Verständnis der Folgekapitel erläutert.

2.1 Eingabedateien

Für ein besseres Verständnis der Problemstellung ist es von Vorteil, zu verstehen, welche Eingaben die Anwendung entgegennehmen soll. Abbildung 1 gibt eine Übersicht über die Eingabedateien der Anwendung. Der Eye Tracking Processor beschreibt die zu entwickelnde Anwendung.

Die Eingabedateien sind hierbei in zwei Gruppen zu unterteilen. In Kapitel 2.1.1 werden die Ausgabedateien des Eye Trackers näher beschrieben, dahingegen befasst sich Kapitel 2.1.2 mit den Ausgabedateien der Logging-Tools.

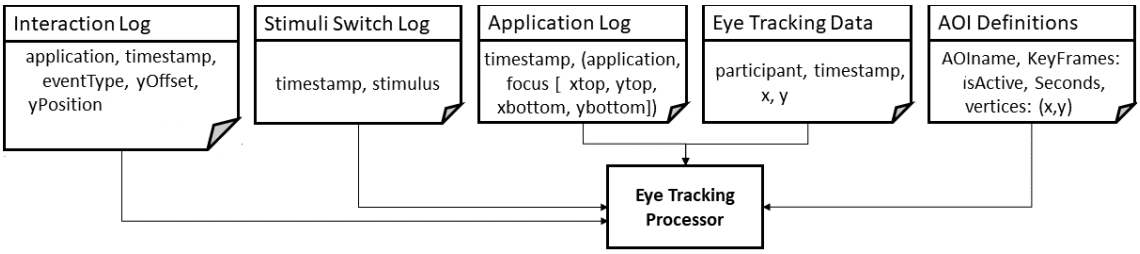

Abbildung 1: Eingabedateien der Anwendung

2.1.1 Eye Tracker Dateien

Die Dateien *Eye Tracking Data* und *AOI Definitions* aus Abbildung 1 sind die Ausgabedateien der Software des Eye Trackers nach einer fertigen Aufzeichnung.

Die *Eye Tracking Data* enthält eine Tabelle mit verschiedenen Informationen über jede Fixation während einer Aufzeichnung. Dabei wird jeder Fixation ein Index zugeordnet. Zusätzlich enthält die Tabelle Daten über Zeitpunkt, Dauer und x/y Koordinaten einer Fixation. Diese werden später benötigt, um den Blickpunkt einer AOI zuordnen zu können. Eine Zeile aus der Eye Tracking Datei sieht wie folgt aus (tsv Format):

Eye movement index	**Recording Timestamp (in ms)**	**Fixation point X (in px)**	**Fixation point Y (in px)**	**Gaze event duration (in ms)**
10	42497	1066	898	442

Tabelle 1: Eintrag aus einer Eye Tracking Datei

Die *AOI Definitions* Datei beinhaltet die Definitionen aller AOIs, die während einer Aufzeichnung erstellt wurden. Für jede AOI wird ein Name und eine Liste mit sogenannten Keyframes angelegt. Wenn eine AOI zum ersten Mal aktiv (sichtbar) wird, wird ein Keyframe dieser Liste hinzugefügt. Ein Keyframe enthält Informationen bezüglich des Zeitpunkts, zu dem der Keyframe erstellt wurde, Koordinaten der AOI zu ebendiesem Zeitpunkt und ein Wahrheitswert darüber, ob die AOI auf aktiv oder inaktiv gesetzt wurde. Dabei ist zu beachten, dass AOIs Vielecke (Polygone) sind und somit beliebig viele x/y-Koordinaten haben können. In Abbildung 2 ist beispielhaft die Definition einer AOI mit dem Namen „Logo“ zu sehen (JSON Format). Diese AOI wird ca. 9,6 Sekunden nach dem Beginn der Eye Tracking Aufzeichnung zum ersten Mal aktiv. Zusätzlich sind die Pixelkoordinaten der Eckpunkte dieser AOI gegeben.


```
{
    "Name": "Logo",
    "KeyFrames": [
        {
            "IsActive": true,
            "Seconds": 9.666805,
            "Vertices": [
                {
                    "X": 1251.73144876325,
                    "Y": 333.286219081272
                },
                {
                    "X": 1616.81978798587,
                    "Y": 333.286219081272
                },
                {
                    "X": 1616.81978798587,
                    "Y": 442.685512367491
                },
                {
                    "X": 1251.73144876325,
                    "Y": 442.685512367491
                }
            ]
        }
    ]
}
```


Abbildung 2: Eintrag aus einer AOI Definitions Datei

2.1.2 Logdateien

Die Dateien *Interaction Log*, *Stimuli Switch Log* und *Application Log* aus Abbildung 1 sind Ausgabedateien der Logging-Tools. Die Informationen aus den Logs werden während einer Aufzeichnung gesammelt. Bei den Zeitstempeln der Logdateien handelt es sich um Unix Timestamps (vergangene Sekunden seit dem 1. Januar 1970).

Das Interaction Log (Interaktionslog) enthält Informationen zu Scroll- und Edit-Events. Zu jedem Event werden der Zeitpunkt und die Distanz angegeben, die durch das Event verschoben oder gescrollt wurde. Zusätzlich wird die Position, an der ein Edit stattgefunden hat, angegeben (yPosition). Ein Edit-Event beschreibt beispielsweise das Hinzufügen oder Löschen einer Zeile aus einem Textdokument. Ein beispielhaftes Interaktionslog einer Word-Datei sieht wie folgt aus (csv Format):

Application	Timestamp	Event_Type	Offset (in px)	yPosition (in px)
Microsoft Word	1590434247	Scroll_Vertical	13	0
Microsoft Word	1590434259	Edit_Vertical	8	44

Tabelle 2: Interaktionslog einer Word-Datei

Das Stimuli Switch Log (Stimuluslog) gibt an, ob der betrachtete Stimulus einer Anwendung gewechselt wird. Stimuli sind betrachtete Objekte innerhalb von Anwendungen, zwischen denen gewechselt werden kann. Beispielsweise können Stimuli in einem Webbrowser gewechselt werden, indem zwischen verschiedenen Tabs hin- und hergewechselt wird. Falls der Proband während einer Aufzeichnung einen Stimuluswechsel vornimmt, so wird ein Eintrag im Stimuluslog erstellt. Dieser enthält den Zeitpunkt des Wechsels und den Namen des ab diesem Zeitpunkt betrachteten Stimulus. Das Stimuluslog eines Browsers könnte wie folgt aussehen (csv Format):

Application	Timestamp	Stimulus
Mozilla Firefox	1590434293	Tab1
Mozilla Firefox	1590434299	Tab2

Tabelle 3: Stimuluslog eines Browsers

Das Application Log (Applikationslog) gibt an, wenn sich während einer Aufzeichnung etwas an den sichtbaren Fenstern ändert (Fenster verschieben, Fenster minimieren o.ä.). Für jede Veränderung wird ein Eintrag im Applikationslog vorgenommen, in dem der Zeitpunkt der Veränderung und die Fensterhierarchie in absteigender Reihenfolge mit den jeweiligen Eckkoordinaten der Applikationen angegeben werden. Die Anwendung, welche zum Zeitpunkt des Eintrags im Fokus lag, wird ebenfalls markiert. Ein Applikationslog mit zwei Einträgen ist in Abbildung 3 zu sehen. Hier fokussiert der Proband vorerst seinen Browser und hat zusätzlich die Applikation Microsoft Word geöffnet. Nach 30 Sekunden wechselt der Proband zu der Word Applikation, weshalb ein neuer Eintrag in dem Applikationslog angelegt wird. Der neue Eintrag zeigt auf, dass sich nun Microsoft Word im Fokus befindet und bezüglich der Fensterhierarchie ganz oben liegt.

```
{
    "ApplicationLog": [
        {
            "Applications": [
                {
                    "ApplicationName": "Mozilla Firefox",
                    "Focus": true,
                    "Coordinates": {
                        "XTop": 0,
                        "YTop": 0,
                        "XBottom": 1920,
                        "YBottom": 1080
                    }
                },
                {
                    "ApplicationName": "Microsoft Word",
                    "Focus": false,
                    "Coordinates": {
                        "XTop": 960,
                        "YTop": 0,
                        "XBottom": 1920,
                        "YBottom": 1080
                    }
                }
            ],
            "Timestamp": 1590434160
        },
        {
            "Applications": [
                {
                    "ApplicationName": "Microsoft Word",
                    "Focus": true,
                    "Coordinates": {
                        "XTop": 960,
                        "YTop": 0,
                        "XBottom": 1920,
                        "YBottom": 1080
                    }
                },
                {
                    "ApplicationName": "Mozilla Firefox",
                    "Focus": false,
                    "Coordinates": {
                        "XTop": 0,
                        "YTop": 0,
                        "XBottom": 1920,
                        "YBottom": 1080
                    }
                }
            ],
            "Timestamp": 1590434190
        }
    ]
}
```


Abbildung 3: Applikationslog einer Aufzeichnung

Während einer Aufzeichnung können mehrere Applikationen geöffnet werden. Jede Applikation, die geöffnet wird, kann seinen eigenes Stimuluslog und/oder Interaktionslog haben. Die jeweilige Applikation, zu dem ein Log gehört, wird ebenfalls in den Logs vermerkt (siehe Tabelle 2 und Tabelle 3).

2.2 Ausgabemetriken

Für die Analyse einer Eye Tracking Aufzeichnung können viele verschiedene Metriken von Interesse sein. Im Folgenden wird beschrieben, welche Metriken von der zu entwickelnden Anwendung berechnet werden sollen. Abbildung 4 gibt eine Übersicht über die verschiedenen Ausgabedateien der Anwendung.

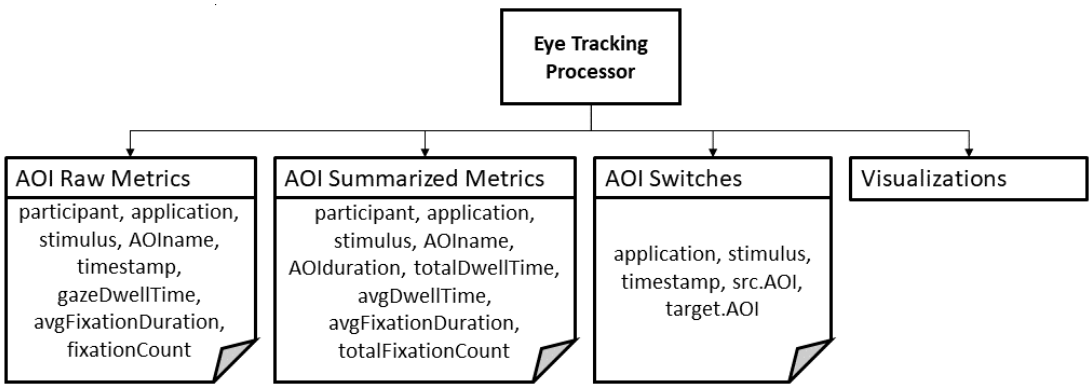

Abbildung 4: Ausgabedateien der Anwendung

Die *AOI Raw Metrics* sollen Informationen zu jedem AOI Visit liefern. Mehrere aufeinanderfolgende Fixationen in dieselbe AOI werden gruppiert und als AOI Visit definiert. Mittels der Raw Metrics können Beobachtungsmuster des Probanden im Verlauf der Aufzeichnung, besonders herausstechende AOIs oder Zeiträume, in denen der Proband in keine AOI geschaut hat, erfasst werden [2]. Für jeden AOI Visit soll der Zeitpunkt, der Name der AOI, die Applikation, der betrachtete Stimulus, die gesamte Verweildauer des AOI Visits, die Anzahl der Fixationen und die durchschnittliche Verweildauer der einzelnen Fixationen des Visits dokumentiert und ausgegeben werden.
Die *AOI Summarized Metrics* fassen die Metriken für jede AOI im Verlauf der gesamten Aufnahme zusammen. Hierfür sollen ebenfalls Metriken für jede AOI ausgeben werden: Wie lange die AOI im Verlauf der Aufzeichnung aktiv war, die gesamte Verweildauer der Blickpunkte in einer AOI, Anzahl der Fixationen innerhalb der AOI und die durchschnittliche Verweildauer der einzelnen Fixationen innerhalb der AOI.
Die *AOI Switches* geben an, wann zwischen AOIs gesprungen wurde. Der Wechsel eines Blickpunktes von einer AOI zu einer anderen soll erkannt und dokumentiert werden.
Letztendlich soll die Anwendung mit den erfassten Metriken eine geeignete Visualisierung ausgegeben. In Kapitel 4 wird genaueres bezüglich der Visualisierung beschrieben.

