Dokumentation der praktischen Arbeit zur Prüfung zur mathematisch-technischen Softwareentwicklerin

12. Mai 2022

Johanna Schindler

Prüfungs-Nummer: 142 - 18741

Programmiersprache: Java

Inhalt

1 AUFGABENANALYSE 1

1.1 Allgemeines 1
1.2 Eingabe 1
1.3 Auswertung der Messwerte 2
1.3.1 Umrechnung und Normierung der Daten 3
1.3.2 Glättung der Daten 3
1.3.3 Obere Einhüllende 4
1.3.4 Pulsbreite 5
1.4 Ausgabe 5
1.5 Parallelisierung 5
1.6 Grenz- und Fehlerfälle 6
1.6.1 Grenzfälle 6
1.6.2 Sonderfälle 7
1.6.3 Fehlerfälle 7
1.7 Anforderungsliste 9

2 VERFAHRENSBESCHREIBUNG 10

2.1 Eingabe und Initialisierung 11
2.2 Verarbeitung und Auswertung der Messwerte 11
2.2.1 Konvertierung 12
2.2.2 Auswertung der Messwerte 12
2.3 Ausgabe 15

3 PROGRAMMBESCHREIBUNG 17

3.1 Grobentwurf 17
3.1.1 Module 17
3.1.2 Ein- und Ausgabe 19
3.1.3 Model 20
3.1.4 Controller 21
3.2 Ablauf 21
3.3 Auswertungsmethoden 24
3.3.1 Normieren und Umrechnen 24
3.3.2 Glätten 25
3.3.3 Obere Einhüllende berechnen 26
3.3.4 Pulsbreite berechnen 27
3.4 Threads 28
3.4.1 Einleser 28
3.4.2 Verarbeiter 28
3.4.3 Ausgeber 29
3.5 Ein- und Ausgabekonvertierung 30
3.5.1 Eingabe 30
3.5.2 Ausgabe 31

4 TESTDOKUMENTATION 32

4.1 Technische Fehler 32

5 ZUSAMMENFASSUNG UND AUSBLICK 34

5.1 Zusammenfassung 34
5.2 Ausblick 34

5.2.1 Ein- und Ausgabe 34
5.2.2 Nebenläufigkeiten 35
5.2.3 Graphische Oberfläche 35
5.2.4 Auswertung 36

6 ABBILDUNGSVERZEICHNIS 37

7 LITERATURVERZEICHNIS 37

A ABWEICHUNGEN UND ERGÄNZUNGEN ZUM VORENTWURF 38

B BENUTZERANLEITUNG 39

A. VERZEICHNISSTRUKTUR 39
B. SYSTEMVORRAUSSETZUNGEN 39
C. INSTALLATION 40
D. KOMPILIEREN 40
E. PROGRAMMAUSFÜHRUNG 40
F. TESTAUSFÜHRUNG 40

C ENTWICKLERDOKUMENTATION 42

A. VERFÜGBARE KLASSEN UND SCHNITTSTELLEN 42
B. NUTZUNGSHINWEISE 42
C. FEHLERBEHANDLUNG 43

D ENTWICKLUNGSUMGEBUNG 45

E VERWENDETE HILFSMITTEL 45

F ERKLÄRUNG 1

1 Aufgabenanalyse

1.1 Allgemeines

Im Rahmen dieser Arbeit soll ein Software-System entwickelt werden, welches die Signalauswertung für einen optischen Autokorrelator vornimmt und dabei Messwerte mittels mathematischen Methoden verarbeitet. Die zu verarbeitetenden Messwerte werden dabei aus einem optischen Autokorrelator geliefert und beinhalten die Intensität eines Signals und die Position eines Spiegels, der das Signal reflektiert.

Das System nimmt zunächst simulierte Messdaten von einer Signaldatenquelle entgegen und bestimmt daraus pro Datensatz die obere Einhüllende und die Pulsbreite, nachdem die Messwerte normiert, umgerechnet und geglättet wurden. Diese Eingabe erfolgt dabei in regelmäßigen Intervallen, sodass das Programm in unabhängigen Threads implementiert wird. In der Zukunft soll ein optischer Autokorrelator die Daten in regelmäßigen Abständen liefern.

Das Software-System soll nach dieser Eingabe im Folgenden die mathematische Auswertung vornehmen und die normierten und geglätteten Daten, sowie Pulsbreite und obere Einhüllende zurückgeben.

1.2 Eingabe

Dazu müssen zunächst die Datensätze eingelesen werden. Dies soll in Textdateien im geschehen, welche die Dateiendung „.txt“ aufweisen müssen. Alle Zeilen, welche mit „#“ beginnen, werden als Kommentar betrachtet und ignoriert. Auch Leerzeilen werden vom Programm übersprungen.

Eingabedateien

Die Eingabedateien sollen nicht im Rahmen dieses Prüfproduktes erstellt werden, sondern wurden per Download zur Verfügung gestellt.

Die Eingabedateien beinhalten jeweils alle Messwerte zu einer Messreihe (im Folgenden Datensatz genannt) in zwei Spalten: die Intensität des Messsignals aus dem Detektor, also `y` und nach einem oder mehreren Tabulatoren die Position des Spiegels `x_schlange`. Diese werden als zwei nicht-negative Ganzzahlen, geliefert.

