HAW HAMBURG

BACHELORTHESIS
Lilli-Jo Kertscher

Entwicklung eines Tools zur Unterstützung der Analyse und Interpretation neuropsychologischer Diagnostik

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Lilli-Jo Kertscher

Entwicklung eines Tools zur Unterstützung der Analyse und Interpretation neuropsychologischer Diagnostik

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
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. Stefan Sarstedt
Zweitgutachter: Prof. Dr. Marina Tropmann-Frick

Eingereicht am: 28. Februar 2021

Lilli-Jo Kertscher

Thema der Arbeit
Entwicklung eines Tools zur Unterstützung der Analyse und Interpretation neuropsychologischer Diagnostik

Stichworte
Diagnostik, Skalen, Konfidenzintervall, Normierung, Analyse, Interpretation

Kurzzusammenfassung
Die vorliegende Bachelorarbeit befasst sich mit der Entwicklung eines Analyse- und Interpretations-Tools für die neuropsychologischen Diagnostik (AI-Tool) in Kooperation mit dem Evangelischen Krankenhaus Alsterdorf in Hamburg (EKA). Das Tool dient der Unterstützung der Neuropsychologen in deren Arbeitsalltag mit dem Fokus der Berechnung und Darstellung von Konfidenzintervallen durchgeführter Skalen. Die Funktionalität und Bedienbarkeit des Tools wurde in einer Testphase im EKA überprüft und abschließenden von den Benutzern bewertet. Diese ergab insgesamt eine hohe Zufriedenheit mit dem entwickelten Tool.

Lilli-Jo Kertscher

Title of Thesis
Development process of a Tool to support the Analysis and Interpretation of neuropsychological Diagnostics

Keywords
diagnostics, scales, confidence interval, standardization, analysis, interpretation

Abstract
This present bachelor thesis addresses the development of an analysis- and interpretation-tool for neurpsychological diagnostics (AI-Tool) in cooperation with the Evangelic Hospital in Alsterdorf, Hamburg (EKA). The tools purpose is the support of neuropsychologists with their dailty tasks, especially in regard to the calculation and display of confidence intervals of conducted scales. In order to asses the functionality and operability of the tool, a trial phase was conducted at the EKA as well as a final rating of its users. This showed an overall contentment with the developed tool.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Hintergrund

In diesem Abschnitt wird der Kontext, in den sich die Arbeit einordnet, definiert, sowie wichtige Hintergründe erläutert und Motivationen offengelegt.

1.1.1 Neuropsychologie

Die Neuropsychologie gilt als ein Teilgebiet der Neurologie und stellt ein interdisziplinäres Fachgebiet der Medizin und Psychologie dar. Die Aufgaben der Neuropsychologie befassen sich mit dem Erforschen der Zusammenhänge neuronaler, behavioraler und psychischer Vorgänge oder Strukturen, sowie deren Veränderungen und Ausfällen. Sie spielt daher eine wichtige Rolle in der Diagnostik und Behandlung neurologischer Krankheiten (Vgl. [3]).

Ziele Neuropsychologischer Diagnostik:

- "Beurteilung, ob ein Hirnschaden oder eine Hirnfehlfunktion vorliegt" [3] und "deren Lokalisation" [3]
- "Erleichterung und Optimierung von Pflege und Rehabilitation des Patienten" [3]
- "Feststellung durch wiederholtes Testen, ob und wie schnell sich der Patient erholt" [3]
- "Abschätzung der Rehabilitationschancen und Wissen um die Defizite; auch, um den Patienten und seine Angehörigen zu informieren und um eine Grundlage für eine realistische Lebensführung zu geben" [3]

- ”Nachweis leichter Störungen, wo andere Testverfahren versagen oder zweideutige Ergebnisse liefern, z.B. bei Schädel-Hirn-Traumen oder ersten Auswirkungen einer degenerativen Erkrankung” [3]
- ”Identifikation ungewöhnlicher Lokalisationen, z.B. bei manchen Linkshändern, bei Entwicklungsabweichungen nach kindlichen Hirnschädigungen oder nach chirurgischen Eingriffen; dies ist auch wichtig, um bei Operationen ggf. wichtige Hirnareale wie das primäre Sprachzentrum nicht zu zerstören” [3]

1.1.2 Neuropsychologische Diagnostik

Eine neuropsychologische Diagnostik umfasst, neben der Objektivierung von Funktionsbeeinträchtigungen die Feststellung und Beschreibung des aktuellen kognitiven und affektiven Zustands eines Patienten, sowie Verlaufsuntersuchungen bei Patienten mit fortschreitenden oder reversiblen Krankheitsverläufen.
Bei der Vorgehensweise einer solchen Diagnostik wird sich an groben theoretischen Strategien zur Planung und Durchführung einer neuropsychologischen Untersuchung orientiert. Ein Beispiel solch einer Strategie wird in dem Flussdiagramm der folgenden Abbildung dargestellt.

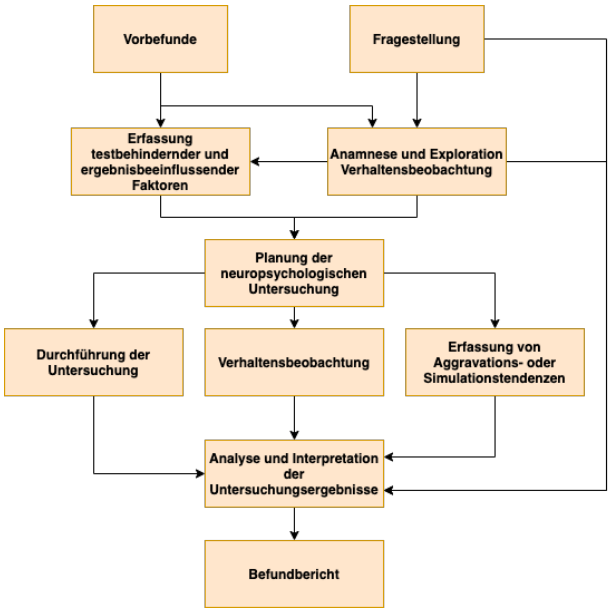


Abbildung 1.1: Flussdiagramm zur Vorgehensweise bei der neuropsychologischen Diagnostik (Angelehnt an [21, S.324])

Grundlegend wird eine neuropsychologische Diagnostik aufgrund einer Fragestellung eines Arztes oder von Vorbefunden ausgehend beauftragt und durchgeführt. Die Auswahl der durchgeführten Tests ist abhängig von der Fragestellung, dem Patienten (z.B. Alter), aber auch von testbehindernden oder ergebnisbeeinflussenden Faktoren (z.B. Medikation oder motorische Störungen).
Bei den Testverfahren handelt es sich um standardisierte Prüfverfahren, die auf statistischen Normen basieren. Als Ergebnis werden z.B. Skalenwerte angegeben, anhand derer ein Vergleich der getesteten Person mit 'gesunden' Personen möglich ist. Der Vergleich erfolgt durch Normwerte, z.B. in Form von Prozenträngen.
Die Ergebnisse werden analysiert, interpretiert und anschließend in Form eines Befundberichts festgehalten und archiviert, sowie eine Kopie an den Arzt bzw. der Person, welche die Diagnostik beauftragt hat, übergeben [21].

1.2 Motivation

Der treibende Gedanke hinter dieser Arbeit ist, einen Mehrwert zu schaffen. Durch das Entwickeln von etwas Nützlichem soll anderen das Leben bzw. die Arbeit erleichtert werden. Aus eigenen Erfahrungen in der klinischen Neuropsychologie und Gesprächen mit ausgebildeten Neuropsychologen bezüglich ihrer Tätigkeiten, werden einige Unannehmlichkeit oder mögliche Probleme in dem Prozess der Diagnostik deutlich, wo eine Automatisierung von Teilschritten sinnvoll und hilfreich sein könnte.

Daher ist das Ziel meiner Arbeit, Neuropsychologen mit einem Programm bei ihrer Tätigkeit zu unterstützen und Unannehmlichkeiten zu reduzieren, sodass diese ihre Arbeitszeit effektiver nutzen können und die Produktivität gesteigert werden kann.

1.3 Problemstellung

Dieser Abschnitt soll die Problemstellung der Arbeit aufzeigen, welche einem verbesserungsbedürftigen Umstand in der praktischen neuropsychologischen Diagnostik zugrunde liegt.

Wurde eine Diagnostik erfolgreich durchgeführt, müssen alle Rohdaten des Patienten auf Abweichungen analysiert und entsprechend interpretiert werden. Da es sich bei den Testverfahren um standardisierte Verfahren handelt, erfolgt dies nach einem vorgegebenen

Schema, das in dem jeweiligen Manual des Testverfahrens dargestellt wird. Zur Analyse der Daten werden daher die Rohdaten, anhand des jeweils erläuterten Schemas, in vergleichbare Normdaten überführt.
Diese Testverfahren kategorisieren die erbrachte Leistung des Patienten anhand unterschiedlicher Patienten-Variablen. Da diese Testverfahren jedoch zu unterschiedlich Zeitpunkten sowie von unterschiedlichen Forschern entwickelt wurden, können die benötigten Variablen, sowie die Darstellung und Form der Ergebnisse unter den verschiedenen Testverfahren variieren.
Die Normdaten werden im aktuellen Prozess typischerweise händisch von den Neuropsychologen in den jeweiligen Manualen herausgesucht und eingetragen. Zur Interpretation müssen gegebenenfalls zusätzlich die Konfidenzintervalle der Testverfahren ebenfalls händisch berechnet werden.
Es kann vorkommen, dass einzelne Excel-Dateien zu Hilfe genommen werden, um einen gewissen Grad der Automatisierung zu erreichen. Diese Dateien erfüllen jedoch in der Regel nur einen einzigen Arbeitsschritt und sind zudem oft von einer einzigen Person angefertigt. Dies führt zu vielen Wechseln zwischen mehreren Programmen und Fenstern während des Prozesses. Zudem kann Wissen schnell verlorengehen, wenn diese Person das Unternehmen verlässt. Grundsätzlich sind Änderungen für den durchschnittlichen Benutzer nur mit Hilfestellung und viel Aufwand umzusetzen.

Insgesamt braucht ein Neuropsychologe für die Analyse und Interpretation unverhältnismäßig lange, da einzelne Schritte des Prozesses viele triviale aber aufwändig händisch ausgeführte Arbeiten beinhalten.

1.4 Aufbau der Arbeit

Nach dem in diesem Abschnitt die relevante Problematik dieser Arbeit definiert und erläutert wurde, erfolgt im nächsten Kapitel das Vorstellen wichtiger theoretischer Grundlagen, an denen sich bei der Umsetzung orientiert wurde. Anschließend wird die Projektorganisation mit dem zugrundeliegenden Vorgehensmodell vorgestellt, nach der in der Entwicklung vorgegangen wurde.

Der weitere Aufbau wird angelehnt an die wichtigen Software-Lebenszyklusphasen des Software-Engineerings. Diese Phasen beschreiben den Prozess, welcher zur Erstellung und Erhaltung eines Softwaresystems führt und bei jeder Art der Softwareerstellung abläuft (Vgl. [16, Kap. 1.2.1]).

Das Vorgehen zur Anforderungsanalyse, sowie die daraus gewonnen Erkenntnisse werden festgehalten und näher erläutert. Anschließend werden diese in eine Spezifikation der Anforderungen überführt.
Auf Basis der Spezifikation wird ein Architektur-Entwurf erstellt und dieser mit Hilfe verschiedener Sichten und entsprechenden Diagrammen dargestellt. Es folgt die Implementierung und das Testen der Anwendung sowie ihre finale Evaluierung.

Zuletzt wird das Projekt kritisch reflektiert und abschließend zusammengefasst.

2 Theoretische Grundlagen

2.1 Vorgehensmodelle

Vorgehensmodelle sind Prozessmodelle zur Erstellung einer Software und bieten eine abstrakte Darstellung der Vorgehensweise. Sie stellen nicht nur die Lebenszyklen von Software dar, sondern legen auch Aktivitäten und deren Reihenfolge fest und unterteilen diese in Phasen. Der Einsatz von Vorgehensmodellen gilt als obligatorisch und vorteilhaft, da es einen Leitfaden für die Entwicklung vorgibt, wodurch die Planbarkeit im Projekt verbessert werden kann (Vgl. [16]).

2.1.1 Agiles Modell

Das *Agile Modell* ist eines der bekanntesten Vorgehensmodelle in IT-Projekten und basiert auf dem *Iterativen Paradigma*. Im Fokus steht hierbei die stetige Anpassung von Änderungswünschen des Kunden. Entsprechend bedeutet dies jedoch auch, dass ein Projektergebnis durch diese Flexibilität in der Regel nicht voraussagbar ist, da die Anforderungen im Laufe des Projekts erst definiert und jederzeit wieder verändert werden können (Vgl. [16]).

2.1.2 Scrum

Scrum ist eine Methode des Agilen Modells, wird jedoch auch als eigenes Vorgehensmodell angesehen (Vgl. [16]). In seinem Ursprung ist Scrum ein Framework für agiles Projektmanagement mit dem Ziel der Bewältigung von Aufgaben und kontinuierlicher Verbesserungen.

Anforderungen werden hier in *User-Stories* definiert und deren Aufwand bezüglich Zeit und Ressourcen eingeschätzt. Diese werden in einer *Story-Map* als graphische Übersicht abgebildet und in einem *Produkt-Backlog* festgehalten (Vgl. [16]).

Der sogenannte *Sprint* ist eines der zentralen Prinzipien von Scrum. Hierbei wird zu einem festen Zeitraum von maximal einem Monat ein lauffähiges und verwendbares Produktinkrement hergestellt. Ein solcher Sprint umfasst dessen Planung, tägliche Treffen (*Daily Scrums*), die Entwicklungsarbeit und entsprechende *Reviews* zur Überprüfung, sowie eine abschließende *Retrospektive* als Feedback-Runde bezüglich des durchgeführten Sprints. Wichtig ist, dass während eines Sprints keine Änderungen bezogen auf das Entwicklungsziel zugelassen sind (Vgl. [16]).

Die Zuständigkeiten der Scrum-Beteiligten können in drei Rollen unterteilt werden:

- **Product-Owner**:
  Verantwortlich für Eigenschaften und Erfolg des Projektes
- **Scrum-Master**:
  Verantwortlich für das Einhalten der Scrum-Methode
- **Entwicklungs-Team**:
  Verantwortlich für die Umsetzung und Testung der Software

(Vgl. [16])

2.2 Domain-Driven-Design

Das Domain-Driven-Design bedeutet ein System nach Fachlichkeit zu entwerfen. Es wird dafür plädiert, die Fachdomäne in der ein System entwickelt werden soll, zu verstehen und sein Wissen darin zu vertiefen, bevor mit dem Entwurf einer Software begonnen wird. Es wird geraten, die Domäne sowohl mit Fachexperten als auch mit den Entwicklern, gemeinsam zu modellieren und diskutieren. Als Resultat entsteht ein Domänenmodell, mit dessen Hilfe die Kommunikation zwischen Fachexperten und Entwicklern verbessert werden kann. Das Modell sollte hierbei frei von technischen Aspekten sein und wichtige Begrifflichkeiten, sowie deren Bedeutungen klar definiert werden. Dadurch kann eine gemeinsame Terminologie, auch genannt *Ubiquitous Language*, im Team entwickelt werden (Vgl. [20]).

2.2.1 Event-Storming

Event-Storming ist ein von Alberto Brandolini entwickeltes und flexibles Workshop-Format zum gemeinsamen Erkunden komplexer Fachdomänen. Durch die Flexibilität ermöglicht Event-Storming fachübergreifende Unterhaltungen zwischen Stakeholdern aus unterschiedlichen Bereichen (Vgl. [7]).
Ziel ist eine Übersicht von (fachübergreifenden) relevanten Ereignissen und Prozessen. Nach der traditionellen Methode erfolgt dies anhand von farbigen Klebezetteln an einer bereitgestellten Wand. Die Wand repräsentiert dabei einen Zeitstrahl auf dem die Klebezettel zeitlich angeordnet werden. Die Zettel an sich haben, abhängig von ihrer Farbe, unterschiedliche Bedeutungen (Vgl. [1]). Die grundlegenden Schritte sind dabei folgende:

- **Richtige Personen einladen**
  Idealerweise nehmen 6 bis 8 Personen aus den relevanten Abteilungen teil, die Fragen und beantworten können.

- **Uneingeschränkte Modellierfläche bieten**
  Um das Problem im vollen Umfang zu betrachten, sollte den Teilnehmern ausreichend Platz zur Verfügung gestellt werden. Es soll damit vermieden werden, dass ein Teil der Domäne auf Grund von Platzproblemen nicht ausreichend betrachtet wird.

- **Die Domäne anhand von Domain-Events erforschen**
  Ein *Domain-Event* ist ein relevantes Ereignis der Domäne und stößt eventuell weitere Events an. Sie werden (mit einem *orangene* Klebezettel) ihrem Zeitpunkt entsprechend auf der Timeline platziert.

- **Ursprung der Domain-Events erforschen**
  Hier wird unterschieden, ob der Ursprung eine direkte Konsequenz einer Benutzeraktion ist (*Command* mit *blauem* Klebezettel), durch ein externes System erzeigt wurde, auf Grund einer Zeitvorgabe erfolgt (durch *lila* Klebezettel repräsentiert) oder durch ein anderes Domain-Event ausgelöst (*orangene* Klebezettel näher zusammen)

- **Suche nach Aggregaten**
  Ein Aggregat ist ein Teil des Systems, welches Anweisungen bekommt und entscheidet, ob ein Domain-Event ausgeführt wird oder nicht.

(Vgl. [6])

Brandolini unterscheidet Workshops zudem nach dessen Zweck in verschiedene Varianten, zum Beispiel in *Big-Picture-, Process-Modelling-* und *Software-Design-*Workshops. Ein *Big-Picture Workshop* dreht sich dabei hauptsächlich um das Erforschen. Es wird meist für Kick-Off-Veranstaltungen eingesetzt (Vgl. [7]).

Es existieren jedoch auch viele Abwandlungen des hier beschriebenen Event-Stormings. Brandolini macht deutlich, dass sich das Event-Storming seit dem hier referenzierten Artikel weiterentwickelt hat (Vgl. [6]).

2.3 Entwurf-Prinzipien

Ein Entwurfs-Prinzip kann als eine Gesetzmäßigkeit oder übergreifende Regel definiert werden, an denen sich beim Entwurf einer Software orientiert werden kann. Sie basieren auf Erfahrungen und haben sich über die Jahre in der Praxis bewährt. Dennoch sollte der Fokus der Entwicklung auf den konkreten Anforderungen liegen und nicht auf de Einhaltung von Prinzipien (Vgl. [20, Kap. 4.1.2]). Im Folgenden werden einige wichtige Prinzipien vorgestellt.

2.3.1 Geringe Kopplung

Kopplung kann auch als Beziehung zwischen Komponenten verstanden werden und ermöglicht eine Zusammenarbeit dieser. Der Nachteil einer solchen Zusammenarbeit ist eine Abhängigkeit, die zwischen den Komponenten entsteht. Wenn nämlich eine Änderung an einer Komponente durchgeführt wird, könnte dies Auswirkungen auf die abhängige Komponente haben, wodurch diese eventuell auch angepasst werden muss. Bei vielen Abhängigkeiten im System kann dies zu einer hohen Komplexität und verringerten Flexibilität führen. Da eine Zusammenarbeit aber nötig ist und dadurch Abhängigkeiten bestehen müssen, wird eine möglichst geringe Kopplung angestrebt (Vgl. [20, Kap. 4.1.2]).

2.3.2 Trennung von Verantwortlichkeiten

Dieses Prinzip wird auch *'Separation of Concern' (SoC)* genannt und plädiert dafür bestimmten Teilen der Anwendung spezifische Zuständigkeiten zuzuteilen. Diese Trennung

betrifft alle Ebenen des Entwurfs, von der Klasse bis hin zu Subsystemen an sich. Verantwortlichkeiten können dabei als *Wissen* (für bestimmte Informationen zuständig) oder aber als *Handeln* (für bestimmte Aktivitäten zuständig) verstanden werden (Vgl. [20, Kap. 4.1.2]).

2.3.3 Geheimnisprinzip

Das Geheimnisprinzip, auch genannt *Information Hiding*, kann als eine Art Spezialisierung des SoC-Prinzips interpretiert werden. Es strebt die Trennung des *'Was'* vom *'Wie'* an. Das bedeutet, dass die innere Funktionsweise einer Komponente durch eine Abstraktion von der Außenwelt gekapselt wird. Es sollte nur nach außen getragen werden, was die Funktion ist, aber nicht wie diese implementiert ist. Ohne diese Abstraktion der Funktionalität kann es zu ungewollten Abhängigkeiten und damit zu mehr Komplexität im System führen (Vgl. [20, Kap. 4.1.2]).

2.4 Architektur- und Entwurfsmuster

Im folgenden Abschnitt werden relevante Architektur- und Entwurfsmuster vorgestellt, an denen sich bei diesem Projekt orientiert wurde. Einige Problemstellungen treten häufiger in der Softwareentwicklung auf, die im Kern das gleiche Problem in abgewandelten Formen beschreiben. Aus diesem Grund wurden Muster entwickelt, die als abstrakte *Best-Practice-Lösungen* für diese wiederkehrenden Kern-Probleme beschrieben werden(Vgl. [18, Kap. 8.1]).

2.4.1 Schichtenarchitektur

Die Schichtenarchitektur ist ein beliebter Architektur-Stil, der oft in der Software- Entwicklung verwendet wird. Die Software wird hierbei in Schichten bzw. Layer geteilt. Jeder dieser Layer hat eine klar definierte Rolle und bietet der darüber liegenden Schicht definierte Dienste an. Eine Schichtenarchitektur kann daher die Umsetzung des *'Separation of Concerns'* (SoC) -Prinzips unterstützen. Mit dem Architektur-Stil gehen jedoch auch bestimmte Regeln einher, die bei der Umsetzung eingehalten werden müssen (Vgl.[18, Kap. 7.5]):

- Eine Schicht gibt weder etwas über die innere Funktionsweise bekannt noch die darunterliegenden Schichten. Es kapselt also seine Details der Implementierung ab.
- Es besteht eine *Top-Down-Kommunikation* zwischen den Schichten. Demnach können höhere Schichten die Dienste unterer Schichten in Anspruch nehmen, aber nicht umgekehrt, sonst könnte es zu zirkulären Abhängigkeiten in dem System kommen.
- Die Kommunikation ist über Schnittstellen klar definiert.

Der bekannteste und beliebteste Stil der Schichtenarchitektur ist die Aufteilung in drei Schichten (3-Schichtenarchitektur). Hierbei wird die Anwendung in folgende Schichten strukturiert: Präsentationsschicht, Anwendungsschicht und Persistenzschicht.

Die **Persistenzschicht** (auch genannt *Datenschicht*), bezeichnet den untersten Layer und beinhaltet Datenquellen, z.B. Datenbanken (relational und objekt-orientiert), in denen die Daten der Anwendung persistent gehalten werden.
Die **Anwendungsschicht** (auch genannt *Logikschicht*), baut auf der Persistenzschicht auf und beinhaltet die Kernfunktionalität des Systems. Typische Zuständigkeiten dieses Layers sind die Verarbeitung und Aufbereitung von Daten für die Präsentationsschicht.
Die oberste Schicht, die **Präsentationsschicht**, beinhaltet die Kommunikation mit dem Benutzer. Aufgrund der Interaktion befindet sich auch etwas Logik in dieser Schicht, z.B. bei der Reaktion auf das Klicken eines Buttons. Diese Logik bezieht sich jedoch eher auf die GUI-Elemente selbst und wird daher nicht als Geschäftslogik interpretiert (Vgl. [18, Kap. 7.5.2]).

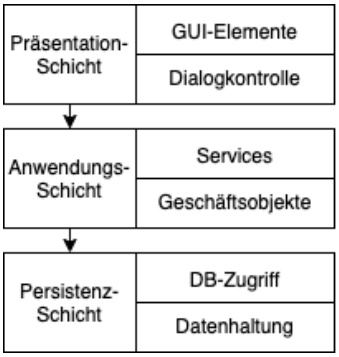


Abbildung 2.1: Aufteilung Schichtenarchitektur (erstellt mit draw.io) angelehnt an (Vgl. [13, S. 12])

Nachteile der Schichtenarchitektur sind, dass die Performance (= Laufzeiteffizienz) eines Systems unter dem Durchreichen der Schichten leiden könnte. Zudem können manche Än-

derungen die Anpassung sämtlicher Schichten bedeuten, wenn zum Beispiel ein Datenfeld hinzugefügt wird, das sowohl persistent gespeichert als auch in der Benutzeroberfläche angezeigt werden soll (Vgl. [20])

2.4.2 MVC-Pattern

„Mit Model-View-Controller (MVC) wird ein Interaktionsmuster in der Präsentationsschicht von Software beschrieben“ [14, Kap. 8.2]. Die Buchstaben im Namen steht hierbei für **Model, View, Controller**.

Das MVC-Pattern ist ein Muster, das schon viele Jahre Bestand hat und einen wichtigen Platz in der objektorientierten Entwicklung einnimmt. Über die Jahre haben sich allerdings viele verschiedene Varianten herausgebildet (Vgl. [14, Kap. 8.2]).

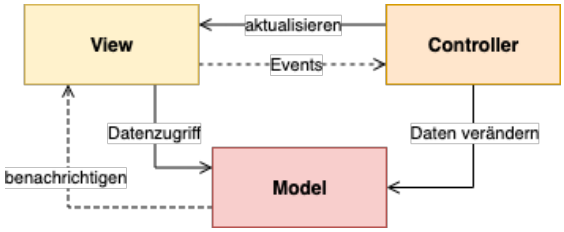


Abbildung 2.2: MVC-Pattern angelehnt an [20] (erstellt mit draw.io)

In dem Muster verwaltet das *Model* den Zustand eines Konstrukts und dient somit als eine Art Referenz für die Darstellung.
Die *View* ist für die Darstellung bzw. Präsentation der Daten verantwortlich und sollte selbst keine fachliche Logik beinhalten. Der Darstellung ist dabei jedoch keine Grenze gesetzt und kann in allen möglichen Formen erfolgen (Vgl. [14, Kap. 8.2.3]).
Der *Controller* nimmt die Benutzereingaben entgegen und gibt diese in konsolidierter Form an die View oder das Model weiter (Vgl. [20]).

Passive-View-Pattern

Das *Passive-View*-Pattern ist eine Variation des MVC. Das Besondere an diesem Pattern ist, dass die View komplett passiv gemacht wird, und sich somit nicht mehr eigenständig von dem Model aktualisieren muss. Entsprechend bestehen keine Abhängigkeiten zwischen dem Model und der View (Vgl. [10]).

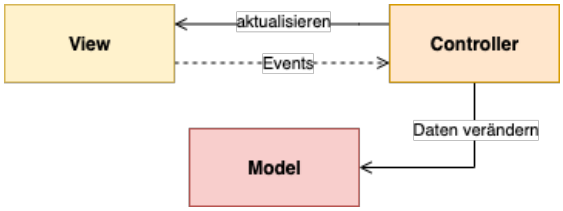


Abbildung 2.3: Passive-View-Pattern angelehnt an [20] (erstellt mit draw.io)

Abgrenzung zur Schichtenarchitektur

Das MVC-Pattern ist als ein Interaktionsmuster zu verstehen und setzt voraus, dass eine wichtige Architekturentscheidung bereits getroffen wurde: Die Verwendung von **Schichten** und die Trennung der Präsentationsschicht von der restlichen Anwendung (Vgl. [14, Kap. 8.2.3]).

Hieraus ergeben sich einige wichtige Abgrenzungen:

- „MVC ist kein Schichtenmodell“ [14, Kap. 8.2.3]
- „MVC ist ein Muster für Interaktionen in der Präsentationsschicht“ [14, 8.2.3].
- „MVC ist keine komplette Architektur“ [14, 8.2.3].

Dies bedeutet, dass die Schichtenarchitektur nicht mit dem MVC-Pattern gleichzusetzen ist, sondern eine Voraussetzung dafür ist. Zudem wird deutlich, dass das Modell im MVC-Pattern nicht dafür zuständig ist Geschäftslogik oder Persistenz zu enthalten. Das Modell im MVC-Pattern dient der „Darstellbarkeit in der Präsentationsschicht“ [14, Kap. 8.2.3] und hält entsprechend Daten vor.

In der Praxis können Darstellbarkeit in der Präsentationsschicht und Anwendungslogik von demselben Objekt übernommen werden. Es ist allerdings wichtig sich vor Augen zu halten, dass hier Aspekte aus der Schichtenarchitektur und dem MVC zusammenfließen (Vgl. [14, Kap. 8.2.3]).

2.4.3 Interface-Pattern

Ein einfaches und grundlegendes Muster ist das Interface bzw. Schnittstellen-Pattern. Bei diesem Muster wird eine Schnittstelle und dessen Implementierung getrennt. Der

Vorteil hierbei besteht darin, dass die Schnittstellen sauber und klar definiert werden können und Klassen, welche das Interface nutzen, nicht die konkrete Implementierung der Schnittstelle wissen müssen. Daher erleichtert diese eine Trennung und die Entkopplung von Klassen.
Angewandt wird dieses Pattern, um eine klar definierte Schnittstelle mit begrenzt benutzbaren Methoden anzubieten, oder um, wenn es für eine Schnittstelle mehrere Implementierungen geben soll, und diese jedoch vor dem Benutzer der Schnittstelle zu verstecken (Vgl.[18, Kap. 8.2.1]).

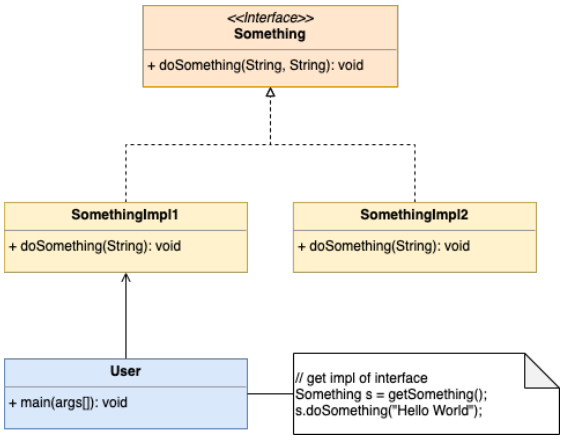


Abbildung 2.4: Beispiel Interface-Pattern (erstellt mit draw.io)

Die Abbildung zeigt ein Interface (*Something*), welches zwei Implementierungen hat (*SomethingImpl1 und SomethingImpl2*). Eine Klasse (*User*) erhält eine konkrete Implementierung des Ínterfaces zugewiesen. Über diese konkrete Implementierung weiß die Klasse (*User*) jedoch nicht mehr, als die in der Schnittstelle definierten Dienste und kann unabhängig von der Implementierung nur diese in Anspruch nehmen.

2.4.4 Facade-Pattern

Das Facade-Pattern ist Teil der Struktur-Muster und definiert ein Interface als zusammengefasster Zugangspunkt zu einem Subsystem. Dieses übergeordnete Interface erleichtert die Benutzung des darunterliegenden Subsystems und kann so eine geringere Kopplung zwischen Subsystemen erreichen. Zudem werden die Komponenten des Subsystems abgeschirmt von den Klassen, welches das Interface ansteuern. Dadurch kann die Anzahl der Abhängigkeiten reduziert und komplexe oder zirkuläre Abhängigkeiten vermieden

werden.
Angewandt wird dieses Pattern, um eine einfache Schnittstelle zu einem komplexen Subsystem anzubieten, um ein Subsystem von anderen zu entkoppeln, oder um Subsysteme in Schichten zu unterteilen und einen Eingangspunkt zu definieren (Vgl.[11, Kap. 4]).

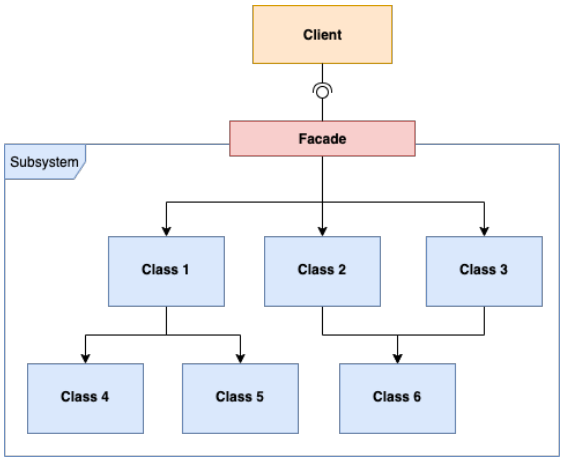


Abbildung 2.5: Beispiel Facade-Pattern (erstellt mit draw.io)

Die Abbildung zeigt die Facade als übergeordnetes Interface, welches von dem Client genutzt wird. Der Client greift nur über diesen Zugangspunkt auf das Subsystem zu, ohne die konkreten Klassen 1-6 im Subsystem zu kennen.

3 Projektorganisation

In diesem Kapitel wird auf die Organisation des Projekts und das zugrundeliegende Vorgehensmodell eingegangen.

3.1 Zeitplan

Es bestand ein fester Endtermin des Projekts am 11. Februar 2021. Dieser ist abhängig von der Abgabe der vorliegenden Arbeit und wird daher von der Bachelorandin als Stakeholder definiert.

3.2 Team

Zur Entwicklung werden vom kooperierenden Krankenhaus drei Neuropsychologen als Unterstützung zur Verfügung gestellt. Da die Unterstützung jedoch während der regulären Arbeitszeit stattfindet, ist diese nur begrenzt verfügbar.
Es wurden wöchentliche Meetings mit den Neuropsychologen vereinbart, sowie eine zentrale Ansprechperson. Auf Grund der Corona-Pandemie sind persönliche Treffen nicht möglich gewesen, was den Kommunikationsbedarf und -aufwand im Projekt deutlich erhöhte.

3.3 Vorgehensmodell

Das Projekt wurde agil durchgeführt, da in dem kooperierenden Krankenhaus zum Projektstart noch kein Programm zur angegebenen Problemstellung existierte. Demnach waren die Anforderungen und Prioritäten an ein solches Programm zu Beginn nicht stabil oder im vornherein klar definiert. Um auf mögliche Änderungen bestmöglich reagieren

zu können, wurde daher ein iteratives Vorgehen bevorzugt.
Orientierung gab dabei die Scrum-Methode. Entsprechend wurde mit den Kunden ein Backlog an User-Stories erarbeitet und in einer Story-Map festgehalten. Vor jedem Sprint wurde gemeinsam entschieden, welche Stories bearbeitet werden sollten und diese nach deren Priorität eingestuft. Im Laufe des Sprints wurden diese Stories implementiert und die neue Funktionalität in das Produkt eingearbeitet. Die Anforderungen während des Sprints konnten nicht geändert werden, aufkommende Anmerkungen wurden jedoch dokumentiert und in der nächsten Iteration besprochen. Dies ist bei Scrum so vorgesehen und sorgt für einen ruhigen Entwicklungsprozess, bei dem vom Kunden verlangt wird, dass er Anforderungen frühzeitig kommuniziert.

3.3.1 Abweichungen von Scrum

Da, abgesehen vom Kunden, alle Rollen und die damit verbundene Verantwortlichkeiten von einer Person getragen wurden und sich Scrum jedoch auf die Zusammenarbeit eines Teams bezieht, waren einige Abweichungen vom Prozess nötig.

Aufgrund der Team-Zusammenstellung und den begrenzten Ressourcen erschien es nicht sinnvoll *Daily-Scrum*-Meetings abzuhalten. Ein Sprint wurde daher auf eine Woche gesetzt, in der Fragen gestellt werden konnten, User-Stories priorisiert, sowie über Anforderungen und den aktuellen Stand des Projektes gesprochen wurde.
Ebenfalls wurde, auf Grund des relativ kurzen Zeitrahmens, nicht davon ausgegangen, dass mehrere kleine Release-Zyklen durchgeführt werden können. Zudem wurde bei der ersten Besprechung bereits festgestellt, dass nicht alle Anforderungen in dem zeitlichen Rahmen umzusetzen sind. Daher wurde ein größeres Release des *Minimum Viable Products* angestrebt. Demnach wurde auf eine Retrospektiven nach den Sprints verzichtet. Stattdessen wurde eine abschließende Retrospektive am Ende der Zusammenarbeit angestrebt.

4 Anforderungsanalyse

In diesem Kapitel wird der Analyse-Prozess und die daraus resultierenden Anforderungen an das zu entwickelnde System aufgeführt.

4.1 Stakeholder

Zunächst wurden die für dieses Projekt relevanten Stakeholder identifiziert:

- *Kooperierendes Krankenhaus*
  Im Wesentlichen ist das Krankenhaus an rechtlichen Aspekten, wie z.B. dem Datenschutz der Patienten und die Sicherheit der IT-Infrastruktur interessiert.
- *Neuropsychologen des Krankenhauses*
  Die Neuropsychologen des Krankenhauses hingegen sind wesentlich an der Funktionalität und Benutzbarkeit des Programms interessiert.
- *Bachelorandin*
  Die Bachelorandin ist am Erfolg des Projektes und den dahinterliegenden Entwicklungs-Prozess und dessen Methoden interessiert.

4.2 Begriffsverzeichnis

Um eine gemeinsame Sprache im Projekt zu fördern, wurden wichtige Begriffe in dem folgenden Verzeichnis festgehalten und definiert.

4.2.1 Rohwert

Als Rohwert wird ein Wert bezeichnet, der durch die Ausführung eines Tests gemessen wurde. Je nach Art des Tests kann dieser Wert unterschiedlich aussehen. Wurde in dem Test eine Zeitmessung durchgeführt, kann der Rohwert die benötigten Millisekunden widerspiegeln, welche die Person für das Erfüllen der Aufgabe benötigt hat. Besteht der Test darin, in einer vorgegebenen Zeit möglichst viele Wörter einer bestimmten Kategorie aufzuzählen, spiegelt der Rohwert die Anzahl der genannten validen Wörter wieder.

4.2.2 Normierung

Als Normierung wird das in Bezug setzen des gemessenen Wertes einer Person mit denen einer repräsentativen Stichprobe bezeichnet. Der erzielte Rohwert muss dafür in einen Normwert transformiert werden. Dadurch kann die erbrachte Testleistung interpretiert und mit anderen Personen vergleichbar gemacht werden. Gängige Normen sind z.B. die IQ-Skala, z-Skala, T-Werte, Prozentränge, usw. (Vgl. [19, Kap. 2.3.5]).

4.2.3 Reliabilität

Die Reliabilität ist ein Gütekriterium diagnostischer Verfahren und gibt Auskunft über die Messgenauigkeit, mit der ein Merkmal von einem Test erfasst wird.
Die Reliabilität kann einen Wert zwischen 0 und 1 annehmen, wobei ein hoher Wert eine hohe Reliabilität angibt. Reliabilitäten basieren auf unterschiedlichen Schätzmethoden. Diese Methoden erzeugen Genauigkeiten für bestimmte Situationen, in denen gemessen wird. Die Werte der unterschiedlichen Methoden sind untereinander nicht austauschbar. Zum Beispiel schätzt die Retest-Reliabilität eines Tests die Genauigkeit bei Testwiederholungen (gleicher Test bei gleicher Stichprobe), die Paralleltestreliabilität hingegen die Genauigkeit zwischen parallelen Versionen eines Tests, usw. (Vgl. [19, Kap. 2.3.3]).

4.2.4 Alpha

Beim Testen von Mittelwerten wird nach einem signifikanten Unterschied zwischen Werten geprüft. Signifikant bedeutet in diesem Zusammenhang *bedeutsam, wesentlich* oder *nicht durch Zufall zustande gekommen.* Ein Alpha-Fehler oder auch *Fehler ersten Typs* beschreibt das Verwerfen einer wahren Nullhypothese aufgrund zu großer Abweichungen

der Werte. Die Größe dieser Abweichung, muss dabei vorher definiert werden, der den *Ablehnungsbereich* von dem *Nichtablehnungsbereich* trennt.
Dieser Wert ergibt sich daraus, dass man eine Wahrscheinlichkeit (Alpha) vorgibt, mit welcher man einen Fehler des ersten Typs zu begehen in Kauf nimmt. Diese Wahrscheinlichkeit (Alpha) wird daher auch *Signifikanzniveau* oder *Irrtumswahrscheinlichkeit* genannt (Vgl. [17, Kap. 10.4]).

4.2.5 Standardmessfehler (SMF)

Die beobachteten Testwerte können von dem wahren Wert abweichen. Wie stark diese Abweichung ist, kann mit Hilfe des Standardmessfehlers geschätzt werden. Dieser Wert wird aus der Reliabilität des Tests und der Standardabweichung des benutzten Normwerts berechnet. Er gibt die Stärke der Streuung der Messfehler um den wahren Wert der Person an (Streuung der Fehlerwerte) (Vgl. [19, Kap. 2.1]).

4.2.6 Konfidenzintervall

Ein Konfidenzintervall gibt an, in welchem Bereich sich der wahre Wert einer Person befindet. Für die Berechnung wird ein vorher definierter Sicherheitswert bzw. eine Irrtumswahrscheinlichkeit (Alpha) festgelegt. Weitere Bezeichnungen für diesen berechneten Bereich sind 'Vertrauensintervall' oder 'Erwartungbereich' (Vgl. [19, Kap. 2.1]).

4.2.7 Test-Richtung

Die Richtung, in die eine Testung durchgeführt wird, hängt von der aufgeworfenen Fragestellung ab. Soll nach einer Abweichung nach oben und nach unten getestet werden, wird eine zweiseitige Test-Richtung gewählt. Eine einseitige Test-Richtung wird in der Regel gewählt, wenn für die Fragestellung lediglich von Interesse ist, ob ein bestimmter Wert über- oder unterschritten wird (Vgl. [19, Kap. 2.1]).

4.3 Workshop mit dem Kunden

Angelehnt an das Domain-Driven-Design (DDD) wurde mit den Kunden, welche ebenfalls die Benutzer des Programms sein werden, ein Workshop durchgeführt. Hierfür wurde

eine Präsentation vorbereitet, um den Benutzern ein grobes Verständnis für den Prozess der Softwareentwicklung zu vermitteln, sowie die verwendete Methodik und deren Nutzen vorzustellen. In dem folgenden Abschnitt wird der Ablauf und die entstandenen Ergebnisse des Workshops zusammengefasst.

4.3.1 Event-Storming

Zu Beginn des Workshops wurde ein 'Big-Picture'-Event-Storming mit drei Teilnehmern durchgeführt. Die Teilnehmer waren allesamt angestellte Neuropsychologen des kooperierenden Krankenhauses. Die Methode des Event-Stormings wurde angewandt, um das Eis zu brechen, sowie gemeinsam den zu modellierenden Prozess in seinen Teilschritten zu erarbeiten und zu verdeutlichen. Es wurden zunächst die Teilschritte (grün) und Domain-Events (orange) modelliert, um Anschluss wurden zudem die benutzten Systeme (gelb) und Kommentare (pink) hinzugefügt. Die folgende Abbildung zeigt die Ergebnisse des Event-Stormings mit den angegebenen Farben.

Abbildung 4.1: Ergebnisse des Event-Stormings

Beim Modellieren des Prozesses zeigte sich, dass aktuell mehrere verschiedene Programme bzw. Dateien von den Neuropsychologen benutzt werden müssen. An den gelben *'Zetteln'* sieht man, dass der Auftrag für eine neuropsychologische Diagnostik zunächst durch das

Krankenhaus-interne System (CC) eingeht. Die Vorbereitung und Durchführung der Diagnostik wird manuell mit *Paper-and-Pencil* oder einer computergestützten Messung am PC durchgeführt. Im Anschluss werden mehrere Excel-Dateien und verschiedene Word-Programme verwendet, um die Daten entsprechend zu transformieren, auszuwerten und zu visualisieren. Einige dieser Dateien beinhalten Excel-Tabellen und Grafiken, die von einem früheren Mitarbeiter des Krankenhauses angelegt wurden. Zu dem Aufbau konnte keiner der anwesenden Neuropsychologen konkret zu Stellung nehmen, ohne sich zunächst tiefer damit zu beschäftigen. Die Dateien wirken insgesamt unübersichtlich und damit auch schwer nachvollziehbar. Zuletzt wird der Bericht mit den normierten Testdaten und der erstellten Visualisierung der Konfidenzintervalle wieder in das Krankenhaus-interne System (CC) eingefügt.
Den Kommentaren ist zu entnehmen, dass Programmwechsel die größten Unannehmlichkeiten im vorliegenden Prozess darstellen, da teilweise ein Programm für eine kleine Unterfunktion verwendet werden muss. Ein Beispiel hierfür ist das Forschungs-Datenbank-Programm *Access*, welches lediglich der Ausgabe einer Fallzeile dient, die wiederum in einem anderen Schritt für eine Excel-Datei benötigt wird.
Es wurde angegeben, dass die grafische Darstellung der Konfidenzintervalle eine sehr nützliche Funktion des bisherigen Prozesses ist und unbedingt beibehalten werden sollte.