2.3 Programmierpraktiken

Im Folgenden werden zwei Programmierpraktiken vorgestellt, welche bei der späteren Umsetzung der Anwendung eine Rolle spielen.

2.3.1 Objektorientierte Programmierung

„Objektorientierte Programmierung ist eine Modellierungstechnik, mit der große Programme übersichtlicher, sicherer und wartbarer gestaltet werden können.“ [3]

Die Objektorientierung ermöglicht es, das Prinzip *divide and conquer* zu realisieren: Ein großes Problem wird in mehrere kleine Probleme aufgeteilt. Die Lösung aller kleinen, lösbaren Probleme führt zur Lösung des großen Problems. Dazu werden in der Objektorientierung sogenannte Objekte für die Umsetzung der einzelnen Probleme einer Software implementiert. Diese Objekte lassen sich mit den Objekten aus dem Alltag eines Menschen vergleichen. Jedes Objekt hat verschiedene Eigenschaften und Methoden, welche die richtige Benutzung eines Objekts suggerieren. Die wesentliche Aufgabe der objektorientierten Programmierung ist das Ermitteln und anschließende Implementieren der nötigen Eigenschaften und Methoden sogenannter *Klassen*. Die Klassen definieren eine Anleitung, um Objekte bzw. Instanzen aus dieser Klasse schaffen und verwalten zu können.

2.3.2 MVC-Pattern

Das Model View Controller Pattern unterteilt eine Software in drei Komponenten.

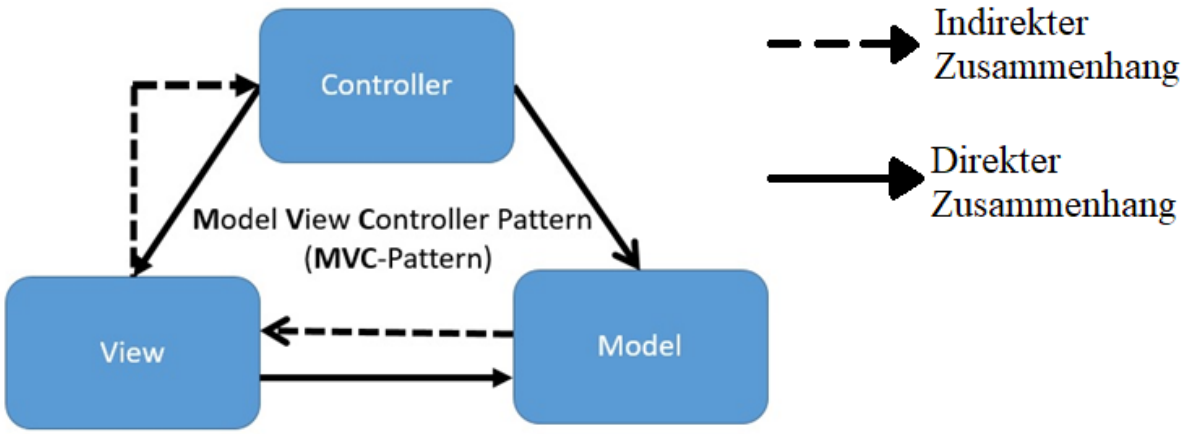

Abbildung 5: MVC-Pattern

Das **Model** beschreibt die Daten eines Programms, welche während des Programmablaufs verändert werden können. Daten können initial aus einer View durch

Interaktionen eines Benutzers mit dem System gewonnen werden. Die **View** beschreibt die grafische Benutzeroberfläche einer Anwendung. Durch diese kann ein Benutzer Einfluss auf die im Model und Controller verarbeiteten Daten nehmen. Zusätzlich hat die View die Aufgabe, gespeicherte Daten aus dem Model zu präsentieren. Letztendlich enthält der **Controller** die gesamte Programmlogik einer Anwendung zur Ausführung der Kernaufgabe. Diese muss die View und das Model verwalten, indem die von einem Nutzer übergebenen Daten aus einer View entgegengenommen werden. Die Daten werden im Controller verarbeitet und während des Programmablaufs durch Zugriff auf eine interne Datenspeicherungs- und Verwaltungslogik des Models gespeichert und bearbeitet [4].

3. Anforderungserhebung

In den folgenden Abschnitten werden die Anforderungen an die zu entwickelnde Software genannt. Die Formulierung der Anforderungen orientiert sich an den „Schablonen für alle Fälle“ von Chris Rupp und den SOPHISTen [5].
Im ersten Abschnitt werden die funktionalen Anforderungen aufgelistet und erläutert. Diese werden mit der Abkürzung „R“ gekennzeichnet. Im zweiten Abschnitt werden Nicht-funktionale Anforderungen an das System genannt, welche mit „NR“ bezeichnet werden.

3.1 Funktionale Anforderungen

Dieses Kapitel liefert einen grundlegenden Überblick über die wichtigsten funktionalen Anforderungen an die Anwendung.

[R01]: *Die Anwendung muss dem Benutzer die Möglichkeit bieten, mehrere Stimulus- und Interaktionslogs anzugeben.*
Jede Applikation kann einen eigenen Stimulus und/oder einen Interaktionslog besitzen. Falls während einer Aufzeichnung mehrere Applikationen geöffnet waren, entsteht die Möglichkeit, dass mehrere Stimulus- und Interaktionslogs zur Analyse des Datensatzes nötig sind.

[R02]: *Falls ein Applikationslog in dem übergebenen Datensatz existiert, muss die Anwendung den AOIs, welche sich innerhalb einer Applikation befinden, den Namen der Applikation zuweisen.*
Mittels der Koordinaten der AOIs und den Informationen aus dem Applikationslog sollen die AOIs der jeweiligen Applikation, in der sie sich befinden, zugeordnet werden.

[R03]: *Falls bei dem übergebenen Datensatz Applikationen existieren, für die keine Logs angelegt wurden, muss die Anwendung anhand der Ausgabedateien des Eye Trackers ermitteln, ob ein Blickpunkt innerhalb der AOIs in ebendiesen Applikationen liegt.*
Hat eine Applikation keinen Stimulus- oder Interaktionslog, so kann die Anwendung davon ausgehen, dass die Positionen und die Verschiebung der AOIs innerhalb dieser Applikationen nach einer Eye Tracking Aufzeichnung manuell mitverfolgt wurden. Für

diese AOIs muss die Anwendung für die Zuordnung der Blickpunkte zu den AOIs die Datentabelle der Eye Tracking Datei heranziehen.

[R04]: *Die Anwendung muss die Events aus den Logs der zeitlichen Reihenfolge nach simulieren und die daraus resultierenden Auswirkungen auf die Positionen und Sichtbarkeit der AOIs ermitteln.*
Diese Anforderung beschreibt die Kernaufgabe der Anwendung. Benutzerinteraktionen können die Positionen der AOIs verschieben. Darunter können Nutzer beispielsweise Fenster verschieben oder innerhalb von Applikationen scrollen und editieren, wodurch eine Veränderung der Positionen bzw. der Sichtbarkeit von AOIs resultieren. Die Anwendung muss die Veränderungen auf die AOIs berechnen und zwischenspeichern.

[R05]: *Falls die Position einer AOI manuell angepasst wurde, muss das System die neuen Koordinaten der AOI übernehmen.*
Es ist möglich, dass die Position einer AOI, für dessen Applikation Logdateien existieren, dennoch manuell angepasst wird. Die Anwendung muss dies erkennen und die neue Position der AOI abspeichern.

[R06]: *Die Anwendung muss dem Benutzer die Möglichkeit bieten, nur die Eye Tracking Ausgabedateien zu übergeben.*
Diese Anforderung ist eine Erweiterung von [R05]. Die Anwendung soll die Möglichkeit bieten, nur die Eye Tracking Dateien entgegenzunehmen. Das System soll somit erkennen, dass keine Logs in dem übergebenen Datensatz existieren und schließlich davon ausgehen, dass die Positionen aller AOIs manuell verfolgt wurden. In solch einem Anwendungsfall sollen die Blickpunkte nach demselben Prinzip wie in [R03] beschrieben den AOIs zugeordnet werden.

[R07]: *Die Anwendung sollte dem Benutzer die Möglichkeit bieten, einen Pfad anzugeben, in dem die Ausgabedateien gespeichert werden.*
Damit ein Benutzer die Ausgabedateien nach der Ausführung wiederfindet, kann dieser einen Pfad zu einem Ordner angeben, in dem die Dateien gespeichert werden.

[R08]: *Die Anwendung muss die Zeitstempel der Eye Tracking Dateien mit den Zeitstempeln der Logdateien synchronisieren.*
Bei der Zuweisung der Blickpunkte zu den AOIs ist zu beachten, welche AOIs zu dem Zeitpunkt des Blickpunktes aktiv (sichtbar) sind und wo sich diese befinden. Um dies zu bewerkstelligen, muss die Anwendung am Anfang der Programmausführung die Zeitstempel der Eye Tracking- und Logdateien synchronisieren.

[R09]: *Falls es Fehler bei der Ausführung der Anwendung gab, muss das System dem Benutzer eine Mitteilung anzeigen.*
Dem Nutzer sollte bei Fehlern in der Programmausführung eine ausführliche Fehlerbeschreibung geliefert werden, damit ein Nutzer schnell den Ursprung des Fehlers erkennen kann.

[R10]: *Die Anwendung muss für jede Applikation eine AOI „whitespace“ erstellen und tracken.*
Blickpunkte, die keiner AOI zugeordnet werden, sollen zwecks einer späteren Datenanalyse nicht vernachlässigt werden. Liegt ein Blickpunkt innerhalb keiner AOI, so soll die Anwendung diesen Blickpunkt in eine manuell erstellte AOI „whitespace“ eintragen. Jede Applikation erhält eine eigene „whitespace“ AOI, sodass diese Blickpunkte einer Applikation zugeordnet werden können. Im Nachhinein kann somit nachvollzogen werden, wie oft, wie lang oder zu welchen Zeitpunkten ein Proband Blickpunkte fixiert hat, die in keiner AOI lagen.

[R11]: *Die Anwendung muss vordefinierte Metriken ausgeben, die während der Ausführung berechnet werden.*
Zweck der Programmausführung ist das Sammeln und Berechnen von Metriken für die Analyse der Eye Tracking Aufzeichnung. Interessant für die Analyse sind Metriken, wie beispielsweise die Fixationshäufigkeit einer AOI, die Dauer der Sichtbarkeit einer AOI oder die durchschnittliche Fixationsdauer einer AOI. Eine detaillierte Beschreibung der Ausgabemetriken befindet sich im Kapitel 2.2.

[R12]: *Die Anwendung muss die Ausgabemetriken im csv-Dateiformat ausgeben.*
Die Ausgabedateien sollen im csv-Dateiformat ausgeben werden, damit diese einfach und schnell in Excel importiert werden können.

[R13]: *Die Anwendung muss eine Visualisierung der gesammelten Daten ausgeben, welche den zeitlichen Verlauf der angeschauten AOIs darstellt.*
Die berechneten Metriken sollten mittels einer geeigneten Visualisierung veranschaulicht werden, sodass auch hier weiterer Raum für eine Analyse der Eye Tracking Aufzeichnung geschaffen wird.

3.2 Nicht-funktionale Anforderungen

Im Folgenden werden die Nicht-funktionalen Anforderungen an die Anwendung genannt. Bei den Anforderungen NR02-NR04 handelt es sich explizit um Qualitätsanforderungen.

[NR01]: *Die GUI für das Übergeben der Eingabedateien sollte schnell benutzbar sein.*
Die GUI dient einzig dem Übergeben der Log- und Eye Tracker Dateien und sollte aus diesen Gründen schnell und einfach bedienbar sein. Da sich die Benutzergruppe im Wesentlichen aus Personen zusammensetzt, die im technischen Bereich tätig sind, sind Usability-Optimierungen zu vernachlässigen.

[NR02]: *Die Anwendung sollte korrekt funktionieren.*
Die Korrektheit der Software in Bezug auf mögliche Randfälle sollte einwandfrei sein, sodass die Anwendung zur Analyse von erstellten Datensätzen genutzt werden kann.

[NR03]: *Bei der Implementierung sollte das Konzept der Modularität angewendet werden.*
Zwecks einer übersichtlichen und flexiblen Softwarearchitektur sollte modulare Programmierung angewendet werden.