Jeder Datensatz enthält N Messdaten und es werden zunächst zehn mit „0.txt“ bis „9.txt“ benannte Eingabedateien eingelesen. Da es sich hier zunächst um simulierte Daten handelt und in der Zukunft ein Autokorrelator in regelmäßigen Zeitabschnitten einen Datensatz liefert, soll diese Anzahl und Benennung jedoch variabel sein.

Eine gültige Eingabedatei („0.txt“) könnte wie folgt aussehen:

```
# int   pos
20065  127505
20110  127506
20074  127508
…
```

Abbildung 1: Beispiel einer gültigen Eingabedatei („0.txt“)

Aufgrund der zukünftigen Anwendung soll das Programm so entwickelt werden, dass die regelmäßig gelieferten Dateien ausgelesen und für 0.05 Sekunden zur Verfügung gestellt werden, da der Autokorrelator die Signale mit einer Frequenz von 20 Hz empfängt. Das Auslesen der Daten soll ein nur für das Einlesen zuständiger Thread übernehmen, der alle gegebenen Eingabedateien nacheinander ausliest und wieder von vorne beginnt, sobald er die letzte ausgelesen hat, so lange, bis jeder Datensatz verarbeitet wurde.

1.3 Auswertung der Messwerte

Damit die Auswertung parallel zum ständigen Auslesen der Dateien passieren kann, soll sie ebenfalls in einem eigenen Thread entwickelt werden. Zu der Auswertung gehört das Initialisieren der Datenstrukturen und die Auswertung anhand der mathematischen Modelle, die im Folgenden vorgestellt werden.

Zum Initialisieren der Datenstrukturen wird der Inhalt der aktuell zur Verfügung gestellten Eingabedatei zeilenweise eingelesen und die vorher beschriebenen Werte aus den Zeilen ausgelesen, geprüft und in die vorgesehene Datenstruktur überführt. Im Anschluss werden mithilfe von vier mathematischen Methoden eine Auswertung der Messwerte vorgenommen.

Bemerkung zu Abkürzungen

Die Benennung der Variablen soll erst in der Verfahrensbeschreibung geschehen. Jedoch handelt es sich bei der Auswertung um eine Abarbeitung mehrerer mathematischer Methoden, welche im Folgenden zu Übersichtszwecken auch teils mit selbst benannten Abkürzungen beschrieben werden. Die Benennung bezieht sich dabei auf das in der Aufgabenstellung verwendete Symbol: $\tilde{x}$, $\hat{x}$, x und y

1.3.1 Umrechnung und Normierung der Daten

$$\hat{x}_k = \frac{\tilde{x}_k}{2^{18} - 1} * 266.3 - 132.3 \quad , \forall\, k \in [0, N-1]$$

Mittels dieser Formel werden die $\tilde{x}$ -Werte in Pikosekunden, kurz ps, umgerechnet. Diese Werte werden im Folgenden $\hat{x}$ genannt.

Außerdem sollen die y -Werte normiert werden, indem sie durch das Maximum aller y -Werte dividiert werden. Dieses muss zunächst bestimmt werden.

$$y_{k\,normiert} = \frac{y_k}{y_{max}}$$

1.3.2 Glättung der Daten

Im nächsten Schritt soll eine Glättung der Daten geschehen, da sie zunächst nicht stetig vorliegen, was auf Messungenauigkeiten zurückzuführen ist. Damit die Kurve der Daten glatter wird, soll für jeden Messwert $\hat{x}$ der gleitende Mittelwert x bestimmt werden, indem die $\hat{x}$-Werte in einer Umgebung von 0.2% der Messdaten um $\hat{x}$gemittelt werden.

Dafür muss zunächst n, das 0.2% der Messdaten große ungerade Mitteilungsfenster, bestimmt werden.

$$n = \begin{cases} \lfloor 0.002 * N \rfloor - 1 & , für\ \lfloor 0.002 * N \rfloor\ gerade \\ \lfloor 0.002 * N \rfloor & , für\ \lfloor 0.002 * N \rfloor\ ungerade \end{cases}$$

Zur Berechnung wird außerdem der Wert τ als Entfernung vom aktuellen Punkt zum Rand des Mitteilungsfensters benötigt, der aus der Größe des Mitteilungsfensters bestimmt, welche Werte links und rechts von $\hat{x}$ für die Berechnung von x verwendet werden sollen.

$$\tau = \frac{n-1}{2} \quad , \forall\ k \in [0, N-1]$$

$$x_k = \frac{1}{n} \sum_{i=0}^{n-1} \hat{x}_{k-\tau+i}$$

Im Sonderfall, dass rechts oder links von $\hat{x}$ weniger als τ Werte existieren, man sich also am rechten oder linken Rand der Datenreihe befindet, soll x wie folgt berechnet werden:

Für den linken Rand sollen alle Werte vom Index 0 bis zum Index n_1* zur Mittelung herangezogen werden, wobei n_1* die Differenz zwischen n und der Anzahl an Werten ist, die links fehlen, um τ Werte aufaddieren zu können.

Für den rechten Rand sollen alle Werte vom Index n_2* bis zum Index N-1 zur Mittelung herangezogen werden, wobei n_2* die Differenz zwischen n und der Anzahl an Werten ist, die rechts fehlen, um τ Werte aufaddieren zu können.

1.3.3 Obere Einhüllende

Weiterhin soll eine Annährung der oberen Einhüllenden der Messwerte bestimmt werden. Dafür wird jedem x -Wert ein $y_{einhuellende}$ -Wert zugewiesen, der den zuletzt höchsten y -Wert beinhaltet. Dabei wird von links bis zum Maximum und danach von rechts bis zum Maximum vorgegangen.