4.3.2 Nutzerrollen

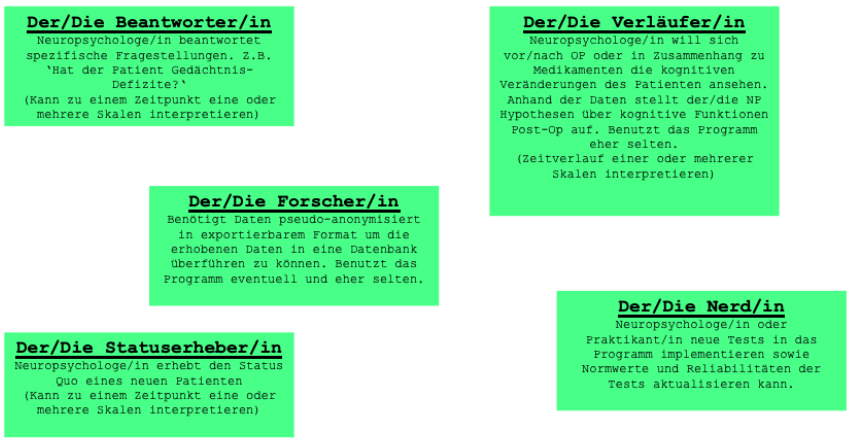


Abbildung 4.2: Nutzerrollen aus dem Workshop

Im Workshop wurden anschließend Nutzerrollen mit den Teilnehmern rausgearbeitet, welche mögliche Benutzergruppen des zu entwickelnden Systems identifizieren sollen. Diese geben Aufschluss über mögliche Funktionen, die das Tool beinhalten sollte.

4.3.3 Personas

Personas werden benutzt, um die Benutzer eines Programms in größerem Detail zu erfassen und zu beschreiben. Sie sind fiktive Personen, die auf Attributen und Verhalten wahrer Personen basieren und dadurch ermöglichen, reale Benutzergruppen zu personifizieren. Diese Personifikation soll das Ableiten von Anforderungen erleichtern, da auf typische Verhaltensweisen, Ziele und Motivationen der fiktiven Persona eingegangen werden kann (Vgl. [15]).

Im Workshop wurden von den Teilnehmern folgende Personas erstellt:

Abbildung 4.3: Personas: I & II

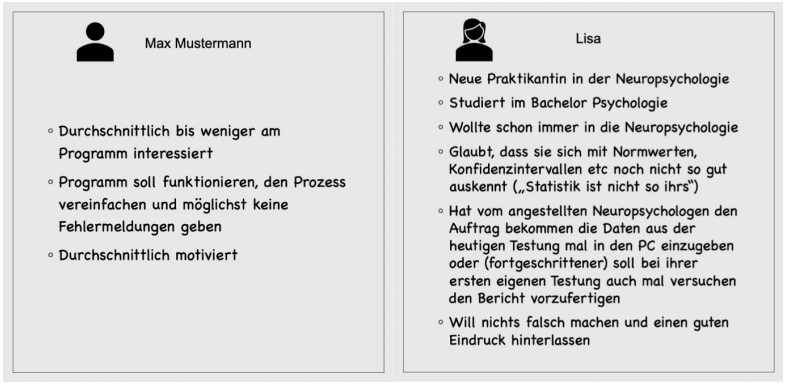


Abbildung 4.4: Personas: III & IV

Von den Personas wurden folgende Qualitätsanforderungen an das zu entwickelnde Programm abgeleitet:

- Intuitive und einfache Benutzeroberfläche (nicht überladen)
- Benutzern aussagekräftige Fehlermeldungen und Hinweistexte ausgeben
- Option, bei Fehlern Werte noch einmal zu überarbeiten (nicht immer ganz von vorne anfangen müssen)
- Auch mit fehlenden Werten klarkommen (nicht immer für alle Felder Werte vorhanden)
- Keine fachlichen Abkürzungen verwenden, die nicht allen geläufig sind
- Bei Berechnungen die benutzten Formeln mit anzeigen, sodass Ergebnisse nachvollziehbar sind
- Benutzer um Bestätigung bitten, bevor Daten unwiderruflich verändert werden
- Bei Fehlern einen Stacktrace ausgeben, um Fehlersuche zu unterstützen

4.3.4 Story-Writing-Workshop

Im Anschluss wurde ein Story-Writing-Workshop durchgeführt. Die Teilnehmer erhielten eine kurze Einführung in das Thema User-Stories. In diesem Teil des Workshops ging es primär darum, möglichst viele Stories anzufertigen, entsprechend stand die Quantität und nicht die Qualität der Stories im Fokus. Nachdem den Teilnehmern keine weiteren Stories einfielen, wurden diese gemeinsam besprochen und zusammengehörige Anforderungen gruppiert.

4.3.5 User-Story-Mapping

Das User-Story-Mapping wurde mit den Teilnehmern durchgeführt, um eine erste Priorisierung der erstellten User-Stories vorzunehmen.
Die Aufteilung erfolgte zunächst in drei *'Releases'*. Für das erste Release wurde das *Minimum Viable Product* (MVP) identifiziert, was die Hauptfunktionalität der Software beinhalten sollte. Die entsprechenden Stories wurden in *must-be*-Anforderungen priorisiert. Im zweiten Release kamen ebenfalls wichtige Anforderungen an das Programm,

die als *expected*-Stories klassifiziert und in Form von Features integriert werden sollen. Das letzte Release beinhaltete Stories, die sehr hilfreich wären, aber eher als zusätzliche Funktionalität, im Sinne von *nice-to-have*-Anforderungen angesehen wurden.
Aus den User-Stories und dem entsprechenden Mapping wurde ein Story-Board erstellt. Die Anforderungen wurden hierbei in folgende drei User-Aktivitäten klassifiziert: *Verwalten, Analysieren und Interpretieren.* Diese Aktivitäten wurden wiederum in Tätigkeiten spezifiziert. Die Kernfunktionalität und entsprechenden *must-be*-Anforderungen wurden als MVP-Stories gruppiert und in dunkelblau farbcodiert. Die verbleibenden Anforderungen wurden im Backlog festgehalten, wobei die *expected*-Stories in grün und die *nice-to-have*-Stories in grau farbkodiert wurden.

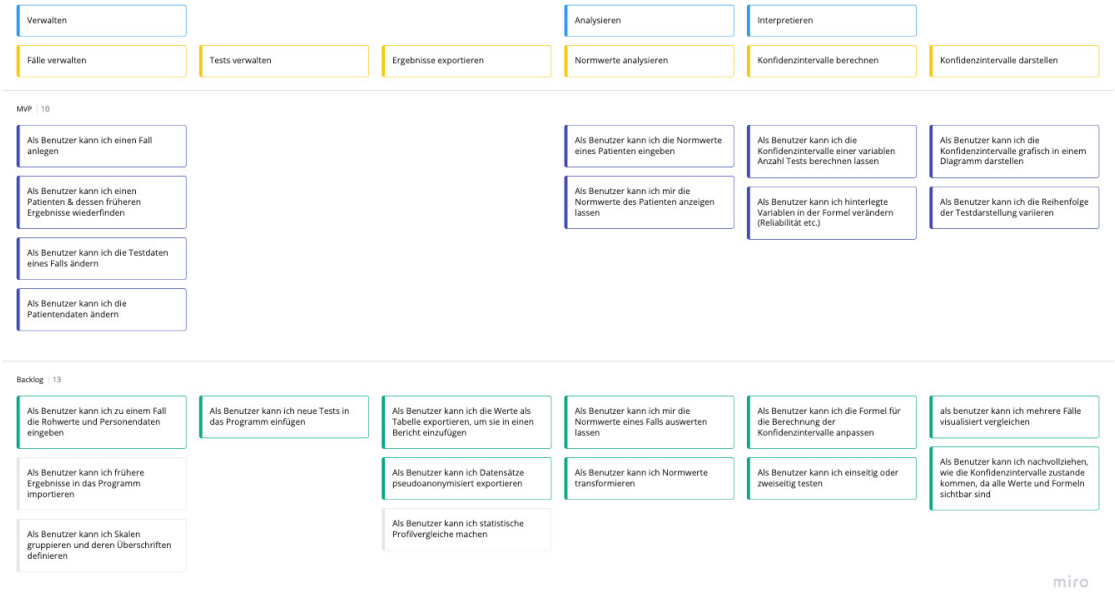


Abbildung 4.5: User Story Board (erstellt mit miro.com)

4.4 Funktionalität

4.4.1 Kernaufgabe des Systems

Das System unterstützt Neuropsychologen bei der Analyse und Interpretation der Testdaten einer neuropsychologischen Diagnostik. Die wichtigsten Aspekte der Fachdomäne sind Testverfahren, Messwerte, Normwerte, Reliabilitäten und Konfidenzintervalle.

4.4.2 Funktionale Anforderungen an das System

Durch die Nutzerrollen und User-Stories ließen sich folgende funktionale Anforderungen an das System ableiten:

- Auswertung der Normwerte anhand von Personendaten und Rohwerten der Testung
- Berechnung und Darstellung der Konfidenzintervalle (wenn Variablen dafür vorhanden)
- Exportieren von Falldaten (Rohwerte, Normwerte, Konfidenzintervalle), um diese in einen Bericht oder eine Datenbank einzufügen
- Übersichtlicher Vergleich von zwei oder mehr Testzeitpunkten einer Person
- Hinzufügen, Entfernen und Updaten von Testverfahren und Skalen

4.4.3 Qualitätsanforderungen an das System

Nicht nur funktionale, sondern auch nicht-funktionale Anforderungen an das System sollten festgehalten und in dem Entscheidungsprozess berücksichtigt werden (Vgl. [20, Kap. 3.2.3]).

Die abgeleiteten Qualitätsanforderungen durch die Personas fallen zusammengefasst laut DIN/ISO 25010 unter die Qualitätsmerkmale der **Betreibbarkeit** (Vgl. [20, Kap. 3.2.3]).

- **Erlernbarkeit**
  Dieses Ziel bezieht sich auf den Aufwand für einen Benutzer, den korrekten Umgang mit dem System zu erlernen (Vgl. [20, Kap. 3.2.3]).
- **Einfachheit der Benutzung**
  Die Einfachheit bezieht sich darauf, inwiefern das Softwareprodukt es den Benutzern leicht macht, es zu bedienen (Vgl. [20, Kap. 3.2.3]).

Anwendungsszenarien

Um die genannten Anforderungen zu konkretisieren, werden Anwendungsszenarien definiert werden. Sie sollen das Verhalten des Systems bei seiner Nutzung in bestimmten Situationen beschreiben (Vgl. [20, Kap. 3.2.3]).

- Das System muss damit umgehen können, dass nicht alle Eingabefelder mit Daten gefüllt werden und im Normalbetrieb weiterarbeiten.
- Vor unwiderruflichen Änderungen muss das System explizit um Bestätigung fragen, um eine Aktion durchzuführen.
- Bei Eingabe unzulässiger Daten in die Eingabefelder muss das System entsprechende spezifische Hinweistexte ausgeben und anschließend im Normalbetrieb weiterarbeiten.

5 Spezifikation

In diesem Kapitel wird der Prozess der Spezifikation dargestellt. Hierfür wurden zusammen mit den Neuropsychologen ein Domänen-Modell erstellt, sowie wichtige Begrifflichkeiten und konkrete Anwendungsfälle definiert.

5.1 Fachlicher Kontext

Entsprechend des Vorgehens beim Domain-Driven-Design, wurde mit den Fachexperten (in diesem Fall die Neuropsychologen) diskutiert, um die erarbeiteten Anforderungen genauer zu spezifizieren. Dies führte zu einem klareren Verständnis des fachlichen Kontexts der zu entwickelnden Anwendung.

5.1.1 Fachliches Datenmodell

Das gemeinsam erstellte Domänenmodell bietet einen Überblick über die relevanten fachlichen Konstrukte und deren Beziehung untereinander.

Person

In der neuropsychologischen Diagnostik wird eine **Person** untersucht - die *Testperson*. Um deren Personendaten zu anonymisieren, wird eine eindeutige Personen-Id verwendet, sowie der Name des Patienten in Form von Initialen festgehalten. Relevant für die spätere Normierung der erzielten Rohwerte, sind in der Regel das Geschlecht, sowie das Alter und die Bildungsjahre zum Testzeitpunkt. Um das Alter als stabiles Personenmerkmal aufzunehmen, wurde das Geburtsjahr der Person als Attribut ausgewählt.

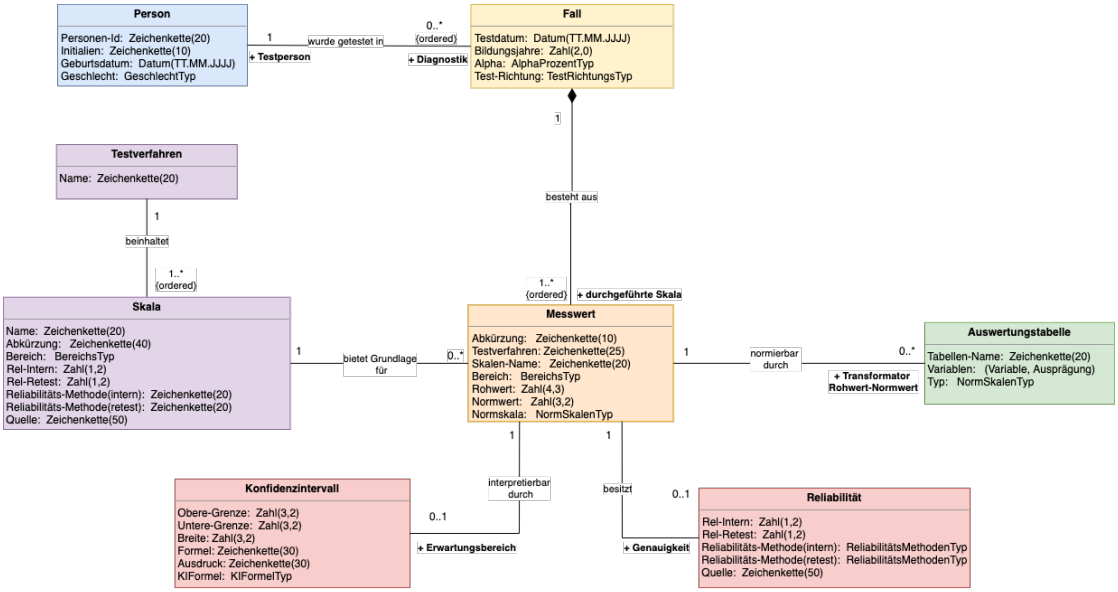


Abbildung 5.1: Fachliches Datenmodell (erstellt mit draw.io)

Fall

Das Durchführen einer *Diagnostik* an einem Testdatum, wird **Fall** genannt. Eine Person kann entweder noch gar nicht oder mehrere Male getestet worden sein. Eine Person kann nur einmal am Tag untersucht werden, daher kann anhand des Testdatums ein Fall einer Person eindeutig identifiziert werden. Zudem sind die Fälle einer Person anhand des Testdatums zeitlich klar zu ordnen. Da bei der Interpretation auch vorherige Fälle zum Vergleich hinzugezogen werden, ist die Ordnung der Fälle wichtig.
Für einen Fall muss zudem im Vorhinein ein Prozentwert tolerierter Alpha-Fehler Wahrscheinlichkeit, sowie eine Test-Richtung definiert werden. Das Attribut *Bildungsjahre* wurde ebenfalls dem Fall zugeordnet, da dies ein eher instabiles Personenmerkmal ist und sich zwischen Fällen unterscheiden kann.

Messwert

Ein Fall besteht aus einem oder mehreren Messwerten. Diese basieren auf einem von Forschern entwickelten **Testverfahren** und haben einen eindeutigen Namen.
Ein Testverfahren kann aus einer oder mehreren **Skalen** bestehen. Eine Skala hat einen eindeutigen Namen und eine eindeutige Abkürzung, um diese in Diagrammen verkürzt

darzustellen. Der Bereich einer Skala, gibt Aufschluss über den neurologischen Funktionen, die mit dem Test erfasst werden (z.B. Aufmerksamkeit, Gedächtnis, ...).
Für die Testverfahren und deren Skalen gibt es Manuale, in denen wichtige Informationen zum Verfahren festgehalten werden. Unter Anderem können die *Reliabilitäten* und benutzten Schätzmethoden für die Skalen angegeben werden. Die Manuale enthalten ebenfalls die Normierungstabellen, zum Transformieren von Roh- in Normwerte der jeweiligen Skala.
Ein Messwert bildet also eine mit der Person *durchgeführten Skala* zu einem bestimmten Testzeitpunkt ab. Ein Messwert hat entsprechend den Namen der verwendeten Skala und des Testverfahrens, zu der diese Skala gehört, einen Bereich und Abkürzung, sowie genau einen von der Testung resultierenden Rohwert und dessen Normierung, der Normwert mit der zugrundeliegenden Normskala. Der Normwert ermöglicht es, den erhobenen Rohwert in ein Verhältnis zu setzen und damit einen Bezug zur Gesamtheit herzustellen.

Auswertungstabellen

Die Normierung von Rohwerten erfolgt über **Auswertungstabellen** in den Testmanualen oder durch das Berechnen einer angegebenen Formel. Bei der Normierung werden Personen oft anhand ihrer Merkmale (Geschlecht, Alter und/oder Bildungsjahre) differenzierter betrachtet. Diese Differenzierung variiert zwischen Testverfahren. Machen unterscheiden nur nach dem Merkmal Alter, andere nach allen drei Merkmalen. Normiert wird zudem immer in Bezug auf eine Normskala.

Beispiel:
Bei einer Person, mit den Merkmalen
(Alter= 30, Geschlecht = Weiblich, Bildungsjahre = 17 Jahre)
wird in einem Testverfahren X das Erinnern von 10 Worten (Rohwert = 10),
durch die Auswertungstabelle Y, unter den Variablen
(Alter = 25-35, Geschlecht = Weiblich, Bildungsjahre= 15-20),
ein IQ (Normskala) von 111 (Normwert) ausgewertet.

Reliabilität

Ein **Messwert** kann zudem **Reliabilitäts**-Werte besitzen. Diese sind aber nicht zwingend nötig. Eine interne Reliabilität schätzt die interne Konsistenz eines Tests und wird in der Regel bei der Konfidenzintervallberechnung verwendet. Eine retest Reliabilität

hingegen schätzt die Konsistenz bei Testwiederholungen und wird bei einem Vergleich von zwei Testzeitpunkten einer Person verwendet. Liegt jedoch nur eine retest Reliabilität vor, kann diese auch zum Berechnen des Konfidenzintervalls benutzt werden. Beide Werte beruhen auf Schätzmethoden und stammen von einer angegebenen Quelle. Alle Angaben zur Reliabilität sind nicht verpflichtend für einen Messwert.

Konfidenzintervall

Das **Konfidenzintervall**, auch *Erwartungsbereich* genannt, macht einen Normwert durch das Einbeziehen von Messfehlern (SMF) und der Messgenauigkeit der Skala (Reliabilität). Das Intervall besteht aus einer oberen und einer unteren Grenze, der Breite des Intervalls. Zudem wurden die benutze Formel und der resultierende Ausdruck hinzugefügt, um die Berechnung transparent und nachvollziehbar zu machen.

5.1.2 Datentypenverzeichnis

Um fachliche Wertebereiche oder Auswahlmöglichkeiten verständlicher und übersichtlicher darzustellen, wurden spezielle fachliche Datentypen definiert und im Modell verwendet. Diese fachlichen Datentypen werden im Folgenden näher erläutert.

GeschlechtTyp

Dieser Typ beschreibt das Geschlecht der Person. In der Neuropsychologie ist es gängig, in *weiblich* und *männlich* zu unterscheiden. Ein zusätzliches Geschlecht z.B. *divers*, wird in der Regel nicht berücksichtigt.

Wertebereich	{*männlich*, *weiblich*}

Tabelle 5.1: GeschlechtTyp Variablen

AlphaProzentTyp

Dieser Typ beschreibt die ausgewählte tolerierte Alpha-Fehler-Wahrscheinlichkeit (Alpha), welcher bei der Analyse und Interpretation der durchgeführten Diagnostik verwendet werden soll.

Wertebereich	{1, 5, 10, 20}
GUI-Darstellung	1: *1%*; 5: *5%*; 10: *10%*; 20: *20%*

Tabelle 5.2: AlphaProzentTyp Variablen

TestRichtungsTyp

Dieser Typ beschreibt die ausgewählte Richtung der Hypothesentestung im Sinne des berücksichtigten Ablehnungsbereichs.

Wertebereich	{einseitig, zweiseitig}
GUI-Darstellung	einseitig: *einseitig*; zweiseitig: *zweiseitig*

Tabelle 5.3: TestRichtungsTyp Variablen

NormSkalenTyp

Dieser Typ beschreibt die Normskala, auf der ein Normwert basiert. Da sich die Normwerte an der Normalverteilung orientieren, haben sie feste Mittelwerte und Standardabweichungen. In diesem Typ sind alle von den Neuropsychologen als relevant betrachteten Normskalen festgehalten. Aufgrund von Ungenauigkeiten eignen sich nicht alle Normskalen zur Berechnung von Konfidenzintervallen oder der Transformation des Normwertes zwischen Normskalen.

Wertebereich	{Z, IQ, T, WP, Prozentrang, Stanine}
GUI-Darstellung	Z: *z-Werte*; IQ: *IQ*; T: *T-Werte*; WP: *Wertpunkte*; Prozentrang: *Prozentrang*; Stanine: *Stanine-Normwerte*;
Mittelwert	Z: *0*; IQ: *100*; T: *50*; WP: *10*; Prozentrang: *10* ; Stanine: *5*
Standardabweichung	Z: *1*; IQ: *15*; T: *10*; WP: *3*; Prozentrang: *3*; Stanine: *2*
KI-Berechenbar	Z: *Ja*; IQ: *Ja*; T: *Ja*; WP: *Ja*; Prozentrang: *Nein*; Stanine: *Nein*

Tabelle 5.4: NormSkalenTyp Variablen

BereichsTyp

Dieser Typ beschreibt die Zugehörigkeit des Messwerts in eine definierte Bereichs-Gruppe. Diese Gruppe beschreibt, welche neuropsychologische Funktion mit dem Messwert untersucht werden. Anhand der Bereiche soll eine Sortierung erfolgen können. Daher gibt es vorgegebene Bereiche, die Neuropsychologen haben jedoch angemerkt, dass sie zudem eigene Kategorien hinzufügen können wollen.

Wertebereich	{Aufmerksamkeit, Gedächtnis, Räumlich-visuell, Verbal, Nicht zugeordnet}
GUI-Darstellung	Aufmerksamkeit: *Aufmerksamkeit*; Gedächtnis: *Gedächtnis*; Räumlich-visuell: *Räumlich-visuelle Konstruktionsfähigkeit*; Verbal: *Verbale Fähigkeit*; Nicht zugeordnet: *Nicht zugeordnet*

Tabelle 5.5: BereichsTyp Variablen

ReliabilitätsMethodenTyp

Dieser Typ beschreibt die gewählte Methode, mit welcher die Reliabilität des Messwerts geschätzt wurde.

Wertebereich	{Odd-Even, Cronbach-a, Split-Half, Paralleltest, Retest}
GUI-Darstellung	Odd-Even: *Odd-Even*; Cronbach-a: *Cronbach-a*; Split-Half: *Split-Half*; Paralleltest: *Paralleltest*; Retest: *Retest*

Tabelle 5.6: ReliabilitätsMethodenTyp Variablen

KIBerechnungsTyp

Dieser Typ beinhaltet die Logik und Formeln zur Berechnung von Konfidenzintervallen. Zentral für die Berechnung ist der z-Wert, welcher von dem gewählten Alpha und der Test-Richtung abzuleiten ist. Zudem werden abhängig von der Test-Richtung unterschiedliche Formeln zur Berechnung des Intervalls angewandt.

Wertebereich	{e-1, e-5, e-10, e-20, z-1, z-5, z-10, z-20}
Richtung und Alpha	e-1: *einseitig, Alpha 1%*, e-5: *einseitig, Alpha 5%*, e-10: *einseitig, Alpha 10%*, e-20: *einseitig, Alpha 20%*, z-1: *zweiseitig, Alpha 1%*, z-5: *zweiseitig, Alpha 5%*, z-10: *zweiseitig, Alpha 10%*, z-20: *zweiseitig, Alpha 20%*
z-Wert	e-1: *2,326*, e-5: *1,645*, e-10: *1,282*, e-20: *0,8416*, z-1: *2,576*, z-5: *1,96*, z-10: *1,645*, z-20: *1,282*
KI-Formel	einseitig: $Normwert \pm SMF \cdot z_{1 - \alpha}$; zweiseitig: $Normwert \pm SMF \cdot z_{1 - \frac{\alpha}{2}}$
SMF-Formel	$SMF = Standardabweichung \cdot \sqrt{1 - Reliabilitaet}$

Tabelle 5.7: KIBerechnungsTyp Variablen

Quellen: z-Werte [8], Berechnung der Konfidenzintervalle [5];

5.2 Anwendungsfälle

Aus den funktionalen Anforderungen der Anforderungsanalyse haben sich Use-Cases ableiten lassen, die in dem folgenden Diagramm zusammengefasst dargestellt werden.

Ein Benutzer des Systems kann Infos zu Testverfahren und Skalen hinterlegen, die beim Start des Programms aus Excel-Dateien von Programm eingelesen werden. Dies erspart dem Benutzer manuelle Eingaben. Zudem können Auswertungstabellen für die Transformation von Roh- in Normwerte hinterlegt werden.

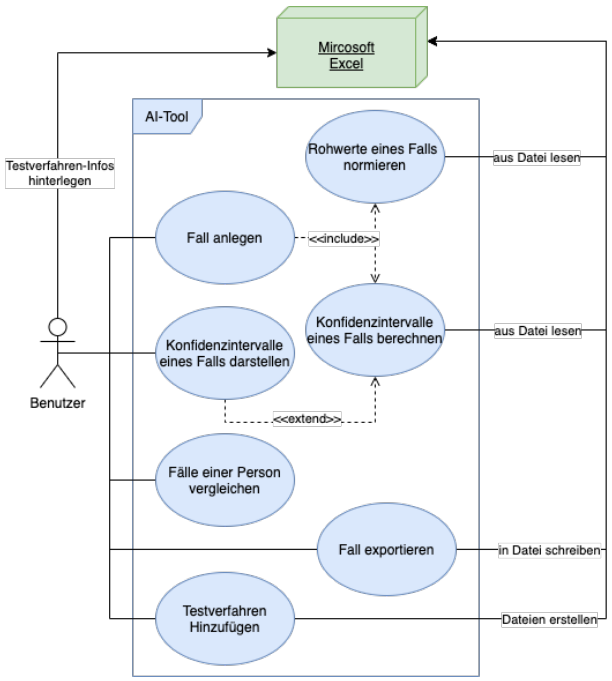


Abbildung 5.2: Use-Case Diagramm (erstellt mit draw.io)

Wird eine neue Testung durchgeführt, für welche die Ergebnisse in einem Bericht zusammengefasst und interpretiert werden müssen, kann der Benutzer einen Fall im Programm anlegen. Beim Anlegen des Falls werden dessen Rohwerte normiert und die Konfidenzintervalle berechnet. Für die Berechnung des Konfidenzintervalls braucht das Programm Informationen zu den Testverfahren und Skalen (z.B. Reliabilitäten), die aus einer Excel-Datei eingelesen werden. Für die Transformation werden Auswertungstabellen benötigt, die in Excel-Dateien hinterlegt sind und ebenfalls vom Programm eingelesen werden.

Für den Bericht soll zudem ein Diagramm der berechneten KIs angefügt werden, um die Ergebnisse zu visualisieren. Daher kann sich der Benutzer solch ein Diagramm von der Anwendung anzeigen lassen, bei dem alle berechneten KIs auf eine festgelegte Normskala vereinheitlicht abgebildet werden.

Sollte eine Person wiederholt getestet werden, kann der Benutzer Fälle einer Person vergleichen. Für diesen Vergleich werden gegebenenfalls andere Schätzmethoden (Reliabilitäten) zur Berechnung der KIs verwendet, weshalb diese bei der Darstellung gegebenenfalls berechnet werden müssen. Die KI-Diagramme der Fälle werden zum Vergleich nebeneinander dargestellt.

Um die Fall-Daten leichter in den Bericht einzufügen, kann der Benutzer einen Fall in eine Excel-Datei exportieren. Zudem kann der Benutzer neue Testverfahren hinzufügen. Um Fehler zu vermeiden, kann das Programm das Anlegen der Ordnerstrukturen und Dateierstellung und -benennung der Auswertungstabellen für den Benutzer übernehmen.

Im Folgenden werden die aufgeführten Use-Cases detaillierter betrachtet und weiter definiert. Zudem wurden Wireframes für die Use-Cases hinzugefügt, um erste Ideen zu der Benutzeroberfläche festzuhalten und gleichzeitig die Anwendung visuell zu unterstützen. Alle Wireframes wurden mit dem Tool *Balsamiq Wireframes* erstellt.

5.2.1 Anwendungsfall: Fall anlegen

Akteur: Benutzer
Ziel: Der Benutzer möchte die Daten eines Falls übersichtlich zusammengefasst haben, sowie die Rohwerte in Normwerte transformiert und die Konfidenzintervalle der im Fall durchegeführten Messwerte berechnet haben

Auslöser: Eine neue Testung wurde durchgeführt und die Testdaten müssen für einen Bericht analysiert und interpretiert werden
Vorbedingung: Die Person, mit der die Diagnostik durchgeführt wurde, ist im System angelegt
Nachbedingung: Der Fall ist im System angelegt, der transformierte Normwert und berechneten KIs sind in einer Tabelle wiederzufinden

Erfolgsszenario:

1. Benutzer startet das System, navigiert zu der Person, mit der die Testung durchgeführt wurde
2. System navigiert zur Übersicht der ausgewählten Person

3. Benutzer klickt auf 'neuen Fall anlegen'
4. System leitet weiter zu einer Tabelle mit aufgelisteten importierten Testverfahren und Skalen
5. Benutzer trägt die Rohwerte für die durchgeführten Skalen ein, die in der Liste aufzufinden sind und klickt 'Hinzufügen'
6. System fragt, ob der Fall mit den ausgewählten Werten wirklich angelegt werden soll
7. Benutzer bestätigt das Anlegen
8. System leitet weiter auf die Fall-Übersicht des soeben angelegten Falls mit einer Tabelle, in der alle relevanten Infos, sowie die jeweiligen Normwerte und Konfidenzintervalle der durchgeführten Messwerte angegeben sind

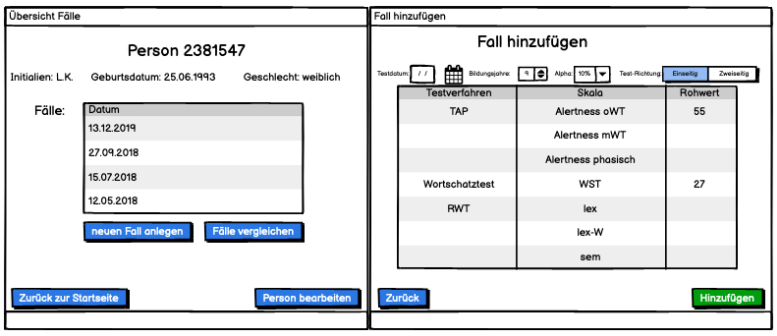


Abbildung 5.3: Wireframes Fall anlegen I

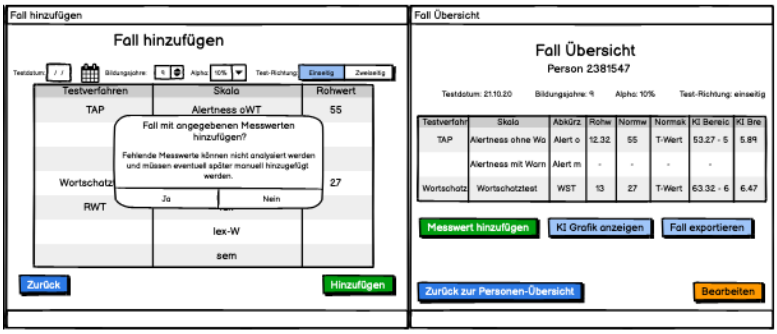


Abbildung 5.4: Wireframes Fall anlegen II

5.2.2 Anwendungsfall: Berechnete Konfidenzintervalle darstellen

Akteur: Benutzer
Ziel: Der Benutzer möchte die Konfidenzintervalle der Messwerte eines Falls in einem Diagramm dargestellt haben

Auslöser: Ein Diagramm der Konfidenzintervalle soll in einen Bericht eingefügt werden
Vorbedingung: Der Fall ist im System angelegt
Nachbedingung: Die Messwerte mit KI werden in einem Diagramm in vereinheitlichter Form dargestellt

Erfolgsszenario:

1. Benutzer startet das System, navigiert zu dem Fall, dessen KI-Diagramm angezeigt werden soll
2. System navigiert zur Übersicht des ausgewählten Falls
3. Benutzer klickt auf 'KI-Grafik anzeigen'
4. System leitet weiter zu einer Diagramm-Übersicht der berechneten Konfidenzintervalle

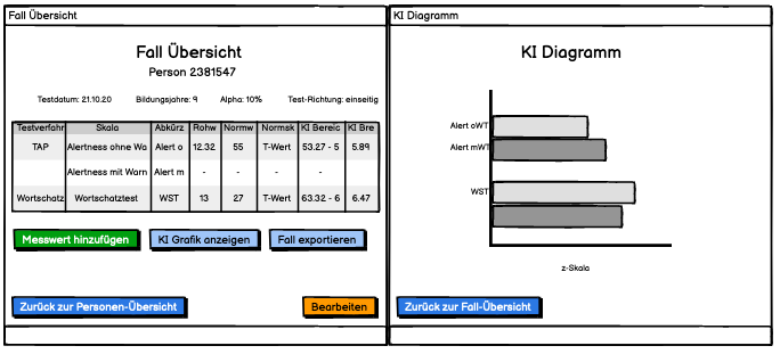


Abbildung 5.5: Wireframes Konfidenzintervalle darstellen

5.2.3 Anwendungsfall: Fälle vergleichen

Akteur: Benutzer
Ziel: Der Benutzer möchte die Konfidenzintervalle zwischen zwei Fällen vergleichen

Auslöser: Die Testwerte einer Person zu verschiedenen Zeitpunkten sollen miteinander verglichen werden
Vorbedingung: Die Person, sowie die zu vergleichenden Fälle sind im System angelegt
Nachbedingung: Die Konfidenzintervall-Grafiken der Fälle werden zum Vergleich nebeneinander dargestellt

Erfolgsszenario:

1. Benutzer startet das System, navigiert zur Person-Übersicht der getesteten Person
2. System navigiert zur Übersicht der ausgewählten Person
3. Benutzer klickt auf 'Fälle vergleichen'
4. System bittet den Benutzer die Fälle auszuwählen
5. Benutzer wählt die zu vergleichenden Fälle aus und klickt 'Vergleichen'
6. System leitet weiter zu einer KI-Diagramm-Übersicht der zu vergleichenden Fälle

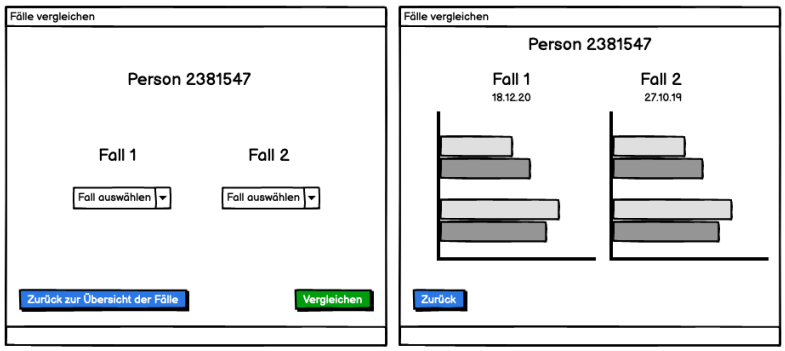


Abbildung 5.6: Wireframes Fälle vergleichen

5.2.4 Anwendungsfall: Fall exportieren

Akteur: Benutzer
Ziel: Der Benutzer möchte die Daten eines Falls in eine CSV-Datei exportieren

Auslöser: Die Daten sollen in ein anderes Programm eingespielt oder gesichert werden
Vorbedingung: Der zu exportierende Fall ist im System angelegt
Nachbedingung: Alle Daten zu dem Fall sind in einer CSV-Datei zugänglich und einsehbar

Erfolgsszenario:

1. Benutzer startet das System, navigiert zu dem zu exportierenden Fall
2. System navigiert zur Übersicht des ausgewählten Falls
3. Benutzer klickt auf 'Fall exportieren'
4. System bittet den Benutzer einen Export-Ordner auszuwählen
5. Benutzer wählt einen Ordner und klickt 'Öffnen'
6. System gib einen Hinweis, dass der Fall erfolgreich exportiert wurde und gibt den Ausgabe-Pfad und den Datei-Namen an
7. Benutzer klickt auf 'Fertig'

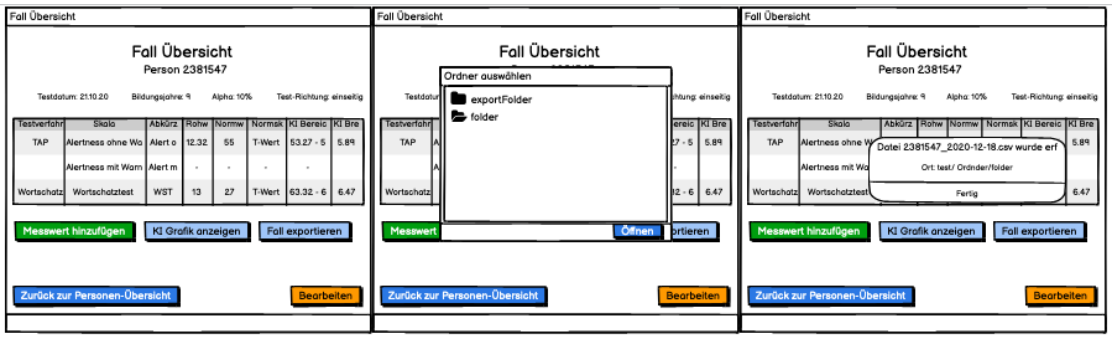


Abbildung 5.7: Wireframes Fall exportieren

5.2.5 Anwendungsfall: Testverfahren hinzufügen

Akteur: Benutzer
Ziel: Der Benutzer möchte die Ordner- und Auswertungsdatei-Struktur eines neuen Testverfahrens generieren lassen

Auslöser: Ein neues Testverfahren soll in das System eingepflegt werden
Vorbedingung: Für das Testverfahren ist noch keine Ordnerstruktur angelegt
Nachbedingung: Die Strukturen wurden erstellt und sind für den Benutzer verständlich und eindeutig identifizierbar, sodass die Auswertungsdateien befüllt werden können

Erfolgsszenario:

1. Benutzer startet das System
2. System navigiert zur Startseite
3. Benutzer klickt auf 'Testverfahren hinzufügen'
4. System bittet den Benutzer den Aufbau des Testverfahren anzugeben (Name, Anzahl Messwerte)
5. Benutzer füllt die Daten entsprechend aus und klickt auf 'Weiter'
6. System bittet den Benutzer nacheinander die enthaltenen Messwerte zu definieren (Name, Benötigte Variablen und deren Ausprägungen, sowie resultierenden Normwert-Typ)
7. Benutzer definiert die Messwerte und klickt jeweils auf 'Weiter'
8. System navigiert zur Testverfahren-Übersicht
9. Benutzer überprüft die Eingaben und klickt auf 'Ordnerstruktur anlegen'
10. System fragt nach, ob die Ordner wirklich angelegt werden sollen
11. Benutzer bestätigt das Anlegen
12. System gib einen Hinweis, dass Dateien und Ordner erfolgreich angelegt wurden
13. Benutzer klickt auf 'Fertig'

14. System navigiert zur Startseite

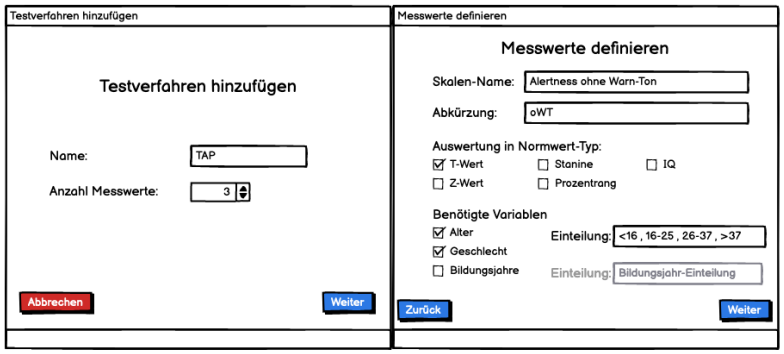


Abbildung 5.8: Wireframes Testverfahren hinzufügen I

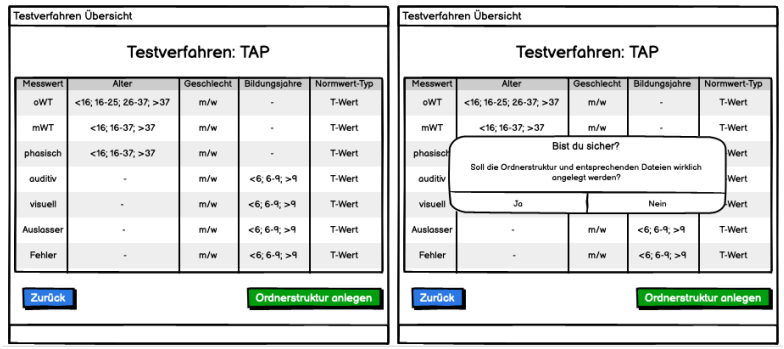


Abbildung 5.9: Wireframes Testverfahren hinzufügen II

6 Architektur

Im folgenden Kapitel wird das Design des Systems näher erläutert. Es wird auf Randbedingungen und getroffene Entwurfs-Entscheidungen eingegangen sowie die Architektur des Systems aus verschiedenen Perspektiven dargestellt.

6.1 Einflussfaktoren

Um bei der Architektur des Systems bewusst Entscheidungen treffen zu können, mussten zunächst beeinflussende oder einschränkende Faktoren identifiziert werden. Es konnten die folgenden organisatorische und politische Einflussfaktoren ermittelt werden.

6.1.1 Datenschutz

Das Krankenhaus muss den Datenschutz der Patienten garantieren, daher müssen die Patienten im System mit ihrer internen Patienten-Kennung referenziert werden.

6.1.2 IT-Sicherheit

Ein externes Unternehmen ist für die IT-Sicherheits-Infrastruktur zuständig. Während der Entwicklungsphase musste mit der IT-Abteilung des Krankenhauses kommuniziert und die Pläne zu der Anwendung offengelegt werden. Um die IT-Sicherheits-Infrastruktur im Krankenhaus nicht zu unterminieren oder zu gefährden, sollte die entwickelte Anwendung möglichst wenig Schnittstellen nach außen haben und nicht web-basiert aufgebaut sein.

6.2 Randbedingungen

6.2.1 Versionsverwaltung

Die Versionsverwaltung erfolgte mit GitLab. Es wurden Komponenten bzw. Features auf Feature-Branches entwickelt. Das hatte den Vorteil, dass diese Features in einer eigenen Geschwindigkeit entwickelt und Zwischenstände gespeichert werden konnten und nicht bei jedem *commit* ein lauffähiges Produkt entstanden sein musste. War die Feature-Entwicklung abgeschlossen, wurde ein Merge-Request erstellt und nur ein getesteter Code in den Master-Branch integriert. Der Master-Branch beinhaltete dadurch stets ein stabiles, geprüftes und lauffähiges Produkt.

6.2.2 Tools

Folgende Tools wurden zur Kommunikation mit den Mitarbeitern des Krankenhauses verwendet:

- E-Mail: Die Kommunikation und Absprache bei allen dringenden Fragen mit den Ansprechpartnern des Krankenhauses erfolgte über E-Mail-Schriftverkehr. Hierfür wurden die Uni- und Arbeitsadressen aller Beteiligten benutzt.
- Zoom: Aufgrund der Corona-Pandemie konnten persönliche Treffen für einen Großteil der Projektzeit nicht stattfinden. Daher fanden die wöchentlichen Meetings ausschließlich über Zoom statt.

6.2.3 Programmiersprachen

Die zu entwickelnde Anwendung in diesem Projekt wurde in Java geschrieben. Dabei wurde auf JDK 8 zurückgegriffen, da zum Entwicklungszeitpunkt die Java-Version *1.8.0_201* auf den Krankenhaus-PCs verfügbar war.
Für Java 8 ist JavaFX im JDK enthalten und musste nicht zusätzlich eingebunden werden. Um möglichst wenig Abhängigkeiten von anderen Technologien zu erzielen, wurde daher JavaFX für die Programmierung der Benutzeroberfläche ausgewählt.

6.2.4 Entwicklungsumgebung

Als Entwicklungsumgebung wurde IntelliJ verwendet, da diese Umgebung viel Unterstützung bei der Entwicklung bietet und die Entwicklungsgeschwindigkeit beschleunigt. Unabhängig von der Entwicklungsumgebung muss das Programm auf jeder Maschine unter Java 8 lauffähig sein.

6.2.5 Frameworks und Dependency Management

Die Anwendung wurde als *SpringBoot*-Applikation aufgesetzt und *Gradle* wurde für das Dependency-Management genutzt. Dies hat den Vorteil, dass bei der Programmierung viel Arbeit abgenommen und die Produktentwicklung beschleunigt werden konnte.
Um mit Microsoft-Dateien arbeiten zu können, wurde das Framework Apache POI genutzt. Diese Java API ermöglicht es, mit Java sowohl aus Excel-Dateien zu lesen als auch in diese schreiben zu können.
Des Weiteren wurde Lombok eingesetzt, um den Code übersichtlich zu halten.