[NR04]: *Die Anwendung sollte wartbar sein.*

Die Anwendung bietet viele mögliche Erweiterungen an (Kapitel 8.2). Um eine weitere Entwicklung ermöglichen und fördern zu können, sollten die in der ISO/IEC 25010[1] definierten Attribute zur Realisierung der Wartbarkeit umgesetzt werden.

[1] https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

4. Konzeptentwicklung

In diesem Kapitel wird die Entwicklung eines Konzepts für die Umsetzung der im vorherigen Kapitel genannten Anforderungen diskutiert. Dazu wird zunächst im Kapitel 4.1 ein Konzept für die Umsetzung der funktionalen Anforderungen entwickelt. Im Kapitel 4.2 wird die Anwendung sinnvoller Programmierpraktiken für die Umsetzung der nichtfunktionalen Anforderungen beschrieben.

4.1 Funktionale Anforderungen

Da das Programm bestimmte Eingabedateien fordert, bietet es sich an, eine GUI für die Problemstellung zu entwickeln (R01, R07). Diese soll den Zweck erfüllen, die Eingabedateien entgegenzunehmen und anschließend die Programmausführung zu starten. Abschließend sollte die GUI eine Meldung darüber ausgeben, ob die Programmausführung erfolgreich oder fehlerhaft war (R09).

Aus den funktionalen Anforderungen ist zu erkennen, dass zwei verschiedene Anwendungsfälle der Software beachtet werden müssen. Der erste Anwendungsfall beschreibt das entgegennehmen eines Datensatzes mit entsprechenden Eye Tracking- und Logdateien (R01). Die Anwendung sollte jedoch auch fähig sein, nur die Eye Tracking Dateien entgegenzunehmen und ohne weitere Logs eine vollständige Datenauswertung auszuführen (R06). Im ersten Fall müssen Events aus den Logs beachtet werden; im zweiten Fall sollen Blickpunkte jedoch nur mittels der gegebenen Informationen aus den Eye Tracking Dateien AOIs zugeordnet werden. Die Anwendung sollte diese Fallunterscheidung beachten und zwei mögliche Programmausführungen zur Verfügung stellen. Jedoch ist auch im ersten Anwendungsfall zu vermerken, dass einige Applikationen nicht geloggt wurden. Hier muss die Anwendung ebenfalls erkennen, für welche Applikationen keine Logs vorliegen. Für solche Applikationen müssen die Blickpunkte den AOIs, welche innerhalb dieser Applikationen liegen, mittels der Eye Tracker Dateien zugeordnet werden (R03).

Durch die Anforderung R02 muss jede AOI der Applikation zugewiesen werden, in welcher sich die AOI befindet. Für jede AOI wird in den AOI-Definitionen ein Keyframe angelegt, welcher definiert, wann eine AOI zum ersten Mal sichtbar wird. Optimalerweise können weitere Keyframes folgen, die die Koordinaten einer AOI nochmals manuell verändern; jedoch besteht die Möglichkeit, dass dieser Fall für einige AOIs nicht eintritt.

Die Zuweisung der AOI zu einer Applikation sollte jedoch genau zum Zeitpunkt erfolgen, zu dem eine AOI das erste Mal sichtbar wird. Nur zu diesem Zeitpunkt darf das System davon ausgehen, dass die Position der AOI korrekt ist und nicht durch Benutzerinteraktionen verschoben wurde. Demensprechend sollte die Anwendung zurückrechnen, welche Applikationen zu der Zeit geöffnet waren und schließlich abfragen, innerhalb welcher Applikationsgrenzen sich die AOI befindet.
Um die verschiedenen Events aus den Logs zu simulieren, bietet es sich an, diese vorerst der zeitlichen Reihenfolge nach abzuspeichern und anschließend zu simulieren. Für jedes Event im Applikations-, Stimulus- und Interaktionslog einer Applikation könnte die Anwendung einen neuen Keyframe für jede AOI innerhalb dieser Applikation anlegen. Für den erstellten Keyframe sollte anschließend berechnet werden, ob die AOI nach dem Event noch sichtbar ist und falls dies der Fall ist, muss die durch das Event entstandene Verschiebung der Koordinaten der AOI berechnet werden (R04).
Die Berechnung der Ausgabemetriken (Kapitel 2.2) sollte während der Zuweisung von Blickpunkten zu AOIs erfolgen. Da in diesem Schritt über jeden Blickpunkt iteriert werden muss, kann leicht überprüft werden, ob ein AOI Switch stattgefunden hat oder ob ein AOI Visit begonnen hat bzw. beendet wurde (R11).
In der funktionalen Anforderung R13 wird eine Visualisierung der ermittelten Daten gefordert, welche den zeitlichen Verlauf der angeschauten AOIs im Laufe der Eye Tracking Aufzeichnung darstellen soll. Hierfür ist das sogenannte „AOI Sequence Chart“ sehr praktisch. Das AOI Sequence Chart visualisiert Reihenfolge und Verweildauer der einzelnen Fixationen in den jeweiligen AOIs [6]. In Kapitel 6 ist ein Sequence Chart abgebildet, welches von der Anwendung erstellt wird.

Aufgrund der oben beschriebenen Überlegungen bietet es sich an, die Anwendung in vier Teile zu unterteilen. In einem ersten Schritt sollte eine GUI für die Übergabe der Daten ausgeführt werden. Nachdem die Programmausführung von der GUI eingeleitet wurde, empfiehlt es sich, in einem zweiten Schritt alle AOIs zu initialisieren und die Events aus den Logdateien zu sammeln. Diese gesammelten Events sollten anschließend simuliert und die Positionen der AOIs, welche von dem Event betroffen sind, angepasst werden. Da nun die Auswirkungen der Events auf die Positionen der AOIs berechnet und abgespeichert wurden, sollte in einem dritten Schritt die Zuweisung von Blickpunkten zu den AOIs erfolgen. In diesem Schritt sollte die Anwendung außerdem die Metriken zu

den AOI Visits und AOI Switches sammeln. Im vierten und letzten Schritt kann die Anwendung die erfassten Metriken auswerten und in Form von csv-Dateien ausgeben (R12). Eine detaillierte Beschreibung der Umsetzung dieses Konzepts ist in Kapitel 5 vorzufinden.

4.2 Nicht-funktionale Anforderungen

Die objektorientierte Programmierung bietet sich gut für die Umsetzung der Anforderungen NR03 und NR04 an. Die Software wird durch die einzelnen Klassen in Module aufgeteilt, wobei eine klare Unterscheidung und Trennung der Klassen – besonders durch eine daraus resultierende vereinfachte Verständlichkeit der Softwarestruktur – einen positiven Einfluss auf die Wartbarkeit der Anwendung hat.

Da das System eine GUI besitzen sollte, eine Programmlogik vorhanden sein muss und während der Programmausführung Daten abgespeichert und manipuliert werden müssen, bietet sich die Verwendung des MVC-Patterns ebenfalls gut für die Umsetzung der geforderten Wartbarkeit an (NR04). Somit können Änderungen oder Erweiterungen an einzelnen Teilen der Anwendung vorgenommen werden, ohne dass daraus negative Auswirkungen hinsichtlich eines anderen Teils der Anwendung resultieren [4].

Weitere Vorteile für die Realisierung der in Kapitel 3.2 genannten Qualitätsanforderungen bringt die Programmiersprache Python mit sich. Aufgrund einer minimalistischen Programmiersyntax ist der Programmcode einfach lesbar und bietet daraus folgernd eine gute Wartbarkeit an [7], womit die Qualitätsanforderung NR04 gewährleistet werden kann. Zusätzlich lässt sich das Konzept der objektorientierten Programmierung in Python anwenden. Dies ermöglicht die Umsetzung der in NR03 geforderten Modularität.

Letztlich lässt sich die Korrektheit der Anwendung (NR02) nur durch ausführliches Testen des Programmcodes realisieren. Die Evaluierung der Anwendung wird in Kapitel 6 ausführlich beschrieben.

5. Implementierung

Dieses Kapitel befasst sich mit der Umsetzung der in Kapitel 3 gesammelten Anforderungen. Dazu soll das in Kapitel 4 entwickelte Konzept realisiert werden. Als Programmiersprache wurde aufgrund der in Kapitel 4 genannten Vorteile Python verwendet.

Kapitel 5.1 beschreibt, welche Frameworks und Bibliotheken während der Implementierung genutzt wurden. In Kapitel 5.2 wird die Architektur der Software mittels eines UML-Klassendiagramms dargestellt und erläutert. Kapitel 5.3 befasst sich mit der Implementierung des Lösungsansatzes.

5.1 Verwendete Frameworks

Unabhängig von den internen Python-Bibliotheken wurden im Wesentlichen zwei weitere externe Bibliotheken genutzt. Für die Implementierung der GUI wurde das Framework *PySide2*[2] genutzt. PySide2 bindet das plattformübergreifende Tool Qt – eines der meistgenutzten Bibliotheken für die Entwicklung von Benutzeroberflächen – in Python ein. Die vielen Vorteile des Qt-Frameworks können somit in Verbindung mit der einfachen Syntax von Python genutzt werden, sodass simple GUI Anwendungen geschrieben werden können [8].

Des Weiteren wurde die Bibliothek *Shapely*[3] verwendet, die es ermöglicht, verschiedenste geometrische Objekte zu erstellen und zu verwalten. In der Entwicklung der Anwendung spielt diese Bibliothek eine wichtige Rolle: Wie in den Grundlagen erläutert, können AOIs beliebige Vielecke sein und müssen während der Programmausführung erstellt, bearbeitet, verändert und gelöscht werden. Die Nutzung der Shapely-Bibliothek übernimmt diese Aufgaben, wodurch die Verwaltung der AOIs in Form von Polygonen vereinfacht wird [9]. Zusätzlich kann durch die Verwendung der Bibliothek ermittelt werden, welche Teile der Applikationen aus dem Applikationslog unter Beachtung der Fensterhierarchie sichtbar sind. So ist es mit diesem Tool möglich, den Fall abzudecken, der eintritt, wenn ein Teil eines sichtbaren Fensters von einem übergelegenen Fenster überdeckt wird. In solch einem Fall muss bestimmt werden, ob

[2] https://doc.qt.io/qtforpython/api.html
[3] https://pypi.org/project/Shapely/

das untere Fenster – in Bezug auf die Hierarchie – gänzlich überdeckt wurde und nicht mehr sichtbar ist. Wurde nur ein Teil des Fensters abgeschnitten, so müssen die Koordinaten des Teilfensters angepasst werden, sodass während der Programmausführung ermittelt werden kann, in welchen Applikationen die AOIs liegen. Eine interne Bibliothek, welche dennoch nennenswert ist, ist die *Matplotlib*[4] Library. Diese dient der Visualisierung von statischen und dynamischen Inhalten [10]. Mittels dieser Bibliothek wird die in R13 geforderte Visualisierung umgesetzt. Dabei wurde diese Bibliothek außerdem intensiv für das Testen und Evaluieren des Programmcodes genutzt. Durch das plotten der verschiedenen Keyframes einer AOI, welche durch Benutzerinteraktionen entstanden sind, konnte getestet werden, ob die Anwendung korrekt funktioniert. Eine weiterführende Beschreibung des Testverfahrens wird in Kapitel 6 beschrieben.

5.2 Architektur

Die *Unified Modeling Language* (UML) dient der Visualisierung einer Softwarearchitektur. Dabei können die verschiedenen Klassen bzw. Objekte einer objektorientierten Anwendung grafisch dargestellt werden. Die Beziehungen und das Verhalten der verschiedenen Klassen werden mittels eines UML-Klassendiagramms präsentiert. Somit können Zusammenhänge und Informationsflüsse zwischen den Objekten nachvollzogen und in einem weiteren Schritt implementiert werden [11].
Im Folgenden wird in Abbildung 6 eine abstrahierte Version des entwickelten UML-Klassendiagrammes präsentiert, welches ermöglicht, die Anforderungen aus Kapitel 3 umzusetzen und zu diesem Zweck das in Kapitel 2.3.2 beschriebene MVC-Pattern nutzt.