1.3.4 Pulsbreite

$$b = L - R$$

Als letztes soll die Pulsbreite b, auch „full width at half maximum“ (FWHM) genannt, als Differenz der x -Werte zweier Punkte L und R berechnet werden. L bzw. R ist dabei der erste Punkt links bzw. rechts vom Maximum, dessen $y_{einhuellende}$ die Hälfte der Höhe zwischen dem Maximum und der Grundlinie erreicht. Die Grundlinie wird als mittlerer y -Wert der äußersten linken 1% der der Messwerte definiert. Diese muss zunächst berechnet werden.

1.4 Ausgabe

Nachdem ein Datensatz ausgewertet wurde, sollen die Ergebnisse wieder in eine Textdatei ausgegeben werden, ebenfalls in einem eigenen Thread.

Die zu erstellende Ausgabetextdatei erhält den Namen der Eingabedatei mit dem Präfix „out“ und könnte für die in Abbildung 1 aufgezeigte Eingabedatei wie folgt aussehen:

```
# FWHM = 0.26781982, 31738, 33831
# pos  int       env
-2.7742696 0.1660625     0.1660625
-2.7742817 0.16643493    0.16643493
…
```

Abbildung 2: Beispiel einer Ausgabe in einer Textdatei („out0.txt“)

In der ersten Zeile steht dabei das Ergebnis des letzten Schrittes 1.3.4 der Berechnung, nämlich die Pulsbreite als Float und die Indizes der Punkte L und R als Integer. Die Inhaltszeilen bestehen aus drei Spalten: den x -Werten, den normierten y -Werten und der $y_{einhuellenden}$. Zwischen den ersten beiden Spalten befindet sich ein Tabulator, zwischen der 2. und 3. zwei Tabulatoren zur Trennung. Diese Werte sind ebenfalls als Float angegeben.

1.5 Parallelisierung

Durch die Anwendung bei einem Autokorrelator, der in Abständen von 0.05 Sekunden einen neuen Datensatz zur Verfügung stellt, soll das Programm in unabhängigen Threads implementiert werden. Dabei ist der erste Thread für das Lesen und immer wieder Überschreiben des Dateiinhaltes zuständig, indem über die Dateien iteriert wird, bis sie alle verarbeitet wurden.

Der zweite Thread nimmt den vom ersten Thread bereitgestellten Inhalt wie oben beschrieben. Der dritte Thread liest bereits verarbeitete Datensätze mit ihren Ergebnissen aus. Das Verhalten der Nebenläufigkeiten, sowie deren Kommunikation soll im Folgenden implementiert werden.

1.6 Grenz- und Fehlerfälle

Während des Programmablaufs, das heißt beim Einlesen, Initialisieren, Verarbeiten und Ausgeben können verschiedene Grenz- und Fehlerfälle auftreten, die im Folgenden beschrieben werden.

1.6.1 Grenzfälle

1. Es wird nur ein Datensatz eingegeben.
2. Der kleinstmögliche (sinnvolle) Datensatz mit 100 Messwerten wird eingegeben.
    - Damit bei im 4. Berechnungsschritt bei der Berechnung der Grundline 1% der Daten mindestens 1 Datensatz ist.
3. Die größtmögliche Anzahl von Messwerten in einem Datensatz N = 2147483647 (maximaler Integerwert) wird eingegeben.
    - Damit das Programm in sinnvoller Zeit die Berechnung beenden kann, sollte vermutlich ein kleinerer Wert gewählt werden. Da aber keine weiteren Testfälle gegeben oder gefordert waren, wird hier nur dieser Wert angegeben.
4. Die größtmögliche Anzahl von Datensätzen M = 2147483647 (maximaler Integerwert) wird eingegeben.
    - Damit das Programm in sinnvoller Zeit die Berechnung beenden kann, sollte vermutlich auch hier ein kleinerer Wert gewählt werden. Da aber keine weiteren Testfälle gegeben oder gefordert waren, wird nur dieser Wert angegeben.

1.6.2 Sonderfälle

1. Es werden alle Datensätze in der ersten Iteration über die Dateien eingelesen, verarbeitet und ausgegeben.
2. Beim Einlesen eines besonders großen Datensatzes dauert das Lesen länger als 0.05 Sekunden.
3. Zu Beginn des Programms liegt kein eingelesener Datensatz vor, da der Einlesethread das Lesen des ersten Datensatzes noch nicht beendet hat.
4. Zu einem Zeitpunkt liegen keine auszulesenden Datensätze vor, da der Verarbeitungsthread gerade noch einen verarbeitet.

1.6.3 Fehlerfälle

Die auftretenden Fehler kommen üblicherweise beim Einlesen und Initialisieren der Datenstrukturen vor und können in drei Fehlerarten aufgeteilt werden. Man unterscheidet zwischen technischen, syntaktischen und semantischen Fehlern. Das Vorkommen eines solchen Fehlers sorgt für einen Programmabbruch.

1.6.3.1 Technische Fehler

Technische Fehler können beim Zugriff auf die Ein- und Ausgabedateien auftreten. Dabei können folgende Fehler auftreten:

1. Es wurde kein Eingabeordner angegeben.
2. Der Eingabeordner existiert nicht.
3. Eine Eingabedatei ist nicht lesbar.
4. Eine Ausgabedatei ist nicht beschreibbar.