6.3 Lösungsstrategie

Um die Neuropsychologen bestmöglich bei ihrer Arbeit zu unterstützen, sollte ein Programm entwickelt werden, dass den Schritt der Auswertung und Interpretation der Ergebnisse digital unterstützt. Das Programm sollte folgende wichtige Funktionalitäten abdecken:

- Überführen der Rohdaten in Normdaten anhand der Patientendaten und Auswertungstabellen der Manuale
- Berechnung und Darstellung der Konfidenzintervalle (bei Vorhandensein relevanter Variablen)
- Exportieren der Daten (Rohwerte, Normwerte, Konfidenzintervalle) in entsprechenden Formaten (Tabelle bzw. Diagramm), um diese in einen Bericht einfügen zu können
- Automatisches Generieren einer Ordnerstruktur und entsprechenden Dateien für Auswertungstabellen beim Hinzufügen neuer Testverfahren

Die Benutzer sollen eigenständig Testverfahren und deren hinterlegte Tabellen zur Normwertermittlung hinzufügen und überarbeiten können, ohne den Quellcode des Programms wissen oder ändern zu müssen.

6.4 Entwurfsentscheidungen

Im folgenden Abschnitt werden wichtige und übergreifende Architekturentscheidungen angegeben, die im Laufe der Entwicklung getroffen wurden, sowie deren Begründungen.

6.4.1 Hybrid aus interner Datenbank & zur Laufzeit Infomationen aus Dateien einlesen

Die ursprüngliche Idee bestand darin, auf eine Datenbank zu verzichten und stattdessen alle nötigen Informationen vom Programm bei Start einlesen zu lassen. Grund für diese Überlegung war, dass die Benutzer zu jedem Zeitpunkt vollen Einfluss auf das Programm und die hinterlegten Informationen haben sollten.
Um dies umzusetzen, sollten alle Personen- und Falldaten in Excel-Dateien gespeichert werden, die für den Benutzer leicht verständliche sind. Diese Excel-Dateien wiederum sollten in einer Ordnerstruktur angelegt werden, anhand derer das Programm gezielt zu einer bestimmten Datei über die Ordnernamen navigieren kann.
Der Vorteil an dieser Umsetzung bestand darin, dass alle Informationen für den Benutzer verständlich sind und Einfluss genommen werden kann, ohne Zugang zu dem Quell-Code oder sonstigen technischen Programmen haben zu müssen.
Diese Variante hatte jedoch auch einige Nachteile. Die Performance des Programms wäre beim Starten durch das Einlesen vieler Daten vergleichsweise schlecht und Änderungen an Personen- bzw. Falldaten würde zu vielen Umbenennungen, Dateiänderungen oder Dateiverschiebungen führen. Zudem ist die Fehleranfälligkeit des Systems höher, da der Benutzer bestimmte Namenskonventionen und Regeln einhalten muss.

Aus diesem Grund wurde sich für eine Kombination zwischen einer internen Datenbank und dem Einlesen von Informationen zur Laufzeit entschieden.
Die interne Datenbank wird für das persistente Abspeichern von Personen- und Falldaten verwendet, da diese Daten als komplex, aber relativ stabil gelten. Die Informationen bezüglich der zur Verfügung stehenden Testverfahren, sowie die Auswertungstabellen

zur Normierung lassen sich hingegen gut in einer wie oben beschriebenen Ordnerstruktur unterbringen und können so leicht von den Benutzern verändert werden.

Da in der Normierung hauptsächlich in maximal drei Variablen (Alter, Geschlecht und Bildungsjahre) unterschieden wird, kann das Programm die Variablen aus den Ordnernamen entnehmen, um durch die Struktur zur richtigen Datei zu navigieren. Solch eine Datei enthält jeweils nur eine Tabelle mit den Transformationswerten der Normierung und trägt einen eindeutig identifizierbaren Namen, dem die Variablenausprägungen und der entsprechende Normwert entnommen werden kann. Dadurch ist der Inhalt der Datei für den Benutzer nachvollziehbar, auch bei versehentlicher Verschiebung der Datei.
Die Informationen zu den Testverfahren sollten zudem ursprünglich in den jeweiligen Ordner des Testverfahrens untergebracht sein, doch dadurch würden zusätzliche Dateien mit eventuell nur einer Zeile Inhalt entstehen. Daher wurde entschieden, die Informationen zu den Testverfahren in einer Übersichtstabelle festzuhalten. Diese Datei hat klar definierte Spalten, in denen gewisse Informationen zu den Testverfahren gelistet und vom Programm eingelesen werden können.

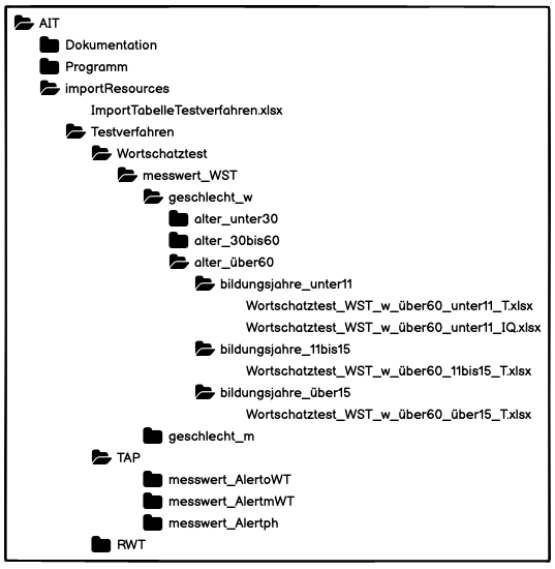


Abbildung 6.1: Beispiel Ordnerstruktur zum Einlesen von Daten

Durch diese Kombination ist die Hauptfunktionalität des Programms, das Berechnen und Darstellen der Konfidenzintervalle vergleichsweise stabil und kann auch bei Problemen beim Einlesen durch manuelles Hinzufügen von Messwert-Informationen erfolgen.
Obwohl die Normierung von Rohwerten auch eine zentrale Funktion der Anwendung ist,

macht es wenig Sinn, die Auswertungstabellen persistent zu speichern. Es entstehen dadurch zwar viele zusätzliche Dateien und Ordner, die jedoch mit der richtigen Benennung den Benutzern viel Transparenz, Nachvollziehbarkeit und Einfluss bieten kann, ohne das System von innen kennen und verstehen zu müssen.

Risiken

Die Fehleranfälligkeit sollte hier erwähnt werden, da es ein großer und relativ monotoner Aufwand ist, die Auswertungstabellen zu digitalisieren. Sollten Fehler in dem Überführungsprozess entstehen, können diese vom Programm nicht entdeckt werden und die Normierung wird fehlerhaft. Die Konsequenz ist, dass die Neuropsychologen mit fehlerhaften Werten weiterarbeiten und dies im schlimmsten Fall zu einer fehlerhaften Einschätzung führen kann. Daher werden die Benutzer gebeten, diese Werte mit großer Sorgfalt zu bearbeiten und durch eine oder zwei Personen überprüfen lassen. Bei Problemen müssen die Normwerte gegebenenfalls manuell nachgetragen werden.

6.4.2 3-Schichtenarchitektur

Um sich an den *SoC-* und *Geheimnis-Prinzipien* zu orientieren und eine klare Trennung der Verantwortlichkeiten bzw. Kapselungen interner Funktionsweisen zu unterstützen, wurde sich für eine Implementierung einer 3-Schichtenarchitektur entschieden. Zudem handelt es sich, wie im theoretischen Hintergrund dargelegt wurde, um ein gängiges Modell. Da es in der Praxis viele Varianten bezüglich der Aufteilung der Anwendungsschicht gibt, zeigt die folgende Grafik die angestrebte Aufteilung in der hier entwickelten Anwendung.

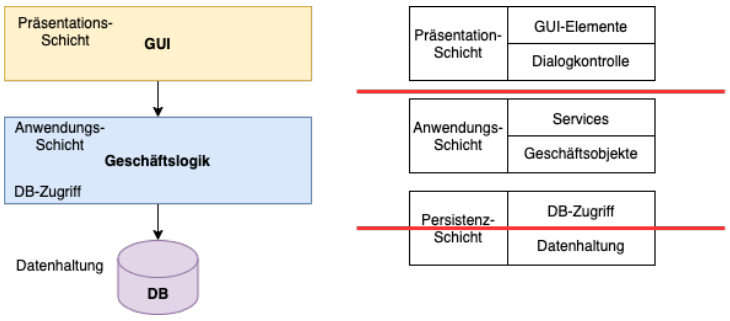


Abbildung 6.2: Schichtenarchitektur (erstellt mit draw.io) angelehnt an [13, S. 29]

Der Datenbank-Zugriff, welcher eigentlich der Persistenzschicht zuzuordnen ist, wird hier mit in die Geschäftslogik-Komponente aufgenommen.

6.4.3 Programm als ausführbare JAR-Datei ausliefern

Aufgrund der hohen Sensibilität bezüglich der Daten sowie der bestehenden IT-Sicherheits-Infrastruktur wurde angestrebt, das Programm ohne Web-Schnittstelle und mit einer lokalen Datenbank zu implementieren. Außerdem sollten Abhängigkeiten zu anderen externen Systemen möglichst reduziert werden.
Es wurde sich mit der IT-Abteilung des Krankenhauses in Verbindung gesetzt und dieser eine *Hello-World*-Test-Applikation als ausführbare JAR-Datei zugesendet. Die IT-Abteilung hat dann anhand dieser Test-Applikationen überprüft, ob der Start der Datei aus der Konsole allgemein möglich ist oder in irgendeiner Weise eingeschränkt wird oder Sicherheitswarnungen auslöst.
Obwohl die Benutzerfreundlichkeit durch den Start aus der Konsole deutlich reduziert ist, sollte sichergestellt werden, dass die Neuropsychologen am Ende ein Produkt haben, welches sie ohne Einschränkungen der Funktionalität verwenden können. Daher wurde nach positiver Rückmeldung der IT-Abteilung bezüglich der Ausführbarkeit die Entscheidung gefasst, das endgültige Programm in dieser Form auszuliefern.

6.5 Datenbank

Um ein Datenbank-Schema zu entwickeln, wurden zunächst die relevanten Objekt-Typen und Attribute aus dem fachlichen Datenmodell zusammengetragen, welche persistent gespeichert werden sollen.

Objekttyp	Attribute
Person	Personen-ID, Initialien, Geburtsdatum, Geschlecht
Fall	Testdatum, Bildungsjahre der Testperson, Alpha, Testrichtung
Messwert	Skalen-Abkürzung, Testverfahren, Skalen-Name, Bereich, Rohwert, Normwert, Normskala
Reliabilität	ID, interne Reliabilität, interne Reliabilitäts-Schätzmethode, retest Reliabilität, retest Reliabilitäts-Schätzmethode, Quelle
Konfidenzintervall	ID, KI-Untergrenze, KI-Obergrenze, KI-Breite, benutzte Formel, resultierender Ausdruck

Tabelle 6.1: Datenbank Objekttypen und Attribute

6.5.1 Entity-Relationship Modell

Diese Objekte wurden anhand des fachlichen Datenmodells in ein Entity-Relationship-Modell (ER-Modell) überführt. Die Beziehungen zwischen den Entitäten wurden hierbei mit Hilfe von Kardinalitäten in der Min-Max-Notation angegeben.

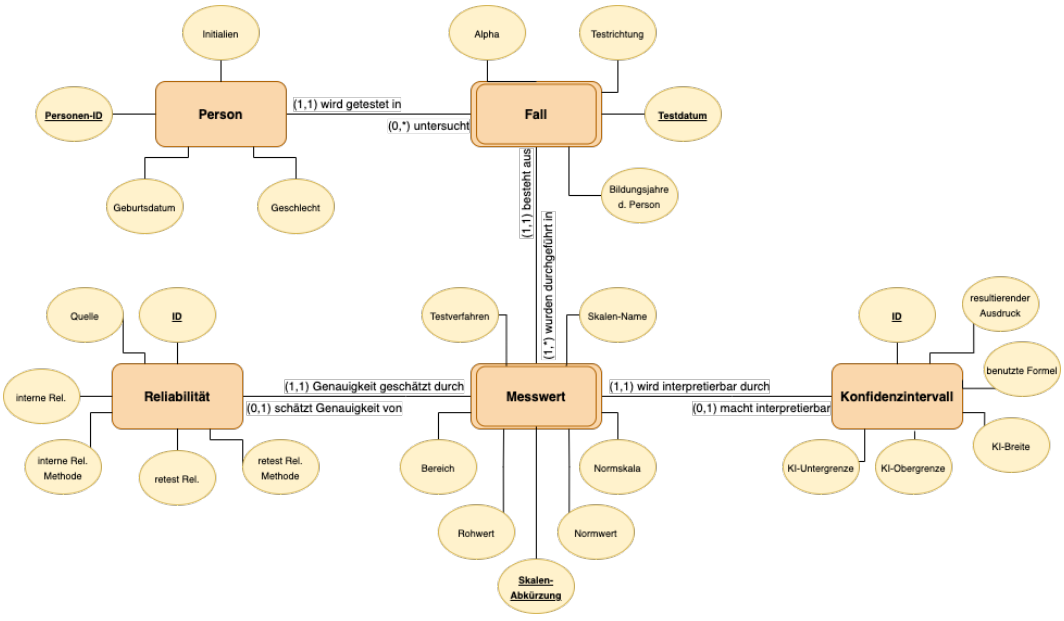


Abbildung 6.3: ER-Modell der Datenbank-Objekte

Was in dem Schema auffällt ist, dass es zwei zentrale, schwache Entity-Typen gibt, *Fall* und *Messwert*, die über ihre Attribute alleine nicht eindeutig identifizierbar sind. Diese Typen benötigen daher zusätzlich einen Schlüssel aus deren Beziehung zu anderen Typen.

6.5.2 Relationstypen

Um ein relationales Schema für die Daten zu entwickeln, wurde das ER-Schema in folgende Relationstypen überführt:

- **PERSON** {Personen-ID, Initialien, Geburtsdatum, Geschlecht},

  Eine **Person** ist eindeutig über dessen Attribut *Personen-ID* identifizierbar und dient daher als Primärschlüssel des Objekts.

- **FALL** {Personen-ID, Testdatum, Bildungsjahre, Alpha, Test-Richtung},

  Das **Fall**-Objekt ist ein schwacher Entity-Typ, daher reicht das Attribut *Testdatum* nicht aus, um eindeutig identifiziert zu werden. Da eine Person aber nur einmal am Tag getestet werden kann, muss der Primärschlüssel *{Personen-ID}* des starken Entity-Typs Person dem Schlüssel hinzugefügt werden. In der Kombination *{Personen-ID, Testdatum}* ist ein Fall eindeutig zu identifizieren und gilt daher als Primärschlüssels des Fall-Objekts.

- **MESSWERT** {Personen-ID, Testdatum, Skalen-Abkürzung, Testverfahren, Skalen-Name, Bereich, Rohwert, Normwert, Normskala},

  Das **Messwert**-Objekt ist ebenfalls ein schwacher Entity-Typ. Das Attribut *Skalen-Abkürzung* reicht alleine als Schlüssel nicht aus, um Eindeutigkeit zu erreichen. Ein Messwert kann in einem Fall jedoch nur einmal durchgeführt werden. Daher muss der Primärschlüssel *{Personen-ID, Testdatum}* des Entity-Typs Fall dem Schlüssel hinzugefügt werden. In der Kombination *{Personen-ID, Testdatum, Skalen-Abkürzung}* ist ein Messwert eindeutig zu identifizieren und gilt daher als Primärschlüssels des Messwert-Objekts.

- **RELIABILITÄT** {Rel-ID, interne Reliabilität, interne Rel.-Methode, retest Reliabilität, retest Rel.-Methode, Quelle, Personen-ID, Testdatum, Skalen-Abkürzung}, mit Fremdschlüssel {Personen-ID, Testdatum, Skalen-Abkürzung}
  bzgl. **MESSWERT**

Eine **Reliabilität** hat kein Attribut, was es eindeutig identifizierbar macht. Daher wurde die künstliche *Rel-ID* eingeführt und dient als Primärschlüssel des Objekts. Da sich eine Reliabilität auf einen eindeutigen Messwert bezieht, muss diese Beziehung zusätzlich modelliert werden. Da ein Messwert keine Reliabilität besitzen muss, eine Reliabilität aber immer zu einem Messwert gehört, wird der Primärschlüssel *{Personen-ID, Testdatum, Skalen-Abkürzung}* des Messwerts als Fremdschlüssel zu dem Reliabilität-Objekt hinzugefügt. Dadurch können undefinierte Werte in Fremdschlüsselattributen vermieden werden.

- **KONFIDENZINTERVALL** {KI-ID, KI-Untergrenze, KI-Obergrenze, KI-Breite, verwendete Formel, resultierender Ausdruck, Personen-ID, Testdatum, Skalen-Abkürzung},
  mit Fremdschlüssel {Personen-ID, Testdatum, Skalen-Abkürzung}
  bzgl. **MESSWERT**

  Das Konfidenzintervall wird analog zur Reliabilität umgesetzt. Daher wird der Primärschlüssel *{Personen-ID, Testdatum, Skalen-Abkürzung}* des Messwerts auch hier als Fremdschlüssel zu dem Konfidenzintervall-Objekt hinzugefügt.

6.5.3 Wahl der Datenbank

Da die Datenbank nicht im Fokus dieses Projektes lag, wurde die Open-Source *H2 Database Engine* ausgewählt. Ein überzeugender Faktor war, dass diese ebenfalls in Java geschrieben ist und mit dieser Datenbank bereits Erfahrungen gemacht wurden. Da die Daten persistent gespeichert und beim Neustart des Systems erhalten bleiben sollen, wurde sich für eine *Embedded (Lokale) Database* entschieden. Die persistenten Daten werden hier in einer Datei lokal im Format *databaseName.mv.db* abgespeichert (Vgl. [2]).

6.6 Sichten

Die Sichten zeigen das System aus verschiedenen Perspektiven. Je nach Sicht sind hierbei unterschiedliche Details und Aspekte im Fokus, die abgebildet bzw. verdeutlicht werden sollen. Als unbedeutend erachtete Details, werden in einer Sicht nur abstrahiert dargestellt (Vgl. [20, Kap. 5.4.1]). Im nächsten Abschnitt werden folgende Sichten abgebildet: Kontextabgrenzung, Bausteinsichten und Laufzeitsichten.

6.6.1 Kontextabgrenzung

Bei der Kontextabgrenzung wird das zu entwickelnde System zunächst als Blackbox dargestellt. Im Fokus liegt die Umgebung, in der das zu entwickelnde System eingebettet ist. Dies beinhaltet welche Schnittstellen es zu externen Systemen (Nachbarsysteme) gibt und welche Benutzergruppen mit dem System interagieren werden (Vgl. [20]).

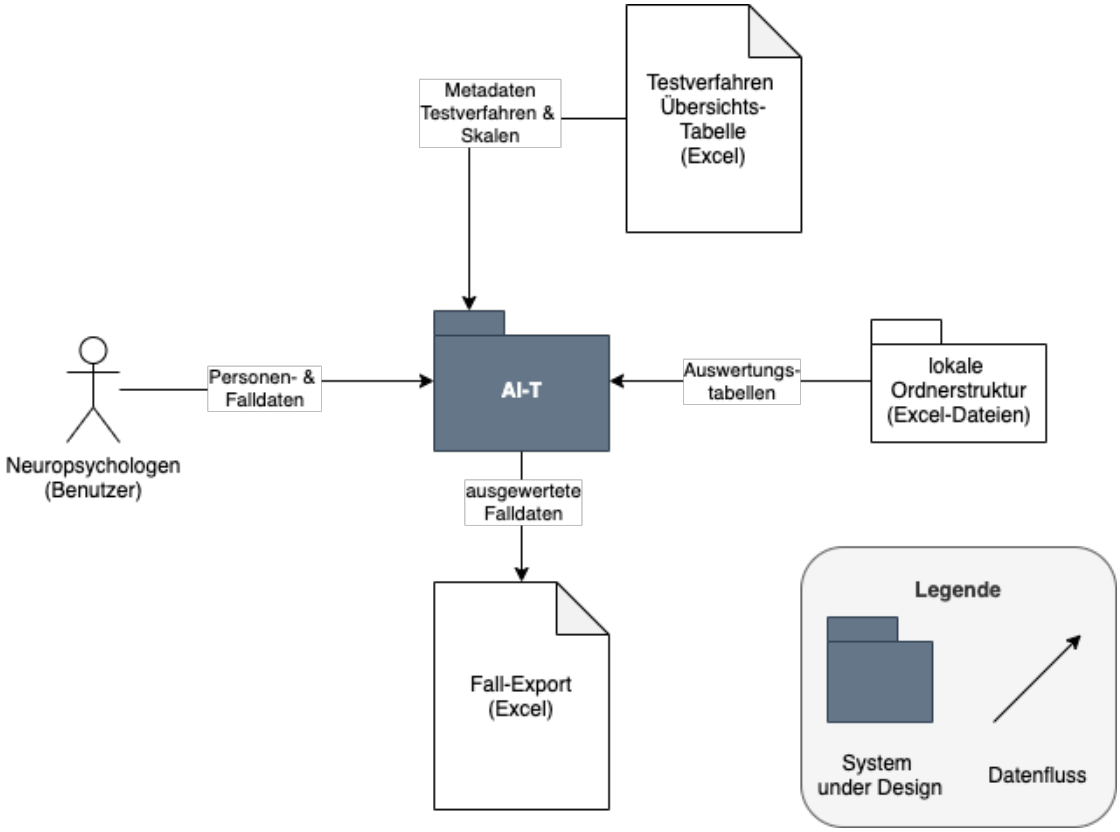


Abbildung 6.4: Kontextabgrenzung (erstellt mit draw.io)

Schnittstelle	Beschreibung	Technologie
Personen- & Falldaten	Stabile Personenmerkmale, die Rohdaten ausgeführter Messwerte, sowie die für alle Messwerte geltenden Falldaten	Eingabe über UI
Metadaten der Testverfahren & Skalen	Tabelle mit Daten zu Testverfahren & Skalen z.B. Name, Normskalen und Reliabilitäten	Aus Excel-Datei mit vorgegebener Struktur eingelesen
Auswertungstabellen	Tabellen zur Transformation von Roh- in Normwerte	In Ordnerstruktur navigieren, um korrekte Excel-Datei mit vorgegebener Struktur einzulesen
Ausgewertete Falldaten	Beinhaltet durchgeführte Messwerte eines Falls, sowie deren Normwerte und berechneten Konfidenzintervallen	In eine Excel-Datei exportiert

Tabelle 6.2: Schnittstellen Kontextabgrenzung

6.6.2 Bausteinsicht

In der Bausteinsicht wird die statische Zerlegung des Systems vorgestellt und zeigt dessen internen Aufbau. Das System wird Top-Down von der Kontextabgrenzung schrittweise in Komponenten, Pakete und Klassen heruntergebrochen und gibt deren Zusammenspiel und resultierende Abhängigkeiten wieder (Vgl. [20]).

Unter Berücksichtigung der 3-Schichtenarchitektur und dem Facade-Pattern wurde das System in vier Hauptkomponenten unterteilt: *UI, Facade, Business-Logic und Persistenz.* Unter Anwendung des Interface-Pattern mit dem Ziel der Entkopplung von Klassen, erfolgt die Kommunikation zwischen diesen Komponenten ausschließlich über Interfaces.

Gesamtsystem

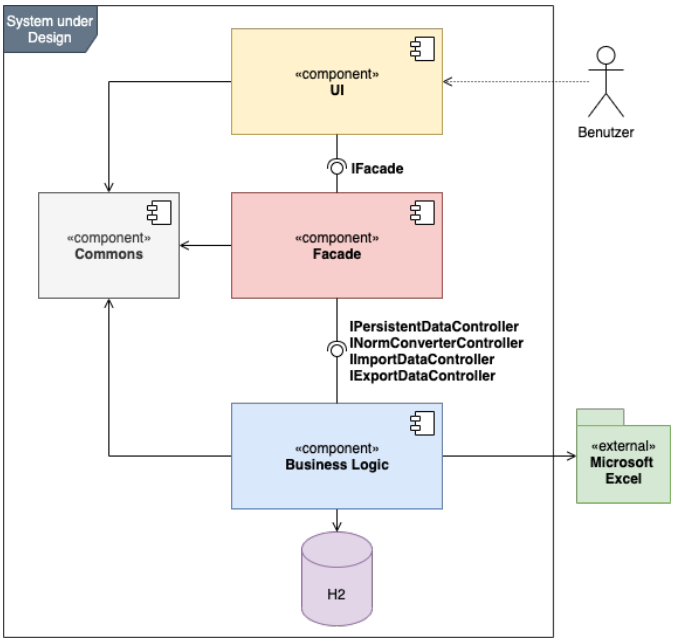


Abbildung 6.5: Whitebox Gesamtsystem

Die **UI**-Komponente ist zuständig für die Darstellung und Interaktion mit dem Benutzer. Entsprechend des *Facade*-Patterns, bietet die **Facade**-Komponente ein übergeordnetes Interface für die Geschäftlogik des Systems. Dadurch wird der UI-Komponente eine zentrale Schnittstelle geboten, ohne dass die Komponente selbst Wissen über den Aufbau der Geschäftslogik benötigt und unterstützt zudem das Geheimnis-Prinzip im System. Ein Nachteile dieser Umsetzung ist allerdings die Entstehung eines *Single-Point-Of-Failure*. Sollte ein unvorhergesehen Fehler in der Facade-Komponente auftreten, kann dies die gesamte Verfügbarkeit des Systems beeinträchtigen.
In der **Business Logik**-Komponente wird die Fachlogik des Systems umgesetzt. Um mit persistenten Daten zu arbeiten, erfolgt zudem der Datenbank-Zugriff in dieser Komponente.

Die **Commons**-Komponente beinhaltet übergreifende Konzepte, wie Konfigurations- und Transportklassen (*DTO*), aber auch Klassen fachlicher Datentypen (*Domain Data Types*), sowie statische Hilfsklassen (*Utils*). Die Hauptkomponenten können auf alle Bausteine dieser Komponente zugreifen.

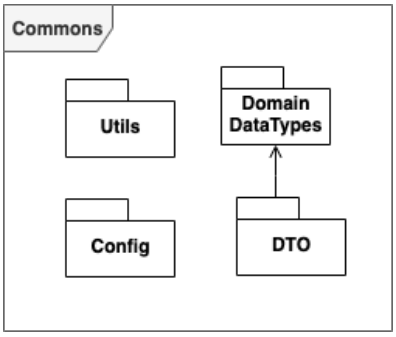


Abbildung 6.6: Whitebox Commons in Paket-Sicht

Die **statischen Hilfsklassen** beinhalten Funktionen, die an mehreren Stellen im Code benötigt werden. Sie helfen einen redundanten Code zu reduzieren und enthalten selbst keine explizite Fachlogik.

Baustein	**Verantwortlichkeit**
TypeConversionUtil	Einheitliches Konvertieren zwischen einfachen Daten-Typen (z.B. LocalDate zu String und umgekehrt)
FileUtil	Unterstützende Methoden zum Finden, Einlesen und Schreiben von (Excel-)Dateien

Tabelle 6.3: Bausteine Commons.Utils

Die **Konfigurationsklassen** (Config) sind der Übersicht halber ebenfalls in der Commons-Komponente angelegt. Sie beinhalten Konfigurations-Variablen, die beim Start aus vordefinierten Dateien eingelesen werden.

Baustein	**Verantwortlichkeit**
ImportConfig	Variablen für den Import z.B. Import-Ordner oder Namen benötigter Import-Dateien
WindowConfig	Variablen zur Größe des Anwendungsfensters
DefaultValueConfig	Default-Variablen fachlicher Enums (z.B. vorausgewähltes Alpha und Testrichtung eines Falls)

Tabelle 6.4: Bausteine Commons.Config

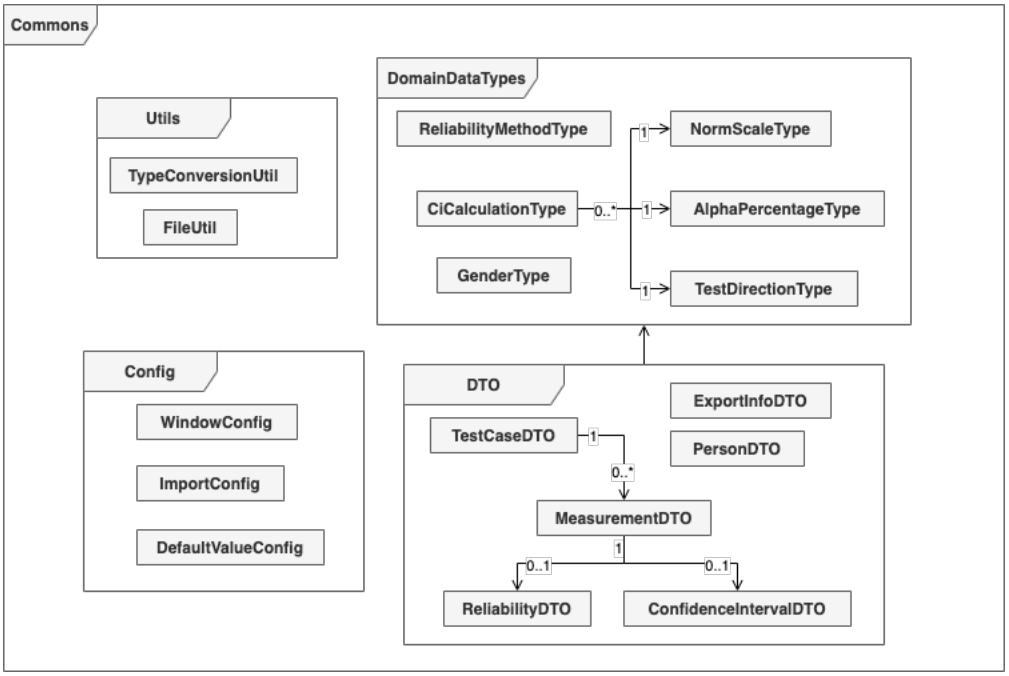


Abbildung 6.7: Whitebox Commons

Die **Transportklassen** (DTO = *Data Transfer Objects*) dienen dem gesammelten Weiterreichen von Informationen über Entität oder Zustände, ohne Referenzen auf diese oder deren Klasse weiterzugeben. Dadurch können Sichtbarkeiten und Abhängigkeiten zwischen Komponenten reduziert werden. Zudem können mit Hilfe der DTO-Objekte Informationen komprimiert zwischen Komponenten transportiert werden, wodurch der Kommunikationsaufwand reduziert werden kann.

Baustein	**Transport von Informationen bezüglich**
ExportInfoDTO	Export einer Datei z.B. Export-Pfad und Name der exportierten Datei
PersonDTO	Person-Entität
TestCaseDTO	Fall-Entität
MeasurementDTO	Messwert-Entität
ReliabilityDTO	Reliabilität-Entität
ConfidenceIntervalDTO	Konfidenzintervall-Entität

Tabelle 6.5: Bausteine Commons.DTO

Die **fachlichen Datentypen** (*DomainDataTypes*) bestehen aus vordefinierten Enum-Klassen. Sie beinhalten die möglichen Ausprägungen, die solch ein Objekt annehmen kann, sowie das dazugehörige fachliche Regelwerk.

Baustein	**Repräsentiert fachlichen Datentyp**
GenderType	GeschlechtTyp
AlphaPercentageType	AlphaProzentTyp
TestDirectionType	TestRichtungsTyp
NormScaleType	NormSkalenTyp
ReliabilityMethodType	ReliabilitätsMethodenTyp
CiCalculationType	KIBerechnungsTyp

Tabelle 6.6: Bausteine Commons.DomainDataTypes

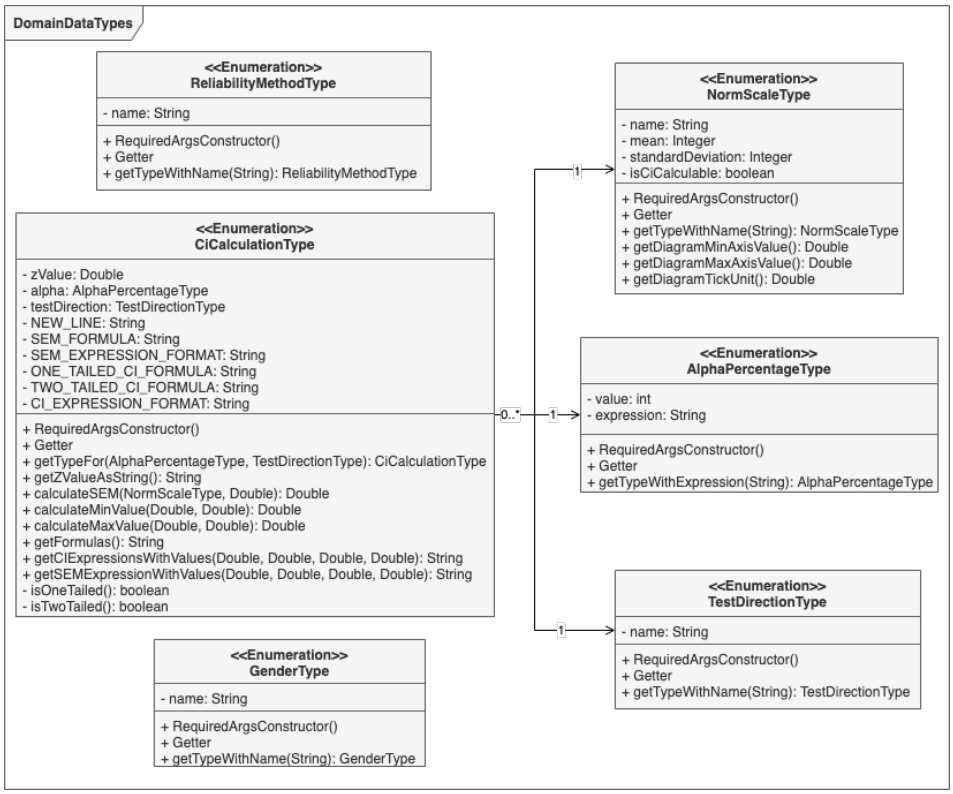


Abbildung 6.8: Whitebox Commons.DomainDataTypes

Anmerkungen:
Die fachlichen Datentypen sind insgesamt voneinander unabhängig, lediglich der *CiCalculationType* ist von den drei Typen *NormScaleType, AlphaPercentageType* und *TestDirectionType* abhängig, da für die Auswahl der korrekten Berechnungs-Formel eines Konfidenzintervalls eine konkrete Instanz jeder dieser Typen benötigt wird.

Der BereichsTyp wurde zunächst implementiert, aber auf Grund der hinzugekommenen Anforderung den Wert zur Laufzeit frei wählen zu können, durch einen String-Wert ersetzt.

UI Komponente

Zweck:
Darstellung und Benutzerinteraktion

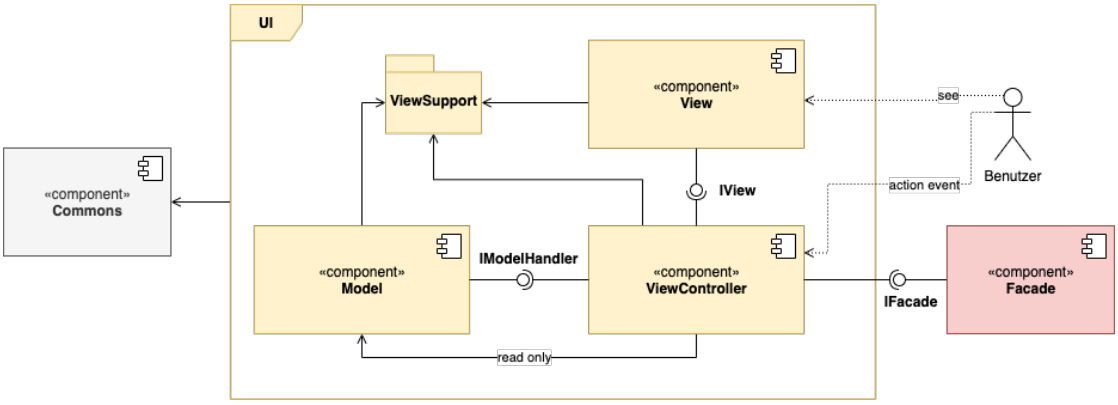


Abbildung 6.9: Whitebox UI

Die UI-Komponente wurde, angelehnt an das *Passive-View*-Pattern, in drei Subkomponenten unterteilt, welche über die jeweils angebotenen Schnittstellen untereinander kommunizieren können. Dadurch kann die mögliche Kommunikation kontrolliert bzw. bei Bedarf eingeschränkt werden. Zudem werden Hilfsklassen, die bei der Darstellung unterstützen, in einem Paket gesammelt.

Baustein	Verantwortlichkeit
ViewController	Darstellungslogik (was und wie), Reaktion auf Benutzerakt-ionen, Kommunikation mit dem Backend
View	Darstellung des Anwendungsfensters (passiv)
Model	Halten und Vereinheitlichen darzustellender Informationen
ViewSupport	Unterstützung bei der Darstellung

Tabelle 6.7: Bausteine UI

UI.ViewSupport Paket

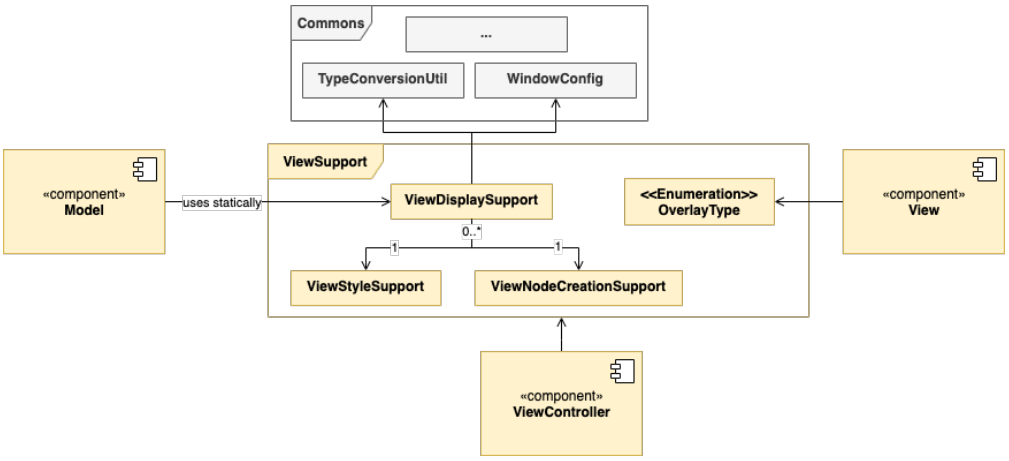


Abbildung 6.10: Whitebox UI.ViewSupport

Anmerkungen:
Aufgrund der zentralen Funktion des ViewControllers, hat dieser Zugriff auf alle Klassen dieses Pakets. Die anderen beiden Subkomponenten hingegen benötigen nur ausgewählte Objekte und Funktionen.
Beim Start der Anwendung wird eine Instanz der ViewDisplaySupport-Klasse erzeugt. Die Abstände, Positionen oder Schriftgröße der Darstellungs-Objekte werden anhand lokaler Variablen dieser erstellten Instanz in Abhängigkeit zur initialisierten Fenstergröße berechnet. Die Model-Subkomponente hat keine Referenz auf die lokale Instanz des ViewDisplaySupports, da diese für sie nicht relevant sind. Somit ist der Zugang des Models auf die statischen Inhalte der Klasse bezüglich des Umgangs mit fehlenden Werten begrenzt.

Baustein	Verantwortlichkeit
OverlayType	Enum-Sammlung darstellbarer Overlays
ViewDisplaySupport	Vereinfachte Darstellung von Werten, einheitlicher Umgang mit fehlenden Werten, Informationen zur Größe des Anwendungsfensters
ViewNodeCreationSupport	Unterstützung beim Erstellen von JavaFX-Objekten (z.B. Alerts, Buttons, ...)
ViewStyleSupport	Unterstützung bei stilistischen Anpassungen der JavaFX-Objekte (z.B. Schriftgroße, Farbe, ...)

Tabelle 6.8: Bausteine UI.ViewSupport

UI.View Subkomponente

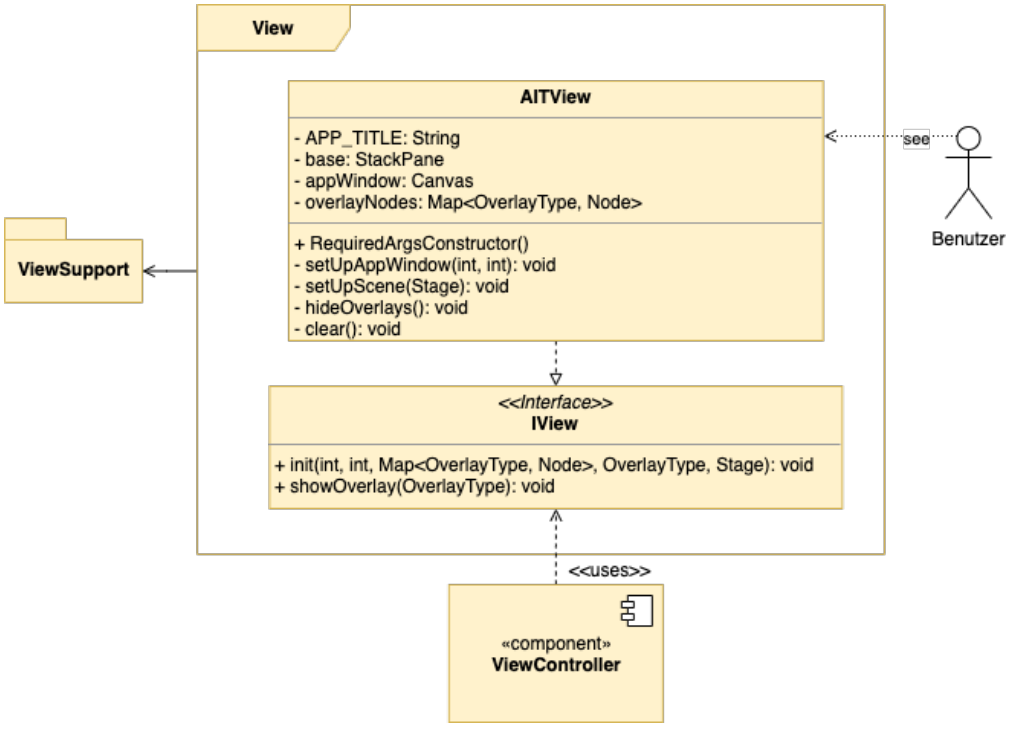


Abbildung 6.11: Whitebox UI.View

Baustein	Verantwortlichkeit
AITView	Darstellung des Anwendungsfensters und der Overlays als Node-Objekte
IView	Angebotene Schnittstelle nach außen, zur Initialisierung der View und um Overlay-Wechsel zu initiieren

Tabelle 6.9: Bausteine UI.View

Anmerkungen:
Da die View in dieser Architektur passiv sein soll, ist die nicht für die Aktualisierung der Inhalte des Overlays zuständig, sondern lediglich der Darstellung.

UI.ViewModel Subkomponente

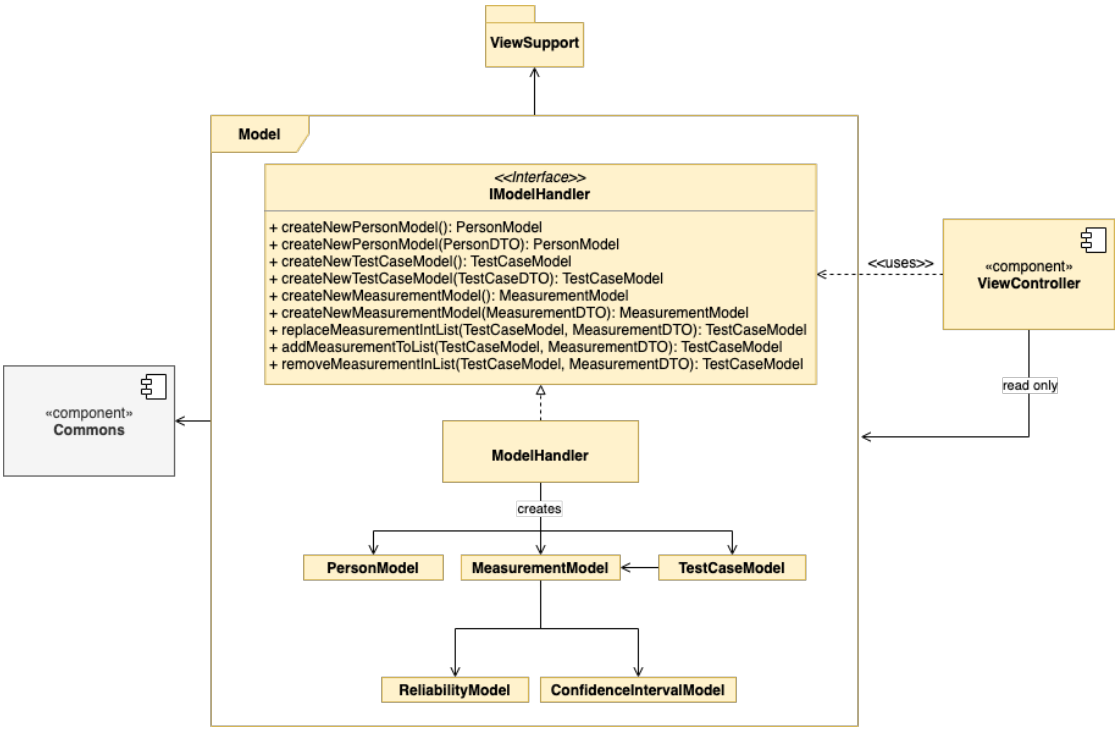


Abbildung 6.12: Whitebox UI.Model

Die Model-Bausteine sind für das Umwandeln und Vereinheitlichen relevanter Informationen aus Transport-Objekten in ein für die View darstellbares Format verantwortlich. Jede Klasse ist hierbei für eine Entität zuständig. Fehlende Werte werden bei Bedarf ersetzt oder in ein darstellbares Format konvertiert.