[4] https://matplotlib.org/

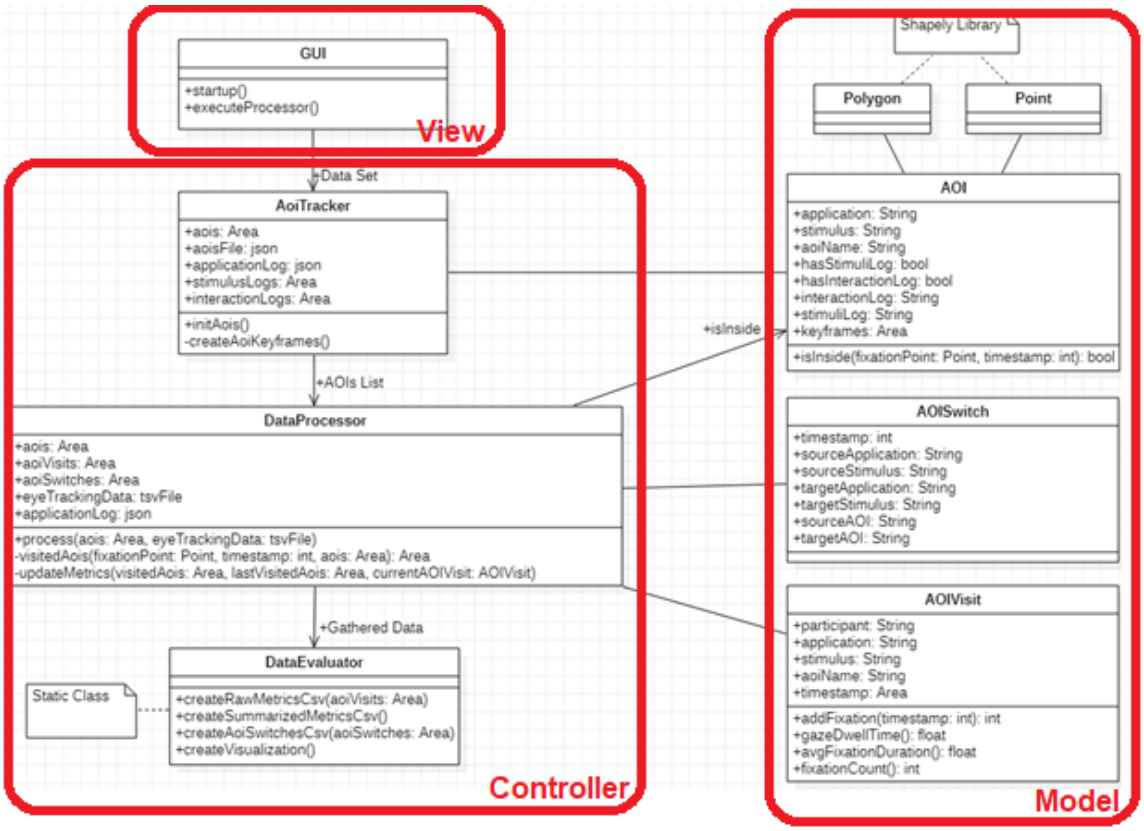

Abbildung 6: UML-Klassendiagramm mit Unterteilung in Model-View-Controller

Für die Initialisierung der Software soll zuerst eine GUI ausgeführt werden. Diese ist die **View** der Anwendung, jedoch ist in diesem Fall eine Differenzierung zwischen zwei verschiedenen Views zu beachten. Zum einen existiert die View aus Abbildung 6, welche Benutzerdaten, also einen Datensatz mit den Eye Tracking Daten und (optional) Logdateien entgegennimmt und anschließend dem Controller übergibt. Des Weiteren ist der Dateibrowser des Betriebssystems als eine View dieser Software zu beachten, welche jedoch nicht in dem Klassendiagramm abgebildet ist, da diese als gegeben zu betrachten ist. In dem Dateibrowser werden die erstellten Dateien mit den Ausgabemetriken, also den gesammelten Daten aus dem Model, ausgegeben.

Das **Model** der Anwendung dient der Abspeicherung der in dem Controller berechneten Daten. Hierbei ist das Erstellen und Abspeichern der Ausgabemetriken und AOIs in jeweilige Objekte von Interesse. In der vorliegenden Architektur wurden dazu die Klassen AOI, AOISwitch und AOIVisit definiert.

In dem **Controller** der Anwendung wird die Hauptfunktionalität der Anwendung ausgeführt. Die Programmausführung besteht im Wesentlichen aus vier Klassen, welche

in Abbildung 6 abstrahiert definiert werden. Die Funktionsweise und Benutzung der einzelnen Klassen wird im Folgekapitel näher beschrieben.

5.3 Umsetzung

Der Controller soll die Problemstellung dieser Arbeit, welche in Kapitel 1.2 beschrieben wurde, lösen. Um dies zu bezwecken, müssen die in Kapitel 3 definierten Anforderungen umgesetzt werden.

Der Controller, welcher die Hauptfunktionalität der Software ausführt, ist in vier Teile aufgegliedert. Abbildung 7 stellt die verschiedenen Abschnitte der Programmausführung dar und deklariert, zu welcher Klasse des UML-Klassendiagramms (Abbildung 6) der jeweilige Abschnitt gehört. Die einzelnen Schritte werden im Text mit der jeweiligen Nummerierung aus der Abbildung referenziert. Im Folgenden wird eine kurze Beschreibung über die Implementierung der einzelnen Phasen geliefert.

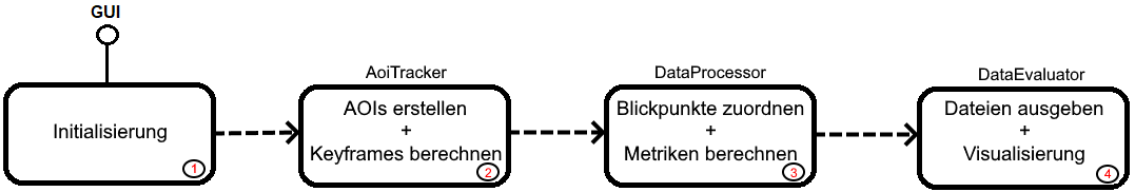

Abbildung 7: Funktionsablauf des Controllers

Im ersten Schritt (①) sind einige Tasks für die Initialisierung des Programms abzuarbeiten. Anfangs werden die Zeitstempel der Logdateien und Eye Tracking Dateien synchronisiert (R08). Dazu sind Datum und die Uhrzeit des Beginns der Eye Tracking Aufzeichnung aus den Dateien zu lesen. Diese sollen anschließend in einen Unix Timestamp umgerechnet werden. Des Weiteren wird das Applikationslog ausgelesen und für jeden Eintrag im Applikationslog (also für jede Veränderung an den sichtbaren Fenstern) der sichtbare Teil jeder Anwendung berechnet und abgespeichert. Mit den berechneten Daten kann später ermittelt werden, in welchen Applikationen sich die AOIs aus den Eye Tracking Dateien befinden.

Im zweiten Schritt (②) wird über jede definierte Area of Interest aus den Eye Tracking Daten iteriert. Für jeden Eintrag in den AOI-Definitionen soll ein Objekt der Klasse *AOI* erstellt werden, in welcher die im folgenden definierten Daten abgespeichert werden sollen. Anhand der Koordinaten der AOI, dem Zeitpunkt des ersten Keyframes (also der

Zeitpunkt, zu dem die AOI als erstes sichtbar ist) und den im ersten Schritt berechneten Positionen der sichtbaren Fenster, wird ermittelt, in welcher Applikation die AOI liegt (R02). Falls diese Applikation einen Stimuluslog besitzt, soll zusätzlich der Stimulus, in dem sich die Applikation zu diesem Zeitpunkt befindet, ermittelt werden (R03). Anschließend werden alle Events aus den Logs der Anwendung, in welcher sich die AOI befindet, gesammelt und zeitlich sortiert. Die einzelnen Events sollen dann simuliert werden. Für ein Event im Stimuluslog muss kontrolliert werden, ob der neue Stimulus dem Stimulus entspricht, in dem sich die AOI befindet. Bei Events in einem Interaktionslog müssen die Koordinaten der AOI abhängig von dem durch das Event entstandene Offset verschoben werden. Daraufhin muss zusätzlich geprüft werden, ob ein Teil der AOI durch die Verschiebung abgeschnitten wurde. Dies kann der Fall sein, falls sich ein Teil der AOI nicht mehr innerhalb der Ränder der Applikation befindet oder über den definierten Bildschirmrand „hinausragt“. Schließlich muss bei einem Event in dem Applikationslog die Position der AOIs angepasst werden, falls die Anwendung verschoben wurde. Analog zu den Interaktionsevents muss dann die Sichtbarkeit der AOI nochmals geprüft werden. Die ermittelten Daten werden in einem neuen Eintrag in der Keyframes-Liste der AOI gespeichert (R04). Ein weiteres Event, welches eintreten kann, betrifft die manuelle Definition einer neuen Keyframe. In solch einem Fall muss die manuell definierte Keyframe der Keyframes-Liste der AOI angefügt werden (R05). Dabei ist zu beachten, dass eine AOI auf inaktiv gesetzt werden kann. In solch einem Fall muss bei Folgeevents beachtet werden, dass die AOI auf inaktiv gesetzt wurde. Falls die Anwendung jedoch keinen Applikationslog finden sollte, wird das zuweisen der AOIs zu den jeweiligen Applikationen und das Sammeln der Events aus Interaktions- und Stimuluslog übersprungen (R06). In solch einem Fall soll die Anwendung nur die manuell definierten Keyframes aus den Eye Tracking Dateien auslesen und in dem AOI Objekt abspeichern.

In einem dritten Schritt (③) sollen die Fixationspunkte aus den Eye Tracking Daten den jeweiligen AOIs zugeordnet werden, deren Keyframes im vorherigen Schritt berechnet wurden. Dazu müssen die Eye Tracking Daten zunächst vorgefiltert werden, da viele Zeilen (z.B. die Kalibrierung des Eye Trackers) irrelevant sind. Anschließend wird über jeden Fixationspunkt iteriert. Für alle AOIs muss dann abgefragt werden, ob der Fixationspunkt innerhalb des Keyframes liegt, welcher zum Zeitpunkt des Fixationspunktes die Koordinaten der AOI repräsentiert hat. Hierbei ist jedoch zu

unterscheiden zwischen AOIs, deren Positionen durch Logdateien mitverfolgt wurden und AOIs, welche manuell mitverfolgt wurden (Applikation der AOI hat keine Logs oder es wurden nur Eye Tracking Dateien übergeben). Für den ersteren Fall muss der Keyframe zum Zeitpunkt des Blickpunktes ermittelt und lediglich abgefragt werden, ob sich der Blickpunkt innerhalb der Eckpunkte des Keyframes befindet. Für manuell mitverfolgte AOIs müssen jedoch die Ausgabedateien des Eye Trackers herangezogen werden. Für jeden Blickpunkt in den Eye Tracking Dateien (siehe Kapitel 2.1.1) werden sogenannte „AOI hit“ Spalten dokumentiert. Diese ermitteln für jede AOI, ob der Blickpunkt aus der Zeile innerhalb der jeweiligen AOI liegt. Beispielsweise sagt die Spalte „AOI hit[Logo] = 1“ aus, dass der Blickpunkt der Zeile innerhalb der AOI mit dem Namen „Logo“ liegt. Falls ein Fixationspunkt innerhalb keiner der AOIs lag, wird die Anwendung diesen Fixationspunkt einer manuell definierten AOI „whitespace“ zuordnen, damit bei einer späteren Inspektion der Ausgabedaten nachvollzogen werden kann, wie lange, wie oft und zu welchen Zeitpunkten ein Proband in keine AOI geschaut hat (R10). Während der Zuweisung von den Blickpunkten zu den AOIs, werden die Metriken zu den AOI Visits und AOI Switches gesammelt und berechnet. Ein AOI Switch entsteht, falls die AOI des letzten Fixationspunktes nicht der AOI der derzeitigen Fixation entspricht. In solch einem Fall wird ein Objekt der Klasse *AOISwitch* (siehe Abb. 4) erstellt, in der die nötigen Daten abgespeichert werden. Das Objekt wird dann einer Liste mit allen während des Programmablaufs erstellten AOISwitch Objekten angefügt. Analog gilt es, die AOI Visits zu betrachten. Falls die AOI des letzten Fixationspunktes nicht der AOI der derzeitigen Fixation entspricht, soll ein neues Objekt der Klasse *AOIVisit* (siehe Abb. 4) angelegt werden. Falls der darauffolgende Fixationspunkt wieder in derselben AOI liegt, soll das derzeitige AOIVisit Objekt aktualisiert werden, indem die Fixation dem Objekt angehängt wird. Ein AOI Visit endet zu dem Zeitpunkt der ersten Fixation außerhalb dieser AOI. Mit den gewonnenen Daten sollen abschließend die geforderten Metriken berechnet und dem letzten Abschnitt des Programms übergeben werden.