Da das Anlegen einer Ausgabedatei bei technischen Fehlern unter Umständen nicht möglich ist, werden diese auf der Standardfehlerausgabe des Systems ausgegeben.

1.6.3.2 Syntaktische Fehler

Syntaktische Fehler liegen vor, wenn die Eingabedatei nicht das vorgegebene Format und in Abschnitt 1.21.2 beschriebene hat.

1. Ein Kommentar ist nicht durch vorangestelltes „#“ gekennzeichnet.
2. Es wurden keine Messwerte eingegeben, also eine Datei ist leer oder besteht nur aus Kommentarzeilen.

Syntaktische Fehler erzeugen eine Fehlermeldung, welche anstatt der eigentlichen Programmausgabe in die Konsole und in die Ausgabedatei geschrieben wird.

1.6.3.3 Semantische Fehler

1. Einer der Messwerte in einer Datei ist keine natürliche Zahl. (nicht-negative Ganzzahl)
2. Es wurden in einer Datei nicht genau zwei Messwerte angegeben.
3. In einer Datei wurden zu wenige Messwerte eingegeben, also eine Datei beinhaltet weniger als 100 Messwerte.

Semantische Fehler erzeugen ebenfalls eine Fehlermeldung, welche anstatt der eigentlichen Programmausgabe in die Konsole und in die Ausgabedatei geschrieben wird.

1.7 Anforderungsliste

Der Übersicht halber wurden die in den vorherigen Abschnitten herausgearbeiteten Anforderungen an das Programm hier noch einmal stichpunktartig aufgeführt.

1. Das Programm liest beliebig viele Textdateien im gewünschten Format ein.
2. Bei technischen Fehlern gibt das Programm eine geeignete Fehlermeldung in der Standardfehlerausgabe des Systems aus.
3. Im Falle der genannten semantischen und syntaktischen Fehlerfälle gibt das Programm eine Fehlermeldung in der Standardfehlerausgabe des Systems und in einer Ausgabedatei aus.
4. Das Programm ermittelt für jeden gegebenen Datensatz die gewünschten Werte.
5. Das Programm gibt das ermittelte Ergebnis für jeden gegebenen Datensatz im gewünschten Format in der Konsole und in einer Ausgabedatei aus.
6. Das Programm führt die Schritte 1.-5. nicht sequentiell aus, sondern in drei Threads, die wie beschrieben alle Dateien in einem gegebenen Ordner abarbeiten.

2 Verfahrensbeschreibung

Da die Aufgabe nun detailliert beschrieben und analysiert wurde, wird im Folgenden der Ablauf des Programms näher erläutert, wobei technische Details verfeinert und Abläufe konkretisiert werden. Besonders die Nebenläufigkeiten Einlese-, Verarbeitungs- und Ausgebethread sollen dabei näher erläutert werden, sodass deutlich wird, in welchem Thread welcher Teil der Verarbeitung durchgeführt wird, wann dieser seine Arbeit beendet oder wartet und wie die Organisation des Dateiflusses implementiert wurde.

Zur Übersicht werden hier die verwendeten geteilten Variablen aufgeführt:

1. *unverarbeitete Eingabedateien* (Liste von Zeichenketten)
2. *verarbeitete Datensätze* (Liste von Datensätzen)
3. *geschlossene Eingabedateien* (Liste von Zeichenketten)
4. *aktuell gelesener Eingabedateiinhalt* und *aktuell gelesener Eingabedateiname* (Zeichenketten)

Auf diese Variablen muss von mehreren Threads zugegriffen werden:

- Einlese-Thread greift auf 1. und 4. zu
- Verarbeiter-Thread greift auf 1., 2. und 4. zu
- Ausgeber-Thread greif auf 2. und 3. zu

Dadurch, dass die Listeninhalte von 1. – 3. während des Programmablaufs stets verändert werden, handelt es sich nicht um Felder fester Länge, sondern Listen.

Gemeinsamer Zugriff auf Variablen

Sowohl die geführten Listen an *unverarbeiteten Dateien*, *verarbeiteten Datensätzen* und *geschlossenen Dateien*, als auch die Zeichenketten für den *Eingabedateinamen* und -*inhalt* werden synchronisiert. Dadurch können mehrere Threads sicher auf diese zugreifen und es kann nicht vorkommen, dass die Threads gegenseitig voneinander abhängen. Insbesondere deswegen werden die Listen zwischengespeichert.

2.1 Eingabe und Initialisierung

Der Eingabeordner wird dem Programmaufruf als Parameter mitgegeben. Nach dem Start des Programms überprüft dieses das Vorhandensein des angegebenen Eingabeordners. Ist dieser vorhanden, wird der Thread zum Einlesen gestartet. Solange es noch unverarbeitete Datensätze gibt, iteriert er immer wieder über diese, liest den Inhalt jeweils zeilenweise aus und speichert ihn als eine Zeichenkette, wobei führende und abschließende Leerzeichen ignoriert werden. Auch der Eingabedateiname wird als Zeichenkette gespeichert.

Sollten technische Fehler beim Einlesen einer der Dateien auftreten, wird das Programm abgebrochen und eine Fehlermeldung, wie in Abschnitt 1.6.3 beschrieben, ausgegeben. Nachdem ein Inhalt ausgelesen wurde, wird dieser für 0.05 Sekunden gespeichert, bis die nächste Datei eingelesen wird. Dies geschieht, indem der Thread für diesen Zeitraum pausiert.