Baustein	**Verantwortlichkeit**
ModelHandler	Erstellen neuer Model-Objekte
IModelHandler	Angebotene Schnittstelle nach außen, um die Erzeugung oder Änderungen von Model-Objekten zu initiieren
PersonModel	UI-Darstellung von Infos zur Person
TestCaseModel	UI-Darstellung von Infos zum Fall
MeasurementModel	UI-Darstellung von Infos zum Messwert
ReliabilityModel	UI-Darstellung von Infos zur Reliabilität
ConfidenceIntervalModel	UI-Darstellung von Infos zum Konfidenzintervall

Tabelle 6.10: Bausteine UI.Model

Anmerkungen:
Abgesehen von Common-Objekten bestehen keine weiteren Abhängigkeiten zu anderen Komponenten des Systems. Die Model-Objekte haben keine öffentlichen Konstruktoren, alle ihre lokalen Variablen sind mit dem *final*-Bezeichner versehen und es wurden keine Setter-Methoden implementiert. Dadurch muss bei einer Änderung jeweils eine neue Instanz erstellt werden. Der ViewController kann dieses Objekt entsprechend nur lesend verwenden. Um die Klassen dieser Komponente nutzen zu können, wird eine klar definierte Schnittstelle nach außen angeboten, welche Model-Instanzen zurückgibt.

UI.ViewController Subkomponente

Da die ViewController-Komponente die zentrale UI-Subkomponente mit den meisten Zuständigkeiten repräsentiert, wurde diese wiederum in kleinere Teile mit klarer Rollenverteilung aufgeteilt. Die Kommunikation zwischen den Bausteinen erfolgt unidirektional über das *IViewDataController*-Interface.

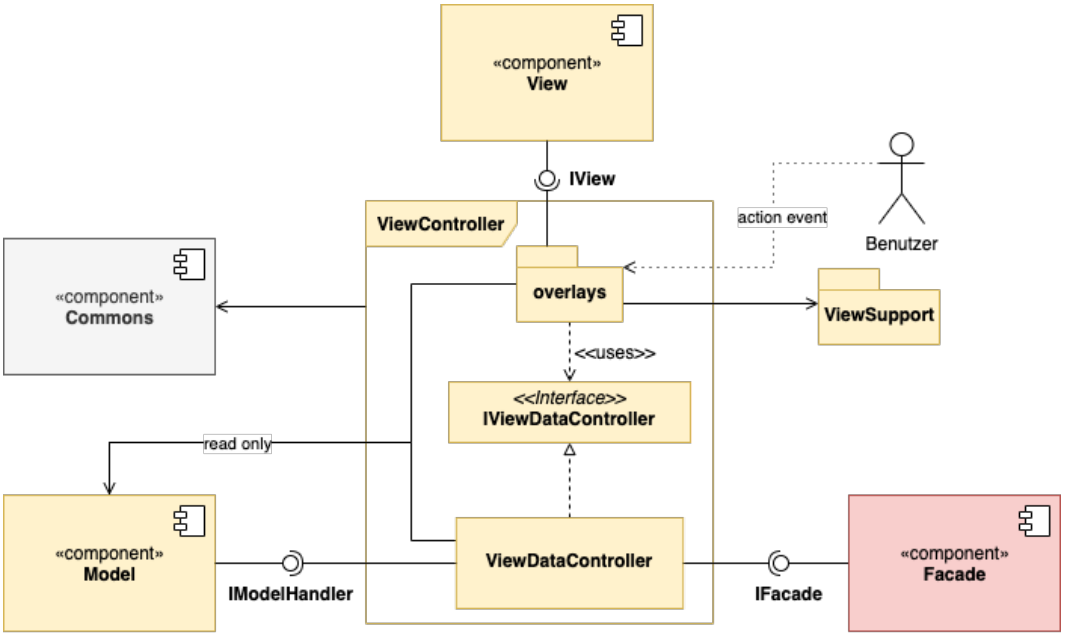


Abbildung 6.13: Whitebox UI.ViewController

Baustein	**Verantwortlichkeit**
Overlays-Paket	Starten des Frontends, Darstellungslogik basierend auf der Model-Komponente (inkl. Umgang mit Benutzeraktionen)
ViewDataController	Aktualisierung der Darstellungsinformationen, Kommunikation mit dem Backend
IViewDataController	Intern angebotene Schnittstelle zum Beantragen von Darstellungsinformationen und deren Änderungen

Tabelle 6.11: Bausteine UI.ViewController

Schnittstelle	**Beschreibung**
IFacade	Kommunikation mit dem Backend
IModelHandler	Aktualisierung der Darstellungsinformationen

Tabelle 6.12: Benötigte Schnittstellen UI.ViewDataController

Als zentrale Subkomponente in der UI greift die *ViewController*-Komponente auf die angebotenen Schnittstellen anderer Komponenten zu, um dessen Service in Anspruch zu nehmen. Sie selbst bietet nach außen keine Schnittstellen an. Die *ViewDataController*-Klasse übernimmt hierbei die Verantwortung bezüglich der darzustellenden Daten. Dies beinhaltet die aktuellen Daten vom Backend anzufordern und Änderungen nach der Geschäftlogik des Backends zu beantragen, sowie die erhaltenen Informationen in Model-Objekten festzuhalten.

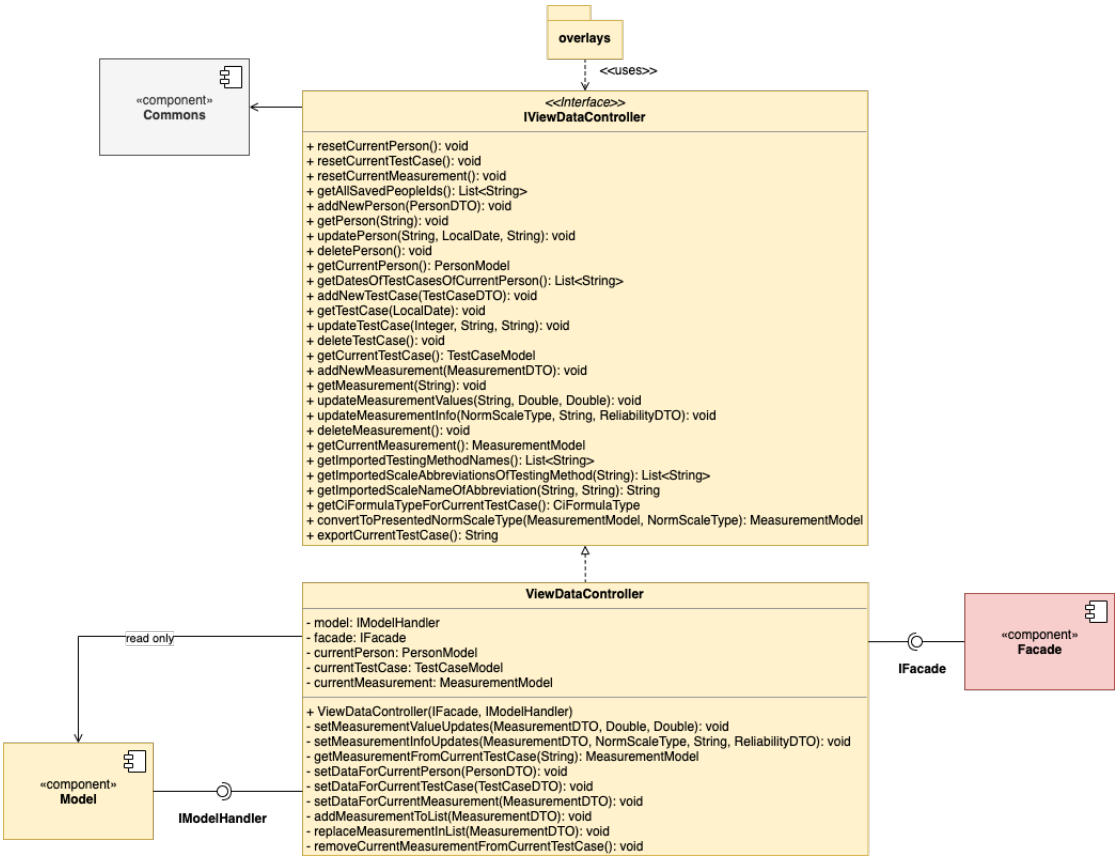


Abbildung 6.14: Whitebox UI.ViewController.ViewDataController

UI.ViewController.Overlays Paket

Das **Overlays**-Paket ist wiederum weiter nach Aufgaben unterteilt. Zentral für die gesamte UI-Komponente ist die *OverlayController*-Klasse. Sie startet die UI-Komponente, in dem auf ein *StageReadyEvent* reagiert wird. Nach außen wird von den enthaltenen Klassen keine Schnittstelle angeboten.

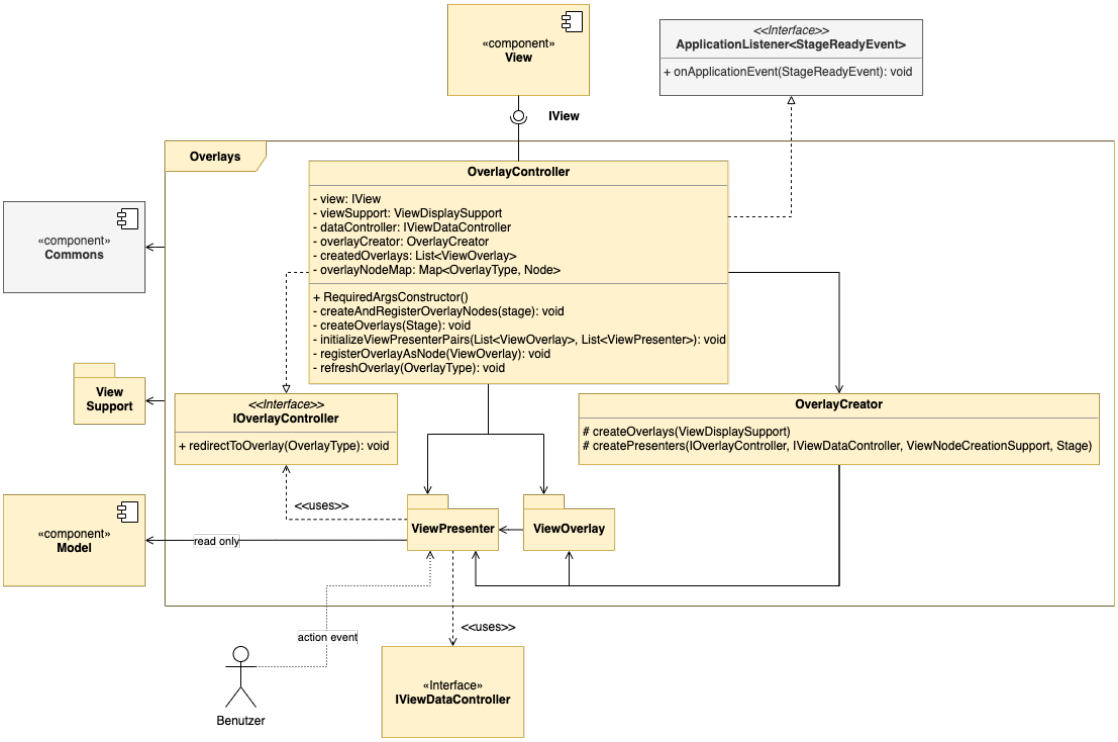


Abbildung 6.15: Whitebox UI.ViewController.Overlays

Schnittstelle	**Beschreibung**
ApplicationListener	Starten des Frontends beim Start der Anwendung
IView	View beim Start initialisieren, Overlay-Wechsel in der View beantragen

Tabelle 6.13: Benötigte Schnittstellen UI.ViewController.Overlays

Baustein	Verantwortlichkeit
OverlayController	Startet UI-Komponente, Initiieren der Erstellung und Darstellung von Overlays, Kommunikation mit der View-Komponente
IOverlayController	Internen Overlay-Wechsel vorbereiten und an die View weitergeben
OverlayCreator	Erstellen der konkreten ViewOverlay- und ViewPresenter-Instanzen
ViewPresenter-Paket	Konkrete Darstellungslogik, Umgang mit Benutzer-Aktionen, Aufbereiten der darzustellenden Informationen in JavaFX-Objekten
ViewOverlay-Paket	Overlay in ein View-darstellbares Format bringen, Anordnung und Stil der JavaFX-Objekte

Tabelle 6.14: Bausteine UI.ViewController.Overlays

Anmerkungen:
Um zu verhindern, dass die konkreten Instanzen der ViewPresenter auf die OverlayController-Klasse frei zugreifen können, wurde ein Interface innerhalb des Pakets hinzugefügt, da angelehnt an das Geheimnis-Prinzip nicht die volle Funktionalität der Klasse für die Presenter nutzbar sein soll.

Für jede Ansicht in der Anwendung gibt es einen klar definierten View-Presenter mit dazugehörigem ViewOverlay. Wird ein neues View-Presenter-Paar implementiert, muss dies dem OverlayCreator hinzugefügt werden. Entsprechend muss darauf geachtet werden, dass die zusammengehörigen Klassen den gleichen OverlayType angegeben haben und dieser nicht mehrfach vorkommt.

UI.ViewController.Overlays.ViewPresenter Paket

Das **ViewPresenter**-Paket beinhaltet alle konkreten Presenter-Implementierungen, deren vereinende abstrakte Superklasse sowie benötigte Hilfsklassen für Bezeichnungen (ViewPresenterTexts) und zur Darstellung von Tabellen (Table-Entities).

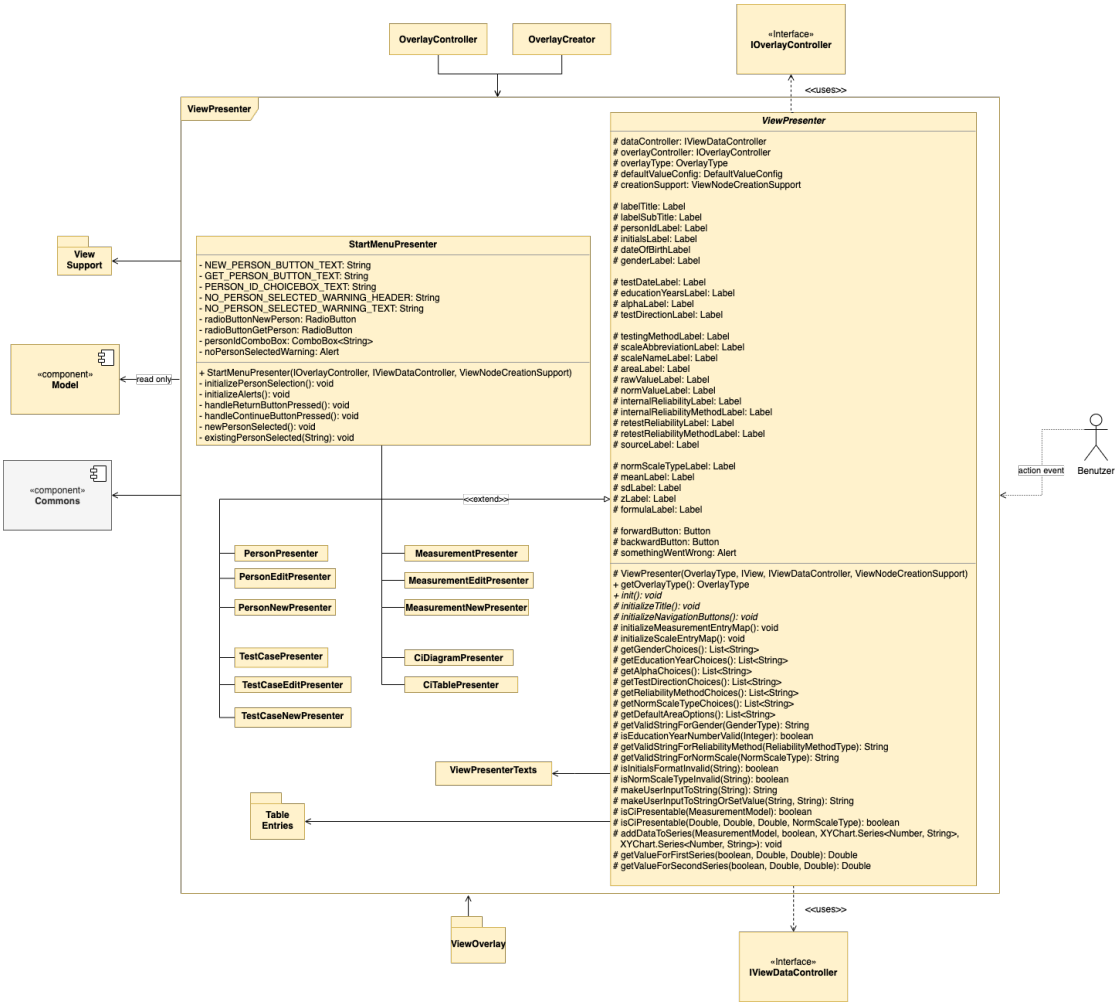


Abbildung 6.16: Whitebox UI.ViewController.Overlays.ViewPresenter

Baustein	Verantwortlichkeit
ViewPresenter	Abstrakte Superklasse aller Presenter, beinhaltet mehrfach genutzte Methoden und Variablen der Presenter
ViewPresenterTexts	Statische allgemeine String-Bezeichnungen, von allen Presentern benutzbar
TabelEntries-Paket	Klassen zur Unterstützung bei der Darstellung von Tabellen
StartMenuPresenter	Darstellungslogik des Startseiten-Overlays
PersonPresenter	Darstellungslogik des Overlays der Übersicht einer ausgewählten Person
PersonEditPresenter	Darstellungslogik des Overlays zum Bearbeiten von Personendaten
PersonNewPresenter	Darstellungslogik des Overlays zum Anlegen einer neuen Person
...	...
CiDiagramPresenter	Darstellungslogik des Overlays zum Abbilden der KI-Grafik
CiTablePresenter	Darstellungslogik des Overlays zum Abbilden der Werte-Tabelle auf denen die KI-Grafik basiert

Tabelle 6.15: Bausteine UI.ViewController.Overlays.ViewPresenter

Die Verantwortlichkeiten der Measurement- und TestCase-Presenter sind jeweils analog zu den Person-Presentern.

Anmerkungen:
Um zu verhindern, dass die konkreten Instanzen der ViewPresenter auf die OverlayController-Klasse frei zugreifen können, wurde ein Interface innerhalb des Pakets hinzugefügt, da angelehnt an das Geheimnis-Prinzip nicht die volle Funktionalität der Klasse für die Presenter nutzbar sein soll.

Der ViewPresenter selbst ist dafür zuständig, welche Informationen in welcher Form im Overlay dargestellt werden sollen. Die konkreten Presenter arbeiten mit den Daten der Model-Subkomponente und bereiten diese mit Hilfe von JavaFX-Scene-Elementen wie beispielsweise Button, Alert und Label für die Darstellung entsprechend auf. Presenter-spezifische Bezeichnungen und Ausgabetexte, werden in der jeweiligen Klasse als statische und unveränderliche String-Objekte definiert.

Zudem sind sie für den Umgang mit Benutzeraktionen zuständig und beinhalten die Logik, wie bei welcher Aktion reagiert werden soll.

Angelehnt an das Geheimnis-Prinzip, nicht mehr preiszugeben als nötig, wurde versucht mit Hilfe von Sichtbarkeiten die Zugriffe auf die ViewPresenter-Objekte zu kontrollieren. Die Konstruktoren der einzelnen Presenter sind jeweils auf *public* gesetzt, damit der OverlayCreator auf diesen zum Erstellen einer Instanz zugreifen kann. Alle lokalen Variablen und Methoden der konkreten Presenter hingegen sind auf *private* gesetzt. Nur für das dazugehörige *ViewOverlay* relevante Variablen können mit der jeweiligen Getter-Methode gelesen werden. Die Superklasse *ViewPresenter* beinhaltet zudem fast ausschließlich *protected*-Elemente. Lediglich die Ermittlung des zugehörigen OverlayTyps, sowie die Initialisierung ist für von außerhalb des Paketes aufrufende Klassen zur Identifizierung der Zugehörigkeit und zum Aktualisieren der Daten möglich.

UI.ViewController.Overlays.ViewPresenter.TableEntries Paket

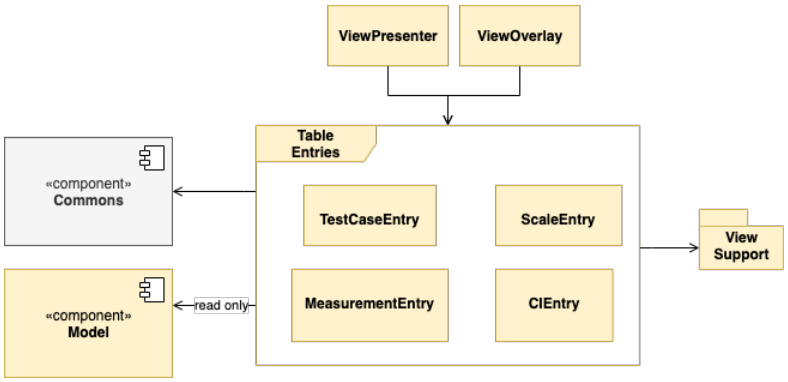


Abbildung 6.17: Whitebox UI.ViewController.Overlays.ViewPresenter.TableEntries

Anmerkungen:
Die Klassen repräsentieren den Aufbau von Tabelleneinträgen. Sie erleichtern die Dateneingabe und Abfrage vom Benutzer in die Tabelle eingegebenen Werte und werden ausschließlich von den konkreten *ViewPresenter*- und den dazugehörigen *ViewOverlay*-Klassen verwendet.

UI.ViewController.Overlays.ViewOverlay Paket

Das **ViewOverlay**-Paket beinhaltet alle konkreten Overlay-Implementierungen und deren vereinende abstrakte Superklasse. Das zu einem *ViewPresenter* dazugehörige **ViewOverlay** ist dafür zuständig, wie die vom Presenter aufbereiteten Informationen darzustellen sind.

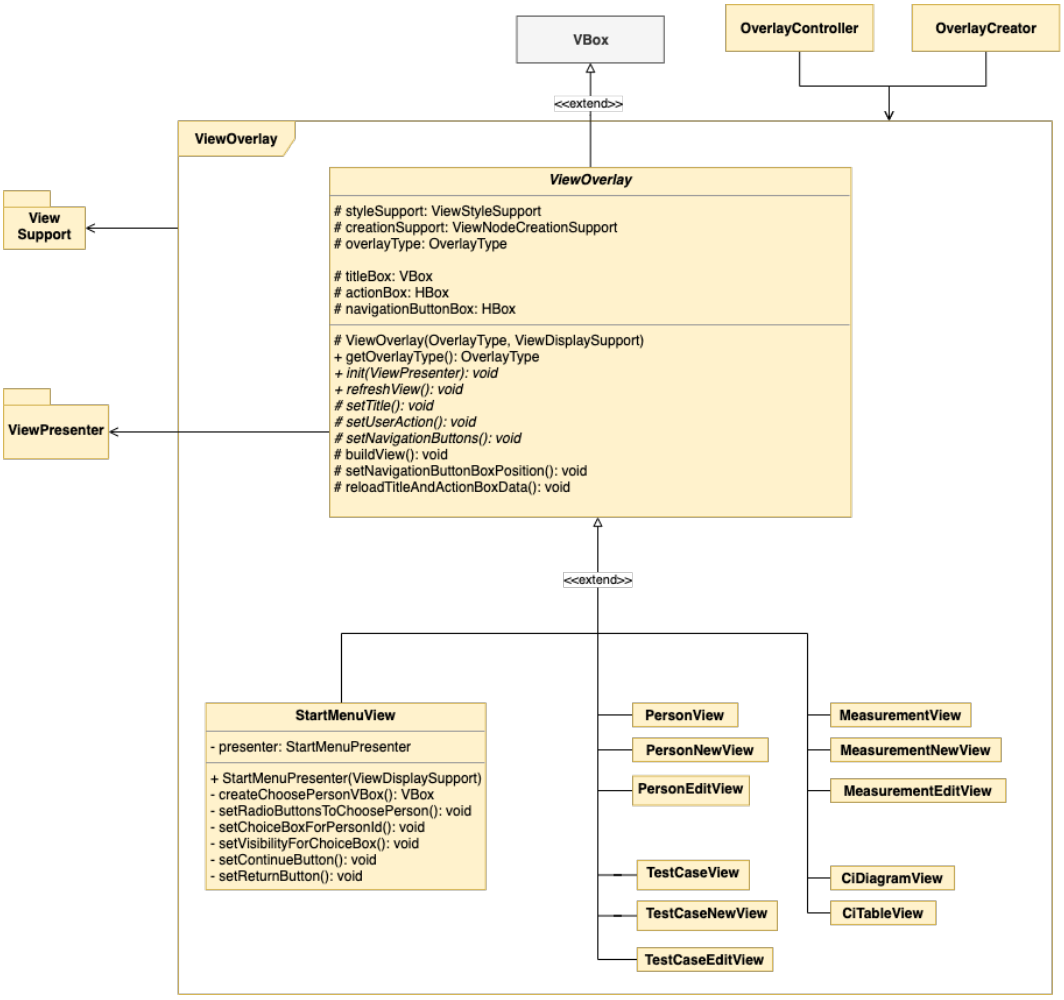


Abbildung 6.18: Whitebox UI.ViewController.Overlays.ViewOverlay

Baustein	Verantwortlichkeit
ViewOverlay	Abstrakte Superklasse aller Overlays, beinhaltet mehrfach genutzte Methoden und Variablen der Overlays
StartMenuView	Overlay der Startseite
PersonView	Overlay der Übersicht einer ausgewählten Person
PersonEditView	Overlay zum Bearbeiten von Personendaten
PersonNewView	Overlay zum Anlegen einer neuen Person
...	...
CiDiagramView	Overlay zum Abbilden der KI-Grafik
CiTableView	Overlay zum Abbilden der Werte-Tabelle auf denen die KI-Grafik basiert

Tabelle 6.16: Bausteine UI.ViewController.Overlays.ViewOverlay

Die Verantwortlichkeiten der Measurement- und TestCase-Overlays sind jeweils analog zu den Person-Overlays.

Anmerkungen:
Da die Instanzen dieser Klassen als Overlays an die View-Subkomponente zur Darstellung weitergereicht werden, wurde versucht keine zusätzlichen Abhängigkeiten zu erzeugen. Daher wurde die Superklasse als eine Erweiterung der JavaFX-Klasse *VBox* (javafx.scene.layout.VBox) implementiert. Die View kann daher mit den Elementen auf der Node-Ebene arbeiten, ohne deren konkrete Implementierung zu kennen.

Um auch hier Code-Wiederholungen zu vermeiden, sowie die Elemente in Relation zu der Fenstergröße darzustellen, werden die Hilfsklassen *ViewStyleSupport* und *ViewNodeCreationSupport* verwendet.

Genau wie die ViewPresenter, müssen die ViewOverlay-Klassen initialisiert werden. Da sie auf den konkreten Daten eines Presenters arbeiten, muss dieser bei der Initialisierung referenziert werden. Um sicherzustellen, dass die Overlays immer mit den aktuellen Daten arbeiten, müssen diese bei einem Wechsel aktualisiert werden. Bei einer solchen Aktualisierung ist die Aufgabe des ViewOverlays lediglich die erneute Initialisierung des dazugehörige Presenters. Der Presenter holt sich dabei eigenständig die relevanten Daten von der Model-Subkomponente. Allgemein wurde wie beim ViewPresenter versucht, das

Geheimnis-Prinzip bezüglich der außerhalb des Paketes liegenden *OverlayController*- und *OverlayCreator*-Klassen mit Hilfe der Sichtbarkeiten umzusetzten.

Facade Komponente

Zweck:
Start des Backends, Zentrales Interface für Anfragen an das Backend und Koordination der Backend-Services

Beim Start der Applikation, wird diese mitsamt allen benötigten Argumenten durch ein StageReadyEvent initialisiert und startet mit Hilfe von Spring und der *@Autowired*-Annotationen das Backend.

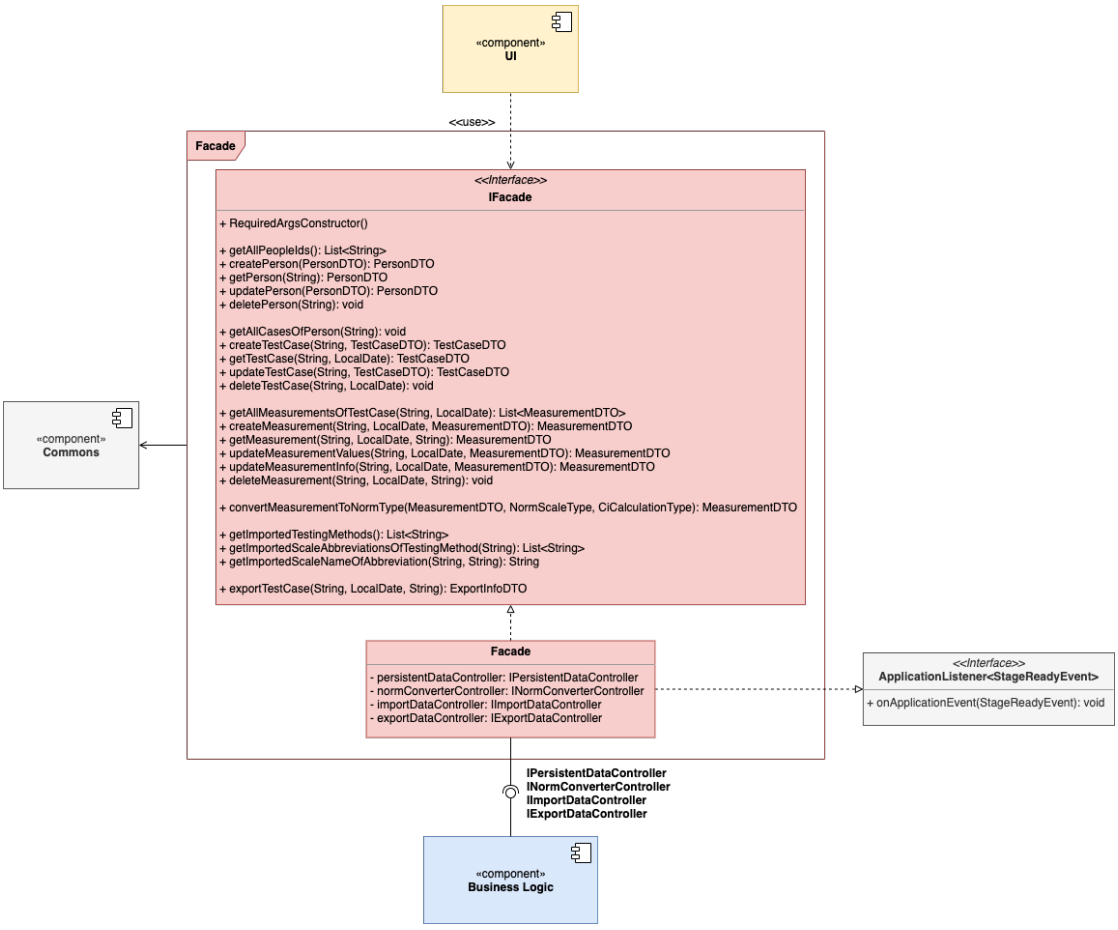


Abbildung 6.19: Whitebox Facade

Baustein	Verantwortlichkeit
Facade	Starten des Backends, Koordination der Backend-Services
IFacade	Nach außen angebotene Schnittstelle als zentrales Interface für Anfragen an das Backend

Tabelle 6.17: Bausteine Facade

Schnittstelle	Beschreibung
ApplicationListener	Starten des Backends beim Start der Anwendung
IPersistentDataController	CRUD-Operationen für persistent gehaltene Daten
INormConverterController	Transformieren von Normwerten auf eine andere Normskala
IImportDataController	Einlesen von Informationen aus Dateien
IExportDataController	Exportieren von Informationen in Dateien

Tabelle 6.18: Benötigte Schnittstellen Facade

Anmerkungen:
Für die Koordination von Anfragen greift die Komponente auf die unterschiedlichen Services des Backends über deren angebotenen Schnittstellen zu.

Um einen *Single-Point-Of-Failure* möglichst zu vermeiden, leitet die Facade Anfragen hauptsächlich weiter, ohne die mitgegebenen Parameter zu betrachten oder zu verändern. Die Prüfung und das Abfangen von Exceptions fallen daher in die Zuständigkeit der jeweiligen Services.

Die Facade arbeitet ausschließlich mit Java-Datentypen und denen der *Commons*-Komponente. Dadurch entstehen keine weiteren Abhängigkeiten und die Facade muss keine Konvertierungen von Objekten durchführen, die zwischen den Front- und Backend Komponenten durchgereicht werden.

Business-Logic Komponente

Die **Business-Logic**-Komponente repräsentiert das Backend der Applikation. Die Facade-Komponente delegiert Anfragen an die zuständigen Business-Logic-Subkomponenten *Norm-*

Converter, PersistentData, ExportData und *ImportData*. Hierfür werden die angebotenen Schnittstellen mit klar definierten Funktionen der jeweiligen Subkomponente verwendet. Es besteht keine direkte Kommunikation unter den Subkomponenten.

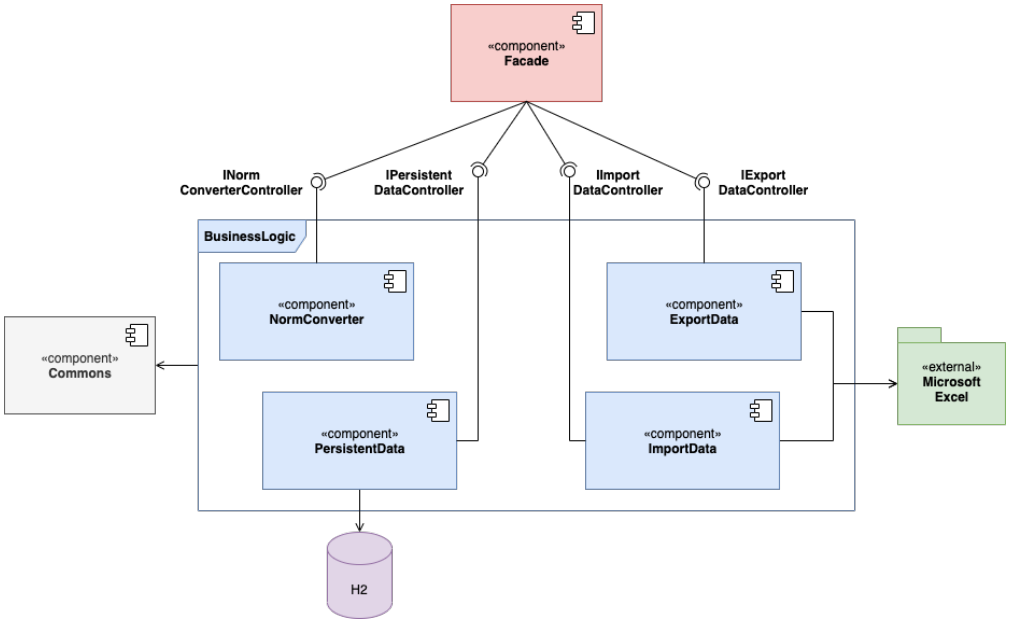


Abbildung 6.20: Whitebox Business-Logic

Baustein	**Verantwortlichkeit**
NormConverter	Transformieren von Normwerten zwischen Normskalen und Anpassung des Konfidenzintervalls an einen transformierten Normwert.
ExportData	Exportieren von Daten in eine Excel-Datei
ImportData	Einlesen und Halten von Daten zur Laufzeit
PersistentData	Umgang mit persistent gehaltenen Entitäten, Datenbankzugriff und deren CRUD-Funktionen

Tabelle 6.19: Bausteine Business-Logic

Business-Logic.Import-Data Subkomponente

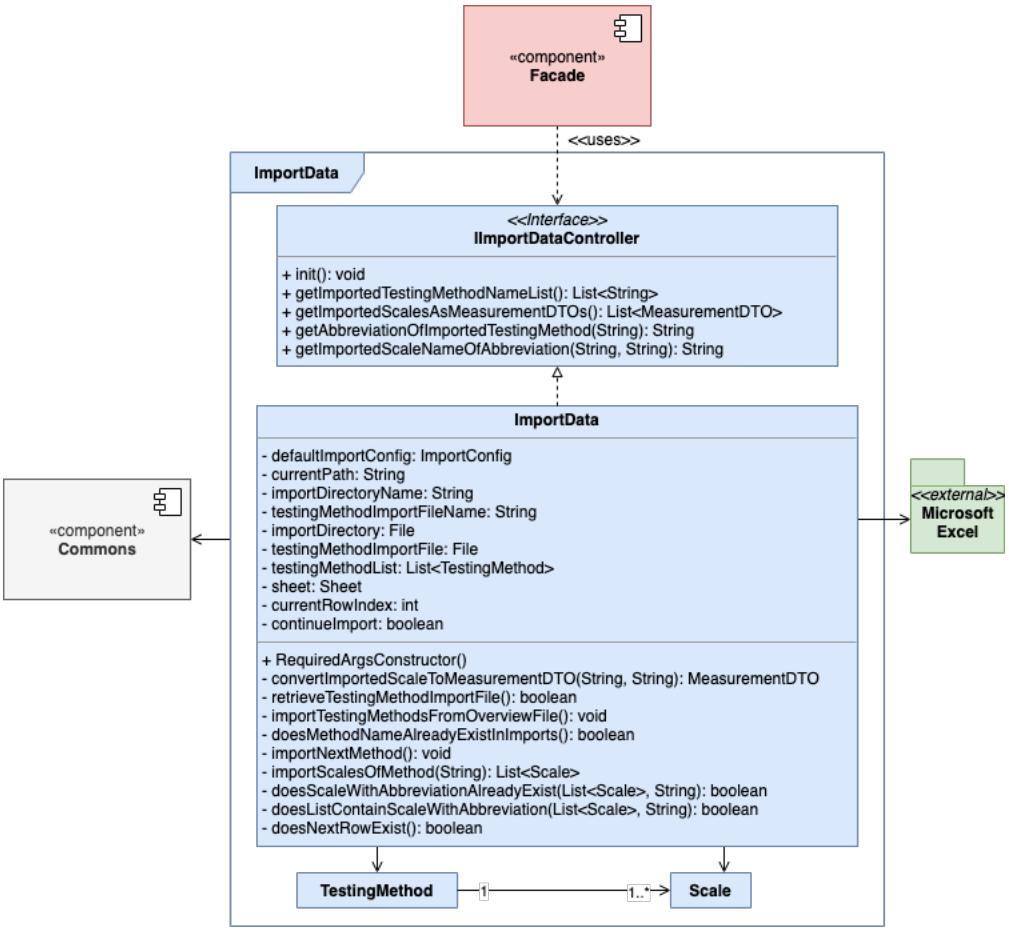


Abbildung 6.21: Whitebox Business-Logic.Import-Data

Baustein	**Verantwortlichkeit**
ImportData	Service zum Einlesen und Halten von Daten zur Laufzeit
IImportDataController	Nach außen angebotene Schnittstelle zum Initiieren der Komponente und Lese-Anfragen auf die eingelesenen Daten
TestingMethod	Daten-Objekt zur internen Repräsentation eines Testverfahrens
Scale	Daten-Objekt zur internen Repräsentation einer Skala

Tabelle 6.20: Bausteine Business-Logic.Import-Data

Anmerkungen:
Aktuell werden in dieser Subkomponente Informationen zu Testverfahren und deren Skalen aus einer Excel-Datei eingelesen.

Beim Initialisieren der *ImportData*-Klasse durch die *init()*-Methode werden die Import-Variablen aus der *ImportConfig* verwendet, um die darin definierten Namen des Import-Ordners und der Testverfahren-Übersichtstabelle zu entnehmen. Nach diesen Dateien wird vom aktuellen Pfad aus gesucht und diese lokal gespeichert. Sind der Import-Ordner, sowie die darin enthaltene Datei der Übersichtstabelle valide Dateien (nicht *null* und existieren), wird der Testverfahren-Import gestartet.

Der Zugriff auf die Excel-Datei, welche die Übersichtstabelle beinhaltet, erfolgt mit der *Workbook*-Klasse des Apache-POI-Framework. Dadurch kann gezielt auf *Tabellenblätter, Zeilen* und *Zellen* innerhalb der Excel-Datei zugegriffen werden.
Die Übersichtstabelle wird zeilenweise nach Testverfahren importiert. Zusammengehörige Skalen des gleichen Testverfahrens müssen daher unmittelbar untereinander aufgelistet werden. Bei Duplikaten wird nur das erste Testverfahren mit dem entsprechenden Namen importiert. Das gleiche gilt für Duplikate bei den Skalen. Dies war eine bewusste Entscheidung, und soll die Benutzer zwingen Ordnung in der Datei zu halten, sowie die Testverfahren und Skalen überlegt zu benennen.

Business-Logic.Export-Data Subkomponente

Baustein	**Verantwortlichkeit**
ExportData	Service zum Exportieren von Daten in eine Excel-Datei
IExportDataController	Nach außen angebotene Schnittstelle zum Initiieren des Exports von Daten

Tabelle 6.21: Bausteine Business-Logic.Export-Data

Anmerkungen:
Bisher wird lediglich der Export von Falldaten als Service angeboten.

Durch das Aufrufen der *init()*-Methode über die Schnittstelle, wird der Default Export-Ordner initialisiert. Sollte bei dem Export der gewünschte Ordner nicht zu finden sein, wird dieser als Ablageort für die erstellte Datei verwendet.

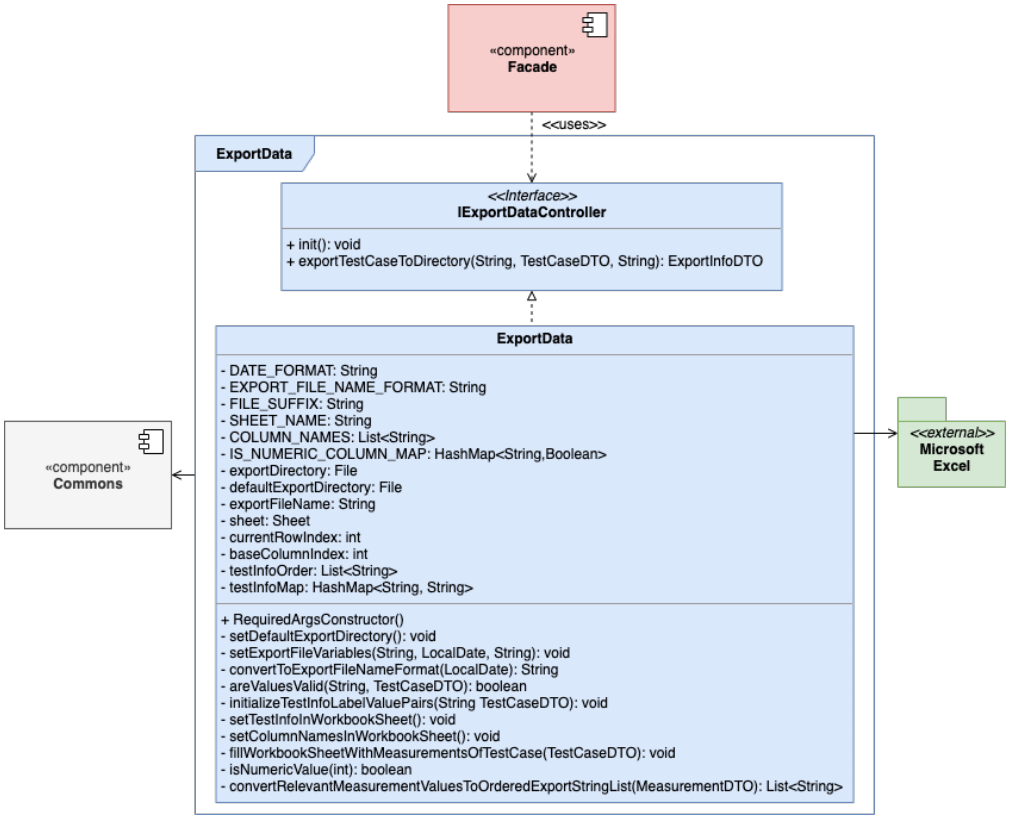


Abbildung 6.22: Whitebox Business-Logic.Export-Data

Die Klasse arbeitet auf DTO-Repräsentationen der Entitäten, um Abhängigkeiten zu anderen Services, speziell zu dem der persistenten Datenhaltung, zu vermeiden.
Das Schreiben der Daten erfolgt ebenfalls mit dem Apache-POI-Framework über die *Workbook*-Klasse, speziell eine *XSSFWorkbook*-Instanz. Hier muss unbedingt die Excel-Version beachtet werden, die auf dem Anwendungs-Rechner installiert ist. Sollte diese älter als 2007 sein, muss die *HSSFWorkbook*-Klasse stattdessen verwendet werden.
Der Aufbau der Export-Datei ist in diesem Service hartkodiert, da nur eine Auswahl der Daten exportiert werden. Diese variieren aus Informationen des Falls, Messwerts und dessen Konfidenzintervall. Um diese in den jeweiligen Spalten korrekt anzuordnen, wurden einige Hilfs-Variablen, wie beispielsweise *testInfoOrder* und *testInfoMap*, sowie entsprechende Hilfs-Methoden zur Umsetzung hinzugefügt.