Im vierten und letzten Schritt (④) werden letztlich die gesammelten Metriken im CSV-Dateiformat ausgegeben (R12). Dazu müssen lediglich die erstellten AOIVisit und AOISwitch Objekte ausgelesen werden. Mittels der Matplotlib und den erfassten Daten wird das in Kapitel 4.1 beschriebene Sequence Chart geplottet, welches im PDF-Format ausgegeben wird (R13). Die erstellten Dateien werden in dem vom Nutzer angegebenen Dateipfad gespeichert (R07). Der ausgewählte Dateipfad und eine Meldung, falls es

Fehler während der Ausführung gab, werden dem Nutzer in der GUI als Meldung angezeigt (R09).

6. Evaluierung

Im Folgenden wird die entwickelte Anwendung evaluiert. Im ersten Unterkapitel wird die angewandte Testmethodik für das Evaluieren der Software beschrieben. In einem weiteren Absatz wird mittels der beschriebenen Testmethodik schließlich überprüft, ob alle Anforderungen an die Anwendung erfüllt wurden.

6.1 Testmethodik

Für die gegenwärtige Problemstellung bietet sich eine Visualisierung von Systemzuständen sehr gut für denkbare Tests an. So können zu einem bestimmten Zeitpunkt alle sichtbaren AOIs und Applikationen visualisiert werden, um zu kontrollieren, ob der visualisierte Systemzustand mit dem Soll-Ergebnis übereinstimmt. Für das Erstellen dieser Visualisierungen wurde die Matplotlib-Bibliothek aus Kapitel 5.1 verwendet.

Aufgrund der komplexen Zusammenhänge zwischen den Eingabedaten der Anwendung ist ein vollständiges Testen aller möglichen Anwendungsfälle bzw. Eingabekombinationen sehr zeitaufwändig. Stattdessen bietet es sich an, eine möglichst hohe Testabdeckung mittels Stichprobentests zu erreichen. Um Testfälle für die Anforderung R05 zu entwickeln, sollte in einem Datensatz nach Zeitpunkten gesucht werden, in denen die Positionen der AOIs manuell angepasst wurden. Schließlich können die Positionen der AOIs vor der manuellen Anpassung und danach geplottet werden. Nach einem Vergleich der visualisierten Koordinaten mit den Soll-Koordinaten kann festgestellt werden, ob die Anforderung R05 von der Anwendung erfüllt wird.

6.2 Verifizierung

Die Verifikation einer Anwendung „demonstriert mit mathematischen Mitteln die Konsistenz zwischen Spezifikation und Implementation“ [12]. Das Ziel dieses Kapitels ist das Veranschaulichen der korrekten Umsetzung der spezifizierten Anforderungen. Somit wird die Korrektheit der Anwendung mittels der oben definierten Testmethodik nachgewiesen (NR02). Im Folgenden wird die Umsetzung der Anforderungen anhand einiger Screenshots veranschaulicht.

In der folgenden Abbildung ist die GUI der Anwendung zu sehen:

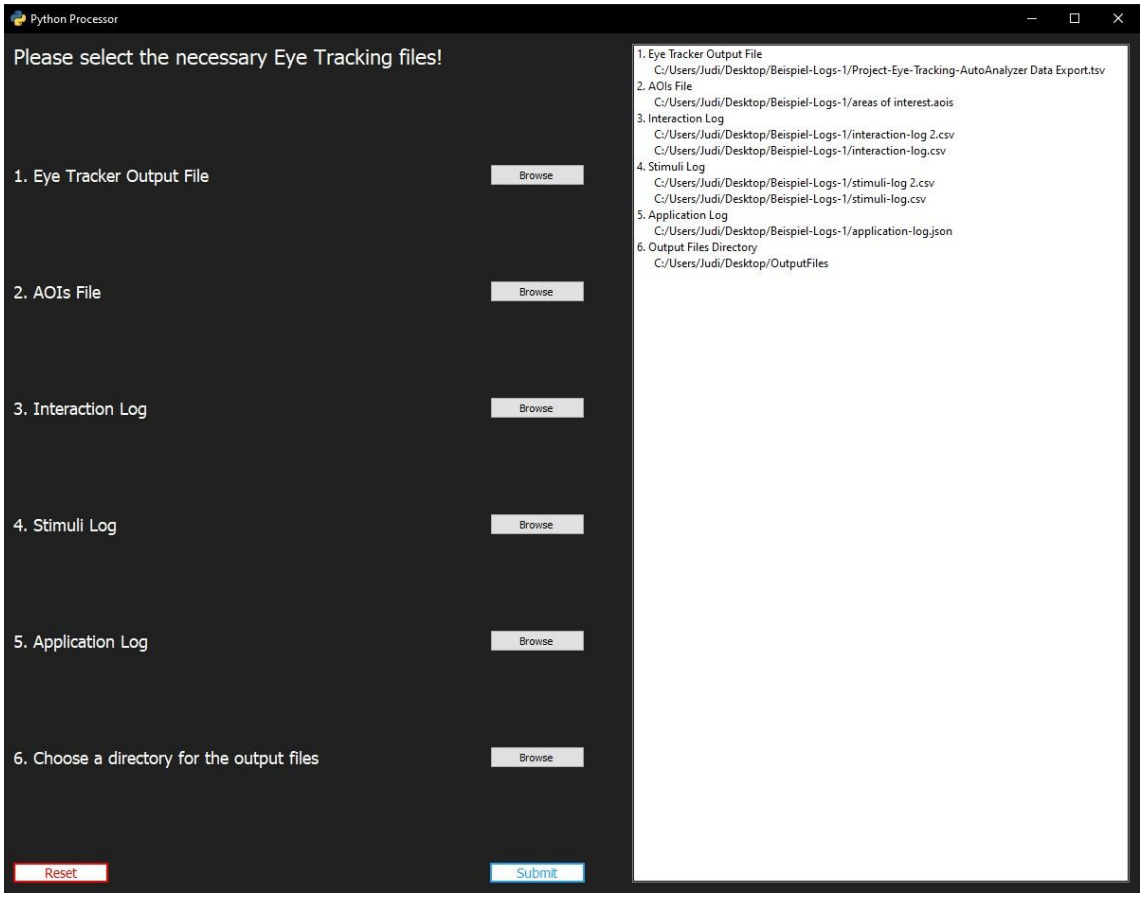

Abbildung 8: GUI der entwickelten Anwendung

Auf der linken Seite kann ein Nutzer die einzelnen Eingabedateien auswählen. Die ausgewählten Dateien werden dem Benutzer auf der rechten Seite in Form einer Liste angezeigt. Im Falle eines Fehlers kann ein Nutzer mittels eines „Reset“-Buttons die ausgewählten Dateien zurücksetzen. Falls der Nutzer jedoch alle Dateien ausgewählt hat und die Programmausführung beginnen möchte, kann dieser auf den „Submit“-Button klicken. Mittels einer minimalistischen GUI, welche für eine Benutzergruppe angelegt wurde, die sich mit der Benutzung von Benutzeroberflächen auskennt, wird die nicht-funktionale Anforderung NR01 realisiert. Zusätzlich ist aus der Abbildung zu erkennen, dass ein Nutzer mehrere Interaktions- und Stimuluslogs auswählen kann (R01). Außerdem kann ein Datenpfad zur Abspeicherung der Ausgabedateien angegeben werden (R07). Falls jedoch eine Datenauswertung ohne Logdateien gewünscht ist, kann der Nutzer die Programmausführung starten, indem nur die Eye Tracker Ausgabedatei und die AOI-Definitionen ausgewählt werden. Die Anwendung erkennt anschließend, dass keine Logdateien vorhanden sind und führt eine Datenauswertung mittels der Informationen aus den Eye Tracking Dateien durch (R03, R06).

Die folgenden Abbildungen veranschaulichen Systemzustände während einer Programmausführung. Der Bildschirm einer Eye Tracking Aufzeichnung wird dabei mittels eines Koordinatensystems veranschaulicht. Die x- und y- Achsen entsprechen der Auflösung des Bildschirms. Das Koordinatensystem veranschaulicht sichtbare Applikationen und AOIs zu verschiedenen Zeitpunkten. In dem für diese Evaluation genutzten Datensatz existieren Die Applikationen „Eclipse“ (IDE) und „Word“ (Textverarbeitungsprogramm). Die Eclipse IDE besitzt einen Stimuluslog, da es möglich ist, in der Applikation zwischen verschiedenen Tabs zu wechseln. Für Word hingegen wurde ein Interaktionslog angelegt, welcher die Benutzerinteraktionen innerhalb der Anwendung mitverfolgt.

Abbildung 9 veranschaulicht die Umsetzung der Anforderung R02:

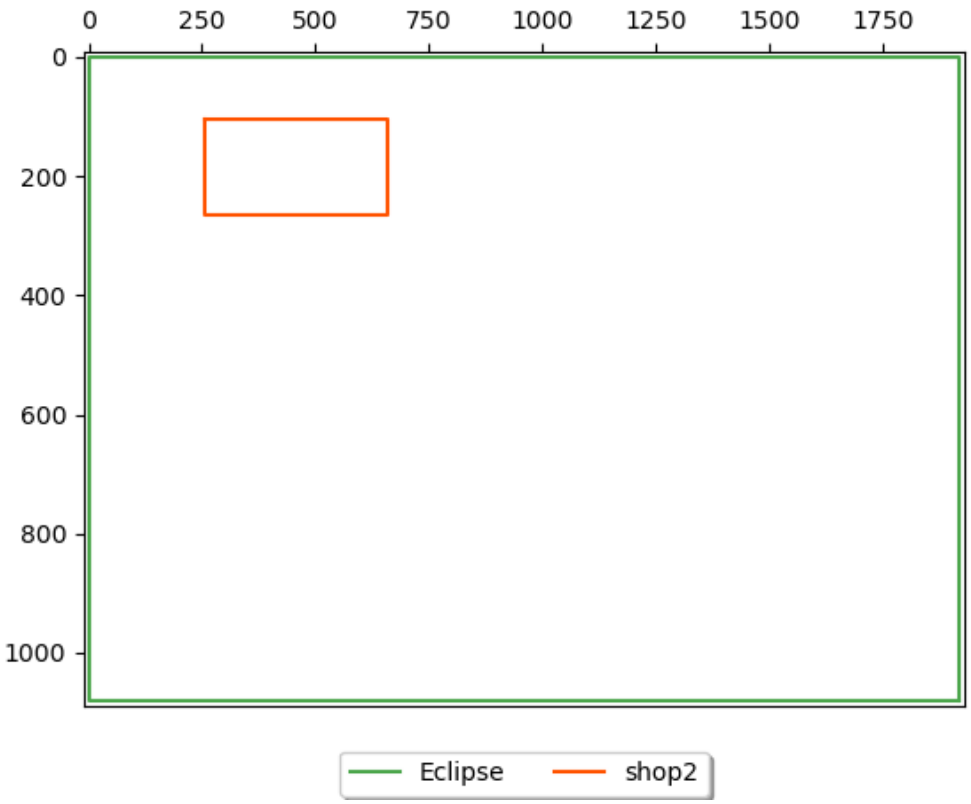

Abbildung 9: Zuweisung von Applikation zu AOI

Zu sehen ist die Anwendung Eclipse, welche derzeit maximiert ist. Im Applikationslog ist die Applikation Word ebenfalls vorzufinden, welche in der Fensterhierarchie jedoch unter Eclipse aufgelistet ist. Da Eclipse jedoch den gesamten Bildschirm einnimmt, ist die Word-Applikation nicht sichtbar. Des Weiteren ist eine AOI mit dem Namen „shop2“

zu sehen, welche zum Zeitpunkt des Plots das erste Mal aktiv geworden ist. Die Anwendung berechnet mittels des Zeitstempels des ersten Keyframes der AOI, welche Applikationen geöffnet waren, als die AOI „shop2“ aktiv geworden ist. Schließlich muss die Anwendung noch abfragen, innerhalb welcher Applikationsgrenzen sich die AOI befindet. In diesem Fall wird der AOI „shop2“ die Applikation Eclipse zugewiesen.