Die Überprüfung, ob es noch unverarbeitete Datensätze gibt, geschieht mithilfe einer Liste, welche die Namen der noch *unverarbeiteten Eingabedateien* beinhaltet. Initial werden hier die Dateinamen aller Eingabedateien, die sich im Eingabeordner befinden, gespeichert. Wird vom Verarbeitungsthread im nächsten Schritt ein Datensatz weiterverarbeitet, so wird der zugehörige Dateiname aus dieser Liste entfernt. So wird verhindert, dass eine Datei unnötigerweise erneut eingelesen wird, auch wenn sie schon vom Verarbeitungsthread verarbeitet wurde.

Sollte diese Liste die leer sein, so ist nichts mehr einzulesen und der Thread beendet sich.

2.2 Verarbeitung und Auswertung der Messwerte

Der zweite Thread wird für das Verarbeiten des Inhalts der Eingabedateien verwendet. Wird dieser direkt nach dem Einlesethread gestartet, so wartet er zunächst 0.05 Sekunden, um dem Einlesethread einen Vorsprung und somit die Möglichkeit zu geben, die erste Eingabedatei zu lesen. Dann fährt er fort, indem er sich den Inhalt aus den gespeicherten Variablen für *Dateiname* und -*inhalt* kopiert. Dies verhindert, dass der Einlesethread den Inhalt der Variablen verändert, während darauf zugegriffen wird.

Steht der Inhalt einer Datei zur Verfügung, der noch nicht verarbeitet wurde, so wird dieser zunächst aus der Liste der *unverarbeiteten Datensätze* gelöscht und dann so konvertiert, dass die späteren Datenstrukturen gesetzt werden können.

Startet der Verarbeitungsthread die Verarbeitung eines Inhaltes und löscht diesen aus der Liste *unverarbeiteter Dateien*, kann es sein, dass der Einlesethread diesen trotzdem noch einmal ausliest. Das kommt aufgrund eines Zwischenspeicherns der Liste auf einer temporären Liste zustande. Daher darf im Verarbeitungsthread nicht blind alles verarbeitet werden, was der Einlesethread liefert, sondern es muss geprüft werden, ob dieser Inhalt tatsächlich noch unverarbeitet, also Element der Liste ist.

Wurde der aktuell gelieferte Inhalt also noch nicht verarbeitet, wird er wie folgt konvertiert:

2.2.1 Konvertierung

Der Verarbeiter durchläuft er den Inhalt und ignoriert jegliche Kommentarzeilen. Eine Zeile wird dann an vorkommenden Tabulatoren getrennt. Ergibt diese Trennung genau zwei Zeichenketten, wird geprüft, ob diese in eine natürliche Zahl umgewandelt werden können. Ist dem so, können diese Werte als `x_schlange` und `y` gespeichert, in eine Liste hinzugefügt und fortgefahren werden, bis das Ende des Inhalts erreicht wurde.

Wurden auf diese Art mindestens 100 Messwerte, jeweils mit `x_schlange` und `y`, erstellt, es wurden also keine syntaktischen oder semantischen Fehler wie in Abschnitt 1.6.3 beschrieben festgestellt, kann aus der Liste von Messwerten und dem Namen der Eingabedatei ein Datensatz erstellt werden. Dafür wird die Liste in ein Feld fester Größe umgewandelt, da sich nach der initialisierung die Anzahl an Messwerten pro Datensatz nicht mehr verändert. Kommt es doch zu syntaktischen oder semantischen Fehlern, wird das Programm abgebrochen und eine Fehlermeldung, wie in Abschnitt 1.6.3 beschrieben, ausgegeben.

Der Konstruktor des Datensatzes setzt die zu speichernden Daten, also Messwerte und Dateiname und berechnet und speichert sich aus den Messwerten die Anzahl `N` und den Index des maximalen Messwerts `maxInd`, also jenem Messwert, dessen `y` -Wert maximal ist.

2.2.2 Auswertung der Messwerte

Die mathematischen Methoden zur Auswertung eines Datensatzes, wurden in der Aufgabenstellung und in Abschnitt 1.3 beschrieben. Daher wird im Folgenden lediglich auf die Bestimmung und Speicherung benötigter Werte detailliert eingegangen.

2.2.2.1 Umrechnung und Normierung

1. Speichere den y -Wert des Messwertes aus der Liste mit Index maxInd als Dezimalzahl zwischen. (Es ist zwar eine Ganzzahl, aber zur Normierung ist das wichtig, damit keine Ganzzahldivison auftritt.)
2. Durchlaufe alle Messwerte in der Liste
    a. speichere pro Messwert x_hut als Dezimalzahl (Berechnung siehe 1.3.1)
    b. normiere pro Messwert den y -Wert (Berechnung siehe 1.3.1)
        i. Da y hier normiert wird, wird y schon im Konstruktor als Dezimalzahl gesetzt.