Der Name für die exportierende Datei wird aus der Personen-ID und dem Testdatum abgeleitet und in folgendem Format für die Speicherung verwendet:

PersonID_ YYYY-MM-DD.xlsx. Dadurch werden die Dateien bei einer Auflistung automatisch in einer nachvollziehbaren und geordneten Reihenfolge dargestellt.

Der Facade wird ein ExportInfoDTO zurückgegeben. Dadurch können alle relevanten Informationen zu dem Export gesammelt übergeben werden und der Export-Vorgang muss nicht für eventuelle Nachfragen lokal gespeichert werden.

Business-Logic.Norm-Converter Subkomponente

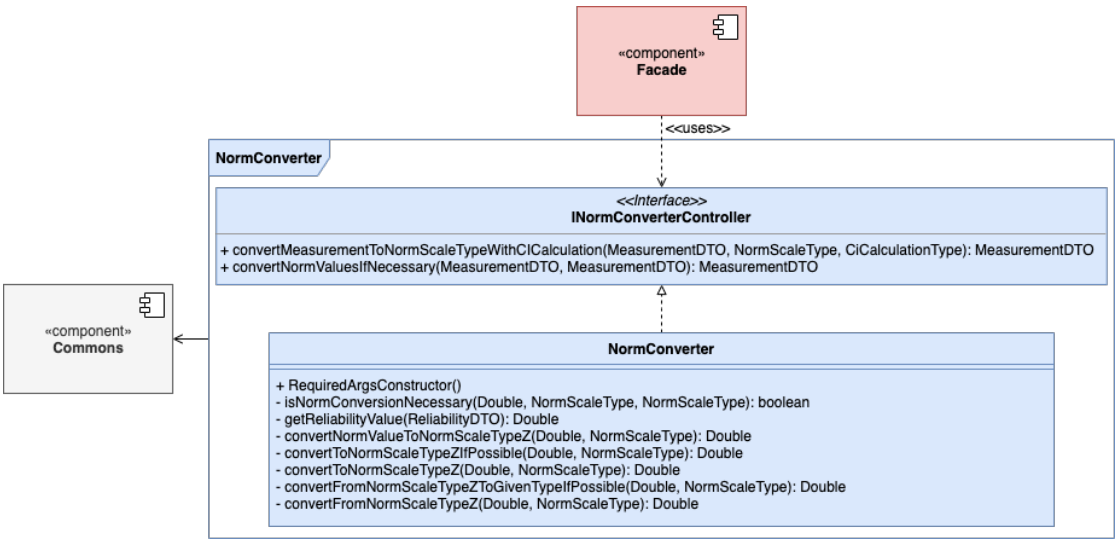


Abbildung 6.23: Whitebox Business-Logic.Norm-Converter

Baustein	Verantwortlichkeit
NormConverter	Transformieren von Normwerten und Anpassung der Konfidenzintervalle an die Norm
INormConverterController	Nach außen angebotenen Schnittstelle zum Transformieren von Normwerten

Tabelle 6.22: Bausteine Business-Logic.Norm-Converter

Anmerkungen:
Die Subkomponente bietet das Transformieren eines Messwerts in einen expliziten Normskalen-Typ an, oder das Überprüfen eines Messwerts mit angegebenen Änderungen und führt

eine Transformation des Normwertes aus, wenn diese nötig ist.
In beiden Fällen erfolgt die Transformation nur unter der Voraussetzung, dass alle benötigten Daten vorhanden sind und für den gewünschte Normskalen-Typ das Konfidenzintervall berechnet werden kann. Diese sind zum aktuellen Stand auf die Normskalen *T-Wert, IQ, Wertpunkte* und *z-Wert* begrenzt, da die übrigen Normskalen teilweise zu ungenau sind oder von den Neuropsychologen nicht benötigt werden.

Der NormConverter arbeitet allgemein ausschließlich auf Objekten der *Commons*-Komponente (DTO und fachliche Datentypen), wodurch keine Abhängigkeiten zu anderen Komponenten im System bestehen.

Business-Logic.Persistent-Data Subkomponente

Baustein	**Verantwortlichkeit**
PersistentData	Prüft die Eingaben und koordiniert die Umsetzung der Anfragen, konvertiert Schlüsselinformationen in jeweilige Schlüssel-Entitäten für die weitere Verarbeitung
IPersistentDataController	Nach außen angebotenen Schnittstelle zum Umgang mit persistent gehaltener Daten
Person-Komponente	Umgang mit persistenten Daten der Person-Entitäten, sowie die damit verbundenen CRUD-Operationen und entsprechendem Datenbankzugriff
TestCase-Komponente	Analog zur Person-Komponente bezogen auf die Fall-Entität *TestCase*
Measurement-Komponente	Analog zur Person-Komponente bezogen auf die Messwert-Entität *Measurement* und Umgang mit zusammenhängenden Reliabilitäten und Konfidenzintervallen des Messwerts.
Reliability-Komponente	Analog zur Person-Komponente bezogen auf die Reliabilitäts-Entität *Reliability*
ConfidenceInterval-Komponente	Analog zur Person-Komponente bezogen auf die Konfidenzintervall-Entität *ConfidenceInterval*
Entity-Keys-Paket	Beinhaltet Schlüssel-Objekte der schwachen Entitäten

Tabelle 6.23: Bausteine Business-Logic.Persistent-Data

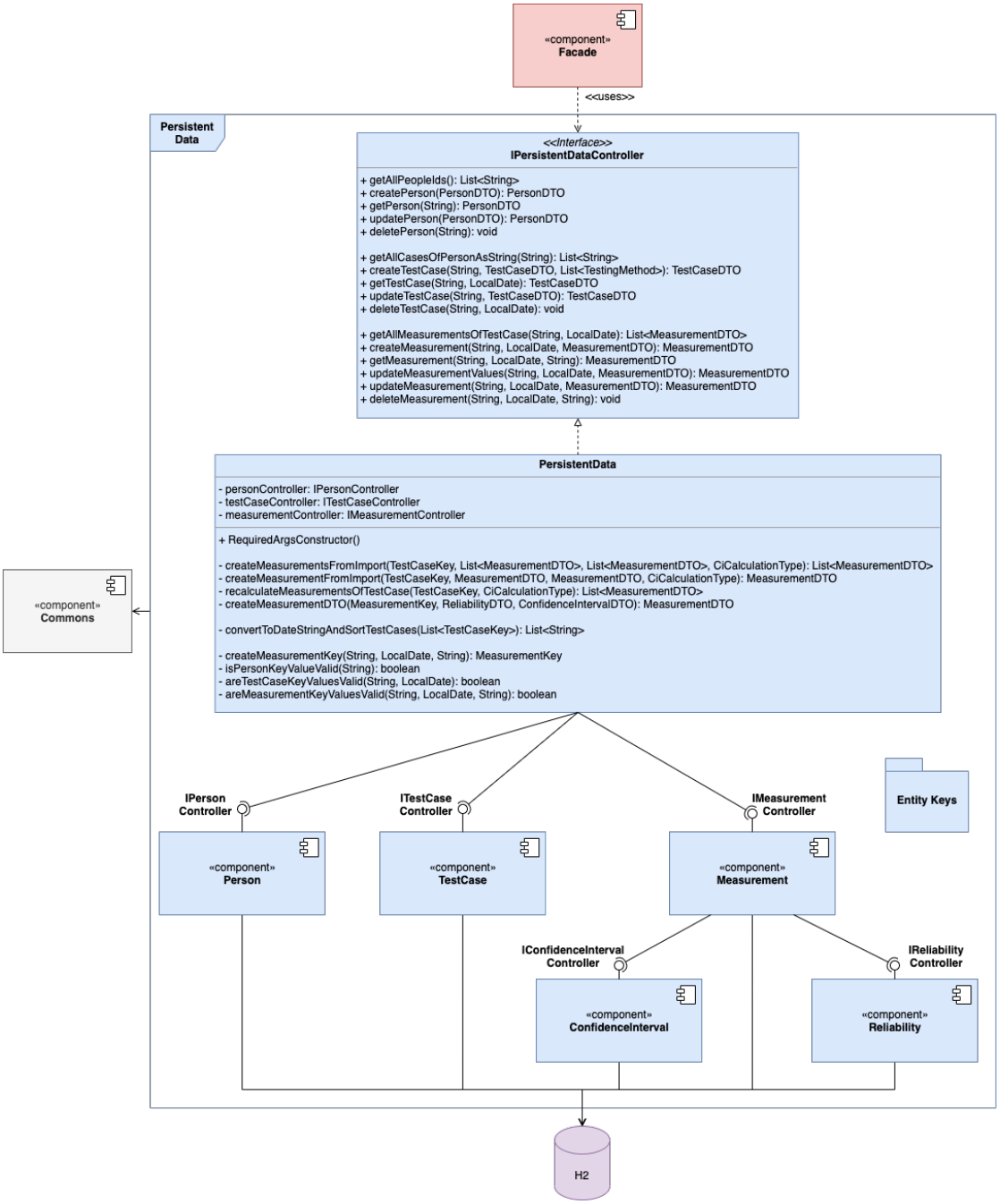


Abbildung 6.24: Whitebox Business-Logic.Persistent-Data

Schnittstelle	Beschreibung
IPersonController	Umgang mit persistent gehaltenen Daten der Person-Entität
ITestCaseController	Umgang mit persistent gehaltenen Daten der Fall-Entität
IMeasurementController	Umgang mit persistent gehaltenen Daten der Messwert-Entität

Tabelle 6.24: Benötigte Schnittstellen Business-Logic.Persistent-Data

Anmerkungen:
Die Kommunikation unterhalb der Persistent-Data-Subkomponenten erfolgt über die angebotenen Schnittstellen der Subkomponenten. Die nach außen angebotene Schnittstelle arbeitet lediglich auf Objekten der *Commons*-Komponente und Java-Typen.

Um die Anzahl der Parameter für die Schlüssel der Fall- und Messwert Entitäten zu reduzieren und Redundanzen zu vermeiden, wurden innerhalb der *Persistent-Data*-Subkomponente Entität-Schlüssel-Klassen eingeführt. Diese sind in dem Paket *Entity-Keys* gesammelt, auf welches alle Elemente der *Persistent-Data*-Subkomponente bei Bedarf zugreifen können.

Da die *Reliability*- und *ConfidenceInterval*-Subkomponenten fachlich zur *Measurement*-Subkomponente zuzuordnen sind und dadurch starke Abhängigkeiten zu dieser bestehen, ist sie für deren Umgang verantwortlich. Dadurch kann die *PersitentData*-Klasse ohne Wissen über spezielle Implementierungen anderer Subkomponenten arbeiten und zugleich können Abhängigkeiten dieser zu den persistenten Entitäten vermieden wird.

Business-Logic.Persistent-Data.EntityKeys Paket

Baustein	Verantwortlichkeit
TestCaseKey	Attribute für eindeutigen Schlüssel der Fall-Entität *TestCase*
MeasurementKey	Attribute für eindeutigen Schlüssel der Messwert-Entität *Measurement*

Tabelle 6.25: Bausteine Business-Logic.Persistent-Data.EntityKeys

Anmerkungen:
Da es sich bei dem *Fall* und *Messwert* um schwache Entitäten handelt, sind diese nicht anhand ihrer eigenen Attribute eindeutig identifizierbar. Diese Schlüssel-Objekte beinhalten dabei alle benötigten Variablen, um solch eine Identifizierung umzusetzen.

Die Variablen der Klassen haben die *@Nonnull*-Annotation von Lombok, wodurch eine NullpointerException geworfen wird, sollte gegen die damit verbundene Bedingung verstoßen werden. Zudem existieren nur Getter- und keine Setter-Methoden. Dadurch kann bei einem Objekt dieser Schlüssel-Klassen davon ausgegangen werden kann, dass die Schlüssel-Attribute nicht undefiniert sind. Der *NoArgsConstructor* ist leider notwendig, da diese Objekte von Entitäten benutzt werden und *Hibernate* sonst eine *org.hibernate. InstantiationException: No default constructor for entity* wirft.

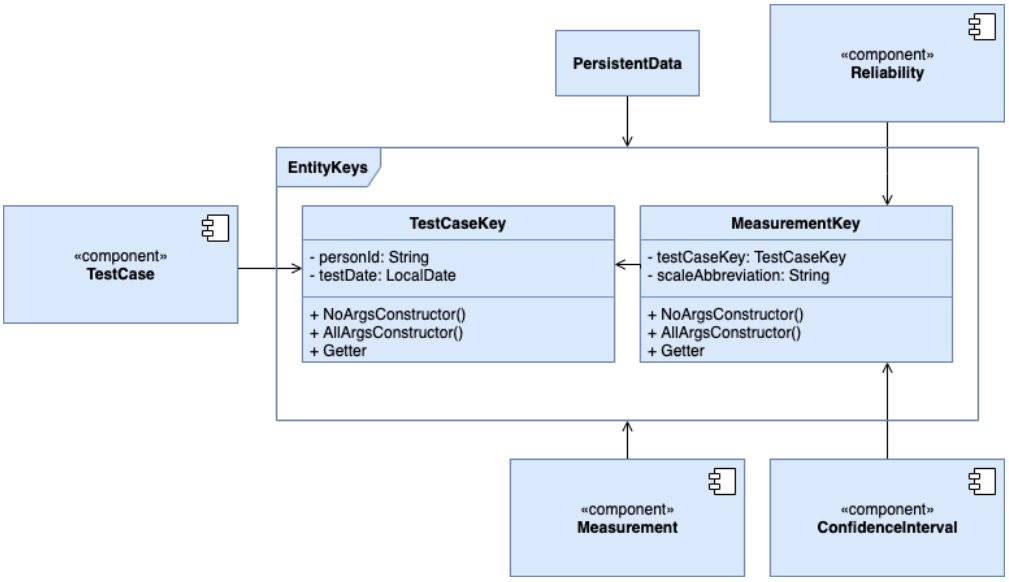


Abbildung 6.25: Whitebox Business-Logic.Persistent-Data.EntityKeys

Die Schlüssel-Klassen werden ausschließlich innerhalb der Persistent-Data-Subkomponente verwendet. Die *PersistentData*-Klasse nutzt diese, um die übergebenen Werte zu überprüfen und diese mit den Anfragen weiterzureichen. Dadurch werden die relevanten Schlüssel-Informationen kompakt in einem Objekt festgehalten und gegebenenfalls mit in der Datenbank abgespeichert.

Angelehnt an die Relationstypen verwenden die *Reliability*- und *ConfidenceInterval*-Subkomponenten diese Klasse als eine Art Fremdschlüssel bezüglich des zugehörigen Messwerts.

Business-Logic.Persistent-Data.Person Subkomponente

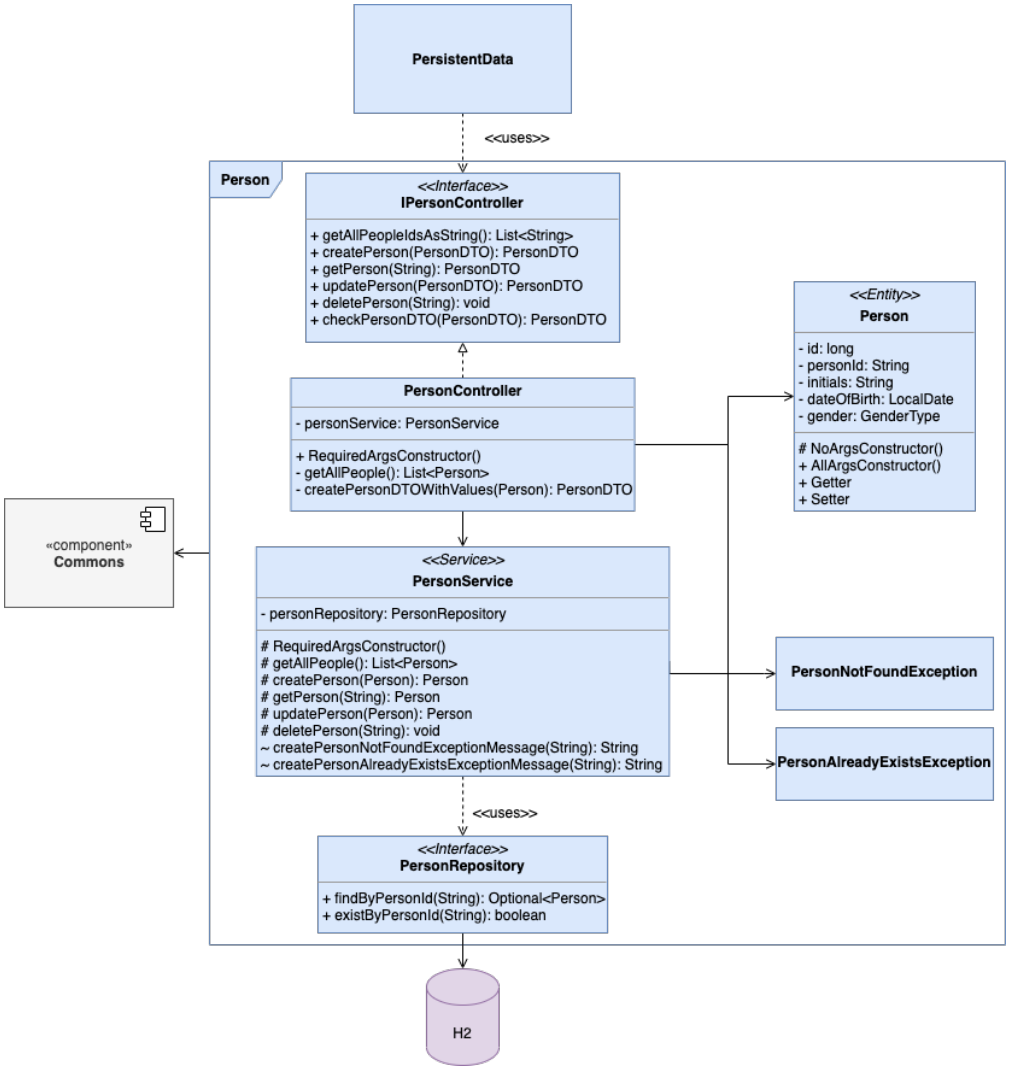


Abbildung 6.26: Whitebox Business-Logic.Persistent-Data.Person

Baustein	Verantwortlichkeit
PersonController	Entgegennehmen und Prüfen der Anfragen des Interfaces, sowie der Daten-Überführung zwischen Person-Entität in dessen Transport-Typ PersonDTO, Umgang mit Fehlern
IPersonController	Intern angebotenen Schnittstelle um CRUD-Operationen der Person-Entitäten zu initiieren
PersonService	Datenbankzugriff (CRUD-Operationen)
Person	Entität als Repräsentation des *Person*-Datentyps im fachlichen Datenmodell
PersonNotFoundException	Hinweis Person mit angebenenem Schlüssel nicht in der Datenbank vorhanden
PersonAlreadyExistsException	Hinweis Person mit angebenenem Schlüssel existiert bereits in der Datenbank
PersonRepository	Schnittstelle zur Person-Tabelle der H2-Datenbank

Tabelle 6.26: Bausteine Business-Logic.Persistent-Data.Person

Anmerkungen:
Die Kommunikation mit der *PersistentData*-Klasse erfolgt ausschließlich auf Objekten der *Commons*-Komponente und Java-Typen. Entsprechend werden die Personendaten anhand eines PersonDTO oder in Form ihres Schlüssels nach außen getragen.

Durch die *checkPersonDTO*-Methode des Controllers, werden die Werte der Transport-Klasse überprüft und gegebenenfalls vordefinierte Default-Werte eingesetzt.

Die *Person*-Entity wurde angelehnt an das fachliche Datenmodell implementiert. Hier wurde eine Objekt-ID des Type *Long* eingeführt. Für die Personen-ID wird durch die javax.persistence Annotationen *@Column* mitgeteilt, dass dieser Wert eindeutig sein muss und nicht undefiniert bleiben kann. Entsprechend sind in dem PersonRepository Funktionen definiert, um eine Person anhand der Person-Id zu finden oder nach ihrer Existenz zu fragen.

Business-Logic.Persistent-Data.TestCase Subkomponente

Anmerkungen:
Die **TestCase**-Subkomponente ist, analog zur Person-Subkomponente aufgebaut. Anders als bei der Person-Entität, hat der TestCase allerdings keine zusätzliche ID. Stattdessen wird der Fall über das *TestCaseKey*-Objekt identifiziert.

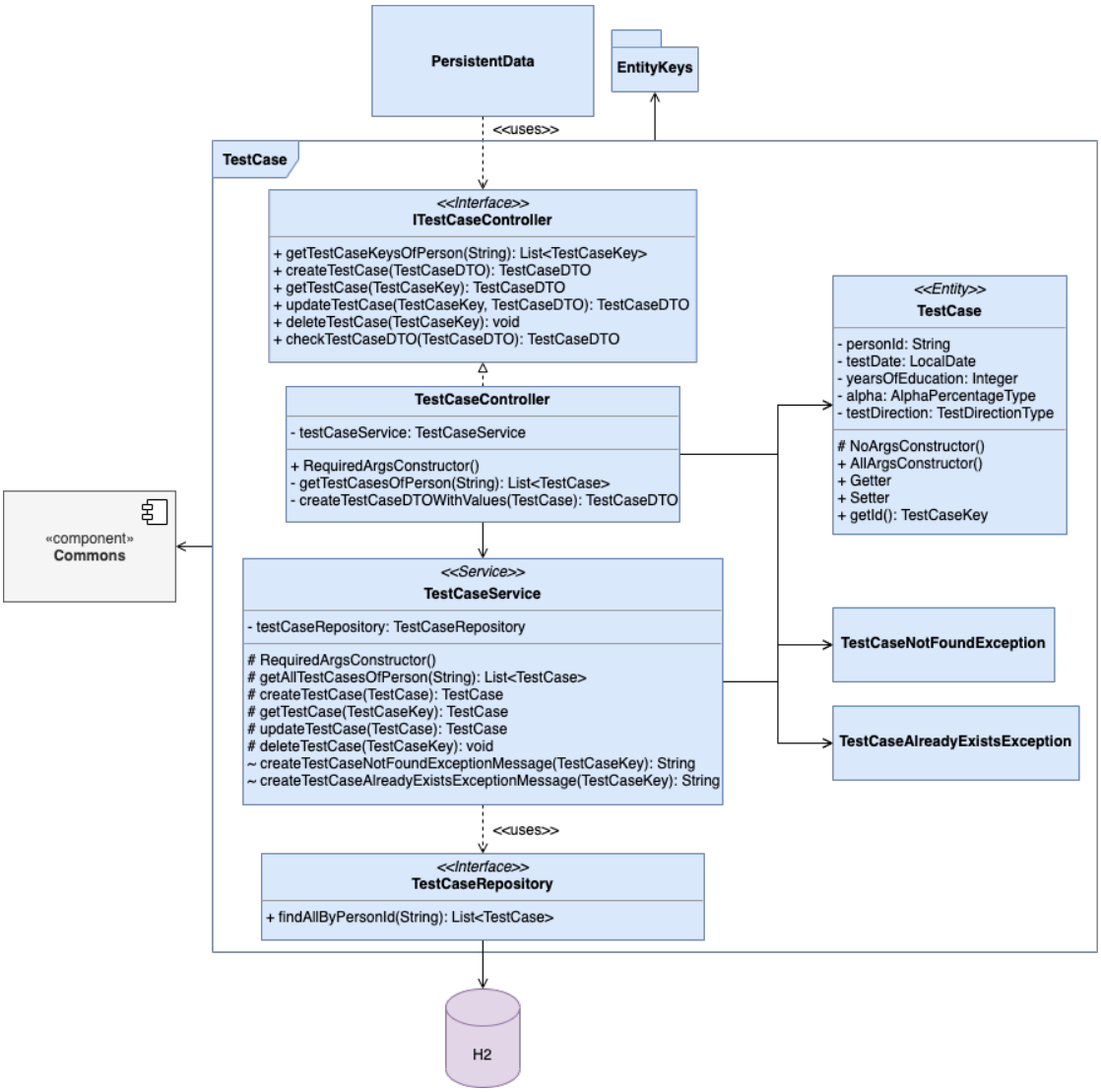


Abbildung 6.27: Whitebox Business-Logic.Persistent-Data.TestCase

Business-Logic.Persistent-Data.Measurement Subkomponente

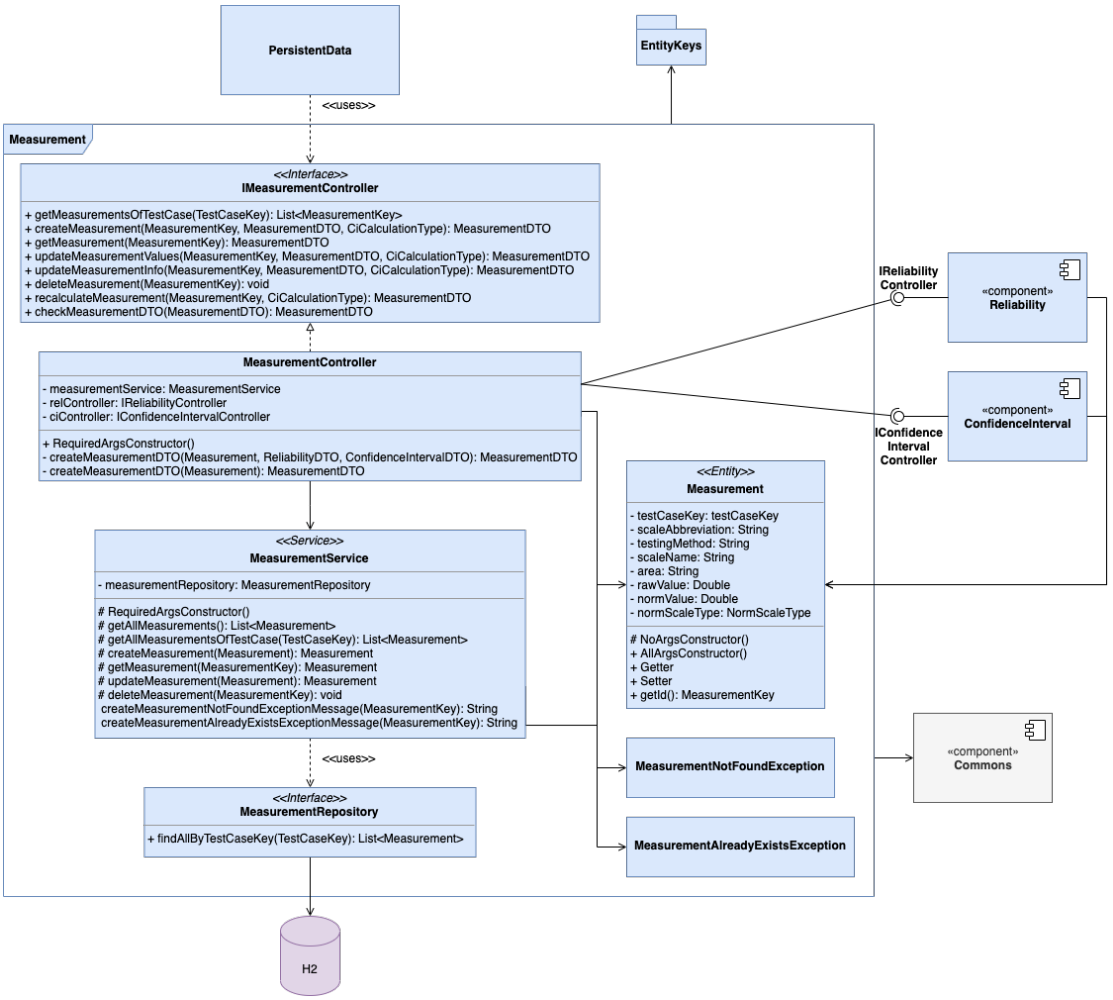


Abbildung 6.28: Whitebox Business-Logic.Persistent-Data.Measurement

Anmerkungen:
Die **Measurement**-Subkomponente ist ebenfalls analog zu der *TestCase*-Subkomponente aufgebaut und hat daher die gleichen Zuständigkeiten bezogen auf die Messwert-Entität *Measurement*. Da es sich auch bei dieser Entität um einen schwache handelt, wurde eine Schlüssel-Klasse *MeasurementKey* eingeführt.

Wodurch sich diese Subkomponente allerdings abhebt, sind die zusätzlichen Zugriff auf die *Reliability*- und *Confidence-Interval*-Subkomponenten. Dies ist notwendig, da diese Komponenten eine Referenz auf die Messwert-Entität besitzen und diese nicht an die

PersistentData-Klasse gegeben werden soll. Der *MeasurementController* kann also bei dem Erstellen eines Messwerts auch die Erstellung der zugehörigen Reliabilität und des Konfidenzintervalls initiieren. Es war zunächst geplant, dass beiden Komponenten nicht mit der Entität selbst, sondern lediglich mit der *Measurement*-Schlüssel-Klasse arbeiten. Bei der Umsetzung gab es allerdings Probleme mit der Datenbank, da der *Measurement-Key*-Wert eine *org.h2.jdbc.JdbcSQLDataException: Wert zu gross* auslöste.

Schnittstelle	**Beschreibung**
IReliabilityController	Initiieren von CRUD-Operationen der Reliabilitäts-Entität
IConfidenceIntervalController	Initiieren von CRUD-Operationen der Konfidenzintervall-Entität

Tabelle 6.27: Benötigte Schnittstellen Business-Logic.Persistent-Data.Measurement

Business-Logic.Persistent-Data.Reliability Subkomponente

Anmerkungen:
Die *Reliability*-Subkomponente ist von den Grundbausteinen analog zur den bisher vorgestellten Komponente aufgebaut.

Eine Unterscheidung bestehen jedoch darin, dass der Controller nicht von der *Persistent-Data*-Klasse aufgerufen wird, sondern vom *MeasurementController* und ein Reliabilitäts-Objekt mit Hilfe der *Measurement*-Klasse in der Datenbank identifiziert wird. Dadurch besteht leider eine Abhängigkeit zur Measurement-Subkomponente, die sonst gerne vermieden worden wäre.

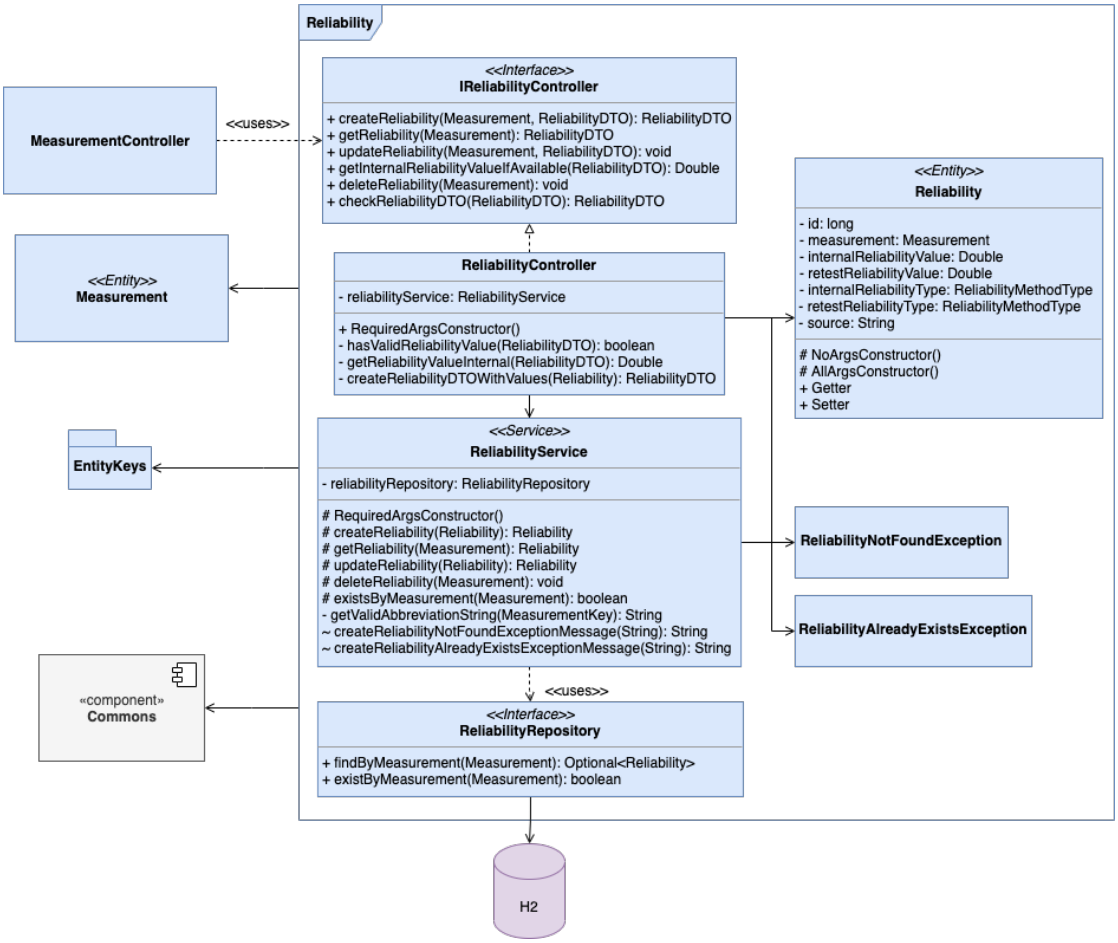


Abbildung 6.29: Whitebox Business-Logic.Persistent-Data.Reliability

Business-Logic.Persistent-Data.Confidence-Interval Subkomponente

Anmerkungen:

Die *Confidence-Interval*-Subkomponente ist analog zur Reliability-Komponente aufgebaut. Die Problematik, wie bereits bei der Reliabilität beschrieben wurde, trifft daher auch auf diese Komponente zu. Entsprechend wurde die gleiche Umsetzung gewählt.

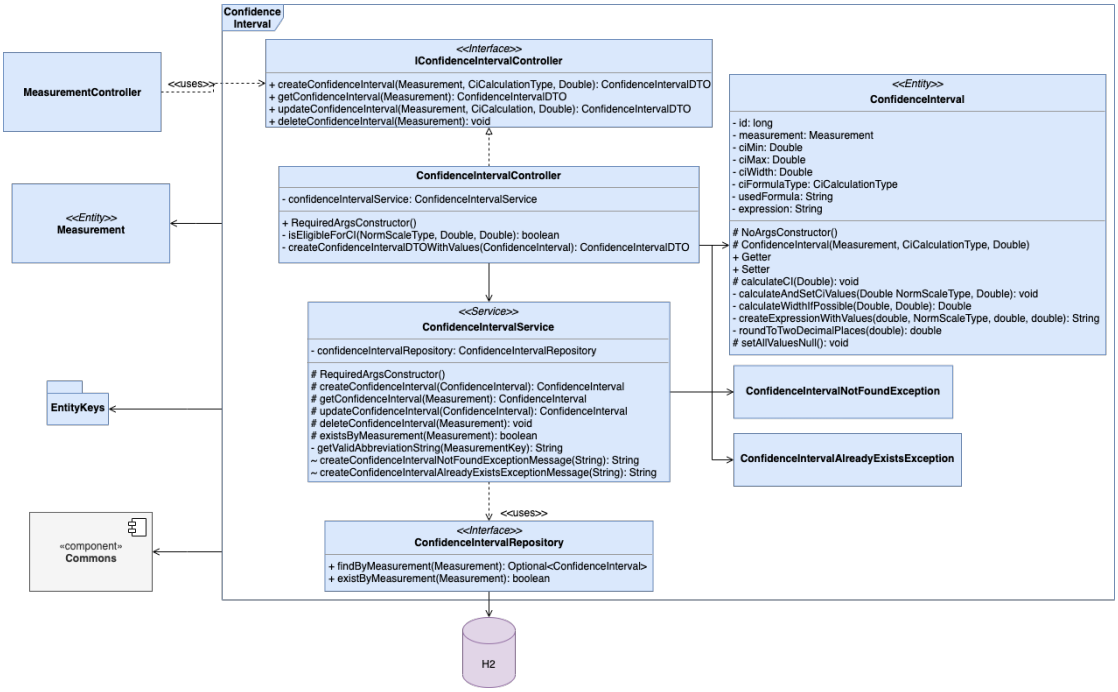


Abbildung 6.30: Whitebox Business-Logic.Persistent-Data.Confidence-Interval

6.6.3 Laufzeitsicht

Bei den Laufzeitsichten steht die Interaktion zwischen den verschiedenen Architektur-Komponenten des Systems im Fokus. Sie zeigt den Ablauf dieser Interaktionen, sowie welche Bausteine zur Laufzeit existieren (Vgl. [20]). Es wurden versucht sowohl die Szenarien der Anwendungsfälle bestmöglichst abzubilden als auch andere als wichtig erachtete Interaktions-Verläufe. Da der Fokus auf der Interaktion zwischen Bausteinen lag, wurden private Methoden innerhalb der Komponente nicht explizit dargestellt.
Die Diagramme der Laufzeitsicht wurden mit Hilfe des Open-Source Tools *sequencediagram.org* erstellt.

Starten der Anwendung

Um die Anwendung zu starten, müssen zunächst Spring Boot und JavaFX integriert werden. Dieses Vorgehen wurde anhand eines Beispiels aus einem Blog-Artikel umgesetzt ([12]).

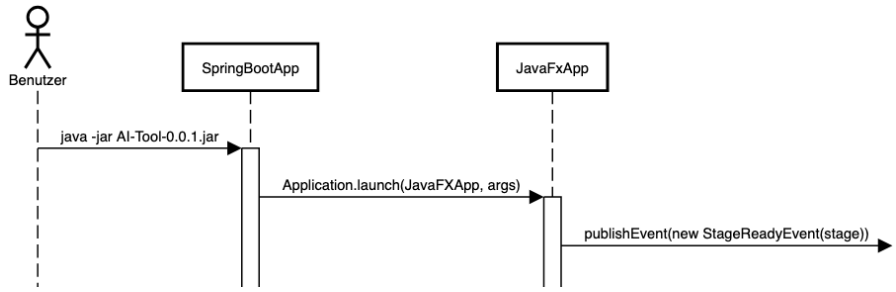


Abbildung 6.31: Sequenzdiagramm Programmstart

Die gesamte Anwendung führt beim Start der ausführbaren JAR-Datei über SpringBoot die einzige Main-Methode im Projekt aus. Diese befindet sich in der SpringBootApp-Klasse. Anstatt *SpringApplication.run(SpringBootApp.class)* wird jedoch die JavaFX-Applikation mit *Application.launch(JavaFxApp.class, args)* gestartet. Um SpringBoot mit dem JavaFx-Lifecycle zu verbinden, wird die *init()*-Methode der JavaFxApp-Klasse überschrieben und verwendet, um Spring zu initialisieren. Außerdem wird die *stop()*-Methode überschrieben, um sicherzustellen, dass beide Applikationen gemeinsam schließen.
Beim Start der JavaFxApp von Spring wird ein *StageReadyEvent* veröffentlicht. Dies ermöglicht es Spring-Komponenten, welche an diesem Event interessiert sind, über einen *Listener* davon informiert zu werden und auf den Applikations-Kontext der mitgegebenen *Stage* zuzugreifen.

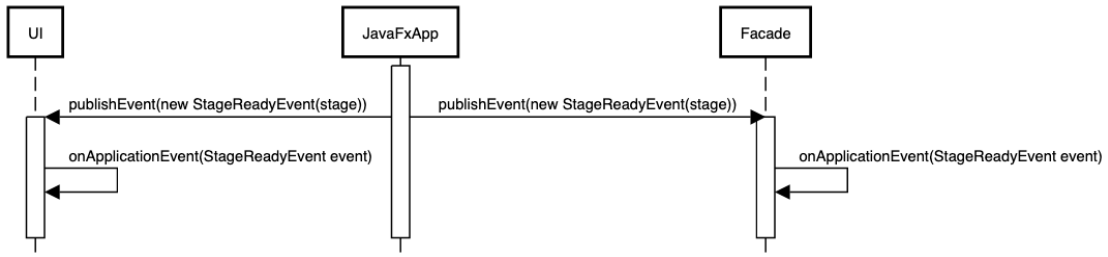


Abbildung 6.32: Sequenzdiagramm StageReadyEvent-Listener

In der Anwendung existieren genau zwei solcher *Listener*, in jeweils einer Klasse der UI- und Facade-Komponente. Durch diese Klassen und dem mitgereichten Applikations-Kontext, werden das Front- und Backend mit allen dafür nötigen Klassen und Komponenten gestartet.

- **Backend** Um das Backend zu starten, wird die *Facade*-Klasse über den Start der Applikation mit Hilfe des StageReadyEvents informiert. Entsprechend wird eine Facade-Instanz von Spring gestartet. Durch eine *@Autowired*-Annotation kann die UI-Komponente Anfragen an die erstellte Instanz über das *IFacade*-Interface schicken.

  Da die Facade zur Koordination die Schnittstellen aller Backend-Services benötigt, werden entsprechende Instanzen, welche diese Schnittstellen implementieren, von Spring initialisiert. Durch eine Art Kettenreaktion werden dadurch auch alle benötigten Variablen der Services usw. als entsprechende Instanzen von Spring gestartet.
  Mit dem StageReadyEvent stößt die *Facade*-Klasse die inhaltliche Initialisierung der Import-Data- und Export-Data-Subkomponenten des Backends an.
  Die *Export-Data*-Subkomponente setzt lediglich den übergeordneten Ordner der Anwendung als Default Export-Ordner. Die *Import-Data*-Subkomponente hingegen setzt bei der inhaltlichen Initialisierung ihre Default Variablen mit Hilfe der *DefaultImportConfig*, versucht diese Dateien anhand Methoden der *FileUtil*-Hilfsklasse zu erfassen und startet den Import der Testverfahren.
  Das Backend ist somit bereit und wartet auf Anfragen vom Frontend.

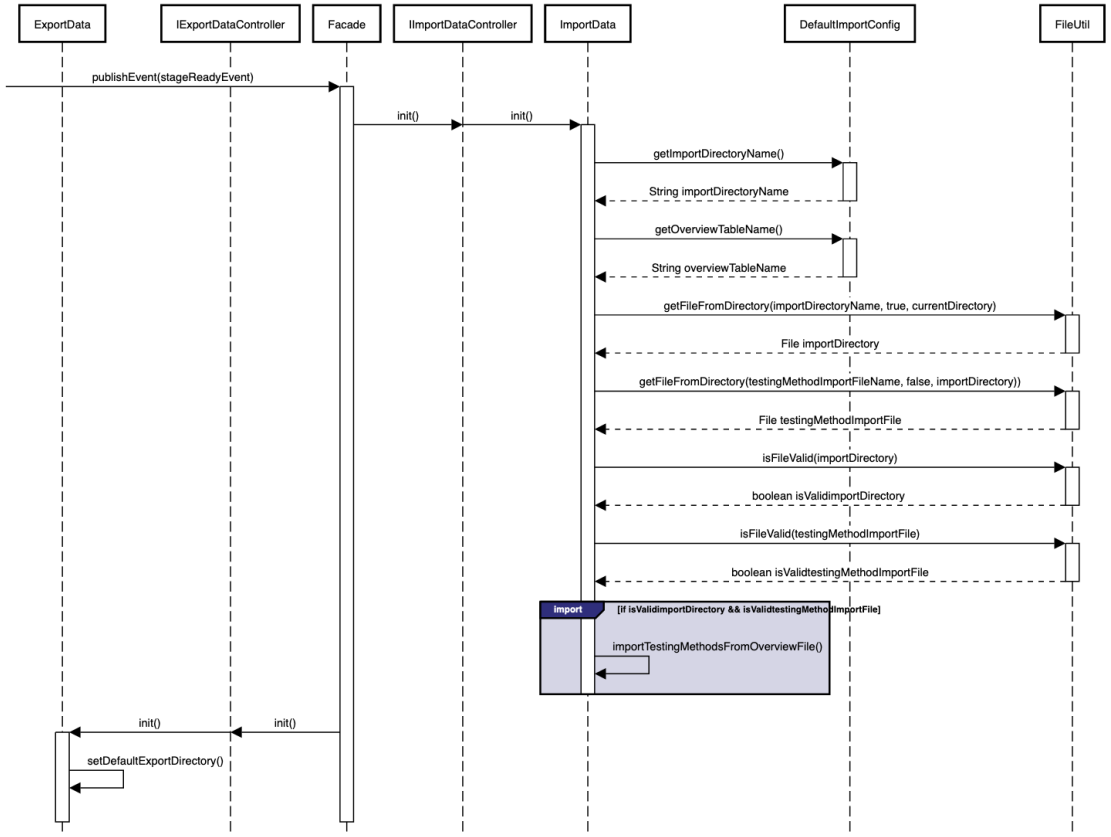


Abbildung 6.33: Sequenzdiagramm Start Facade

- **Frontend**

  In der UI-Komponente wird die *OverlayController*-Klasse über den Start der Applikation informiert und Instanzen aller benötigter Variablen von Spring initialisiert. Es wird mit der *View-Support*-Klasse interagiert, welche relevante Darstellungsinformationen sowie weitere Hilfsklassen beinhaltet, die ebenfalls instanziiert werden. Durch den *OverlayCreator* werden die *Overlays* und deren *Presenter* erstellt und können anschließend paarweise im *OverlayController* registriert werden. Zuletzt werden der **View** über das zugehörige Interface die relevanten Darstellungs-Informationen zur Verfügung gestellt. Die View kann anhand dieser Informationen aufgebaut werden und aktiviert das erste Overlay. Für den Benutzer öffnet sich ein Fenster und das erste Overlay ist sichtbar.