Der folgende Absatz beschäftigt sich mit der Umsetzung des Kerns dieser Arbeit (R04). Events aus den Logdateien sollen der zeitlichen Reihenfolge nach simuliert werden. Die Koordinaten von AOIs, welche sich innerhalb der Applikationen befinden, in denen diese Events stattgefunden haben, müssen schließlich angepasst werden. Die folgende Abbildung visualisiert die Auswirkung der Events aus einem Interaktionslog auf eine AOI mit dem Namen „abo2“:

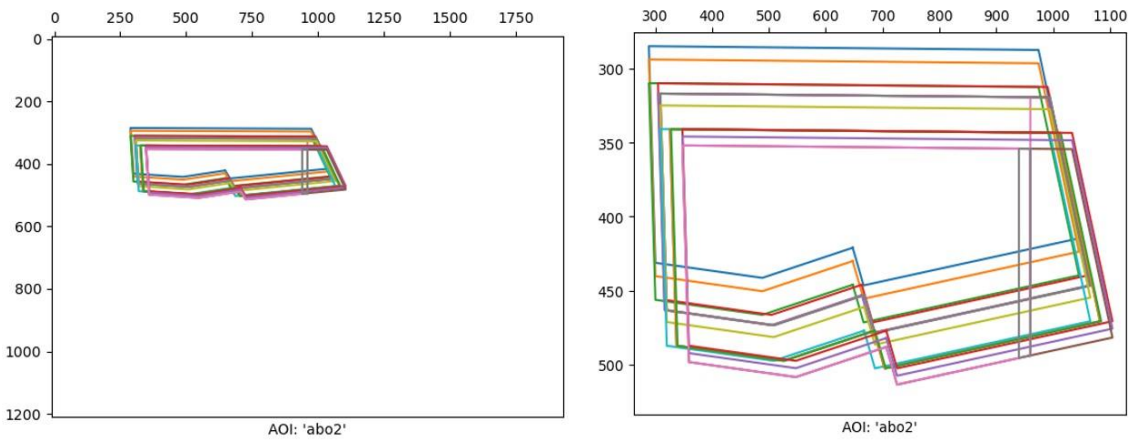

Abbildung 10: Simulation von Events des Interaktionslogs

Die Abbildung veranschaulicht die Keyframes der AOI „abo2“ im Verlauf einer Eye Tracking Aufzeichnung. Der rechte Plot ist ein Zoom des linken Plots, um die Unterschiede zwischen den Keyframes zu erkennen. Es ist zu erkennen, dass neue Keyframes entstehen, wenn in der Anwendung der AOI gescrollt oder editiert wird. Im Falle eines horizontalen Scrolls wird ein neuer Keyframe angelegt, welcher auf der X-Achse um den angegebenen Offset im Interaktionslog verschoben wurde. Analog dazu wird bei vertikalen Scrolls oder Edits ein Keyframe angelegt, welcher auf der Y-Achse um die angegebene Distanz verschoben wird. Bei der Zuweisung von Blickpunkten zu AOIs wird das System zurückrechnen, welcher Keyframe aus Abbildung 10 zum Zeitpunkt des Blickpunktes aktiv war, um letztendlich abfragen zu können, ob dieser Blickpunkt innerhalb der Koordinaten der AOI liegt.

Weitere Events, welche simuliert werden müssen, können aus dem Applikations- und Stimuluslog hervortreten. Im Stimuluslog wird der Stimulus angegeben, zu dem am gegebenen Zeitpunkt gewechselt wurde. Hier muss für jede AOI innerhalb der Applikation, welche den Stimuluslog besitzt, ausschließlich abgefragt werden, ob der neue Stimulus dem ursprünglich zugewiesenen Stimulus der AOI entspricht. Ist dies nicht der Fall, so wird ein neuer Keyframe angelegt, welcher die Sichtbarkeit der AOI auf den Wahrheitswert *false* setzt.

Interessanter wird es jedoch in Events aus Applikationslogs. Hier müssen verschiedene Fälle unterschieden werden. Falls es zu einem neuen Eintrag in einem Applikationslog kommt, so wird berechnet, ob sich das sichtbare Fenster verschoben hat. Ist dies der Fall, so müssen alle AOIs innerhalb dieser Applikation ebenfalls um den entstandenen Offset verschoben werden. Ein weiterer Fall tritt ein, wenn eine Applikation, welche in der Fensterhierarchie ganz oben war, nun von einer anderen Applikation überdeckt wird, welche im neuen Logeintrag hierarchisch als erstes vorkommt. In beiden Fällen muss die Anwendung überprüfen, ob die AOIs in den Applikationen noch gänzlich sichtbar sind oder ein Teil der AOIs nicht mehr sichtbar ist, da dieser von einem übergelegten Fenster abgeschnitten wurde oder sich nicht mehr innerhalb der Grenzen des Bildschirms befindet. Abbildung 11 veranschaulicht den Fall, in dem ein Fenster von einer hierarchisch höheren Applikation in einem neuen Eintrag des Applikationslogs überlappt wird.

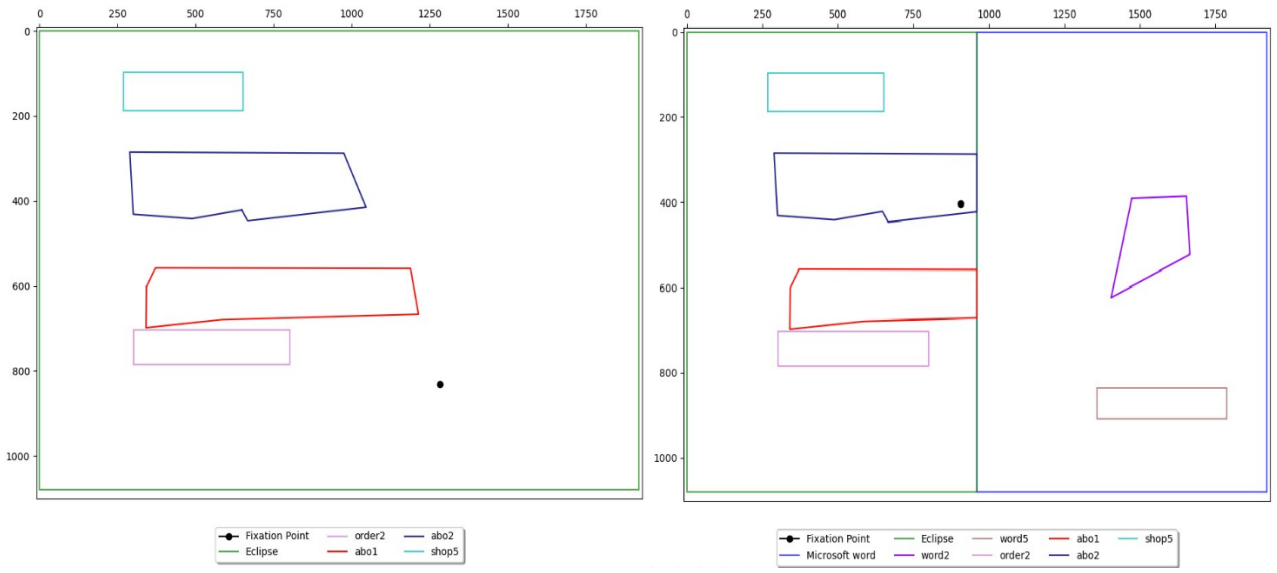

Abbildung 11: Simulation eines neuen Eintrags im Applikationslog

Die beiden Plots visualisieren zwei aufeinanderfolgende Fixationspunkte. Zum Zeitpunkt des ersten Fixationspunktes (links) sind die Applikation Eclipse und die AOIs dieser

Applikation zu sehen. Zwischen den Zeitpunkten der beiden Blickpunkte wurde ein neuer Eintrag im Applikationslog angelegt, welcher definiert hat, dass nun Word in der Fensterhierarchie als erstes vorkommt. Somit ist ein Teil des (vorher maximierten) Eclipse-Fensters nun nicht mehr sichtbar. Für jede AOI innerhalb von Eclipse muss daraus resultierend abgefragt werden, ob diese noch gänzlich oder nur noch teilweise zu sehen ist. Bei den AOIs „abo1“ und „abo2“ ist zu erkennen, dass ein Teil dieser AOIs aufgrund des überliegenden Fensters nicht mehr sichtbar ist. Das System passt somit die Koordinaten dieser AOIs an, sodass nur noch die sichtbaren Eckpunkte im neuen Keyframe vorkommen.

Wie in den Anforderungen erwähnt, können manuelle Keyframes angelegt werden, welche die Anwendung übernehmen muss (R05). Zur Veranschaulichung der Umsetzung dieser Anforderung sollen folgende Keyframes einer AOI mit dem Namen „word5“ betrachtet werden:


```
{
    "IsActive": true,
    "Seconds": 40,
    "Vertices": [
        {
            "X": 1255, "Y": 704
        },
        {
            "X": 1684, "Y": 704
        },
        {
            "X": 1684, "Y": 777
        },
        {
            "X": 1255, "Y": 777
        }
    ]
},
{
    "IsActive": true,
    "Seconds": 42,
    "Vertices": [
        {
            "X": 30, "Y": 30
        },
        {
            "X": 459, "Y": 30
        },
        {
            "X": 459, "Y": 103
        },
        {
            "X": 30, "Y": 103
        }
    ]
}
```


Abbildung 12: Manuell erstellte Keyframe

Die Position der AOI wird initial mittels des ersten Keyframes festgelegt, jedoch werden die Koordinaten der AOI in einem weiteren Keyframe nach zwei Sekunden nochmals verändert. Das System übernimmt den manuell erstellten Keyframe und übernimmt den Keyframe, welches von folgender Abbildung veranschaulicht wird:

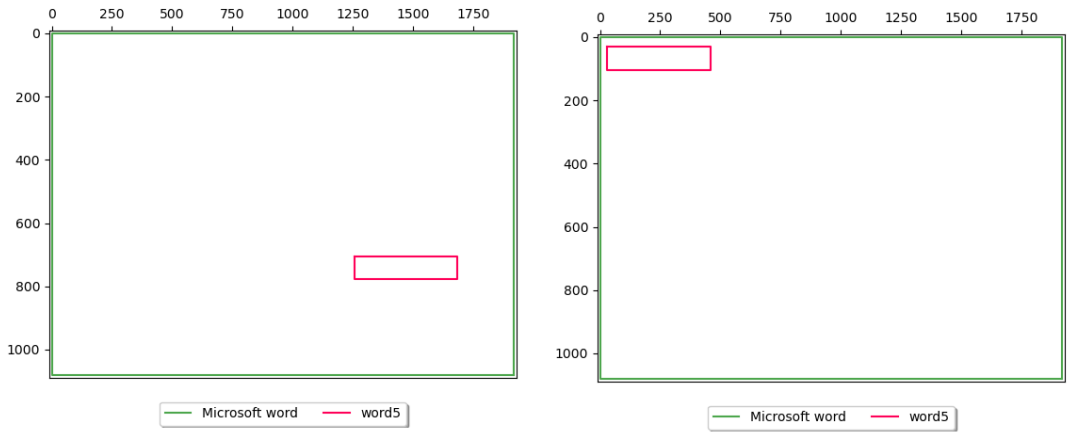

Abbildung 13: Übernahme einer manuell erstellten Keyframe

Der linke Plot visualisiert die Koordinaten der AOI vor dem Zeitpunkt des Eintretens der manuell erstellten Keyframe. Die Anwendung hat den manuell erstellten Keyframe während der Programmausführung übernommen und zeitlich in die Liste der Keyframes der AOI „word5“ einsortiert, welches anhand des rechten Plots sichtbar wird, der nach dem zeitlichen Eintreten des neuen Keyframes erstellt wurde.

Das System sollte dem Benutzer nach einer fehlerhaften Ausführung eine Meldung ausgeben, welche das entstandene Problem aufzeigt (R09). Aufgrund einer möglichen Ausarbeitung bzw. Erweiterung der Anwendung bietet es sich somit an, innerhalb des Programmcodes Exceptions zu werfen und den Traceback Stack auszugeben, sodass ein Entwickler nachvollziehen kann, aus welchen Teilen des Programmcodes ein Fehler hervortritt. Abbildung 14 gibt eine Übersicht über die Ausgabe der GUI für folgenden Fehler: Existiert in dem übergebenen Datensatz ein Stimuluslog, so erwartet die Anwendung, dass sich der erste Eintrag des Stimuluslogs zeitlich vor dem Beginn der Eye Tracking Aufzeichnung befindet, damit das System erkennen kann, in welchem Stimulus sich die Applikation am Anfang der Aufzeichnung befindet.