2.2.2.2 Glättung der Daten

1. Berechne n und speichere als Ganzzahl zwischen. (Berechnung siehe 1.3.2)
2. Berechne τ und speichere als Ganzzahl zwischen. (Berechnung siehe 1.3.2)
3. Durchlaufe alle Messwerte in der Liste und speichere pro Messwert den x -Wert (Berechnung und Behandlung der Ränder siehe 1.3.2)

2.2.2.3 Obere Einhüllende

1. Speichere den Index maxInd als Ganzzahl und den zugehörigen y -Wert als Dezimalzahl zwischen
2. Setze eine lokale Variable y_tempMax auf den ersten y -Wert der Liste und durchlaufe die Liste von oben bis zum maxInd
    a. speichere pro Messwert den bisher größten y -Wert wie folgt:
        i. Ist der y -Wert des Messwertes größer als der bisher größte (y_tempMax)? Dann setze y_tempMax = y -Wert

ii. Setze für diesen Messwert y_einhuellende = y_tempMax

3. Setze die lokale Variable y_tempMax auf den letzten y -Wert der Liste und durchlaufe die Liste von unten hoch bis zum maxInd
    a. Verfahre wie zuvor

2.2.2.4 Pulsbreite

1. Speichere den Index maxInd als Ganzzahl und den zugehörigen y -Wert als Dezimalzahl zwischen.
2. Berechne die Grundlinie wie folgt
    a. Berechne die Anzahl an Messwerten, die zur Berechnung der Grundlinie verwendet werden sollen:
    b.

$$n_{grundlinie} = \begin{cases} \lfloor 0.01 * N \rfloor \; abgerundet \; , f\ddot{u}r \; \lfloor 0.01 * N \rfloor \; abgerundet \; > 0 \\ 1 \quad\quad , f\ddot{u}r \; \lfloor 0.01 * N \rfloor \; abgerundet = \; 0 \end{cases}$$

    c. Durchlaufe die Messwerte in der Liste bis zum Index n_grundlinie, addiere deren y -Werte als Dezimalzahlen auf und teile durch n_grundlinie
    d. Speichere das Ergebnis als Dezimalzahl auf grundlinie
3. Durchlaufe die Messwerte in der Liste vom Index maxInd bis zum Index N (nach rechts)
    a. Speichere den Index des ersten Messwertes, bei Folgendes gilt als indR:

$$y_k \leq \frac{1}{2} * (y_{maxInd} - grundlinie)$$

4. Durchlaufe die Messwerte in der Liste vom Index maxInd bis zum Index 0 (nach links) und verfahre genau so
5. Berechne die Pulsbreite als Differenz der x -Werte von R und L

Nachdem die Berechnung der mathematischen Methoden abgeschlossen wurde, wird der verarbeitete Datensatz der Liste von *verarbeiteten Datensätzen* hinzugefügt. Innerhalb des Datensatz-Objektes befinden sich alle für die Ausgabe relevanten Informationen.

Es wird so lange fortgefahren, bis die Liste an noch *unverarbeiteten Dateien* leer ist. Dann gibt es keine Datensätze mehr einzulesen und somit auch nichts mehr zu verarbeiten.

Sollte zu einem Zeitpunkt die Liste an *verarbeiteten Datensätzen* leer sein, obwohl noch nicht alle Dateien geschlossen sind, so befindet sich mindestens ein Datensatz noch im Lese- bzw. Verarbeitungsschritt. Daher wird in diesem Fall eine kurze Zeit gewartet und erneut auf die Liste zugegriffen.

2.3 Ausgabe

Die Ausgabe der *verarbeiteten Datensätze* ist ebenfalls in einen Thread ausgelagert, der die Konvertierung in eine Zeichenkette und das Schreiben des Inhalts in eine Datei übernimmt. Dafür überprüft der Ausgeber, ob die bereits alle Datensätze verarbeitet und ausgegeben wurden, indem er auf die Liste an *geschlossenen Dateien* zugreift. Diese Liste von Zeichenketten beinhaltet die Dateinamen aller Eingabedateien, die bereits eingelesen, verarbeitet und in eine Ausgabedatei geschrieben wurden. Sollte diese Liste die maximale Länge erreicht haben, also so viele Elemente beinhalten, wie anfangs die Liste unverarbeiteter Eingabedateien, so ist nichts mehr auszugeben und der Thread beendet sich.

Ansonsten wird die Zeichenkette des Ausgabeinhalts erstellt:

1. Die erste Zeile wird gefüllt mit

   „`# FWHM = <fwhm: float>, <indexL: int>, <indexR: int>`“
2. Die zweite Zeile enthält die Spaltennamen, getrennt durch ein, bzw. zwei Tabulatoren:

   „`# pos	int		env`“
3. Danach folgt für jeden Messwert eine Zeile der Form: (ebenfalls getrennt durch Tabulator/-en)

   „`<x: float>	<y: float>	yeinhüllende: float>`“
4. Danach folgt das Dateiende

Wurde die Ausgabe in die Datei beendet, so wird sie geschlossen und der Eingabedateiname des ausgegebenen Datensatzes in die Liste an *geschlossenen Dateien* hinzugefügt. Es wird so lange verfahren, bis diese Liste ihre maximale Größe erreicht hat.

Sollte zu einem Zeitpunkt die Liste an *verarbeiteten Datensätzen* leer sein, obwohl noch nicht alle Dateien geschlossen sind, so befindet sich mindestens ein Datensatz noch im Lese- bzw.

Verarbeitungsschritt. Daher wird in diesem Fall eine kurze Zeit gewartet und erneut auf die Liste zugegriffen.

3 Programmbeschreibung

Anmerkung zu deutschen und englischen Begriffen

Eigene Bezeichner, insbesondere für inhaltliche Elemente, wurden deutsch benannt. Vorkommende englischsprachige Bezeichner wurden verwendet, wenn dies geläufige Begriffe in der Softwareentwicklung sind, da so die Aufgabe der entsprechenden Komponente allgemein ersichtlich ist, beispielsweise bei Reader, Converter oder get/set/is.