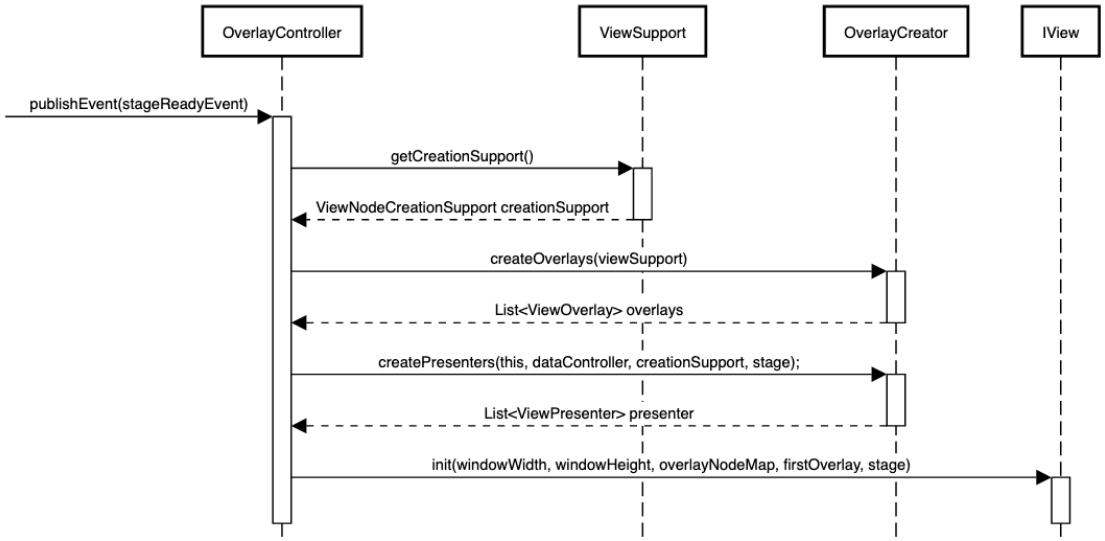


Abbildung 6.34: Sequenzdiagramm Start UI

Fall anlegen

- **Frontend**

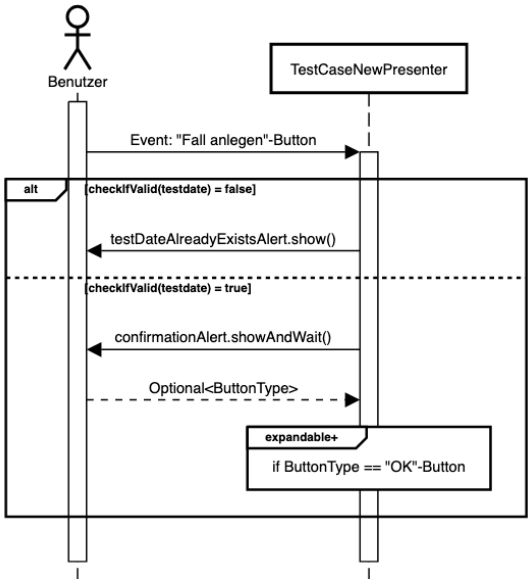


Abbildung 6.35: Sequenzdiagramm Fall anlegen Benutzer-Interaktion

Um einen Fall im System anzulegen, muss dies von dem Benutzer über die GUI initiiert werden. Zuständig für das entsprechende Darstellungs-Overlay ist die *TestCaseNewPresenter*. Wird von dem Benutzer durch das Klicken des *Fall-anlegen*-Buttons ein *ActionEvent* ausgelöst, reagiert der Presenter und prüft die Eingaben des Benutzers. Ist bei der Person bereits ein Fall an dem gewählten Testdatum angelegt, erhält der Benutzer einen Alert, der ihn darauf hinweist. Von dem System werden daraufhin keine weiteren Aktionen initiiert.
Ist das Testdatum hingegen valide, wird der Benutzer mit einem Alert dazu aufgerufen, das Anlegen des Falls zu bestätigen. Wird das Anlegen abgebrochen, werden keine weiteren Aktionen initiiert.
Bestätigt der Benutzer das Hinzufügen des Falls, werden die angegebenen Variablen mit der Anfrage *addNewTestCase* in Form eines TestCaseDTO an das *IViewDataController*-Interface übergeben. Dieses fügt zusätzlich benötigte Parameter hinzu und leitet die Anfrage über die Schnittstelle *IFacade* an das Backend weiter.

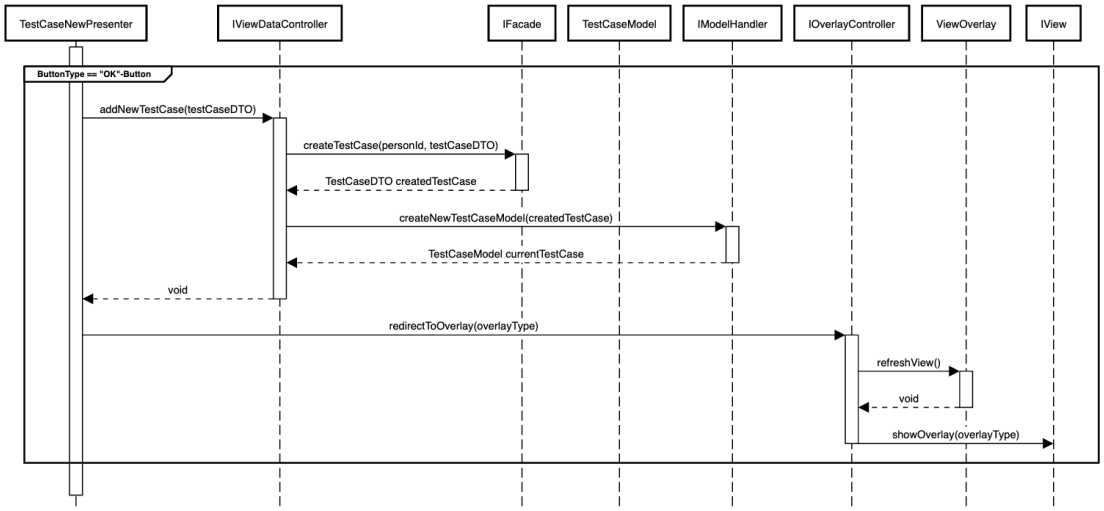


Abbildung 6.36: Sequenzdiagramm Fall anlegen UI-Sicht

Die Antwort des Backends wird anschließend an die *IModelHandler*-Schnittstelle weitergeleitet, um eventuelle Änderungen in einem Model-Objekt zu integrieren und dieses lokal speichern zu können. Eine neue unveränderbare Instanz des *TestCaseModels* wird entsprechend angelegt und zurückgegeben.
Anschließend wird vom *TestCaseNewPresenter* ein Overlay-Wechsel über die *IOverlayController*-Schnittstelle zu dem mitgegebenen *OverlayType* initiiert. Das ent-

sprechende ViewOverlay wird aktualisiert und die View über dessen Interface *IView* angeleitet, dieses darzustellen.

- **Backend (Facade)**

  Aus Sicht der Facade, wird die Komponente über das angebotene *IFacade*-Interface von der UI-Komponente angesprochen. Die Facade fordert von dem Backend über das *IImportDataController*-Interface die importierten Skalen an und erhält diese als eine Liste mit MeasurementDTO-Objekten zurück. Die Facade leitet diese mit den übergebenen Parametern zur Fallerstellung weiter an den Service des Backends zum Umgang persistent gespeicherter Daten über das Interface *IPersistentDataController*. Die Informationen zu dem angelegten Fall werden der Facade zurückgegeben. Diese gibt das resultierende *TestCaseDTO*-Objekt wiederum ohne weitere Prüfungen an die UI-Komponente zurück.

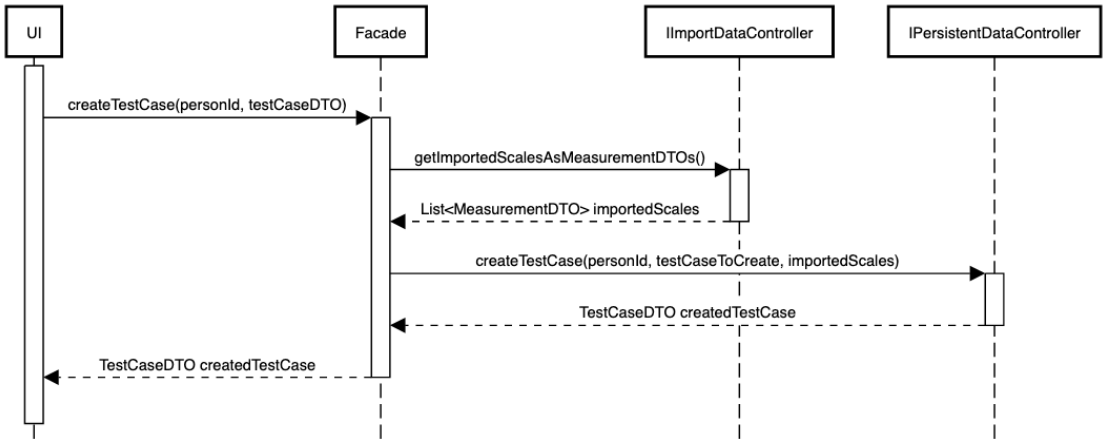


Abbildung 6.37: Sequenzdiagramm Fall anlegen Facade-Sicht

- **Backend (Business-Logic)**

  Aus Sicht der *Business-Logic*-Komponente, wird zunächst die *ImportData*-Klasse vom Interface über dessen Schnittstelle angesprochen und gibt die zur Laufzeit importierten Skalen im Listen-Format an die Facade-Komponente zurück.
  Anschließend wird die *PersistentData*-Klasse der gleichnamigen Business-Logic-Subkomponente über dessen Interface angesprochen und zur Fallerstellung aufgefordert. Diese prüft die mitgegebenen Schlüsselattribute und wendet sich zunächst an den *TestCaseController* zur Erstellung des Fall-Objektes. Der Controller überführt die Informationen aus dem Transport-Objekt in ein Objekt der Fall-Entität

TestCase und leitet dieses zum persistenten Speichern in der Datenbank an den *TestCaseService* weiter. Das zurückgegebene Entitäts-Objekt, wird wieder in ein Transport-Objekt umgewandelt und an die *PersistentData*-Klasse zurückgegeben. Basierend auf dem erhaltenen TestCaseDTO-Objekt und dessen Alpha und Test-Richtung wird mit Hilfe der *CiCalculationType*-Klasse eine entsprechende Instanz des fachlichen Datentyps erstellt. Anschließend wird die Erstellung der Messwert-Objekte des anzulegenden Falls initiiert. Dafür wird durch die mitgegebene MeasurementDTO-Liste iteriert, um die Objekte nacheinander zu erstellen. Bevor die Erstellung an die *MeasurementController*-Klasse delegiert wird, werden die Transportobjekte mit Informationen aus den importierten Skalen vervollständigt, sofern diese vorhanden sind. Der Controller gibt die erstellten Messwerte in Form von deren Transport-Typ *MeasurementDTO* an die *PersistentData*-Klasse zurück. Zuletzt werden die Informationen bezüglich der erstellten Messwerte dem TestCaseDTO-Objekt hinzugefügt, bevor dieses der Facade zurückgegeben wird.

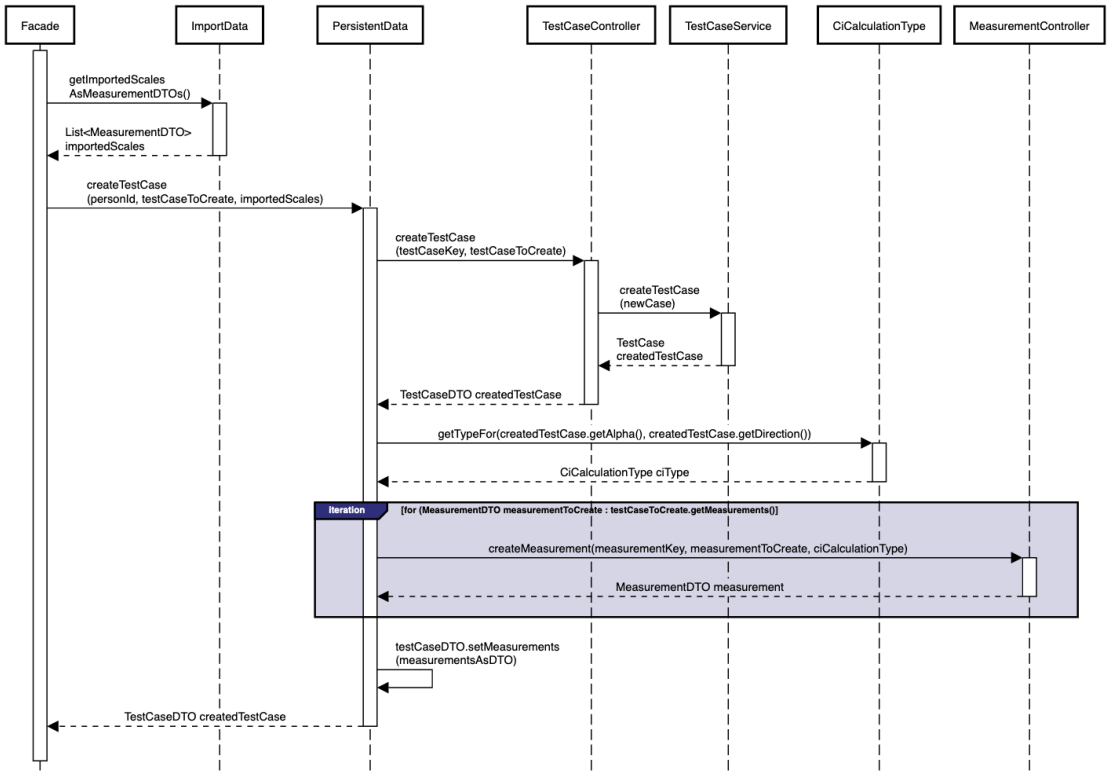


Abbildung 6.38: Sequenzdiagramm Fall anlegen BusinessLogic-Sicht

Innerhalb der Interaktion zu dem Erstellen der Messwerte wird zunächst wie beschrieben der Controller der *Measurement*-Komponente angesprochen. Dieser wandelt das Transport-Objekt in ein Objekt der Messwert-Entität um und initiiert dessen Speicherung durch das Weiterleiten an die entsprechende Service-Klasse. Die in der Datenbank hinzugefügte Entität wird dem Controller zurückgegeben.
Da die Entitäten der Reliabilität und des Konfidenzintervalls eine Referenz auf die zugehörige Messwert-Entität benötigen, werden die entsprechenden Controller der Komponenten von dem Messwert-Controller aus angesprochen und die soeben erstellte Messwert-Entität als Parameter übergeben. Nach dem jeweiligen Erstellen und Speichern der Objekte (sofern die Informationen dazu vorliegen), werden diese in ihrer Transport-Objekt-Form an den Controller der Measurement-Komponente zurückgegeben und von dieser in das MeasurementDTO integriert, bevor es wiederum der PersistentData-Klasse zurückgegeben wird. Dort werden alle erstellten MeasurementDTOs in einer Liste lokal gespeichert, um diese dem TestCaseDTO des angelegten Falls hinzuzufügen.

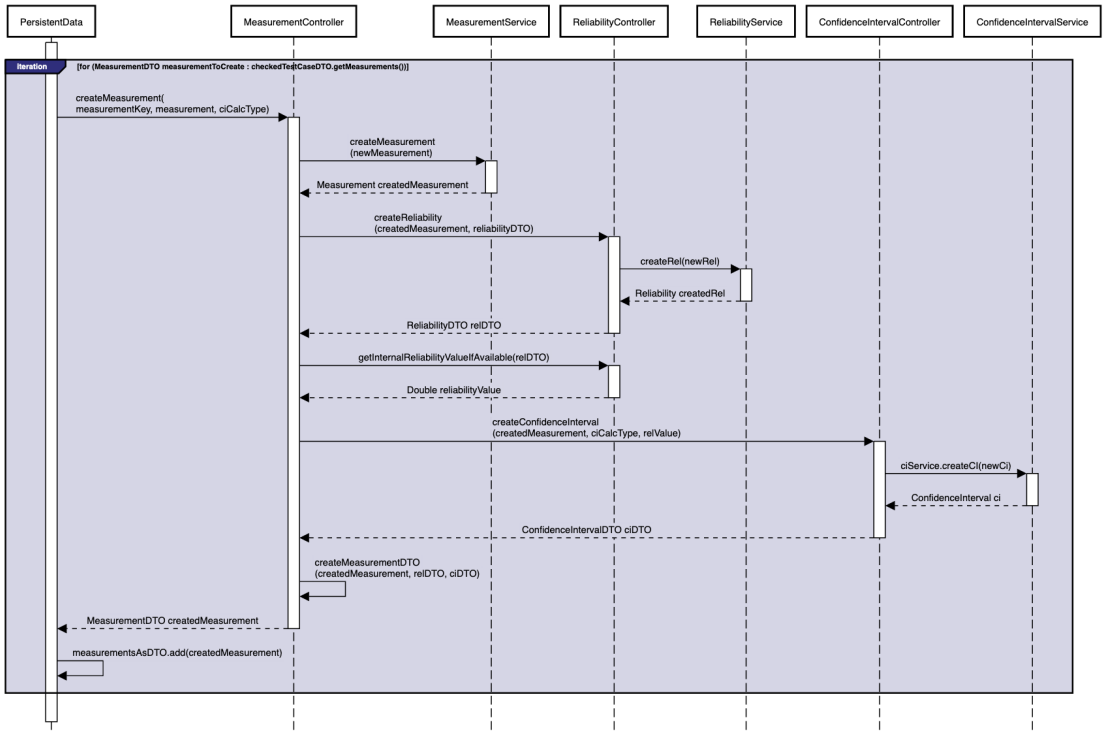


Abbildung 6.39: Sequenzdiagramm Fall anlegen BusinessLogic-Sicht (Messwert Iteration)

Der Datenbankzugriff der Entitäts-Service-Klassen erfolgt über das jeweilige *Repository*-Interface der Entität. Um ein neues Objekt der Fall-Entität in der Datenbank zu persistieren, wird zunächst von dem Service über die Schnittstelle zur Datenbank anhand der Schlüsselattribute (in *TestCaseKey*-Instanz festgehalten) überprüft, ob bereits ein Eintrag mit diesen Attributen existiert. Ist dies nicht der Fall, initiiert der Service das Speichern des *TestCase*-Objekts. Beim erfolgreichen Speichern des Objektes wird dieses dem Service zurückgegeben.

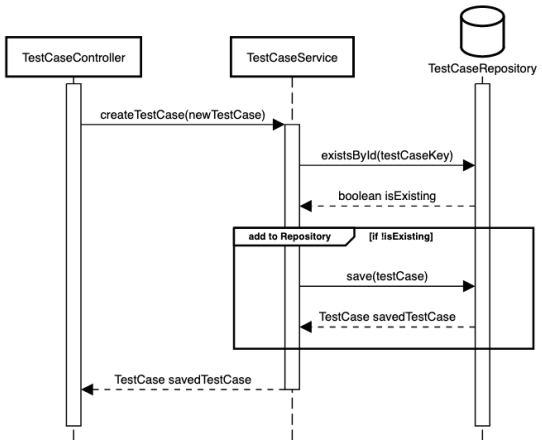


Abbildung 6.40: Sequenzdiagramm Fall anlegen TestCase DB-Zugriff

Das Anlegen einer Messwert-Entität, erfolgt analog mit den Schlüsselattributen in der *MeasurementKey*-Instanz. Dieser Vorgang wird auf Grunde der Iteration für jeden Messwert durchlaufen.

Das Persistieren der Reliabilität- und Konfidenzintervall-Entitäten erfolgt ebenfalls analog zu dem bereits vorgestellten Vorgehen. Allerdings wird anstelle von Schlüsselattributen das Messwert-Objekt selbst zur Identifikation bereits existierender Objekte verwendet.

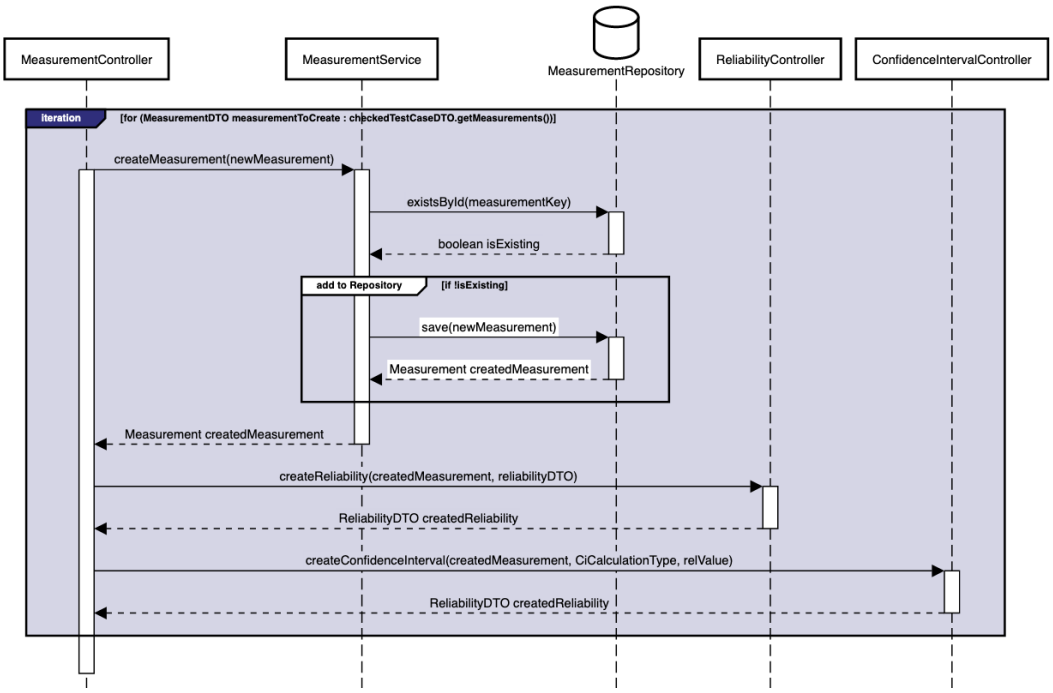


Abbildung 6.41: Sequenzdiagramm Fall anlegen Measurement DB-Zugriff

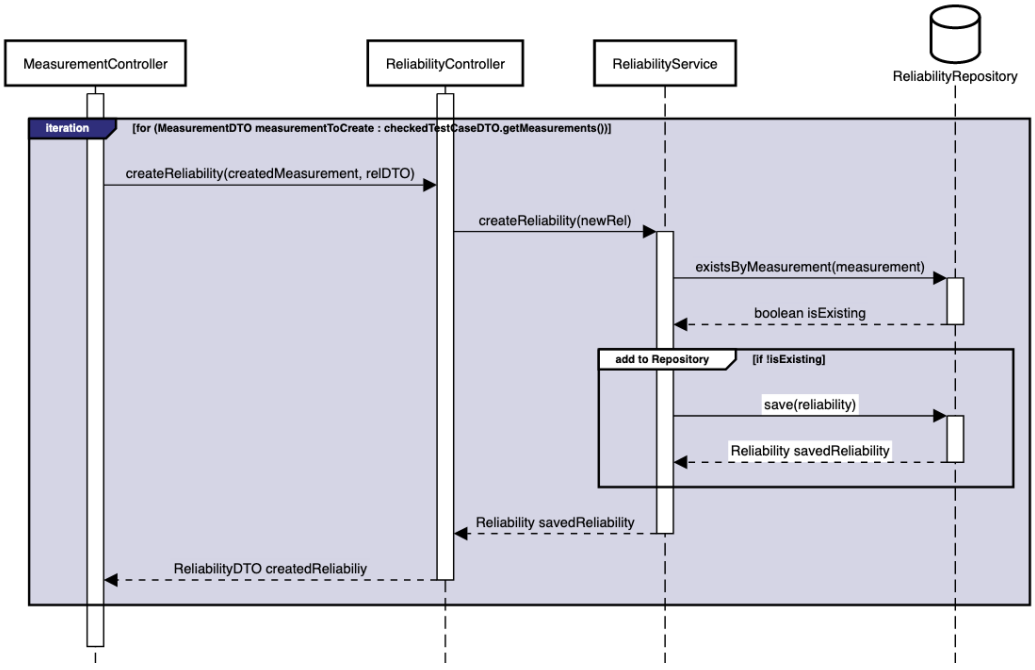


Abbildung 6.42: Sequenzdiagramm Fall anlegen Reliability DB-Zugriff

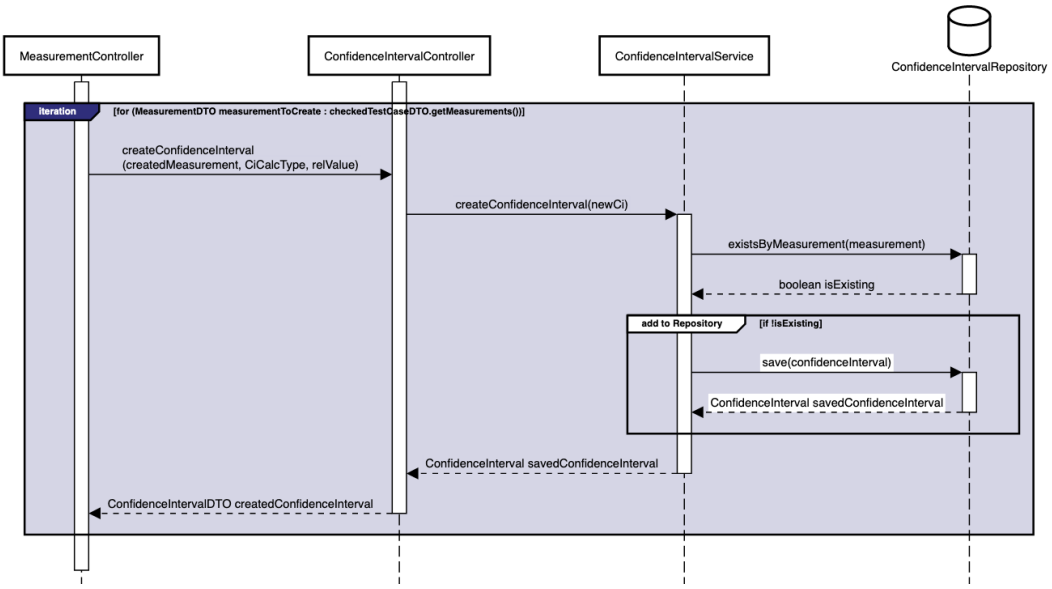


Abbildung 6.43: Sequenzdiagramm Fall anlegen ConfidenceInterval DB-Zugriff

Konfidenzintervalle darstellen

- **Frontend**

  Um die Konfidenzintervall-Grafik von einem angelegten Fall dargestellt zu bekommen, muss der Benutzer zur Übersicht des entsprechenden Falls navigieren. Dort kann durch das Klicken des *Zur Konfidenzintervall-Grafik*-Buttons ein *ActionEvent* in der UI-Komponente ausgelöst werden.

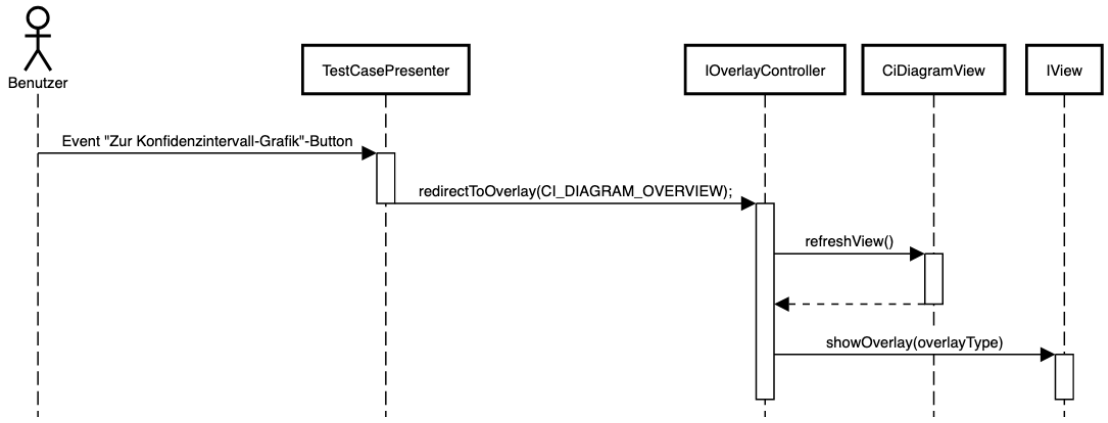


Abbildung 6.44: Sequenzdiagramm KI-Diagramm darstellen Benutzeraktion

Der Presenter des aktuellen Overlays (TestCasePresenter) reagiert auf das ausgelöste *ActionEvent*, indem die *IOverlayController*-Schnittstelle angesprochen wird, um einen Overlay-Wechsel zu initiiert. Das entsprechende Overlay wird von dem OverlayController aktualisiert und anschließend die View-Komponente über dessen Interface *IView* zu dem Wechsel des Overlay-Nodes aufgerufen.
Beim Aktualisieren des Overlays, wird die konkrete Instanz des entsprechenden ViewOverlays angesprochen, die den dazugehörigen Presenter erneut initialisiert.

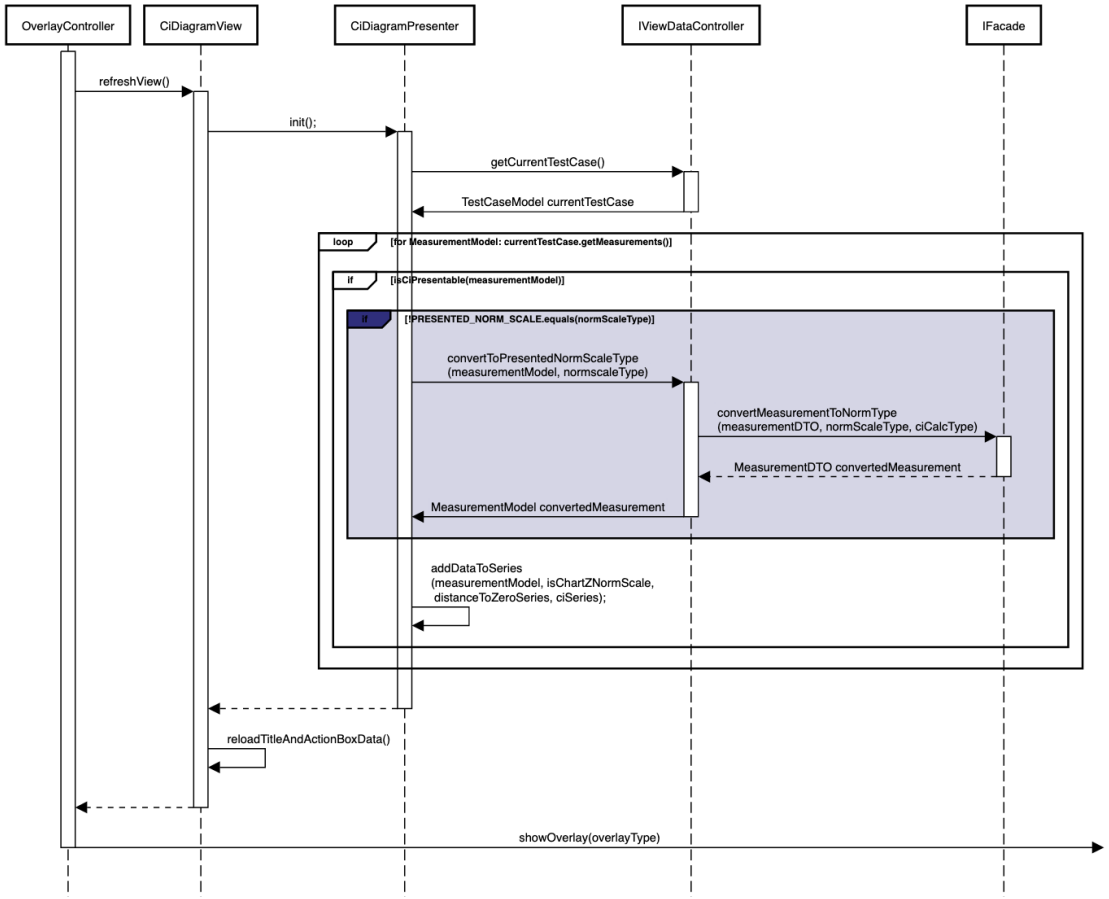


Abbildung 6.45: Sequenzdiagramm KI-Diagramm darstellen Frontend-Sicht

Beim Initialisieren des Presenters, wird die *IViewDataController*-Schnittstelle angesprochen, um die Darstellungsinformationen (TestCaseModel) des aktuell ausgewählten Falls zu erhalten. Diese Messwerte des Falls werden iteriert und jeweils geprüft, ob für den vorliegenden Messwert ein Konfidenzintervall zu berechnen ist. Da in der Grafik alle Messwerte gemeinsam auf eine Normskala dargestellt werden

sollen, müssen diese vor dem Hinzufügen in das Diagramm vereinheitlicht werden. Dafür wird die Normskala mit dem des Diagramms verglichen und bei einer Abweichung eine Transformation des Normwerts initiiert. Dies wird erneut über die *IViewDataController*-Schnittstelle realisiert.

- **Backend**

  Aus Sicht des Backends wird die Facade von der UI-Komponente mit der Transformations-Anfrage angesprochen. Diese delegiert die Anfrage an den zuständigen Service *INormConverterController*, bei dem der Normwert transformiert und das Konfidenzintervall auf Basis der neuen Werte erneut berechnet wird. Da keine explizite Veränderung des NormSkalenTyps von dem Benutzer initiiert wurde, soll die Transformation der Werte nicht die persistent gespeicherten Daten betreffen. Daher wird das MeasurementDTO mit den durchgeführten Änderungen von der Facade direkt zur UI-Komponente zurückgegeben.

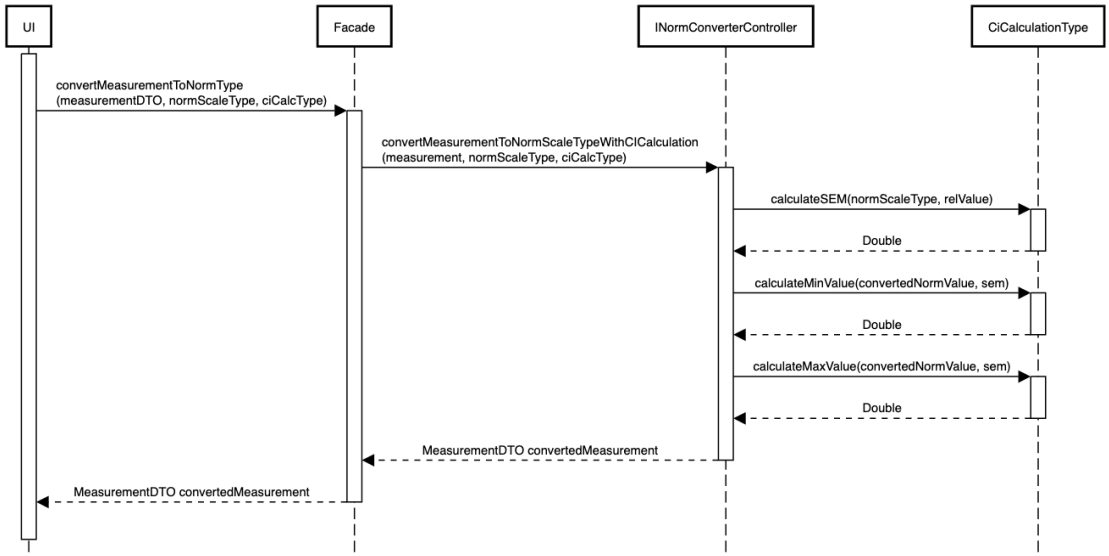


Abbildung 6.46: Sequenzdiagramm KI-Diagramm darstellen Backend-Sicht

Fall exportieren

- **Frontend**

  Um einen angelegten Fall zu exportieren, muss der Benutzer ebenfalls zur Übersicht des entsprechenden Falls navigieren. Der Prozess kann dort durch das Klicken des

Fall exportieren-Buttons ausgelöst werden, wodurch ein *ActionEvent* in der UI-Komponente erstellt wird.

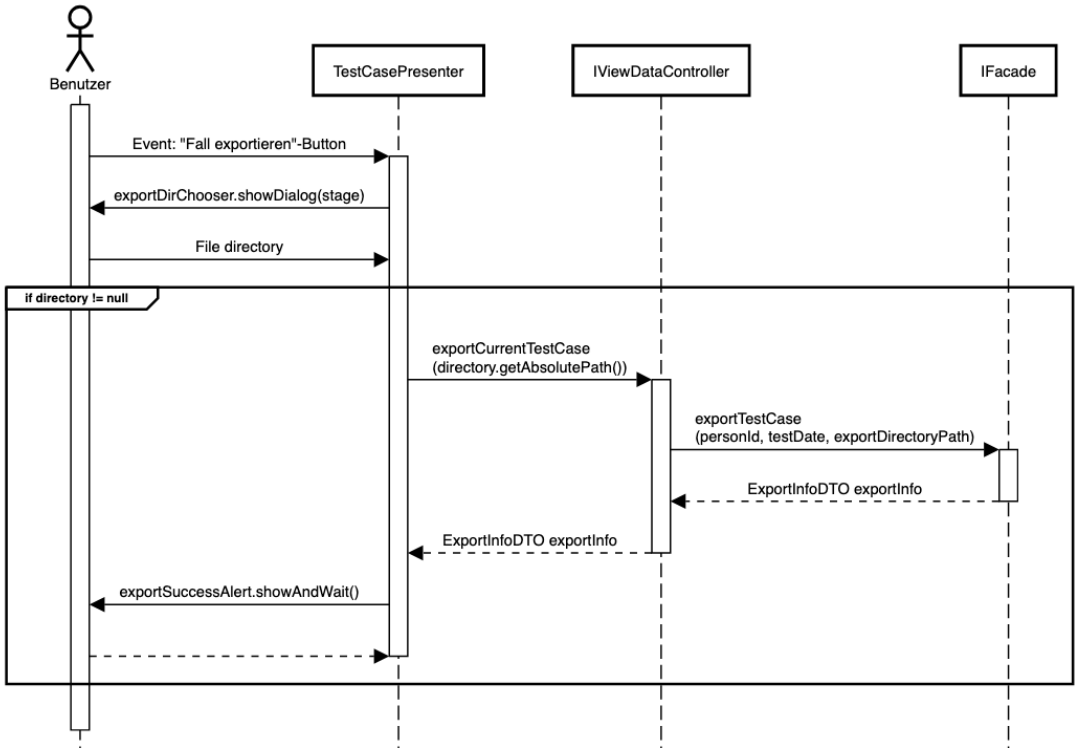


Abbildung 6.47: Sequenzdiagramm Fall exportieren UI-Sicht

Der Presenter des aktuellen Overlays (TestCasePresenter) reagiert auf das ausgelöste *ActionEvent*, indem dem Benutzer ein Fenster zur Auswahl des gewünschten Export-Ordners angezeigt wird. Wurde ein Ordner ausgewählt, wird über die *IViewDataController*-Schnittstelle das Exportieren der aktuellen Falldaten mit dem vom Benutzer ausgewählten Export-Pfad initiiert. Die Anfrage wird mit dem Hinzufügen zusätzlicher und relevanter Informationen an das Backend über die Facade delegiert. Da durch diese Anfrage keine Veränderungen an den darzustellenden Daten hervorgerufen wurde, wird das erhaltene ExportInfoDTO an den Presenter weitergegeben. Dieser entnimmt die relevanten Informationen bezüglich des Datei-Namens und tatsächlichen Exportorts und teilt sie dem Benutzer über einen Bestätigungshinweis mit.

- **Backend (Facade)**

Aus Sicht der Facade erhält diese eine Anfrage von der UI-Komponente zum Exportieren des angegeben Falls und dem Export-Pfad.

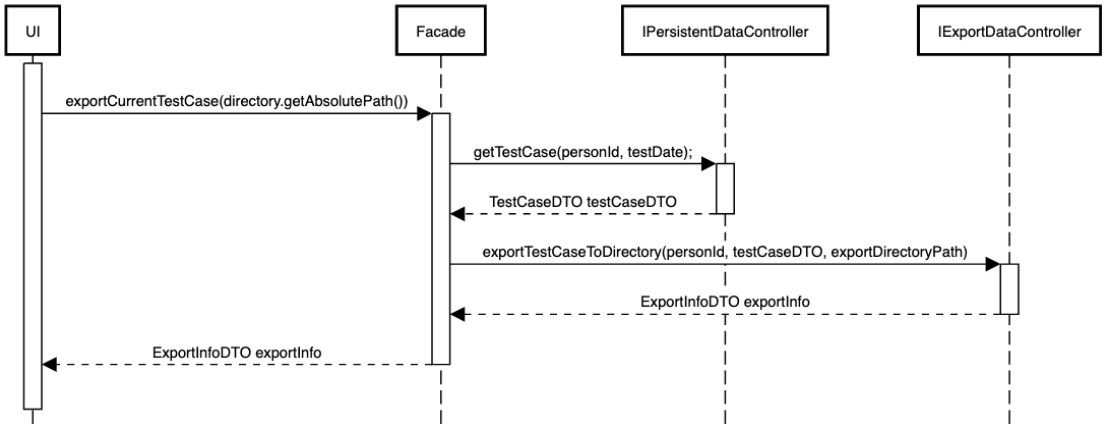


Abbildung 6.48: Sequenzdiagramm Fall exportieren Facade-Sicht

Diese holt den persistent gespeicherten Fall über das *IPersistentDataController*-Interface und übergibt diesen zusammen mit dem Export-Pfad an die *IExportDataController*-Schnittstelle. Das zurückerhaltene ExportInfoDTO wird von der Facade direkt weitergeleitet und an die UI-Komponente zurückgegeben.

- **Backend (Business-Logic)**

  Aus der Sicht der Business-Logic-Komponente, wird zunächst von der Facade die *Persistent-Data*-Subkomponente angesprochen, um die Informationen des persistent gespeicherten Falls zu erlangen. Dafür wird die gleichnamige Klasse angesprochen, welche zunächst den Controller der *TestCase-Komponente* anspricht. Dieser prüft die Eingaben und wendet sich wiederum an die Service-Klasse der Komponente, um die Entität aus der Datenbank zu lesen. Die Entität wird dem Controller zurückgegeben, der diese in Form des Transport-Objekts an die *Persistent-Data*-Klasse zurückgibt.

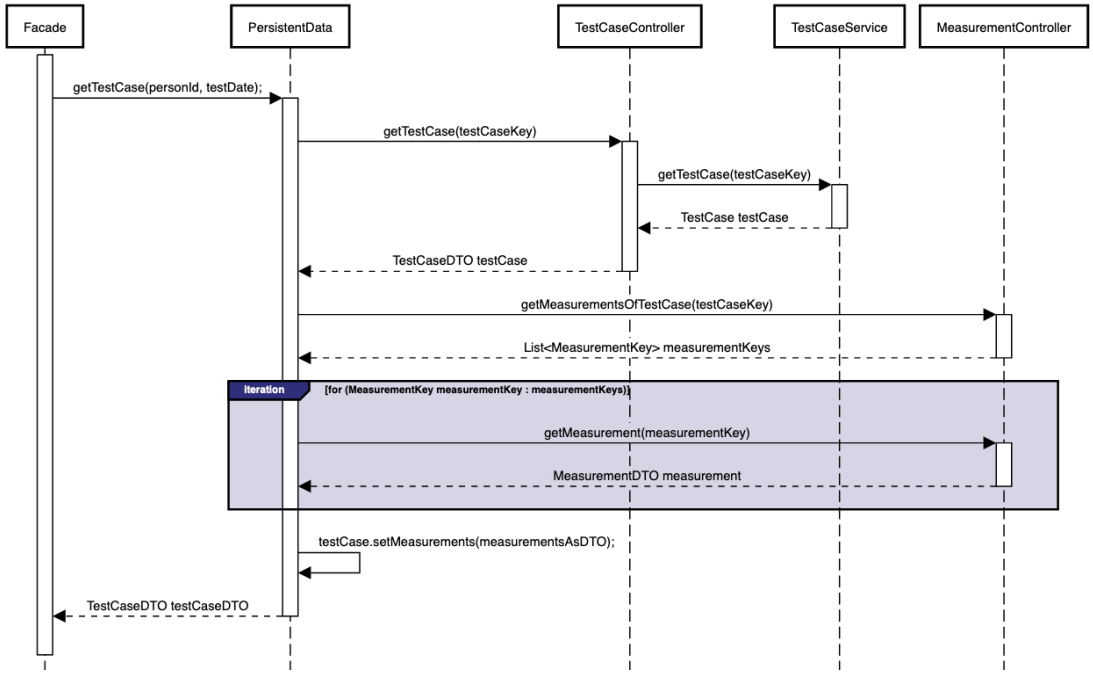


Abbildung 6.49: Sequenzdiagramm Fall exportieren BusinessLogic-Sicht Persistent-Data

Anschließend wendet sich diese an den Controller der *Measurement*-Komponente, um die Informationen zu den zum Fall dazugehörigen Messwert-Entitäten erlangen. Dafür wird zunächst eine Liste der Objekt-Schlüssel angefragt, um diese anschließend iterativ durchzugehen und die Informationen zu den Entitäten aus der Datenbank zu erlangen. Anschließend werden diese Informationen denen des Falls hinzugefügt, bevor dieser der Facade zurückgegeben werden.