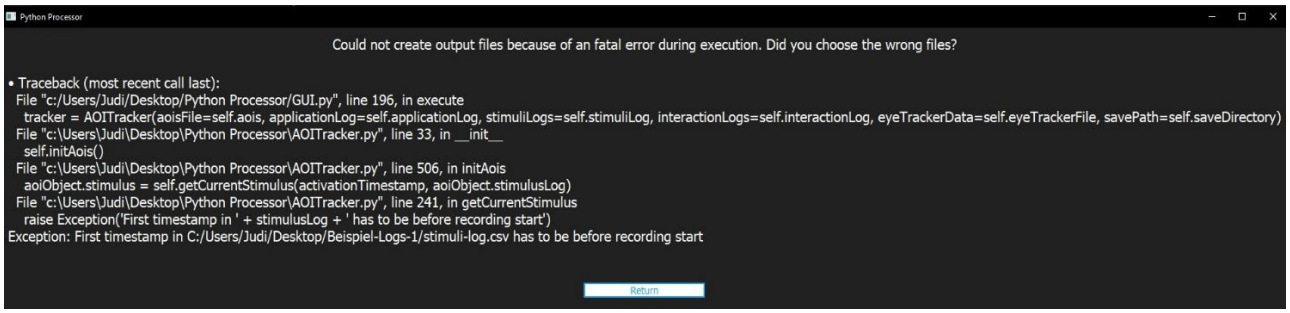

Abbildung 14: Fehlerausgabe der GUI

Zu sehen ist der Traceback Stack und die Exception, welche zu der fehlerhaften Ausführung geführt hat.

Die Anwendung soll letztendlich Ausgabemetriken berechnen und diese in Form von csv-Dateien ausgeben (R11, R12). Im Folgenden werden Ausschnitte aus Ausgabedateien einer Programmausführung veranschaulicht.
Die AOI Raw Metrics sollten alle AOI Visits dokumentieren und für jeden Visit Metriken, wie die Verweildauer oder Anzahl an Fixationen während des Visits, ausgeben. Tabelle 4 veranschaulicht einige Zeilen der Raw Metrics einer Programmausführung:

Participant	Application	Stimulus	AOI Name	Timestamp	Gaze Dwell Time [in ms]	Total Fixation Duration [in ms]	Avg Fixation Duration [in ms]	Fixation Count
Participant1	Microsoft Word		word3	1590434256	349	208	208	1
Participant1	Microsoft Word		whitespace	1590434256	7307	2817	128	22
Participant1	Eclipse	Shop.java	order2	1590434264	166	83	83	1
Participant1	Eclipse	Shop.java	abo3	1590434265	166	83	83	1
Participant1	Microsoft Word		word1	1590434266	1233	633	158	4

Tabelle 4: AOI Raw Metrics

Die Tabelle zeigt außer Visits innerhalb der vordefinierten AOIs auch Besichtigungen innerhalb der „whitespaces“ an. Blickpunkte, die sich nicht innerhalb einer AOI befunden haben, werden einem „whitespace“-Objekt zugeordnet. Dem „whitespace“-Objekt wird zusätzlich die Applikation, in welcher sich der whitespace befindet, zugeordnet. Somit wurde die Anforderung R10 ebenfalls realisiert.
Die AOI Summarized Metrics fassen alle Visits zusammen. Somit soll für jede AOI eine Zeile ausgegeben werden, welche Metriken wie die gesamte Verweildauer, durchschnittliche Verweildauer, Fixationshäufigkeit usw. anzeigen soll. Die folgende Tabelle zeigt einen Ausschnitt aus den Summarized Metrics desselben Datensatzes:

Participant	Application	Stimulus	AOI Name	AOI Duration [in ms]	Total Dwell Time [in ms]	Avg Dwell Time [in ms]	Total Fixation Duration [in ms]	Avg Fixation Duration [in ms]	Total Fixation Count
Participant1			whitespace		124716	3370	72908	156	472
Participant1	Microsoft Word		word2	176261	39197	1088	25311	175	140
Participant1	Eclipse	Shop.java	abo2	179691	32397	790	20497	182	116
Participant1	Microsoft Word		word4	258253	29161	940	19263	201	105

Tabelle 5: AOI Summarized Metrics

Die AOI Switches veranschaulichen die Blicksprünge zwischen zwei AOIs. Für jeden Blicksprung wird eine Zeile in den AOI Switches ausgegeben. Ein Ausschnitt aus den erstellen AOI Switches sieht folgendermaßen aus:

Timestamp	Source AOI	Source Application	Source Stimulus	Target AOI	Target Application	Target Stimulus
1590434283	shipment1	Microsoft Word		word4	Microsoft Word	
1590434291	order1	Microsoft Word		word2	Microsoft Word	
1590434294	word3	Eclipse	Shop.java	order2	Microsoft Word	
1590434295	shipment2	Microsoft Word		word3	Eclipse	Shop.java

Tabelle 6: AOI Switches

Abschließend sollte die Anwendung eine Visualisierung der gesammelten Daten ausgeben (R13). Dazu wurde in Kapitel 4 das AOI Sequence Chart vorgestellt. Abbildung 15 gibt eine Übersicht über das Sequence Chart eines Datensatzes:

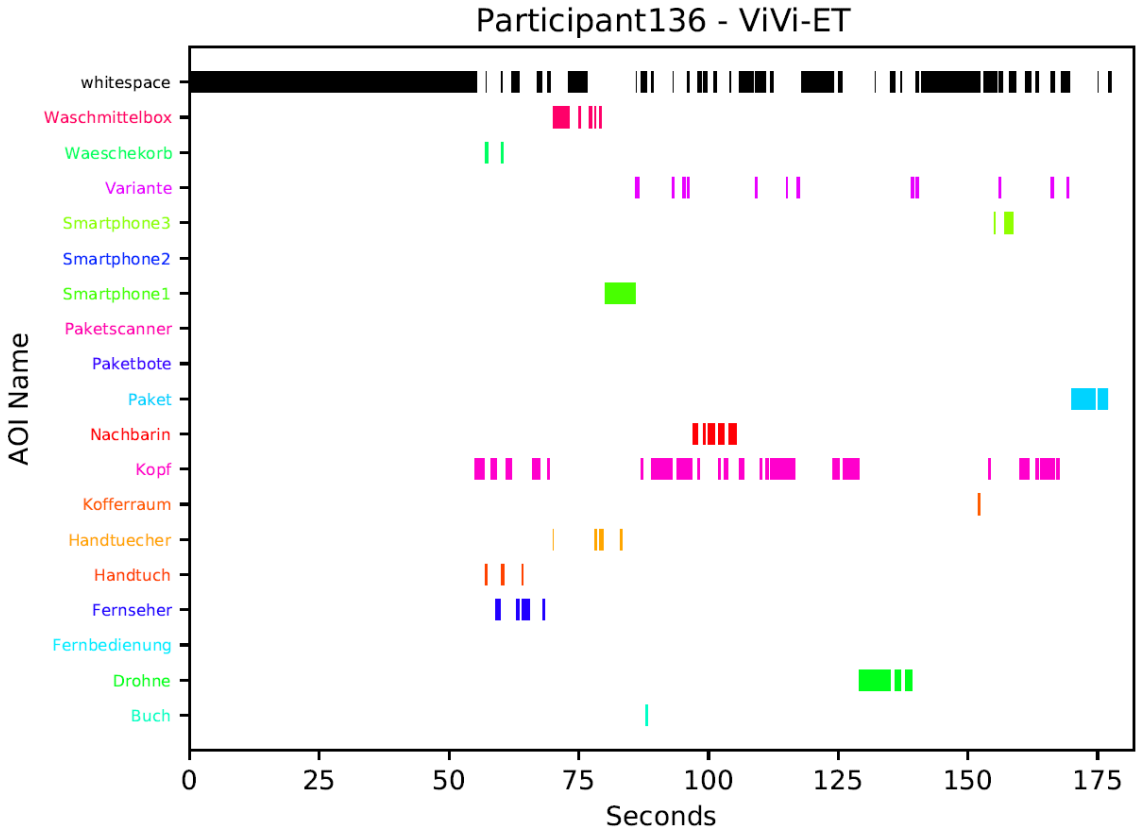

Abbildung 15: AOI Sequence Chart

Mittels des Sequence Charts aus Abbildung 15 wird der zeitliche Verlauf der beobachteten AOIs mitsamt der Verweildauer der jeweiligen AOI Visits ausgegeben. Somit können die Reihenfolge und Verweildauer der Fixationen im Laufe einer Aufzeichnung nachvollzogen werden. Da am Anfang der Aufzeichnung noch keine AOIs aktiv sind, werden viele Blickpunkte dem „whitespace“ zugeordnet.

7. Verwandte Arbeiten

Das Problem, eine Datenauswertung von dynamischen Stimuli innerhalb einer Eye Tracking Aufzeichnung zu ermöglichen, wurde bereits in einigen wissenschaftlichen Arbeiten aufgegriffen. Einige Ansätze für die Lösung der Problemstellung werden im Folgenden vorgestellt.

Papenmeier und Huff stellten ein Konzept vor, welches auf dynamischen AOIs innerhalb von 3-D Modellen der jeweiligen dynamischen Stimuli basiert [13]. In dem Ansatz wird die Position der AOIs zu verschiedenen Zeitpunkten einer Eye Tracking Aufzeichnung mittels der bereits bekannten Keyframes angegeben. Die Koordinaten der AOI werden schließlich von Keyframe zu Keyframe interpoliert bzw. linear verschoben. Die meisten Eye Tracking Softwares, darunter die Software des Tobii Pro Eye Trackers, bieten diese Art der interaktiven Eye Tracking Aufzeichnung an. Papenmeier und Huff stellten in ihrer Arbeit weitere Ansätze anderer Wissenschaftler vor. Für eine Differenzierung der präsentierten Lösungen unterschieden sie zwischen dynamischen Online- und Offline-Ansätzen. Online-Ansätze weisen Fixationspunkte im Laufe einer Eye Tracking Aufzeichnung der jeweiligen AOI zu. Dazu werden meist Schnittstellen der Eye Tracking Anbieter verwendet. Hingegen werden die Eye Tracking Daten bei Offline-Ansätzen mittels eines Post-Processings erst nach einer vollständigen Aufzeichnung ausgewertet. Da die Datenauswertung erst nach einer Eye Tracking Aufzeichnung stattfindet, handelt es sich in dieser Arbeit um eine Offline-Lösung der Problemstellung. Der Ansatz bezüglich einer Interpolation von zeitlich definierten Keyframes entspricht somit ebenfalls einem Offline-Ansatz, da Keyframes erst nach einer vollständigen Eye Tracking Aufzeichnung definiert werden müssen. Das manuelle definieren der Keyframes muss nach jeder Aufzeichnung mit einem Teilnehmer erfolgen, da jeder Teilnehmer während einer Aufzeichnung individuell mit dem System interagiert. Dabei entsteht ein hoher Zeitaufwand, welcher durch die Anwendung dieser Arbeit reduziert werden sollte. Im Folgenden werden weitere Offline-Lösungen vorgestellt, damit diese Arbeit somit von weiteren existierenden Lösungen abgegrenzt werden kann.

T. R. Shaffer et al. präsentieren einen Ansatz in Form eines IDE-Plugins [14]. Das Ziel der Arbeit ist es, eine Eye Tracking Anwendung anzubieten, welche umfangreiche

Studien bezüglich bestimmter Softwareartefakte (z.B. Quellcode) ermöglicht. Das Plugin (iTrace) verbindet sich zunächst mit unterstützten Eye Trackern und nutzt vorgegebene Schnittstellen der jeweiligen Eye Tracking Anbieter, um Blickpunkte eines Probanden während der Benutzung einer IDE aufzuzeichnen. Die Benutzerinteraktionen innerhalb der IDE werden mitverfolgt und Probanden dürfen mit dem System durch scrollen und wechseln zwischen verschiedenen Codedateien interagieren. Schließlich werden mittels eines Post-Processing Moduls die aufgezeichneten Blickpunkte des Nutzers den jeweiligen AOIs zugeordnet. Abschließend kann ein Proband einstellen, in welcher Form die gesammelten Daten aufbereitet werden sollen. Einige Studien bezüglich des Verhaltens von Entwicklern während der Ausführung bestimmter Aufgaben wurden mittels des Plugins bereits durchgeführt. Da es sich bei der Lösung von Shaffer et al. um ein Plugin für eine IDE handelt und derzeit nur wenige Eye Tracker unterstützt werden, ist der entwickelte Ansatz jedoch sehr begrenzt und nur für wenige Anwendungsfälle ausgerichtet.