3.1 Grobentwurf

3.1.1 Module

Das Programm wird durch eine drei-Schichtenarchitektur ähnlich der Model-View-Controller Architektur umgesetzt, welche im folgenden Paketdiagramm grob skizziert wird.

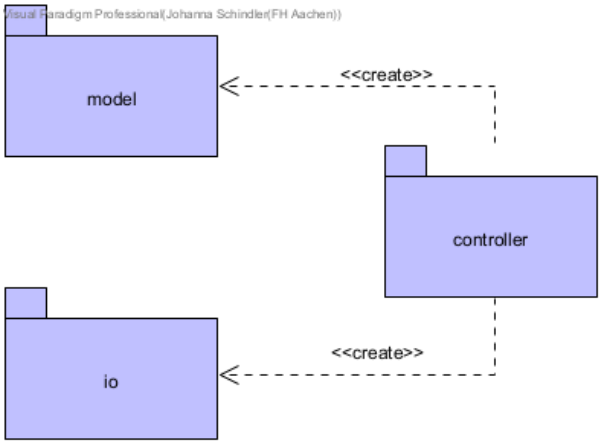

Abbildung 3: Module im Paketdiagramm

Das Model beinhaltet dabei die benötigten Daten in strukturierter Form, also den Datensatz mit einem Array von Messwerten, sowie weiteren Werten, die aus der Verarbeitung entstammen.

Der Controller beinhaltet ein Signalauswertungsprogramm, einen Konvertierer für die Ein- und Ausgabe, sowie den eigentlichen Auswertungsalgorithmus. Weiterhin befinden sich die Klassen, die die Threads benötigen in einem Unterpaket Runnables und eigene Exceptions im Unterpaket Exceptions.

Der Konvertierer wandelt die eingelesenen Daten in Objekte der Klassen aus dem Model Package um, erstellt diese also. Der Algorithmus operiert dann auf diesen. Eine klassische View existiert

in diesem Programm nicht, da keine visuelle Anzeige des Programms in der Aufgabenstellung gefordert war. Stattdessen gibt es das Modul IO, welches Klassen beinhaltet, die für das Ein- und Auslesen zuständig sind.

Hier zunächst eine Übersicht der Klassen des kompletten Programms, der Übersicht halber ohne Methoden und Attribute.