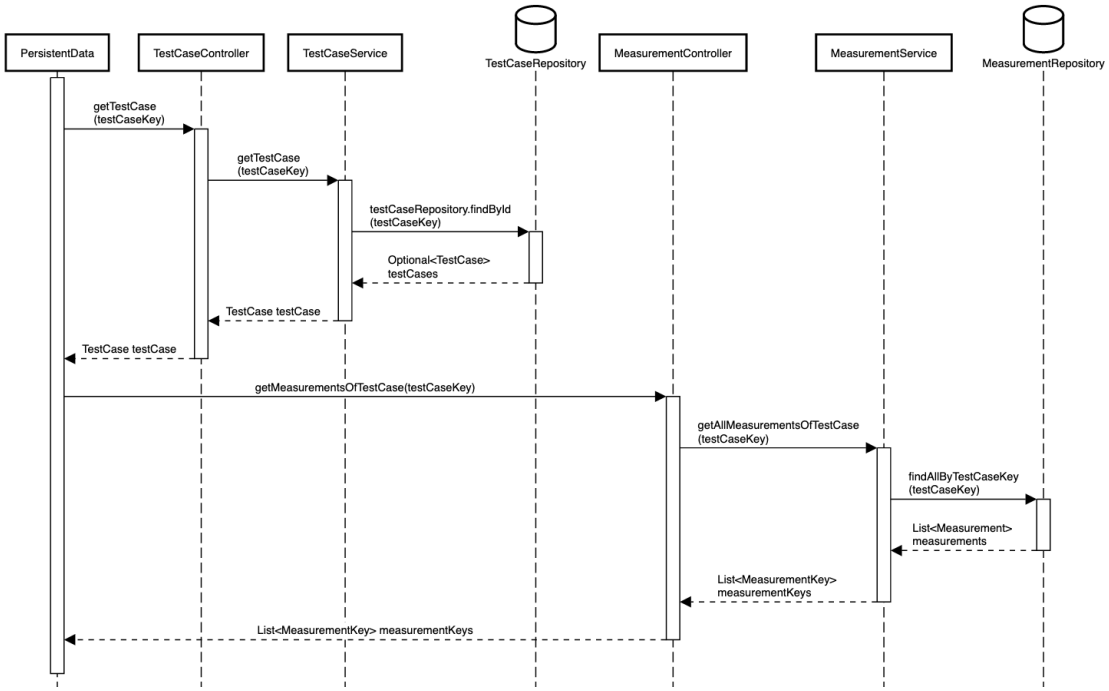


Abbildung 6.50: Sequenzdiagramm Fall exportieren TestCase DB-Zugriff

Bei der Iteration wird zunächst der Service der *Measurement*-Komponente angesprochen, um die entsprechende Entität aus der Datenbank zu lesen.

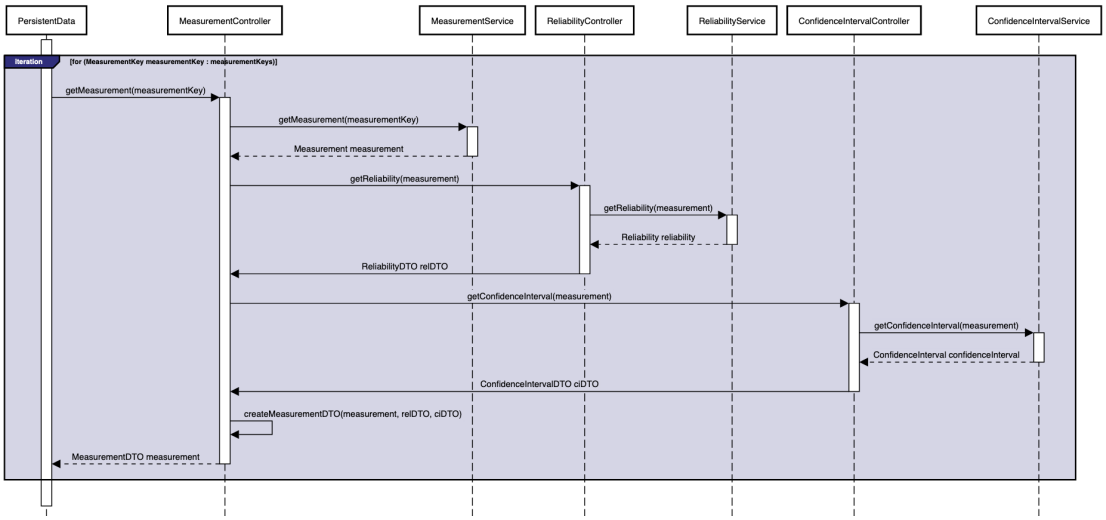


Abbildung 6.51: Sequenzdiagramm Fall exportieren BusinessLogic-Sicht Persistent-Data (Messwert Iteration)

Anschließend wendet sich der MeasurementController nacheinander an die Controller der *Reliability*- und *ConfidenceInterval*-Komponente, um die Informationen der jeweilig dazugehörigen Entitäten der Komponenten zu den Messwert-Informationen hinzuzufügen, bevor das entsprechende MeasurementDTO-Objekt zurückgegeben wird.

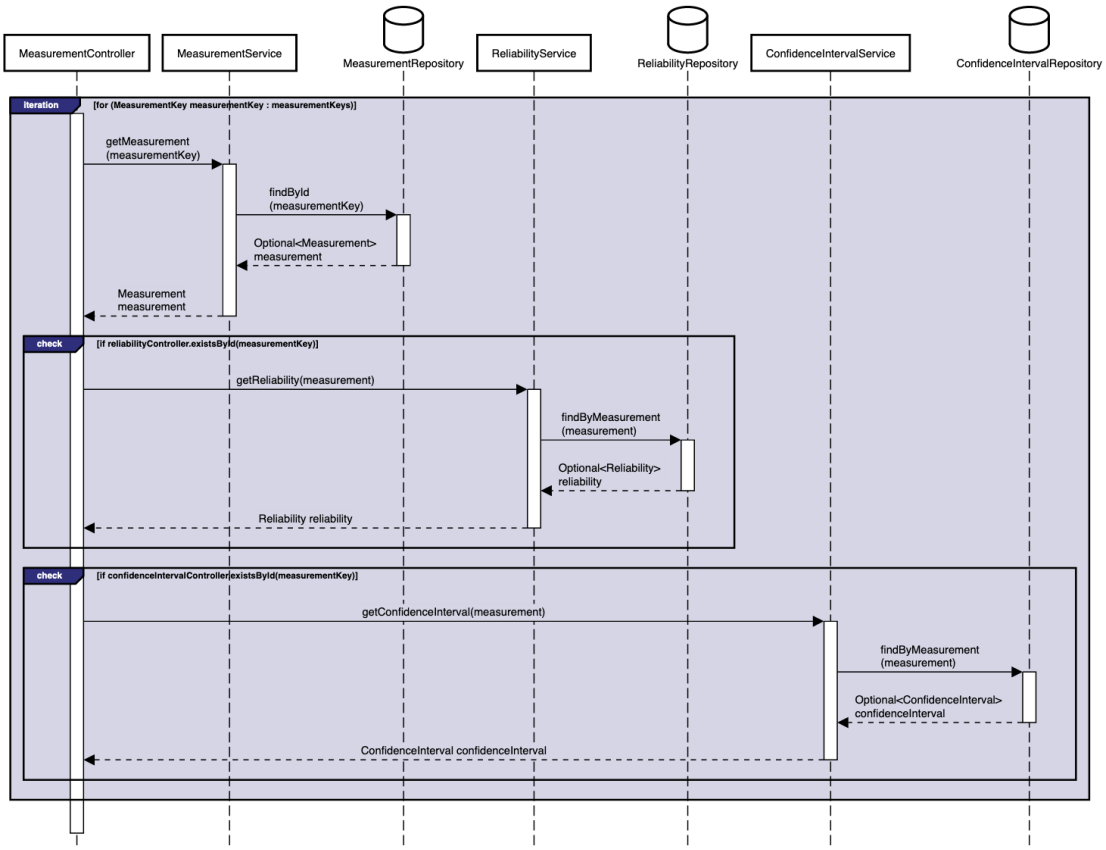


Abbildung 6.52: Sequenzdiagramm Fall exportieren Measurement DB-Zugriff

Nachdem die Facade die persistent gespeicherten Informationen zu dem Fall erhalten hat, wird diese zusammen mit dem Export-Pfad an die *ExportData*-Subkomponente der *Business-Logic*-Komponente weitergegeben. In der zuständigen Klasse wird mit Hilfe der *FileUtil*-Hilfsklasse der Export-Ordner ausfindig gemacht und die Fall-Informationen in eine Excel-Datei geschrieben. Anschließend werden die relevanten Export-Informationen in ein Transport-Objekt eingefügt und der Facade zurückgegeben.

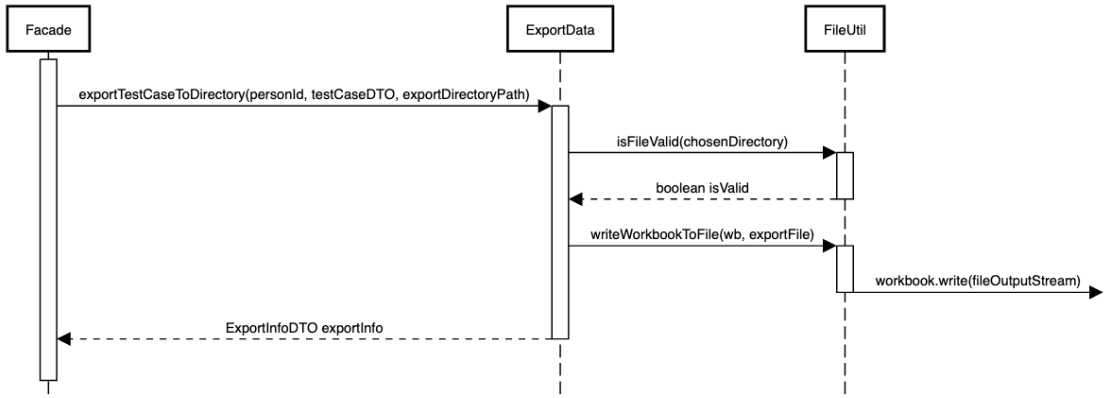


Abbildung 6.53: Sequenzdiagramm Fall exportieren BusinessLogic-Sicht Export-Data

6.7 Risikomanagement

Um die Qualität und Vorhersagbarkeit des Projektes während der Entwicklung zu verbessern, wurden laufend potenzielle Probleme identifiziert, gesammelt und bewertet. Der Prozess dient der Kontrolle von Risiken, die mit dem Projekt zusammenhängen. Die Bewertung orientierte sich an der Eintrittswahrscheinlichkeit und der Schwere der Folgen (Risiko = Eintrittswahrscheinlichkeit x Auswirkungen) (Vgl. [9]).

6.7.1 Unzureichende Ressourcen

Identifikation	Manpower_Risiko
Art	Wirtschafliches Risiko
Beschreibung	Es stehen nicht genügend Neuropsychologen an sich oder nur mit sehr beschränkter Zeit zur Verfügung
Folgen	Fragen können nicht geklärt werden, Implementierung zieht sich in die Länge, eventuell entstehen Fehler
Eintrittswahrscheinlichkeit	mittel
Auswirkungen	mittel
Priorität	4
Maßnahmen	Literatur zum Nachlesen suchen und Aufgaben anders einteilen

Tabelle 6.28: Risikoeinschätzung Manpower_Risiko

6.7.2 Programm auf den Krankenhaus-Rechnern deployen

Identifikation	Deploy_Risiko
Art	Technisches-& Implementierungs-Risiko
Beschreibung	IT-Security-Infrastruktur externer Firma könnte das Programm am Laufen hindern oder einschränken
Folgen	Programm kann nur eingeschränkt oder garnicht benutzt werden
Eintrittswahrscheinlichkeit	hoch
Auswirkungen	hoch
Priorität	9
Maßnahmen	Mit dem IT-Team des Krankenhauses in Kontakt treten und nach Unterstützung fragen

Tabelle 6.29: Risikoeinschätzung Deploy_Risiko

6.7.3 Umgang mit Excel Macros (Programmecode in Zelle)

Identifikation	Data_Injection_Risiko
Art	Implementierungs-Risiko
Beschreibung	Der Inhalt einer Excel-Zelle kann ein Befehl oder ähnliches sein
Folgen	Befehl könnte unbewusst ausgeführt werden
Eintrittswahrscheinlichkeit	gering
Auswirkungen	mittel
Priorität	2
Maßnahmen	eingelesene Daten als String über die *format*- Methode aufnehmen

Tabelle 6.30: Risikoeinschätzung Data_Injection_Risiko

6.7.4 Dateien, die eingelesen werden auf Manipulation prüfen

Identifikation	Read_File_Risiko
Art	Implementierungs-Risiko
Beschreibung	Eine einzulesende Datei kann manipuliert worden sein, wodurch die Daten garnicht oder fehlerhaft im Programm aufgenommen werden
Folgen	Programm arbeitet mit feherhafte Dateien oder ist in der Funktion stark eingeschränkt
Eintrittswahrscheinlichkeit	gering
Auswirkungen	mittel
Priorität	2
Maßnahmen	Dateien vor dem einlesen auf Manipulation checken

Tabelle 6.31: Risikoeinschätzung Read_File_Risiko

7 Implementierung

7.1 Konfidenzintervall-Diagramm Darstellung

Das Konfidenzintervall-Diagramm bildet alle Messwerte ab, von denen ein Konfidenzintervall berechnet werden konnte und transformiert die Normwerte auf eine einheitliche Skala, um diese in einem Übersichtsdiagramm darzustellen. Auf der Y-Achse werden die verwendeten Messwerte als Kategorien aufgeführt. Auf der X-Achse des Diagramms werden die Werte der Normskala abgebildet. Das Diagramm bildet immer eine Range von drei Standardabweichungen über und unter dem Mittelwert der ausgewählten Normskala ab.

7.1.1 Problemstellung

Nach einer Teambesprechung der Neuropsychologen wurde entschieden, dass das KI-Diagramm eines Falls in der z-Skala abgebildet werden soll. Als Orientierung für das Diagramm wurde eine Excel-Datei genommen, mit der die Neuropsychologen bisher arbeiten.

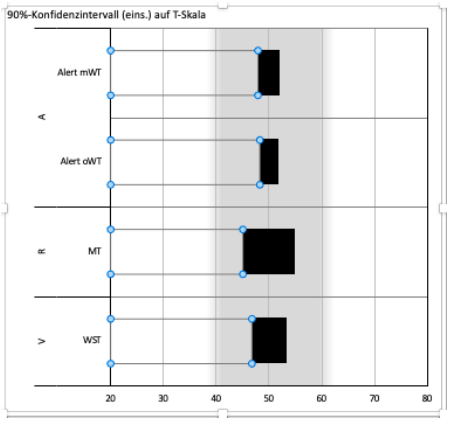


Abbildung 7.1: KI-Diagramm aus der Excel-Datei

Das bisherige Diagramm der Excel-Datei ist auf der T-Skala abgebildet. Die T-Normskala hat einen Mittelwert von 50 und eine Standardabweichung von 10, daher bildet die X-Achse Werte von 20 (50-3x10 = 20) bis 80 (50 + 3x10 = 80) ab.
Das Diagramm basiert auf einem Stacked-Bar-Chart und ist folgendermaßen aufgebaut:

1. Als erste Serie bzw. 'Bar' wird die Untergrenze des KI-Intervalls angegeben, die Farbe auf 'transparent' gesetzt und soll die Distanz zu 0 repräsentieren

2. Als zweite Serie bzw. 'Bar' wird die Breite des KI-Intervalls angegeben und in schwarzer Farbe auf die erste Bar draufgesetzt (stacked), dieser Wert repräsentiert das tatsächliche Konfidenzintervall

Dies ließ sich ohne Probleme mit JavaFX und dem dort zur Verfügung stehenden *StackedBarChart (javafx.scene.chart.StackedBarChart)* auf der T-Skala umsetzen. Bei der Testung fiel jedoch auf, dass dieses Vorgehen auf alle anderen Normskalen abgesehen von der z-Skala anzuwenden ist.
Der Grund hierfür ist, dass die z-Skala die einzige der aufgenommenen Normskalen ist, welche negative Werte annehmen kann.
(Mittelwert = 0 und Standardabweichung = 1, also eine Range von -3 bis +3 abbildet)

Entsprechend gibt es drei Szenarien, in die unterschieden werden kann:

- **1. Szenario:** KI ist im negativen Bereich
- **2. Szenario:** KI geht vom negativen in den positiven Bereich
- **3. Szenario:** KI ist im positiven Bereich

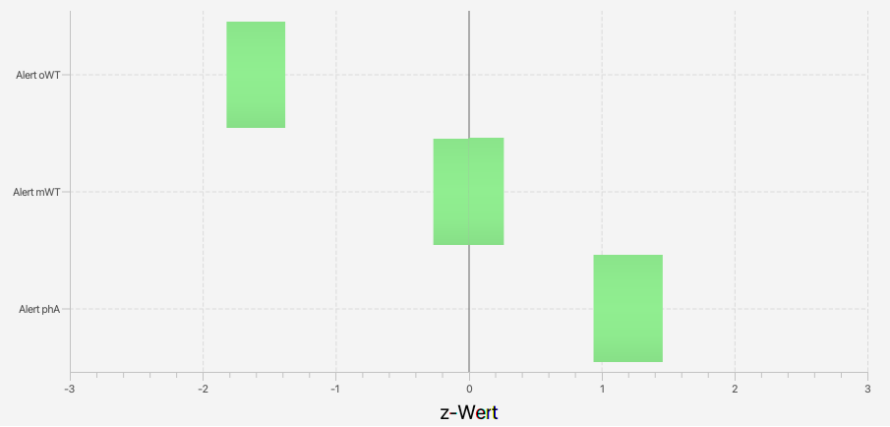


Abbildung 7.2: KI-Diagramm Szenarien auf z-Skala

Die Implementierung des JavaFX-StackedBarChart scheint zu sein, dass die erste *Bar* vom 0-Wert ausgehend bis zum angegebenen Wert abgebildet wird unabhängig, ob dies ein positiver oder negativer Wert ist. Da es sich um ein *Stacked*-Bar-Chart handelt, werden alle folgenden *Bars* auf dem vorherigen aufgebaut. Je nachdem, ob der Wert der zweiten *Bar* größer oder kleiner ist als der erste, wird die *Bar* also ins Negative oder positive *gestapelt*.
Um das Diagramm von oben umzusetzen, müssen die *Bars* wie in folgender Abbildung aufgebaut sein. Die erste Datenserie wurde hier rot eingefärbt, die zweite grün.

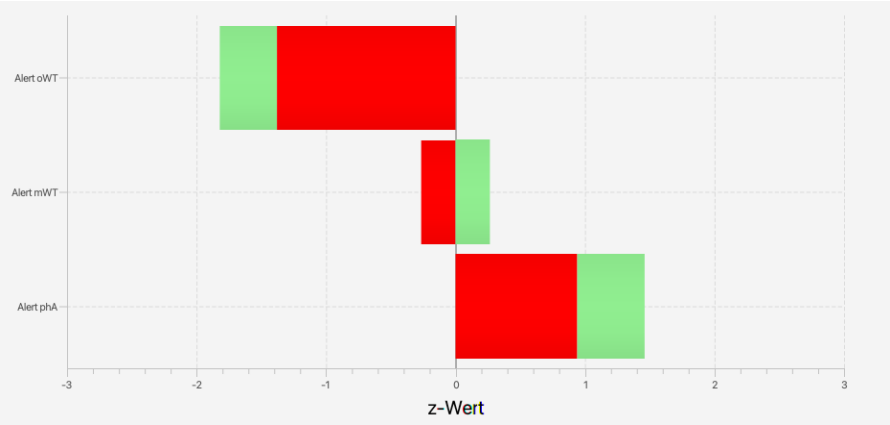


Abbildung 7.3: KI-Diagramm Farbcodiert

Im ersten und dritten Szenario fällt auf, dass das vorher angewandte Prinzip mit der *Distanz zu 0* und dem Daraufstapeln der Breite des KI-Intervalls weiterhin anwendbar ist. Szenario 3 spiegelt hierbei das bisherige Vorgehen wider. Bei Szenario 1 muss lediglich die KI-Obergrenze als Distanz zu 0 angegeben und die Breite als negativer Wert hinzugefügt werden, damit die zweite *Bar* ins Negative gestapelt wird.
Im 2. Szenario sieht man, dass beide Daten-Serien gemeinsam das Intervall abbilden, daher müssen beide Balken eingefärbt werden. Hier funktioniert das bisher angewandte Prinzip nicht, da sich der 0-Wert mitten im Intervall befindet. Um dieses Intervall abbilden zu können, muss die erste Datenserie den Wert der Untergrenze annehmen, die zweite Datenserie den der Obergrenze.
Durch diese Veranschaulichung wird klar, dass abhängig von dem Intervall nicht nur die Balken unterschiedlich eingefärbt werden müssen, sondern auch unterschiedliche Kennwerte in die Datenserie eingetragen werden müssen.

7.1.2 Implementierung

Das oben geschilderte Problem, tritt in den Klassen **CiDiagramPresenter** und **MeasurementPresenter** auf, daher wurde die Implementierung in der abstrakten Superklasse **ViewPresenter** folgendermaßen umgesetzt:

```
243 @   protected void addDataToSeries(MeasurementModel measurementModel, boolean isChartZNormScale,
244                                    XYChart.Series<Number, String> firstSeries, XYChart.Series<Number, String> secondSeries) {
245         String scaleAbbreviation = measurementModel.getScaleAbbreviation();
246         ConfidenceIntervalModel ciModel = measurementModel.getConfidenceIntervalModel();
247         Double ciMin = ciModel.getCiMinAsDouble();
248         Double ciMax = ciModel.getCiMaxAsDouble();
249         Double ciWidth = ciModel.getCiWidthAsDouble();
250
251         Double series1Value = ciMin;
252         Double series2Value = ciWidth;
253         if (isChartZNormScale && ciMin < 0) {
254             boolean isCiMaxAlsoNegative = ciMax < 0;
255             series1Value = getValueForDiagramFirstSeries(isCiMaxAlsoNegative, ciMin, ciMax);
256             series2Value = getValueForDiagramSecondSeries(isCiMaxAlsoNegative, ciMax, ciWidth);
257         }
258         firstSeries.getData().add(new XYChart.Data<>(series1Value, scaleAbbreviation));
259         secondSeries.getData().add(new XYChart.Data<>(series2Value, scaleAbbreviation));
260     }
261
```

Abbildung 7.4: Funktion: Serien-Daten hinzufügen

Die Methode *addDataToSeries* soll die Messwertdaten in die angegebenen Serien hinzufügen. Hier wird zunächst geprüft, ob die Intervall-Werte des Messwerts auf der z-Skala basieren und ob die Untergrenze des Intervalls ein negativer Wert ist. Sind beide Bedingungen nicht erfüllt, kann das Intervall mit Szenario 3 umgesetzt werden und die Datenserien werden wie oben beschrieben mit dem Untergrenzen Wert und der Breite des Intervalls in die Serien hinzugefügt.

Treffen beide Bedingungen zu, muss zwischen Szenario 1 und 2 spezifiziert werden. Dafür wird geprüft, ob es sich bei der Obergrenze des Intervalls ebenfalls um einen negativen Wert handelt. Trifft diese Bedingung zu, gilt Szenario 1 und es müssen der Obergrenzen Wert und der negative Wert der Breite in die Serien hinzugefügt werden. Ist der Obergrenzen Wert hingegen positiv, handelt es sich um Szenario 2 und es müssen der Untergrenzen- und Obergrenzen Wert der Serie hinzugefügt werden. Die jeweils korrekten Werte werden mit Hilfe der Methoden *getValueForFirstSeries* bzw. *getValueForSecondSeries* entsprechend zurückgegeben.

```
262     protected Double getValueForFirstSeries(boolean isCiMaxNegative, Double ciMin, Double ciMax) {
263         return isCiMaxNegative ? ciMax : ciMin;
264     }
265
266     protected Double getValueForSecondSeries(boolean isCiMaxNegative, Double ciMax, Double ciWidth) {
267         return isCiMaxNegative ? 0 - ciWidth : ciMax;
268     }
```

Abbildung 7.5: Funktionen: Rückgabe der korrekten Serien-Werte für Szenario 1 und 2

Da nun die korrekten Daten in den Daten-Serien hinzugefügt wurden, müssen diese für die Darstellung noch entsprechend eingefärbt werden. Dies geschieht in der Hilfsklasse **ViewStyleSupport**, die von den ViewOverlay-Klasse der entsprechenden Presenter aufgerufen wird.

```
153     /**
154      * Sets Color of bars in stacked bar chart, depending on the sign of the numerical values in the Data-Series
155      * color options are either transparent or above defined color
156      *
157      * if both signs are the same (both negative or both positive), then the first bar needs to be transparent
158      * otherwise both bars need to be colored
159      *
160      * @param stackedBarChart for which the bars need to be colored
161      */
162 @   public void setBarColoursForChart(StackedBarChart<Number, String> stackedBarChart) {
163         ObservableList<XYChart.Data<Number, String>> firstSeriesEntries = stackedBarChart.getData().get(0).getData();
164         ObservableList<XYChart.Data<Number, String>> secondSeriesEntries = stackedBarChart.getData().get(1).getData();
165
166         for (int index = 0; index < firstSeriesEntries.size(); index ++) {
167             XYChart.Data<Number, String> firstSeriesEntry = firstSeriesEntries.get(index);
168             XYChart.Data<Number, String> secondSeriesEntry = secondSeriesEntries.get(index);
169
170             Double firstBarValue = firstSeriesEntry.getXValue().doubleValue();
171             Double secondBarValue = secondSeriesEntry.getXValue().doubleValue();
172
173             String firstBarColor = haveSameSign(firstBarValue, secondBarValue) ? barChartTransparent : barChartColour;
174
175             Node ciStartNode = firstSeriesEntry.getNode();
176             ciStartNode.setStyle(String.format(barColourSettingFormat, firstBarColor));
177
178             Node ciEndNode = secondSeriesEntry.getNode();
179             ciEndNode.setStyle(String.format(barColourSettingFormat, barChartColour));
180         }
181
182         stackedBarChart.setLegendVisible(false);
183     }
```

Abbildung 7.6: Funktion: KI-Diagramm Balken einfärben

Auch hier wurde eine Regel für das Einfärben herausgefunden:
Da lediglich bei Szenario 2 die erste *Bar* nicht transparent, sondern farbig sein muss, werden die X-Achsenwerte der Serie betrachtet. Haben die Werte das gleiche Vorzeichen (Szenario 1 und 3), muss die erste *Bar* transparent sein. Andernfalls (Szenario 2) wird die erste *Bar* mit der gleichen Farbe der zweiten *Bar* eingefärbt.

Mit dieser Implementierung ist es gelungen, das oben abgebildete Diagramm auf der z-Skala darzustellen. Da Szenario 3 auf alle anderen Normskalen zutrifft, können auch diese korrekt abgebildet werden.

7.2 Export/Import-Pfade unabhängig vom Betriebssystem

Da sich die Pfad-Konventionen von Windows- und Mac-Betriebssystemen im Aufbau unterscheiden, muss dies bei der Programmierung berücksichtigt werden, um eine Anwendung auf beiden Betriebssystemen ohne Einschränkungen lauffähig zu machen.

7.2.1 Problemstellung

In dem kooperierenden Krankenhaus haben die Neuropsychologen Windows-Rechner am Arbeitsplatz zur Verfügung. Die Entwicklung des Programms erfolgt hingegen auf einem Apple-Gerät. Um Daten zur Laufzeit des Programms korrekt ein- und auszulesen, sowie diese Funktionen ausreichend zu testen, muss die Anwendung mit diese Pfad-Unterschieden umgehen können.

7.2.2 Implementierung

In der Implementierung muss zwischen dem Export und Import von Dateien unterschieden werden. Beim Export wurde dieses Problem umgangen, indem der Benutzer selbst den Export-Ordner in der UI mit einem *Directory-Chooser* (javafx.stage.DirectoryChooser) auswählt und der absolute Pfad weitergereicht wird. Beim Import war dies keine Option, da sonst der Benutzer bei jedem Start der Anwendung den Import-Ordner manuell auswählen müsste. Daher wurde dies wie folgt umgesetzt:

Es wurde festgelegt, dass Daten, die für das Einlesen bzw. Importieren bestimmt sind, in einem Import-Ordner gesammelt werden. Dieser Ordner muss sich am gleichen Ort befinden, an dem sich die ausführbare JAR-Datei befindet (im gleichen Ordner).
Zunächst wurde eine Konfigurations-Datei *import.properties* angelegt, in welcher Import-Variablen definiert werden können. Diese beinhalten bisher die Namen des Import-Ordners, sowie der Übersichtsdatei der Testverfahren.

```
import.importDirectoryName=importResources
import.overviewTableName=ImportTabelleTestverfahren.xlsx
```

Abbildung 7.7: Konfiguration Import-Variablen

Bei dem Dateinamen muss zudem das Suffix mit angegeben werden. Auf diese festgelegten Bezeichnungen müssen die Benutzer explizit hingewiesen werden. Zudem sollten die Bezeichnungen zum Nachschlagen im Benutzerhandbuch aufgenommen werden.

Die benötigte Funktionalität betrifft die *ImportData*-Klasse in der Geschäftslogik der Anwendung. Da sich der Import-Ordner neben der JAR-Datei befinden muss, kann die Anwendung beim Start den eigenen Pfad bestimmen, aus dem sie gestartet wurde. Dieser wird als String-Objekt *currentDirectory* lokal gespeichert.

```
49      this.currentDirectory = FileSystems.getDefault().getPath( first: ".").toAbsolutePath().normalize().toString();
```

Abbildung 7.8: Funktion: Aktuellen Ordner bestimmen

Anhand des Pfades ist es nun möglich, den Import-Ordner und die Übersichtsdatei zu lokalisieren und im System als lokale Variable festzuhalten.

```
132  /**
133   * Method to search for Testing-Method import file and set variables
134   */
135  private boolean retrieveTestingMethodImportFile() {
136      File currentDirectory = new File(this.currentDirectory);
137      this.importDirectory = FileUtil.getFileFromDirectory(importDirectoryName, isSearchedFileADir: true, currentDirectory);
138      this.testingMethodImportFile = FileUtil.getFileFromDirectory(testingMethodImportFileName, isSearchedFileADir: false, importDirectory);
139      return FileUtil.isFileValid(importDirectory) && FileUtil.isFileValid(testingMethodImportFile);
140  }
```

Abbildung 7.9: Funktion: Import-Variablen setzen

Die Dateien werden mit Hilfe der *FileUtil*-Klasse und dessen *getFileFromDirectory*-Methode erfasst. Diese Methode benötigt insgesamt drei Parameter: den Namen der gesuchten Datei, ob es sich bei dieser um einen Ordner handelt und in welchem Ordner nach dieser Datei gesucht werden soll.
Zunächst wird nach dem Import-Ordner in dem aktuellen Ordner gesucht und angegeben, dass es sich bei der gesuchten Datei um einen Ordner handelt (*isSearchedFileADir* = *true*). Danach kann nach der Testverfahren-Übersichtsdatei im Import-Ordner gesucht werden. Hier wird angegeben, dass die gesuchte Datei kein Ordner sein soll (*isSearchedFileADir* = *false*).
Die Methode selbst ist folgendermaßen aufgebaut:

```java
public static File getFileFromDirectory(String fileName, boolean isSearchedFileADir, File directory) {
    try {
        if (directory.isDirectory()) {
            File[] files = directory.listFiles();
            if (files != null && files.length > 0) {
                return Arrays.stream(files)
                        .filter(f -> fileName.equals(f.getName()) && f.isDirectory() == isSearchedFileADir)
                        .findAny().orElse( other: null);
            }
        }
    } catch (Exception ignored) {}
    return null;
}
```

Abbildung 7.10: Funktion: Datei in Ordner suchen

Um mit unvorhergesehenen IO-Exceptions umgehen zu können, wird ein try-catch-Block verwendet. Da die Methode bei Fehlern ohnehin *null* zurückgibt, wird auf eine explizite *null*- und Existenz-Prüfung des angegebenen Ordners verzichtet. Es wird allerdings geprüft, ob es sich bei der mitgegebenen Ordnerdatei um einen Ordner handelt. Trifft die Bedingung zu, werden alle Dateien des Ordners gefiltert. Der Filter sucht nach einer Datei, dessen Namen dem angegeben entspricht und prüft zudem, ob diese vom Typ (Ordner oder kein Ordner) mit der gesuchten Datei übereinstimmt. Da Dateien in einem Ordner nicht den gleichen Namen haben können, dürfte nur eine oder keine Datei nach diesem Filter-Prozess übrig bleiben. Wurde eine Datei mit den gegebenen Anforderungen gefunden, wird diese zurückgegeben, ansonsten wird *null* zurückgegeben.

Durch diese Implementierung wurden vordefinierte Pfad-Namen und -konventionen umgangen und funktioniert sowohl auf Mac- als auch auf Windows-Geräten. Um sich zu vergewissern, dass die hier beschriebene Umsetzung tatsächlich funktioniert, wurde die Implementierung mit einem Windows-Rechner vor der Übergabe des Programms an das Krankenhaus getestet.

8 Testen

Der folgende Abschnitt beschreibt, wie die Testung der Anwendung umgesetzt wurde. Hierbei wurde versucht, sowohl die Eignung der Software als auch deren Funktionalität zu überprüfen.

8.1 Continuous Integration

Es wurde ein *Continuous Integration* in das Projekt eingebunden, welches bei jedem *Push* ins das Git-Repository die Testfälle ausführt und das Projekt baut. So wurde jede neue Änderung sofort überprüft und Fehler konnten bereits auf den Feature-Branches identifiziert werden.

8.2 Unit-Tests

Zum Testen der Funktionalität wurden Unit-Tests implementiert. Der Fokus lag hauptsächlich auf der Verifikation der Backend-Geschäftslogik.
Diese beinhaltet neben der Interaktion über die Schnittstellen auch den Datenbankzugriff und die vom Programm durchgeführten Berechnungen. Die Berechnung der Konfidenzintervalle, sowie die Normwert-Transformation wurden mit Hilfe von Beispielen aus Fachbüchern und der Neuropsychologen genau überprüft, ob diese inhaltlich korrekt umgesetzt wurden. Die Schnittstellen wurden ebenfalls verstärkt getestet, um zu vermeiden, dass das System in einen undefinierten Zustand gerät und Komponenten des Backends ausfallen. Da die Facade-Komponente eine zentrale Rolle im Backend einnimmt, wurde versucht diese umfangreich zu testen.

8.3 Manuelle Tests

Das Frontend wurde hauptsächlich manuell getestet. Grund hierfür war, dass die Layouts der verschiedenen Overlays mehrfach angepasst und verändert wurden. Stattdessen wurde sich auf die Schnittstellen zwischen den Hauptkomponenten fokussiert.

8.4 Testphase mit erster Version der Anwendung

Um die Anwendung zu verifizieren, wurde dem Kunden die erste Version der Anwendung Mitte Januar übergeben. An dieser Testphase haben alle Neuropsychologen der Abteilung teilgenommen. Dadurch waren Testpersonen involviert, die nicht an der Entwicklung des Tools beteiligt waren. Die Testphase umfasste einen Zeitraum von etwa zwei bis drei Wochen, um die Funktionalität im Berufsalltag zu testen. Die Übergabe und Vorstellung der Anwendung erfolgte über Zoom mit einem der Neuropsychologen. Es wurde zudem ein Handbuch angefertigt, welches wichtige Bedienungshinweise, aber auch Erklärungen zu den benötigten Dateien und dem Aufbau des Systems beinhaltet. Dieses Handbuch ist der vorliegenden Arbeit angehängt. Während dieser Testphase wurde von den Neuropsychologen eine Liste mit Problemen beziehungsweise Anmerkungen geführt.

9 Evaluation

In diesem Abschnitt soll auf die Evaluation der Anwendung und der Zusammenarbeit im Team eingegangen werden. Um Feedback zur Anwendung und dem Projekt an sich zu erhalten, wurde nach Ende der Testphase eine abschließende Retrospektive mit dem Kunden durchgeführt. Hierfür wurden Übungen aus einem Retromat-Blog [4] verwendet. Zudem wurde die Liste mit Anmerkungen und Problemen, welche von den Neuropsychologen während der Testphase geführt wurde, gemeinsam besprochen. Da noch immer keine persönlichen Treffen möglich waren, wurde dieser Termin ebenfalls über *Zoom* durchgeführt. Allgemein sind die Ergebnisse nicht sehr repräsentativ, da es lediglich drei Teilnehmer gab, die zudem alle selbst an der Entwicklung beteiligt waren. Sie werden daher eher richtungsweisend interpretiert.

9.1 Feedback zur Zusammenarbeit

Um Feedback über die Zusammenarbeit zu erhalten, wurde nach passenden Übungen auf dem Retromat-Blog gesucht. Da die Team-Zusammensetzung nicht dem klassischen Scrum-Team entsprach, waren die durchzuführenden Übungen sehr begrenzt.
Es wurden die Übungen *Round of Admiration (#76)*, *Three Words (#82)* und *Original 4 (#55)* ausgewählt.
Um eine positive und wertschätzende Atmosphäre zu schaffen, wurde zunächst mit der *Round of Admiration* begonnen, bei der jeder den Satz *'Was ich am meisten an dir bewundere ist...'*, bezogen auf die Person an deren linker Seite, vervollständigt. Da Zoom benutzt wurde, durfte sich jeder eine Person aussuchen mit der Bedingung, dass eine Person nur einmal ausgewählt werden durfte.
Als nächste Übung wurden die Teilnehmer gebeten das Projekt in drei Worten zu beschreiben. Die Antworten der Teilnehmer waren insgesamt sehr positiv. Die Ergebnisse werden in Form einer *Word-Cloud* in der unteren Abbildung dargestellt.

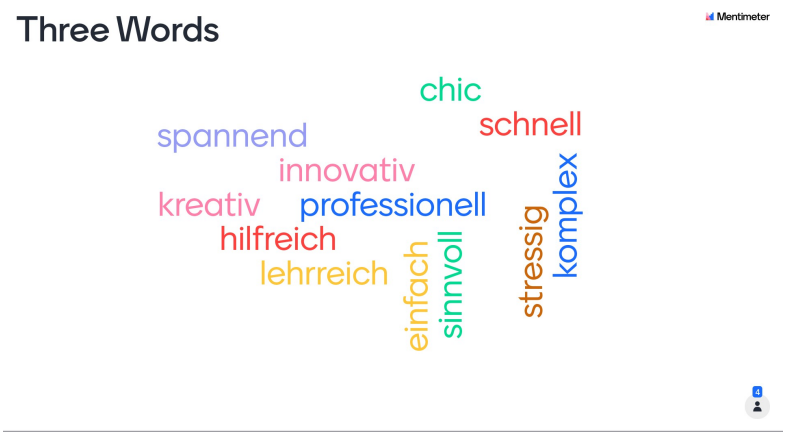


Abbildung 9.1: Ergebnisse der 'Three Words'-Übung (erstellt mit Menti.com)

Bachelorandin	spannend, lehrreich, stressig
NP 1	hilfreich, sinnvoll, innovativ
NP 2	komplex, kreativ, professionell
NP 3	chic, einfach, schnell

Tabelle 9.1: Three-Words-Übung nach Personen

Abschließend zum Feedback bezogen auf die Zusammenarbeit, wurden den Teilnehmern vier Schlüsselfragen nach Norman Keith gestellt:

1. Was haben wir gut gemacht?
2. Was haben wir gelernt?
3. Was sollten wir nächstes Mal anders machen?
4. Was beschäftigt uns noch immer?

Insgesamt waren alle Beteiligten stolz auf das resultierende Produkt, zudem wurde die Wichtigkeit der Kommunikation und Dokumentation im Projekt hervorgehoben und, dass diese eventuell für zukünftige Projekte weiter optimiert werden könnte.

9.1.1 Probleme / Anmerkungen

Der während der Testphase geführten Listen zu entnehmen wurden bisher keine Bugs oder Fehler in der Anwendung gefunden. Es gab jedoch eine wichtige Anmerkung weiterer Neuropsychologen der Abteilung, die nicht am Entwicklungsprozess beteiligt waren.

Diese haben festgestellt, dass die Bildungsjahre fachlich nicht genügend differenziert wurden. Die bisher implementierten Bildungsjahre beziehe sich auf den sogenannten *MNND*-Wert. Dieser sei ein allgemein gefasster Wert und umfasse sowohl Schuljahre als auch Ausbildungsjahre und Studienjahre. Manche Messwerte, würden sich jedoch in der Normierung allerdings nur auf die absolvierten Schuljahre beziehen.

Entsprechend müsste die Variable *Bildungsjahre* überarbeitet und eventuell erweitert werden, sodass die Anwendung mit solchen Messwerten umgehen kann. Bei dem aktuellen Stand des Programms und dessen Funktionalität ist diese Unterscheidung nicht relevant, muss aber unbedingt bei der Implementierung der automatischen Transformation von Roh- in Normwerte berücksichtigt werden.

9.2 Feedback zur Anwendung

Die Teilnehmer wurden zudem zu einer Einschätzung des Systems in Bezug auf die Qualitätsanforderungen gebeten. Der Fragebogen bestand aus fünf Items mit jeweils einer Aussage und einer Bewertungsskala von 0 (*starker Ablehnung*) bis 5 (*starke Zustimmung*). Wie in der Anforderungsanalyse definiert, wurde bei der Anwendung ein Fokus auf die Betreibbarkeit gelegt. Speziell wurden die Qualitätsmerkmale der *Erlernbarkeit* und *Einfachheit der Benutzung* angestrebt. Die Teilnehmer stimmten der *generellen Erlernbarkeit der Benutzung als einfach* mit einem Mittelwert von 4.7 zu.

Die Frage nach der *generellen Einfachheit der Bedienbarkeit* erhielt eine Zustimmung von durchschnittlich 4.3 Punkten.
Gezielter wurde zudem nach der intuitiven Bedienung und Verständlichkeit der Hinweistexte gefragt, welche eine Zustimmung von 3.3 bzw. 4 erhielten.

Da das Ziel dieser Arbeit war, den Neuropsychologen in ihrem Arbeitsalltag zu unterstützen, wurde zudem die Aussage 'Das Tool erleichtert mir meine Arbeit' hinzugefügt. Diese Aussage erhielt eine Zustimmung von 3 Punkten. Anhand der Verteilung der Kurve ist jedoch zu erkennen, dass diese Werte sehr gestreut waren. Wenn bei der Bewertung

bedacht wird, dass das Tool bisher nur eine der Hauptfunktionalitäten besitzt, ist diese Wertung jedoch insofern positiv zu beurteilen, als dass sie einer der dort arbeitenden Personen im Alltag hilft.

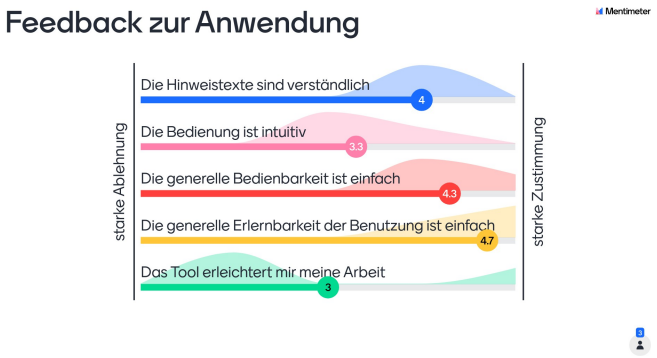


Abbildung 9.2: Ergebnisse des Fragebogens zu den Qualitätszielen (erstellt mit Menti.com)

Abschließend wurden die Teilnehmer nach der Zufriedenheit mit dem Tool gefragt. Um den Teilnehmern eine differenziertere Auswahl zu geben, wurde eine Bewertungsskala von 0 (überhaupt nicht zufrieden) bis 10 (sehr zufrieden) angeboten. In dieser Gesamtbewertung erhielt die entwickelte Anwendung eine durchschnittliche Bewertung von 9.3 Punkten. Das Tool scheint daher die Funktionalität, welche es bisher besitzt, mit großer Zufriedenheit der Kunden auszuführen.

Abbildung 9.3: Ergebnisse der Gesamtbewertung des Tools (erstellt mit Menti.com)

10 Kritik

Durch das Treffen von Entscheidungen sowie der Einfluss äußerer Faktoren werden einige Schwachpunkte der Anwendung deutlich. Im folgenden Abschnitt werden einige dieser Schwachpunkte vorgestellt.