Der WebEyeMapper von R. W. Reeder et al. ist eine weitere Lösung, welche Benutzerinteraktionen innerhalb des Internet Explorer Browsers aufzeichnet und anschließend die Blickpunkte den AOIs zuordnet [15]. Das Ziel der Arbeit war es, eine Anwendung zu entwickeln, welche Forschungen in dem Bereich der Web-Usability ermöglichen. Die entwickelte Anwendung nimmt, analog zu der Anwendung dieser Arbeit, einen Datensatz des Eye Trackers und weitere Logdateien entgegen, welche die Benutzerinteraktionen innerhalb des Browsers während einer Aufzeichnung mitverfolgen. Anschließend rekonstruiert der WebEyeMapper die Webseiten, welche vom Nutzer besucht wurden und berechnet die Auswirkungen der Benutzerinteraktionen auf die Positionen der AOIs, um schließlich die Blickpunkte den jeweiligen AOIs zuordnen zu können. Die Anwendung ist jedoch nur auf den Internet Explorer Browser ausgerichtet und unterstützt keine anderen Applikationen oder Webbrowser, da das in der Anwendung verwendete Logging-Tool nur auf diesen Browser zugeschnitten ist. Jedoch war der WebEyeMapper zu der damaligen Zeit (2001) ein Durchbruch in den Technologien bezüglich der Nutzung von Eye Trackern im Bereich der Web-Usability.

8. Fazit

Dieses Kapitel liefert einen abschließenden Überblick der Inhalte dieser Arbeit. Zusätzlich werden in einem Ausblick mögliche und sinnvolle Erweiterungsmöglichkeiten der entwickelten Anwendung genannt.

8.1 Zusammenfassung

Das Ziel dieser Arbeit war es, eine Anwendung zu entwickeln, welche automatische Auswertungen von Eye Tracking Aufzeichnungen mit dynamischen Stimuli ermöglicht. Wenn sich der Blickpunkt eines Nutzers zu einem bestimmten Zeitpunkt an einer Bildschirmposition befindet, könnte es sein, dass sich die Informationen an dieser Bildschirmposition zwischenzeitlich durch Benutzerinteraktionen verändert haben.

Für die Realisierung einer solchen Anwendung wurden zunächst die Anforderungen an das System erhoben. Schließlich wurde basierend auf den Anforderungen ein Konzept entwickelt, welches Beschränkungen und Ideen der zu entwickelnden Anwendung beschreibt. Mittels des entwickelten Konzepts wurde schließlich eine Softwarearchitektur entwickelt, welche in einem letzten Schritt implementiert werden musste. Abschließend wurde das System evaluiert, indem über Visualisierungen von Systemzuständen die Erfüllung von Anforderungen veranschaulicht wurde.

Durch die entwickelte Anwendung ist es einem Nutzer gestattet, während einer Eye Tracking Aufzeichnung mit dem System zu interagieren. Diese Interaktionen – beispielsweise das Scrollen innerhalb von Anwendungen – werden aufgezeichnet und in Logdateien gespeichert. Durch ein Zusammenspiel von Interaktions-Logging und Eye Tracking Daten kann die Anwendung im Nachhinein die Koordinaten von definierten AOIs bezüglich der durch die Interaktionen entstandenen Verschiebungen anpassen. In einem letzten Schritt können die Blickpunkte des Nutzers den jeweiligen AOIs, unter Beachtung der Verschiebung von Koordinaten zu verschiedenen Zeitpunkten, zugeordnet werden. Ohne die entwickelte Anwendung wäre eine manuelle Zuordnung der Blickpunkte zu AOIs notwendig, welches eines hohen Zeitaufwands bedarf.

Die Lösung dieser Arbeit ist im Gegensatz zu bereits entwickelten Ansätzen und Lösungen (siehe Kapitel 7) sehr flexibel, da sie nicht auf bestimmte Anwendungen begrenzt ist. Applikationen, für die bereits ein Logging-Tool entwickelt wurde, können durch die Software dieser Arbeit verarbeitet werden. Zusätzlich können jedoch auch

Applikationen, für die keine Logging-Tools vorhanden sind, in die Datenauswertung eingebracht werden, indem die Verschiebungen und Koordinaten von AOIs innerhalb dieser Applikationen manuell mitverfolgt werden. Eine Kombination beider Anwendungsmöglichkeiten ermöglicht somit eine flexible Datenauswertung.

8.2 Ausblick

Die entwickelte Anwendung bietet viele Ausbaumöglichkeiten an. Eine wesentliche Erweiterung ist das Entwickeln oder Nutzen von weiteren Logging-Tools. Derzeit werden während einer Eye Tracking Aufzeichnung nur Interaktionen, Stimuluswechsel und Verschiebungen von Applikationsfenstern aufgezeichnet. Weitere Logging-Tools würden die Präzision der Auswertung erhöhen und somit zu besseren Ergebnissen führen. Möglich wäre beispielsweise die Entwicklung eines Zoom-Logs, welches verfolgt, wie sich der Zoom von Objekten innerhalb einer Anwendung verändert. In Abbildung 16 ist anhand der Windows „Foto“-Applikation zu erkennen, wie sich die Größe bzw. der Zoom von Objekten innerhalb der Applikation mit der Fenstergröße der Applikation verändert.

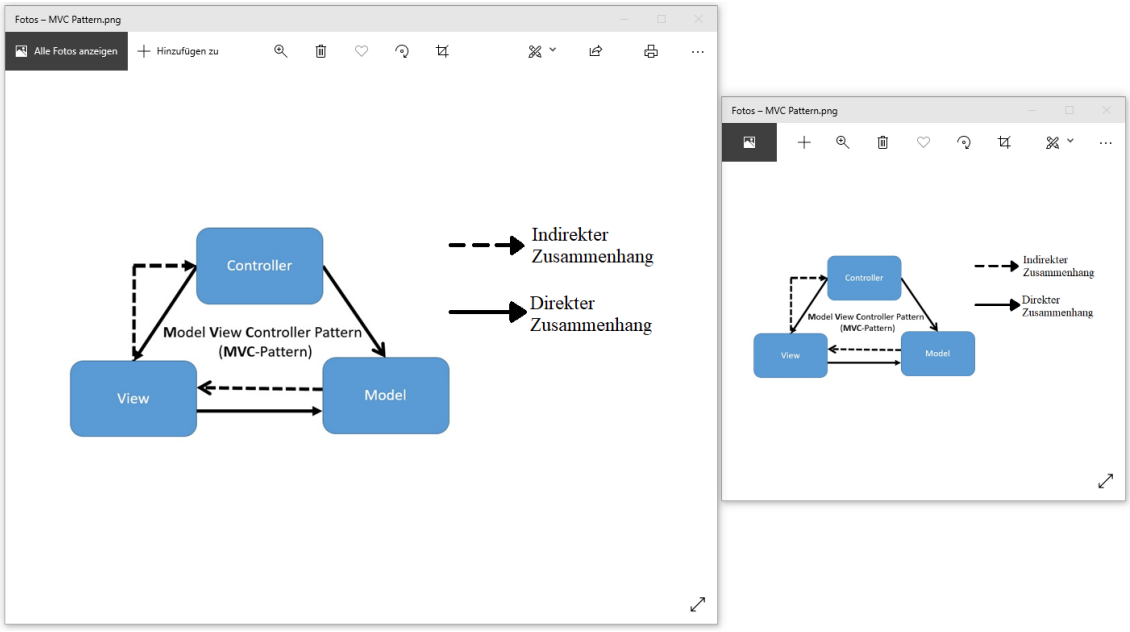

Abbildung 16: Zoom einer Applikation

Falls sich eine AOI innerhalb des geöffneten Bildes in Abbildung 16 befindet, so könnte in einem Zoom-Log, nach dem Verändern der Fenstergröße, ein neuer Eintrag angelegt werden, welcher den neuen Zoom-Faktor der Objekte innerhalb der Anwendung definiert. Die Größe der AOI müsste dementsprechend mit dem Zoom-Faktor multipliziert werden, damit die Koordinaten der AOI angepasst werden können. Solch ein Logging-Tool

müsste jedoch für jede Applikation einzeln angelegt werden, da verschiedene Applikationen individuelle Zoom-Faktoren haben.

Eine weitere nennenswerte Erweiterung ist das Einführen einer ausführlicheren Visualisierung von Ausgabemetriken. So könnten weitere Visualisierungsformen und Filteroptionen eingeführt werden, um eine ausführlichere Analyse zu bewerkstelligen. Beispielsweise ist das Erstellen von sogenannten Scanpaths für die Analyse von Eye Tracking Daten praktisch. Diese visualisieren die Reihenfolge und Blicksprünge eines Nutzers während der Aufzeichnung. Dazu könnte die Anwendung die Applikationsfenster und aktiven AOIs plotten und schließlich die Scanpaths eines Nutzers einzeichnen. Durch Filteroptionen könnten in der Visualisierung beispielsweise einige AOIs ein- oder ausgeblendet werden.

Letztendlich unterstützt die Anwendung derzeit nur Tobii Pro Eye Tracker, mit denen Rohdaten nach einer Aufzeichnung mittels des Tobii Pro Labs ausgegeben werden können. Hier wäre eine Erweiterung auf andere Eye Tracker möglich, indem die Anwendung verschiedene Eingabeformate für Eye Tracking Daten entgegennimmt.

Zusammenfassend hat die entwickelte Anwendung ein hohes Erweiterungspotenzial und könnte durch die Einführung von weiteren Logging-Tools und einer flexibleren Gestaltung der Ausgabedateien eine breit gefächerte Software für die Analyse von Eye Tracking Daten darbieten.

Literaturverzeichnis

[1] G. Rakoczi, „Eye Tracking in Forschung und Lehre: Möglichkeiten und Grenzen eines vielversprechenden Erkenntnismittels“ in *Digitale Medien – Werkzeuge für exzellente Forschung und Lehre*, Münster, Waxmann, 2012, p. 87.

[2] M. Wu und T. Munzner, SEQIT: Visualizing Sequences of Interest in Eye Tracking Data, Chigaco, 2015.

[3] H. Mössenböck, Objektorientierte Programmierung in Oberon-2, Linz: Springer, 1993.

[4] Datenbanken-verstehen.de Team, „Model View Controller Pattern“, 2020. [Online]. Available: https://www.datenbanken-verstehen.de/lexikon/model-view-controller-pattern/. [Zugriff am 10 Juli 2020].

[5] C. Rupp und die SOPHISTen, Requirements-Engineering und -Management, Carl Hanser Verlag München, 2014.

[6] S. Hareide und R. Ostnes, Maritime Usability Study by Analysing Eye Tracking Data, Glasgow, 2016.

[7] M. Lutz und D. Ascher, Einführung in Python, O'Reilly, 2007.

[8] V. Loganathan, PySide GUI Application Development, Birmingham: Packt Publishing, 2013.

[9] S. Gillies, „Shapely Project description“, 28 Januar 2020. [Online]. Available: https://pypi.org/project/Shapely/. [Zugriff am 15 Juli 2020].

[10] J. Hunter, D. Dale, E. Firing, M. Droettboom und the Matplotlib development team, „Matplotlib: Visualization with Python“, 17 Juni 2020. [Online]. Available: https://matplotlib.org/. [Zugriff am 15 Juli 2020].

[11] B. Kahlbrandt, Software Engineering - Objektorientierte Software-Entwicklung mit der Unified Modeling Language, Berlin Heidelberg: Springer, 1998.

[12] P. Liggesmeyer, H. M. Sneed und A. Spillner, Testen, Analysieren und Verifizieren von Software, Berlin Heidelberg: Springer, 1992.

[13] F. Papenmeier und M. Huff, DynAOI: A tool for matching eye-movement data with dynamic areas of interest in animations and movies, Tübingen, 2010.

[14] T. R. Shaffer et al., iTrace: Enabling Eye Tracking on Software Artifacts withinthe IDE to Support Software Engineering Tasks, Bergamo, 2015.

[15] R. W. Reeder et al., WebEyeMapper and WebLogger: Tools for Analyzing EyeTracking Data Collected in Web-use Studies, Palo Alto, 2001.

Tabellen- und Abbildungsverzeichnis