10.1 Kommunikationsaufwand unterschätzt

Da Corona-bedingt zum Großteil des Projekts keine persönlichen Treffen möglich waren, wurde versucht, die wöchentlichen Treffen in Form von E-Mails und Zoom-Meetings umzusetzen. Zudem wurde von der Krankenhausleitung aus Kostengründen entschieden, dass nur noch ein Neurologe an den Treffen teilnehmen könne, wodurch die verfügbaren Ressourcen weiterhin begrenzt wurden. Daher entstand ein erhöhter Kommunikationsaufwand im Team selbst, aber auch unter den Neuropsychologen. Dafür mussten Lösungen gefunden werden. Dies resultierte in erhöhtem Arbeitsaufwand im Projekt.

10.2 Anfangs keine klare Rollenverteilung im Team

Zu Beginn des Projekts wurden die Treffen im Team mit allen drei beteiligten Neuropsychologen angestrebt. Da diese in ihrer Position jedoch alle gleichgestellt sind und in ihrem Arbeitsalltag teilweise sehr unterschiedliche Aufgabe ausführen, bestand zunächst kein klarer Konsens über manche Details der Anforderungen. Als die Ressourcen vom Krankenhaus weiter eingeschränkt wurden, wurde vereinbart, dass ein gleichbleibender Ansprechpartner mit voller Entscheidungsmacht für die Treffen zur Verfügung stehe. Dieser Ansprechpartner war zudem für die Kommunikation unter den Kollegen verantwortlich. Dies hat etwas an Arbeitsaufwand abgenommen, allerdings auch die Kontrolle oder Einflussnahme reduziert um sicherzustellen, dass Fragen oder Bitten an die anderen Team-Mitgliederangekommen oder richtig verstanden wurden.

10.3 Probleme bei der Datenbank-Implementierung

Die Entitätstypen *TestCase* und *Measurement* sind beide sehr zentrale Konstrukte in dem erstellten Modell und wurden als schwache Entitäten identifiziert, die zudem aufeinander aufbauen.
Es wurde versucht die Datenbank nach dem ER-Modell aufzubauen. Allerdings gab es Probleme mit Spring und Hibernate, sodass Datenbankspalten nicht verbunden werden konnten. IntelliJ hat jegliche Versuche als nicht zulässig markiert, was zu Build-Problemen geführt hat. Auch nach ausgiebigen Recherchen wurde keine Lösung zu dem Problem gefunden. Es entstand sogar eher der Eindruck, dass der Einsatz der Hibernate-Annotationen (@OneToMany, ...) sehr bedacht gewählt und bidirektional implementiert werden sollte. Da für die bidirektionale Umsetzung jedoch das Verbinden von Tabellenspalten empfohlen wurde, ist nach anderen Möglichkeiten der Umsetzung gesucht worden.

10.4 Facade Single-Point-Of-Failure

Die Facade soll als zentrale Anlaufstelle für das Backend gelten. Dadurch entsteht allerdings auch ein potentieller *Single-Point-Of-Failure*. Es wurde versucht diesen zu vermeiden, in dem die Komponente lediglich koordiniert und Parameter, ohne sie zu betrachten, an die entsprechenden Services weiterleitet. Dadurch muss jedoch darauf vertraut werden, dass die entsprechenden Services alle Fehler und möglichen Exceptions abfangen. Die Interaktion mit der Facade und den angesprochenen Services sollte daher umfangreich getestet werden. Allgemein besteht bei dem System keine Ausfallsicherheit, falls irgendeine Komponente in einen undefinierten Zustand kommt. Da die Funktionalität des Systems jedoch lediglich unterstützend agieren soll, wurde hier auf zusätzliche Sicherheitsvorkehrungen bezüglich des Ausfalls oder der Verfügbarkeit verzichtet.

10.5 Nicht für zeitgleiche Nutzung ausgelegt

Während des gesamten Projekts wurde davon ausgegangen, dass die Anwendung auf einem gemeinsam genutzten Computer verwendet werden soll. Bei der Übergabe der Anwendung an das Krankenhaus wurde erst festgestellt, dass diese in einer gemeinsamen Ordnerstruktur abgelegt werden sollte und von vielen Mitarbeitern benutzt und eventuell

zum gleichen Zeitpunkten verwendet werden könnte. Um Konflikte bei Datenbankzugriffen zu vermeiden, wurden die Benutzer gebeten, auf einer lokalen Kopie der Anwendung zu arbeiten. Diese Anforderung hätte vorher mit dem Kunden besprochen und in einer Verteilungssicht festgehalten werden müssen.

11 Zusammenfassung

Zur Entwicklung eines Tools zur Unterstützung der Analyse- und Interpretation neuropsychologischer Diagnostik wurde eine Kooperation mit einem lokalen Krankenhaus in Hamburg eingegangen und mit der Abteilung der klinischen Neuropsychologie zusammengearbeitet. Da die Neuropsychologen die Benutzer des zu entwickelnden Tools repräsentieren, wurde ein Workshop zur Anforderungsanalyse mit drei Mitarbeitern der Abteilung durchgeführt, die auch im Verlauf der Entwicklung als Fachexperten unterstützend zur Seite standen.

Auf Grund von Corona konnten abgesehen vom Workshop keine persönlichen Treffen stattfinden. Dadurch entstand ein erhöhter Kommunikations- und Arbeitsaufwand für alle Beteiligten des Projekts. Dennoch konnte anhand der ausgearbeiteten Ergebnisse des Workshops mit den Neuropsychologen eine Spezifikation der Anforderungen erfolgen. So wurden ein fachlichen Datenmodell mit einem Datentypen-Verzeichnis angelegt, sowie Use-Cases mit Wireframes erstellt.
Bei der Architektur wurde sich an bestehenden Einflussfaktoren, Randbedingungen und bekannten Entwurfsmustern, wie der Drei-Schichten-Architektur und dem Facade-Pattern orientiert und eine Lösungsstrategie entwickelt. Ein sehr wichtiger Faktor, mit dem ein hohes Risiko für das Projekt verbunden war, bestand in der bestehenden IT-Sicherheitsstruktur des Krankenhauses. Entsprechend beeinflusste dieser Faktor zentrale Entwurfsentscheidungen wie beispielsweise die Reduktion von Abhängigkeiten außerhalb des Systems, sowie die Form, in welcher das Programm an das Krankenhaus ausgeliefert werden konnte.

Um den Benutzers Kontrolle über die Informationen zu den verwendeten Verfahren zu geben, mit denen die Anwendung arbeitet, wurde sich für eine Kombination aus einer Datenbank und dem Einlesen von Informationen zur Laufzeit entschieden. Im Hinblick auf den IT-Sicherheits-Faktor in Verbindung mit dem Datenschutz wurde sich für eine lokale Datenbank entschieden und ein Relationales-Datenmodell und Relationstypen definiert. Dies ließ sich mit Hilfe des fachlichen Datenmodells ableiten. Als auffällig stellten

sich dabei zwei zentrale Objekt-Typen dar (Fall und Messwert), die jeweils als schwache Entitäts-Typen einzuordnen sind und zur eindeutigen Identifizierung auf Attribute anderer Objekte angewiesen sind. Dies führte zu Architektur-Entscheidungen wie dem Hinzufügen von Schlüssel-Klassen sowie der Art und Weise, wie diese Objekte gespeichert und auf sie zugegriffen wird.
Um die Architektur verständlich und nachvollziehbar darzustellen, wurden mehrere Sichten angefertigt. Diese beginnen mit einer Kontextabgrenzung zu dem Systemumfeld und wird gefolgt von einer statischen Top-Down Zerlegung des internen Aufbaus in einzelne Bausteine. Um die Interaktion der Systembausteine zu verdeutlichen, wurden zudem für zentrale Funktionen und Abläufe Laufzeitsichten erstellt. Diese zeigen nicht nur die Interaktion zwischen den Komponenten, sondern auch deren zeitlichen Abläufe.
Während des Projektes wurde darüber hinaus eine Risikoliste geführt und stetig aktualisiert. Einige der Risiken konnten abgewandt werden, vielesind hingegen offengeblieben.

Spannend bei der Implementierung war vor allem die Darstellung der Konfidenzintervalle, da das Diagramm auf einem *StackedBarChart* basiert und das Darstellen negativer Zahlen oder Ranges ein Umdenken im Umgang mit diesen Werten bedurfte.
Die Anwendung wurde auf einem Mac-Rechner implementiert, eingesetzt wird sie allerdings auf einem Windows-System. Da bei dem Importieren und Exportieren von Informationen mit Pfaden gearbeitet wird, wurde eine Implementierung angestrebt, die unabhängig vom zugrundeliegenden Betriebssystem funktioniert.

Die Funktionalität der Anwendung wurde hauptsächlich durch manuell- und Unit-Tests überprüft. Dabei wurde sichergestellt, dass die Berechnung der Konfidenzintervalle sowie die Normierung inhaltlich korrekt umgesetzt wurde. Nach einer Testphase der Anwendung durch die Benutzer, sind Anregungen für Verbesserungen und zusätzliche Anforderungen aufgekommen, ein Fehlverhalten der Anwendung konnte hingegen nicht identifiziert werden.

Zum Abschluss des Projektes, wurde eine Retrospektive mit den drei Neuropsychologen durchgeführt. Dieses bestand aus Übungen und Gesprächen, um Feedback zur Zusammenarbeit, Funktionalität und Qualität des Tools zu erlangen. Insgesamt war die Rückmeldung der Neuropsychologen positiv. Bei einer Gesamtbewertung am Ende erhielt das Tool 9.3 von 10 Punkten.

Die Entwicklung des Tools sollte Neuropsychologen als eine Unterstützung bei der Arbeit dienen und triviale aber aufwändige Arbeitsschritte automatisieren. Obwohl das Tool

noch nicht seine volle Funktionalität besitzt, kann es schon jetzt von den Neuropsychologen im Arbeitsalltag eingesetzt werden und einen Mehrwert schaffen.

Literaturverzeichnis

[1] *Event Storming 101: Accelerating Your Software Development in Domain-Driven Design.*. – URL https://www.lucidchart.com/blog/ddd-event-storming. – abgerufen: 12.02.21

[2] *H2 Database offizielle Website.* – URL www.h2database.com/html/main.html. – abgerufen: 14.12.20

[3] *Spektrum. Lexikon der Neurowissenschaft. Neuropsychologie.* – URL https://www.spektrum.de/lexikon/neurowissenschaft/neuropsychologie/8714. – abgerufen: 09.10.20

[4] Baldauf, C.: *The Retromat-Blog.* – URL https://retromat.org. – abgerufen: 05.02.21

[5] Bates, N.: *Diagnostik I. Formeln.* – URL https://www.unet.univie.ac.at/~a9806297/Files/Formeln.pdf. – abgerufen: 24.01.21

[6] Brandolini, A.: *Introducing Event Storming.* 2013. – URL https://ziobrando.blogspot.com/2013/11/introducing-event-storming.html. – abgerufen: 02.10.20

[7] Brandolini, A.: *Remote EventStorming.* 2020. – URL https://blog.avanscoperta.it/2020/03/26/remote-eventstorming/. – abgerufen: 02.10.20

[8] Dear, K.: *Standardnormalverteilung.* 1999. – URL http://eswf.uni-koeln.de/glossar/surfstat/normal.htm. – abgerufen: 14.01.21

[9] Ebert, C.: *Risikomanagement kompakt. Risiken und Unsicherheiten bewerten und beherrschen.* Springer-Vieweg

[10] Fowler, M.: *Passive View.* 2006. – URL https://martinfowler.com/eaaDev/PassiveScreen.html. – abgerufen 24.11.20

[11] Gamma, R.; Johnson R.; Vlissides J.: *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley

[12] Gee, T.: *Fully Reactive: Spring, Kotlin, and JavaFX Playing Together [Links]. Spring Framework 5 brings full reactive support to developers everywhere. Learn more about reactive programming with Spring, JavaFX, and Kotlin!* 2019. – URL https://dzone.com/articles/fully-reactive-spring-kotlin-and-javafx-playing-to. – abgerufen 02.10.20

[13] Haase, O.: *Verteilte Systeme. Verteilte Architekturen..* – URL http://www-home.fh-konstanz.de/~haase/lehre/versy/slides/v3VerteilteArchitekturen.pdf. – abgerufen: 28.10.20

[14] Lahres, G.: *Praxisbuch Objektorientierung. Professionelle Entwurfsverfahren.* Rheinwerk Computing, 2006. – URL http://openbook.rheinwerk-verlag.de/oo/index.htm

[15] Mayas, S.; Kromker H.: Personas for Requirements Engineering. In: *Usability- and Accessibility-Focused Requirements Engineering*, Springer International Publishing, 2016, S. 34–46

[16] Metzner, A.: *Software-Engineering - kompakt.* Carl Hanser Verlag München, 2020

[17] Puhani, J.: *Statistik. Einführung mit praktischen Beispielen, 13. Auflage.* Wiesbaden: Springer Gabler, 2020

[18] Schatten, M.; Winkler D.; Biffl S.; Gostischa-Franta E.; Ostreicher T.: *Software-Architektur.* S. 199–227. In: *Best Practice Software-Engineering: Eine praxiserprobte Zusammenstellung von komponentenorientierten Konzepten, Methoden und Werkzeugen*, Heidelberg: Spektrum Akademischer Verlag, 2010

[19] Schmidt-Atzert, M.: *Psychologische Diagnostik, 5. Auflage.* Springer Verlag, 2012

[20] Starke, G.: *Effektive Softwarearchitekturen. Ein praktischer Leitfaden.* Carl Hanser Verlag München, 2014

[21] Sturm, M.; Münte T.: *Lehrbuch der Klinischen Neuropsychologie. Grundlagen, Methoden, Diagnostik, Therapie, 2. Auflage.* Heidelberg: Spektrum akademischer Verlag, 2009

A Anhang

A.1 Glossar

- **AI-T**
  Analyse- und Interpretations-Tool
- **API**
  Application Programming Interface (engl.); Schnittstelle zur Anwendungsprogrammierung
- **CC**
  Clinic Centre (engl.) internes System des EKA
- **CI**
  Konfidenzintervall bzw. Confidence Interval (engl.)
- **CRUD**
  Create, Read, Update und Delete als grundlegende Funktionen persistent gehaltener Daten
- **DB**
  Datenbank; Database (engl.)
- **DTO**
  Data Transfer Object (engl.)
- **EKA**
  Evangelisches Krankenhaus Alsterdorf
- **GUI**
  Graphical User Interface (engl.); grafische Benutzerschnittstelle bzw. grafische Benutzeroberfläche

- **IO**
  Input/Output (engl.); Ein- und Ausgaben
- **IT**
  Information Technology (engl.); Informationsetechnologien
- **KI**
  Konfidenzintervall bzw. Confidence Interval (engl.)
- **MVP**
  Minimum Viable Product (engl.); minimal lauffähges Produkt
- **NP**
  Neuropsychologin/e
- **Rel.**
  Reliabilität; Reliability (engl.)
- **SD**
  Standardabweichung, Standard Deviation (engl.)
- **SMF bzw. SEM**
  Standard-Messfehler; Standard Error of Measurement (engl.)
- **SoC**
  Separation of Concern (engl.); Trennen von Verantwortlichkeiten
- **UI**
  User Interface (engl.); Benutzerschnittstelle bzw. Benutzeroberfläche

A.2 Benutzerhandbuch

AI-Tool Benutzerhandbuch

VERSION 1.0.0

TOOL ZUR UNTERSTÜTZUNG DER ANALYSE UND INTERPRETATION NEUROPSYCHOLOGISCHER DIAGNOSTIK

Erstellt am:
January 25, 2021

Authorin
Lilli-Jo KERTSCHER
Kontakt: lj.kertscher@gmail.com

Contents

1 Das Programm und dazugehörige Dateien

1.1 Programm-Datei JAR

Das Programm selbst ist in der ausführbaren JAR-Datei **AI-Tool-x.x.x.jar** komprimiert und lässt sich, wie im nächsten Abschnitt erklärt, über die Eingabe-Konsole starten.

Figure 1: *ausführbare Jar-Datei*

1.2 Import Ordner

Um Informationen in das Programm einzuspielen, muss neben der JAR-Datei ein Ordner mit dem Name **importResources** liegen.

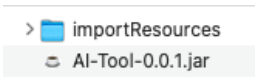


Figure 2: *Ordner für importe in das Programm*

1.2.1 Testverfahren Import-Tabelle

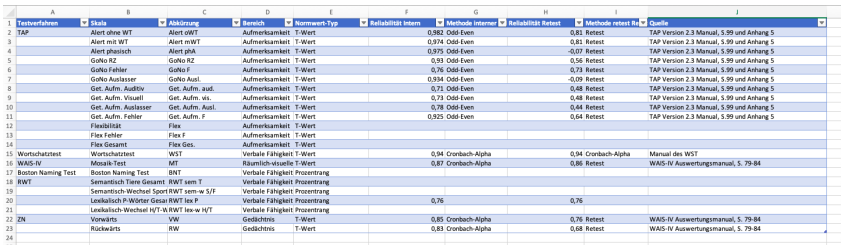
Um vordefinierte Skalen (inkl. verwendeter Norm-Skalen, Reliabilitäten und deren Quellen) in das Programm zu importieren, muss der *importResources*-Ordner die Datei **ImportTabelleTestverfahren.xlsx** enthalten. Sie beinhaltet eine Tabelle mit vorgegebenen Spalten.

Figure 3: *Beispiel Import-Datei für Skalen*

Excel-Tabelle mit folgenden Spalten:

Spalte	Bedeutung	Beispiel
Testverfahren	Namen des übergeordneten Testverfahrens der Skala	*TAP, WAIS-IV, ...*
Skala	Namen der Skala	*Wortschatztest, Mosaik-Test, Boston Naming Test*
Abkürzung	Abkürzung der Skala	*WST, MT, BNT*
Bereich	Bereich der Skala	*Gedächtnis, Aufmerksamkeit, ...*
Normwert-Typ	Art der Norm-Skala des Normwerts	*Prozentrang, T-Wert, ...*
Reliabilität Intern	Interne Reliabilität der Skala	*0.932*
Methode interner Reliabilität	Methode interner Reliabilität	*Cronbach-Alpha, Odd-Even, ...*
Reliabilität Retest	Retest Reliabilität der Skala	*0.82*
Methode retest Reliabilität	Methode retest Reliabilität	*Retest, Odd-Even, ...*
Quelle	Quelle auf der die Reliabilität-Infos der Skala basieren	*WAIS-IV Manual, S. 79-84*

Wichtig!:
Die Spalten müssen in genau dieser Reihenfolge in der Datei vorliegen, und dürfen nicht verschoben werden. Das Einlesen und die richtige Interpretation dieser Daten hängt mit dieser Reihenfolge zusammen. Bei Änderungen der Struktur kann das Programm die Daten nicht mehr korrekt einlesen.



	A	B	C	D	E	F	G	H	I	J
1	Testverfahren	Skala	Abkürzung	Bereich	Normwert-Typ	Reliabilität Intern	Methode interner	Reliabilität Retest	Methode retest Re	Quelle
2	TAP	Alert ohne WT	Alert oWT	Aufmerksamkeit	T-Wert	0,982	Odd-Even	0,81	Retest	TAP Version 2.3 Manual, S.99 und Anhang 5
3		Alert mit WT	Alert mWT	Aufmerksamkeit	T-Wert	0,974	Odd-Even	0,81	Retest	TAP Version 2.3 Manual, S.99 und Anhang 5
4		Alert phasisch	Alert phA	Aufmerksamkeit	T-Wert	0,975	Odd-Even	-0,07	Retest	TAP Version 2.3 Manual, S.99 und Anhang 5
5		GoNo RZ	GoNo RZ	Aufmerksamkeit	T-Wert	0,93	Odd-Even	0,56	Retest	TAP Version 2.3 Manual, S.99 und Anhang 5
6		GoNo Fehler	GoNo F	Aufmerksamkeit	T-Wert	0,76	Odd-Even	0,73	Retest	TAP Version 2.3 Manual, S.99 und Anhang 5
7		GoNo Auslasser	GoNo Ausl.	Aufmerksamkeit	T-Wert	0,934	Odd-Even	-0,09	Retest	TAP Version 2.3 Manual, S.99 und Anhang 5
8		Get. Aufm. Auditiv	Get. Aufm. aud.	Aufmerksamkeit	T-Wert	0,71	Odd-Even	0,48	Retest	TAP Version 2.3 Manual, S.99 und Anhang 5
9		Get. Aufm. Visuell	Get. Aufm. vis.	Aufmerksamkeit	T-Wert	0,73	Odd-Even	0,48	Retest	TAP Version 2.3 Manual, S.99 und Anhang 5
10		Get. Aufm. Auslasser	Get. Aufm. Ausl.	Aufmerksamkeit	T-Wert	0,78	Odd-Even	0,44	Retest	TAP Version 2.3 Manual, S.99 und Anhang 5
11		Get. Aufm. Fehler	Get. Aufm. F	Aufmerksamkeit	T-Wert	0,925	Odd-Even	0,64	Retest	TAP Version 2.3 Manual, S.99 und Anhang 5
12		Flexibilität	Flex	Aufmerksamkeit	T-Wert					
13		Flex Fehler	Flex F	Aufmerksamkeit	T-Wert					
14		Flex Gesamt	Flex Ges.	Aufmerksamkeit	T-Wert					
15	Wortschatztest	Wortschatztest	WST	Verbale Fähigkeit	T-Wert	0,94	Cronbach-Alpha	0,94	Cronbach-Alpha	Manual des WST
16	WAIS-IV	Mosaik-Test	MT	Räumlich-visuelle	T-Wert	0,87	Cronbach-Alpha	0,86	Retest	WAIS-IV Auswertungsmanual, S. 79-84
17	Boston Naming Test	Boston Naming Test	BNT	Verbale Fähigkeit	Prozentrang					
18	RWT	Semantisch Tiere Gesamt	RWT sem T	Verbale Fähigkeit	Prozentrang					
19		Semantisch-Wechsel Sport	RWT sem-w S/F	Verbale Fähigkeit	Prozentrang					
20		Lexikalisch P-Wörter Gesa	RWT lex P	Verbale Fähigkeit	Prozentrang	0,76		0,76		
21		Lexikalisch-Wechsel N/T	N/T-N RWT lex-w N/T	Verbale Fähigkeit	Prozentrang					
22	ZN	Vorwärts	VW	Gedächtnis	T-Wert	0,85	Cronbach-Alpha	0,76	Retest	WAIS-IV Auswertungsmanual, S. 79-84
23		Rückwärts	RW	Gedächtnis	T-Wert	0,83	Cronbach-Alpha	0,68	Retest	WAIS-IV Auswertungsmanual, S. 79-84
24										

Figure 4: *Beispiel-Tabelle*

1.3 Datenbank-Datei

Die Datenbank-Datei wird von dem Programm selbst erstellt, falls diese nicht bereits existiert. Sie erhält den Name **aitDatabase.mv.db** und wird neben der auführbaren JAR-Datei abgelegt.

Figure 5: *Ordner für importe in das Programm*

Das sind alle notwendigen Dateien, die das Programm benötigt, um zu starten.

2 Programm starten

Das Programm wird aus der Konsole aus gestartet. Dazu müssen folgende Schritte befolgt werden:

2.1 Programm-Ordner öffnen

Über den Dateien-Explorer (Windows) bzw. Finder (Mac) zu dem Ordner navigieren, in dem sich die ausführbare JAR-Datei *AI-Tool-x.x.x.jar* befindet

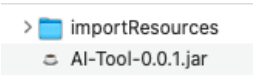


Figure 6: *Ordner für importe in das Programm*

2.2 Konsole öffnen

Windows:

1. Windowstaste + R: "Ausführen"-Fenster öffnet sich
2. "cmd.exe" eingeben und bestätigen mit Enter

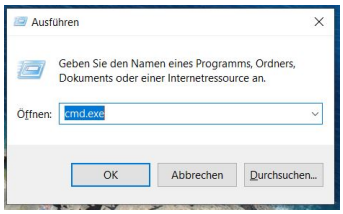


Figure 7: *Konsole öffnen unter Windows*

Mac:

1. cmd + Leertaste: "Spotlight-Suche" öffnet sich
2. "terminal" eingeben und bestätigen mit Enter

Figure 8: *Konsole öffnen unter Mac*

Die Konsole öffnet sich. In der letzten Zeile sehen wir, an welchem Ort wir uns befinden. In diesem Fall sind wir auf dem *C:*-Laufwerk im Ordner *"User"* und dort wiederum im Ordner *"Jan"*

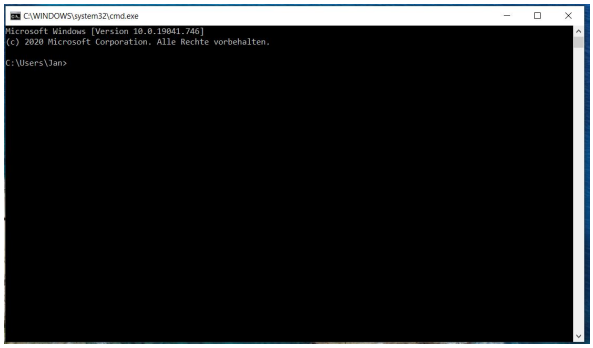


Figure 9: *geöffnete Konsole*

2.3 Mit der Konsole in den Programm-Ordner wechseln

Um das Programm zu starten, müssen wir in der Konsole in den in Schritt 1. geöffneten Ordner wechseln.
Das machen wir wie folgt:

2.3.1 In das richtige Laufwerk wechseln (Windows)

Liegt der Ziel-Ordner in einem anderen Laufwerk (z.B. Laufwerk "P:"), muss dieses vorher gewechselt werden.
Der Wechsel erfolgt durch die Eingabe des gewünschten Laufwerks.
z.B. um vom Pfad *C:* in das Laufwerk *P:* zu wechseln muss lesiglich **P:** in die Konsole eingegeben werden und ENTER gedrückt werden.

Der Wechsel war erfolgreich, wenn die letzte Zeile der Konsole mit dem gewünschten Laufwerk beginnt: *P:>*

2.3.2 "Change Directory"-Befehl geben

Den*"Change Directory"*-Befehl können wir der Konsole geben, indem wir **cd** und einmal die Leertaste eingeben

WICHTIG! Noch nicht ENTER drücken, der Befehl ist noch nicht vollständig und durch das Klicken von ENTER wird der Befehl ausgeführt!

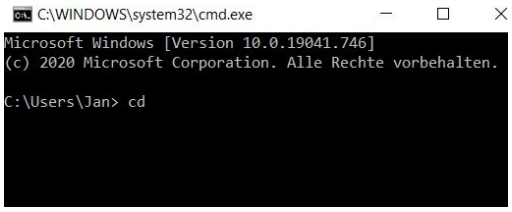


Figure 10: *"Change-Directory"-Befehl*

2.3.3 JAR-Datei *AI-Tool-x.x.x.jar* in die Konsole ziehen

Wenn wir die Datei per *Drag-And-Drop* in die Konsole ziehen, wird automatisch dessen Pfad hinter unsere "cd"-Eingabe eingefügt

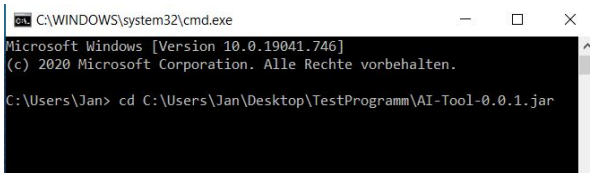


Figure 11: *JAR-Pfad in Konsolel*

2.3.4 Pfad am Ende anpassen

Da wird in den Übergeordneten Ordner der JAR-Datei wollen, müssen wir den Datei-Namen entfernen. Wenn man möchte kann man auch den letzten Backslash aus de Pfad entfernen, muss man aber nicht.
Alternativ kann man auch den Ziel-Ordner direkt in die Konsole ziehen (Ordner in dem sich die JAR-Datei befindet)

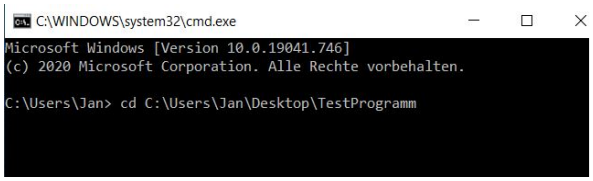


Figure 12: *JAR-Name aus Pfad entfernt*

2.3.5 Bei Bedarf: Pfad am Anfang anpassen

Da wir uns ja schon im Ordner *Jan* befinden, müssen wir mit den **Pfeiltasten!** im Pfad zurück gehen und den überflüssigen Teil des Pfads entfernen. Würden wir dies nicht machen, wird nach einem Ordnern "Users" in dem Order "Jan" gesucht, anstatt nach dem nächsten Ordner im Pfad "Desktop". Das resultiert darin, dass anstatt in den gewünschten Ordner zu wechseln, die Konsole zurückgeben wird, dass es den gesuchten Ordner nicht finden kann oder einfach nichts machen und den aktuellen Pfad in der letzten Zeile angeben...

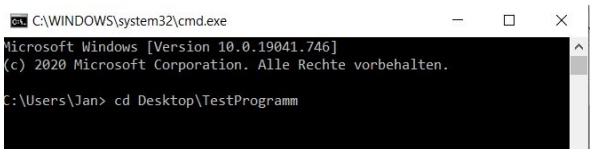


Figure 13: *Korrekter Pfad*

2.3.6 ENTER drücken

Nun können wir endlich Enter-drücken und die Konsole wechselt in den richtigen Ordner. Wir können erkennen, ob der Ordner-Wechsel erfolgreich war, wenn die Konsole in der letzte Zeile genau den Pfad angibt, in den wir wechseln wollten

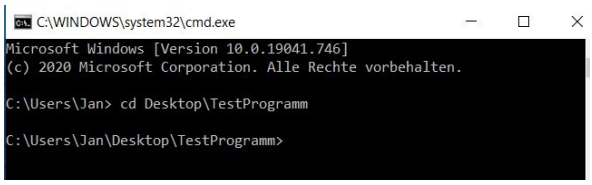


Figure 14: *Erfolgreicher Ordner-Wechsel*

2.4 Die JAR-Datei Starten

Wenn wir uns im richtigen Ordner befinden, können wir das Programm ganz leicht mit dem Befehl **java -jar AI-Tool-x.x.x.jar** und der ENTER-Taste starten (anstatt der "x"e entsprechend die Versionsnummer eingeben)

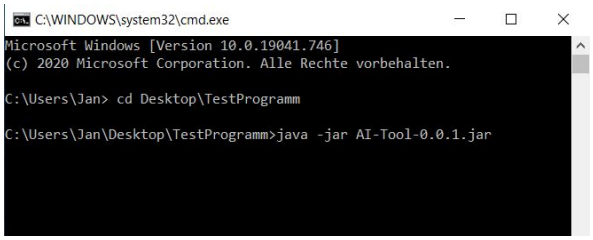


Figure 15: *Start-Befehl*

Hier hilft uns die Konsole auch wieder: wenn du *java -jar AI-T* schon eingegeben hast, kannst du einfach die TAP-Taste drücken und der Name der Datei wird automatisch vervollständigt (Bedingung: es ist eindeutig welche Datei du meinst, gibt es mehrere Dateien die so anfangen weiß sie nicht, welche sie nehmen soll)

2.5 Erfolgreicher Start

Läuft alles wie gewünscht, startet das Programm (das dauert leider etwas) und ein neues Fenster für die Benutzer-Interaktion öffnet sich.

Figure 16: *Erfolgreicher Start*

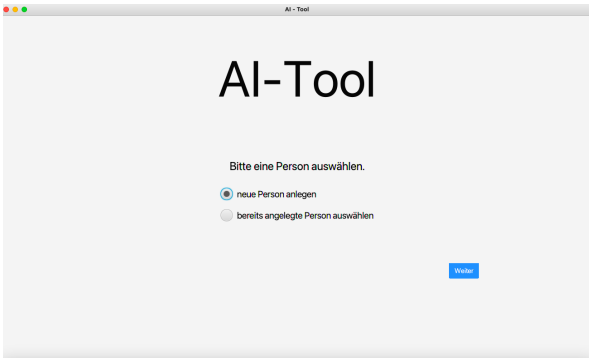


Figure 17: *Benutzer-Interaktions-Fenster*

3 Benutzungshinweise

3.1 Person auswählen

Soll eine bereits angelegte Person geöffnet werden, kann der entsprechende Punkt ausgewählt werden. Es wird automatisch ein Textfeld erscheinen. Dort kann man die gewünschte ID eingeben, oder durch klicken des Pfeil-Symbols am Ende des Textfelds sich eine Liste angelegter Personen anzeigen lassen und die gesuchte ID auswählen. Ist die gesuchte Person dort nicht gelistet, muss diese Person neu angelegt werden.

Durch das klicken des *Weiter*-Buttons gelangt man zu seiner gewählten Option.

3.2 Person anlegen

Um eine Person anlegen zu können, muss eine Personen-ID angegeben werden. Anhand dieser ID können Personen eindeutig identifieziert werden, daher kann es hier keine Duplikate geben. Die ID ist demnach fest einer Person zugeordnet. Es ist nicht möglich diese später zu ändern.

Das Programm weist darauf hin, wenn versucht wird eine Person mit einer ID anzulegen, die bereits existiert. Es erscheint ebenfalls eine Warning, wenn dieses Feld leer bleibt. Alle anderen gelisteten Personen-Variablen können hingegen frei bleiben, oder gefüllt werden.

Die Initialien müssen nach einen bestimmten Format angegeben werden: Nach einem Buchstaben muss immer entweder ein Bindestrich (-) oder ein Punkt kommen (.) damit das Format akzeptiert wird. Auf Groß- und Kleinschreibung muss hingegen nicht geachtet werden.

Nach Bestätigung des Anlegens gelangt man automatisch zur Personen-Übersicht der soeben angelegten Person.

3.3 Personen-Übersicht

Die Personen-Übersicht zeigt die angegeben Variablen der Person, sowie eine Tabelle mit deren im Programm angelegten Fällen. Die Fälle sind nach Datum sortiert, mit dem am nähsten zurückliegenden Datum an oberster Stelle.

Durch einen Doppelklick auf ein Fall in der Tabelle, wird der entsprechende Fall geöffnet. Zudem gibt es die Option einen neuen Fall hinzuzufügen durch das klicken des *"Neuen Fall anlegen"*-Buttons.

Um die Personen-Daten zu Ändern, wird man durch das klicken des *"Person bearbeiten"*-Buttons entsprechend weitergeleitet.

3.4 Personen bearbeiten

Hier lassen sich alle Variablen der Person anpassen, abgesehen von der Personen-ID.

Wie bei dem Anlegen einer Person müssen die Initialien nach dem oben angegebenen Format vorliegen, sonst kommt eine Warnung, die auf ein inkorrektes Format hinweist.

Zudem besteht die Möglichkeit die Person zu löschen. Hierbei werden auch alle angelegten Fälle der Person mitgelöscht.

3.5 Fall anlegen

Um einen neuen Fall anzulegen, muss die getestete Person im Programm angelegt und geöffnet sein. Auf der Seite der Personen-Übersicht muss der *"Neuen Fall anlegen"*-Button geklickt werden.

Man wird weitergeleitet zu einer Tabelle mit importierten Testverfahren/Skalen. Über der Tabelle kann man die Fall-Variablen angeben.

Das Testdatum ist auf den aktuellen Tag voreingestelltund lässt sich über das Textfeld selbst, oder über eine Kalender-Ansicht auswählen bzw. ändern. Um einen Fall eindeutig zu indentifizieren, muss das Testdatum, ähnlich wie die Personen-ID, innerhalb einer ausgewählten Person eindeutig sein. Daher kann eine Person nur einen Fall an einem Datum haben.

Zudem kann man die Bilsungsjahre einer Person zum Testzeitpunkt angeben. Die Bildungsjahre sind vordefiniert auf 0-25, wird eine Zahl außerhalb dieser Spanne eingegeben, wird diese ignoriert.

Die letzten beiden Test-Variablen sind das gewünschte Alpha und die Testrich-

tung. Diese Werte können nicht leer bleiben, daher ist ein Alpha von 10% und eine zweiseitige Testrichtung vorausgewählt.

In der Tabelle kann man für alle durchgeführten Skalen deren Roh- und Normwerte eintragen. Dies erfolgt in folgenden Schritten:

1. Zelle, in der was eingetragen werden soll, mit Doppelklick fokussieren.
2. Wert eintragen
3. ENTER-Taste drücken!

Wird die ENTER-Taste nicht gedrückt, wird der Wert nicht angenommen.

Es kann auch nur der Roh- oder Normwert eingegeben werden. Das Programm filtert automatisch Alle Testfälle raus, in denen es einen Wert findet.

Durch das Bestätigen des Anlegens wird man auf die Fall-Übersicht des soeben angelegten Falls weitergeleitet.

3.6 Fall-Übersicht

Die Fall-Übersicht zeigt die zugehörige Personen-ID, die angegebenen Test-Variablen, sowie eine Tabelle der angelegten Messwerte des Falls.

In der Tabelle werden die mit der Person durchgeführten Skalen (Messwerte) angezeigt. Das Programm berechnet automatisch die Konfidenzintervalle der Messwerte, vorausgesetzt es stehen ihm alle relevanten Variablen zur Verfügung (Normwert, und Reliabilität) und der Normskalen-Typ ist für eine Berechnung geeignet (alle außer Prozentrang).

Durch einen Doppelklick auf einen Messwert in der Tabelle, werden die Infos zu dem entsprechenden Messwert und der KI-Berechnung geöffnet. Zudem gibt es die Option einen Messwert zum Fall hinzuzufügen durch das klicken des *"Messwert hinzufügen"*-Buttons.

Um sich das Konfidenzintervall-Übersichts-Diagramm anzeigen zu lassen, muss der *Zur Konfidenzintervall-Grafik*-Button geklickt werden.

Es besteht zudem Die Möglichkeit einen Fall zu exportieren. Hierfür muss der *Fall exportieren*-Button geklickt werden. Es öffnet sich ein Fenster, an

dem der Export-Ordner ausgewählt werden kann. Sollte es Probleme beim Export geben, wird man mit einem Error darauf hingewiesen, dass der Export nicht erfolgen konnte.

Ist der Export erfolgeich, wird eine Export-Bestätigung angezeigt mit dem Namen und Ort der abgespeicherten Datei.

Um die Fall-Daten, sowie die Roh- und Messwerte, zu Ändern, wird man durch das Klicken des *"Fall-Werte bearbeiten"*-Buttons entsprechend weitergeleitet.

3.7 Fall-Daten bearbeiten

Hier lassen sich alle Variablen des Falls anpassen, abgesehen vom Testdatum.

Zudem besteht die Möglichkeit den Fall zu löschen. Hierbei werden auch alle angelegten Messwerte des Falls mitgelöscht.

In der Tabelle kann man die Roh-und Normwert-Spalten bearbeiten, indem man folgende Schritte befolgt:

1. Zelle, in der was eingetragen werden soll, mit Doppelklick fokussieren.
2. Wert eintragen
3. ENTER-Taste drücken!

Wird die ENTER-Taste nicht gedrückt, wird der Wert nicht angenommen.

Die Normskala wird lediglich zur Hilfe beim Eintragen des Normwertes angezeigt, lässt sich jedoch nur beim Bearbeiten des Messwerts selbst anpassen. Dies wurde so umgesetzt, da bei einer Änderung der Normskala auch der Normwert automatisch transfomiert wird, wenn möglich.

Nach dem Speichern der Änderungen wird man zurück zur Fall-Übersicht geleitet. Die Tabelle und deren Werte werden bei Änderungen entsprechend angepasst und neu berechnet.

3.8 Messwert hinzufügen

Um einen weiteren messwert zu einem Fall hinzuzufügen, muss in der Fall-Übersicht der *"Messwert hinzufügen"*-Button geklickt werden. Man wird weitergeleitet zu einem Eingabeformular mit allen Infos zu einem Messwert.

Um einen Messwert eindeutig zu indentifizieren, muss die Skalen-Abkürzung, ähnlich wie die Personen-ID, innerhalb eines Falls eindeutig sein. Daher kann ein Fall nur einen Messwert mit einer bestimmten Skalen-Abkürzung haben.

Ebenfalls müssen das Testverfahren, sowie der Skalen-Name angegeben werden, da diese Werte nicht nachgetragen werden können. Außerdem muss ein Normskalen-Typ ausgewählt werden, um den Normwert korrekt zu interpretieren. Alle anderen Daten können leer gelassen oder angegeben werden.

Wurde das Anlegen des Messwerts bestätigt, wird man zurück zur die Fall-Übersicht geleitet. Wurden alle relevanten Daten für die KI-Berechnung angegeben und der Normskalen-Typ ist geeignet für die Berechnung, wird das Intervall für den neu hinzugefügten Messwert automatisch berechnet und in der Tabelle angezeigt.

3.9 Messwert-Übersicht

Die Messwert-Übersicht zeigt die angegebenen Messwert-Variablen des ausgewählten Messwerts, sowie eine KI-Grafik des Messwerts auf der entsprechenden Norm-Skala, die benutzte Formel sowie der resultierende Ausdruck durch einsetzen der Werte (sofern das KI berechnet wurde). Zudem werden alle relevanten Infos angezeigt, die benötigt werden, um die Berechnung des KIs nachvollziehen zu können.

Um die Messwert-Daten zu Ändern, wird man durch das Klicken des *"Daten bearbeiten"*-Buttons entsprechend weitergeleitet.

3.10 Messwert-Infos bearbeiten

Hier lassen sich alle Messwert-Infos anpassen, abgesehen von den Testverfahren/Skalen-Namen und der Abkürzung, sowie des Roh- und Normwerts. Der Roh-

und Normwert müssen über die Fall-Daten bearbeitet werden, da bei Änderung der Normskala der Normwert automatisch transfomiert wird, sofern möglich, und die Änderungen eindeutig nachvollziehbar sein müssen. Zulässige Transfomrationen sind zwischen T-Wert, z-Wert und IQ möglich, alle anderen Normskalen-Änderungen resultieren darin, dass der Normwert entfernt wird, da die Umrechnung nicht bekannt ist. Entsprechend muss der korrekte Normwert bei der Bearbeitung der Falldaten nachgetragen werden.

Pflichtfeld ist hier lediglich die Angabe eines validen Normskalen-Typs, ansonsten können alle anderen Angaben leer sein.

Zudem besteht die Möglichkeit den Messwert zu löschen.

3.11 Konfidenzintervall-Grafik

Um sich die Konfidenzintervall-Grafik eines Falls anzeigen zu lassen, muss der entsprechende Fall der Person geöffnet werden. In der Fall-Übersicht wird durch das klicken des *"Zur Konfidentintervall-Grafik"*-Buttons zum Diagram weitergeleitet.

Das Diagramm zeigt alle Messwerte an, bei denen ein Konfidenzintervall berechnet werden konnte und transformiert alle Werte auf die z-Skala. Die Reihenfolge ist die Gleiche, wie in der Fall-Übersichts-Tabelle. Zudem wird das benutzte Alpha und die Testrichtung angezeigt.

Um die Transformation nachvollziehbar zu machen, kann man sich die Werte in einer Tabelle anzeigen lassen. Dafür muss der *"Tabelle mit umgerechneten Werten"*-Button geklickt werden.

3.12 Neues Testverfahren in die Import-Tabelle einfügen

Um ein Testferfahren, oder auch nur eine einzelne Skala, in die Tabelle hinzuzufügen, müssen folgende Schritte erfolgen:

1. Datei **ImportTabelleTestverfahren.xlsx** im Ordner *importResources* öffnen.
2. Die Tabelle um beliebige Anzahl Zeilen erweitern (An der rechten, unteren Ecke der Tabelle klicken und nach unten ziehen)

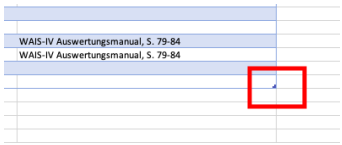


Figure 18: *Hier klicken, um Tabelle zu erweitern*

3. Testverfahren-/Skalen-Werte eintragen

   Zulässige Normskalen:
   T-Wert, z-Wert, IQ, Wertpunkte, Prozentrang, Stanine-Normwerte

   Zulässige Reliabilitäts-Methoden:
   Odd-Even, Split-Half, Cronbach-Alpha, Paralleltest, Retest

   (Die angegebenen Werte müssen genau so geschrieben werden, sonst können sie nicht korrekt erfasst werden)

4. Datei speichern und schließen!

Und schon sollten die Änderungen beim nächsten Start des Programms integriert werden.

Wichtig!:
Die Skalen-Abkürzung muss in der gesamten Tabelle eindeutig sein! Existiert bereits eine Skala mit der gleichen Abkürzung, wird die Skala beim Import ignoriert!
Grund hierfür ist, dass anhand der Skalen-Abkürzung ein Messwert eindeutig identifiziert werden kann. Würde eine Abkürzung öfter vorkommen, ist diese eindeutige Zuordnung nicht mehr gegeben und das Programm könnte nicht mehr wissen, welche Skala genau gemeint ist.

Auch Wichtig!:
Das Programm orienitert sich an den Testverfahren-Namen. Es wird ein Testverfahren nach dem anderen imporiter, daher müssen Skalen eines Testverfahrens direkt untereinander stehen! Wird kein Testverfahren-Name angegeben, interpretiert es dies als zugehörig zu dem zuletzt bekannten Testverfahren.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Entwicklung eines Tools zur Unterstützung der Analyse und Interpretation neuropsychologischer Diagnostik

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original