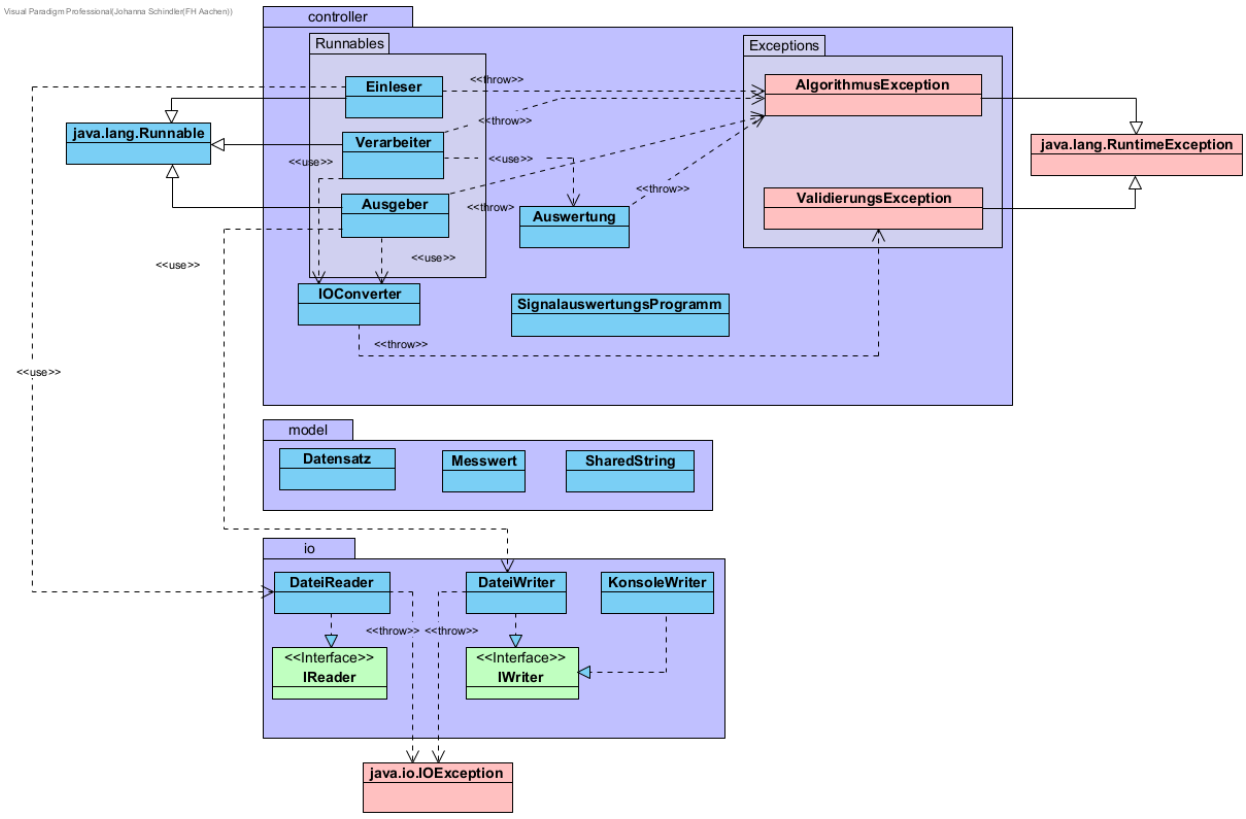

Abbildung 4: Klassendiagramm inklusive Pakete

3.1.2 Ein- und Ausgabe

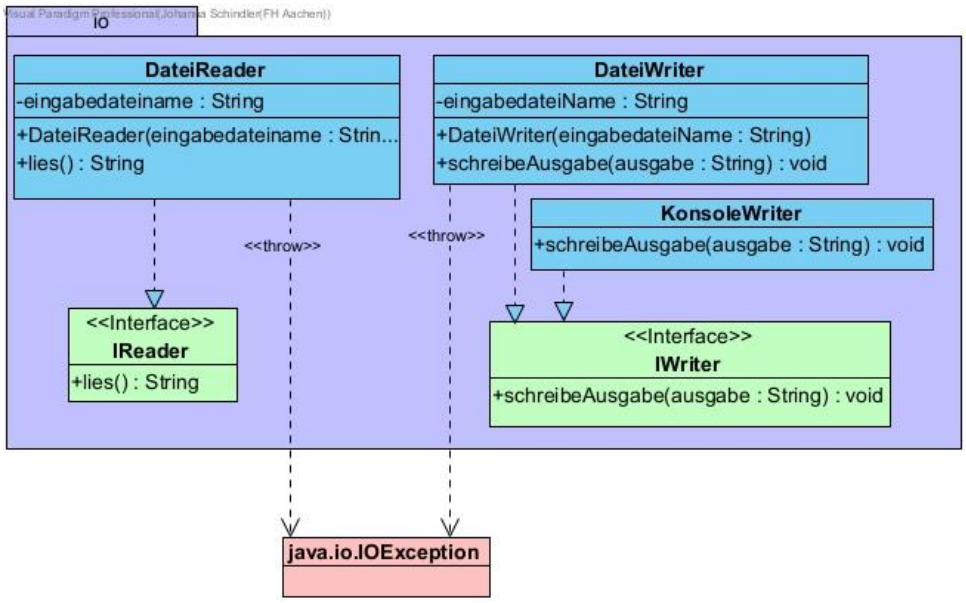

Abbildung 5: Klassendiagramm von IO

Die Ein- und Ausgabe werden durch jeweils ein Interface abstrahiert, also mittels des Strategy-Entwicklungsmusters implementiert. Dies erleichtert die Erweiterung des Programms um weitere Formen der Ein- und Ausgabe. (siehe Absatz 5.2.1)

Das erstellte Objekt der Klasse DateiReader liest in der Methode *lies()*, sofern keine technischen Fehler auftreten, beliebige Eingabedateien mit der Endung „.txt“ ein und lässt dabei Leerzeilen aus. Hier findet noch keine Syntax- oder Semantiküberprüfung statt. Die Methode gibt den gesamten Inhalt der Datei als String zurück. Der IOConverter konvertiert diesen Inhalt in ein Objekt der Klasse Datensatz unter Berücksichtigung der Syntax- und Semantikregeln. Da von der Konvertierer - Klasse kein Objekt erstellt werden soll, sondern eine statische Methode die Konvertierung übernimmt, ist der Konstruktor der Klasse privat.

Nachdem ein Ergebnis gefunden wurde, erstellt der IOConverter die Ausgabe des Objektes der Klasse Datensatz als String. Dieser wird dann an ein Objekt der Klasse DateiWriter gereicht und die Ausgabe umgesetzt. Der KonsolenWriter wird hier nur für die Ausgabe von Fehlermeldungen genutzt. Es ist aber denkbar auf gleiche Art wie die Ausgabe über den DateiWriter die Ausgabe eines Datensatzes auch in die Konsole zu schreiben.

Da der IOConverter im Gegensatz zum DateiReader, DateiWriter und KonsolenWriter somit Kenntnis über die Bedeutung des Dateiinhaltes hat und Objekte der Klassen im Model Package erstellt, befindet er sich im Package Controller.

Beim reinen Ein- und Auslesen können durch die Entkopplung von dem eigentlichen Dateiinhalt nur Exceptions der Klasse java.io.IOException auftreten. Für die syntaktischen und semantischen Fehler, die danach bei der Validierung des Dateiinhalts auftreten können, (siehe Absatz 1.6.3) wurde die Klasse ValidierungsException implementiert, welche von java.lang.RuntimeException erbt, da sie Fehler behandelt, die zur Laufzeit auftreten können. Es handelt sich demnach nicht um Eingabe oder Ausgabefehler, wie bei der Klasse java.io.IOException, da das Lesen des Inhalts der Textdatei zu diesem Zeitpunkt bereits abgeschlossen ist. Die Klasse ValidierungsException liegt im Package Controller, ebenso wie die erstellende Klasse IOConverter.

Die Klassen Einleser, Verarbeiter und Ausgeber implementieren das Interface java.lang.Runnable und können somit einem Objekt der Klasse java.lang.Thread zur Verarbeitung mitgegeben werden. Einleser bedient sich dabei der Klasse DateiReader, Verarbeiter der Klassen IOConverter und Auswertung und Ausgeber der Klassen IOConverter und DateiReader. Das Starten der Threads geschieht im SignalauswertungsProgramm, welches von der Klasse Main erstellt und gestartet wird.

3.1.3 Model

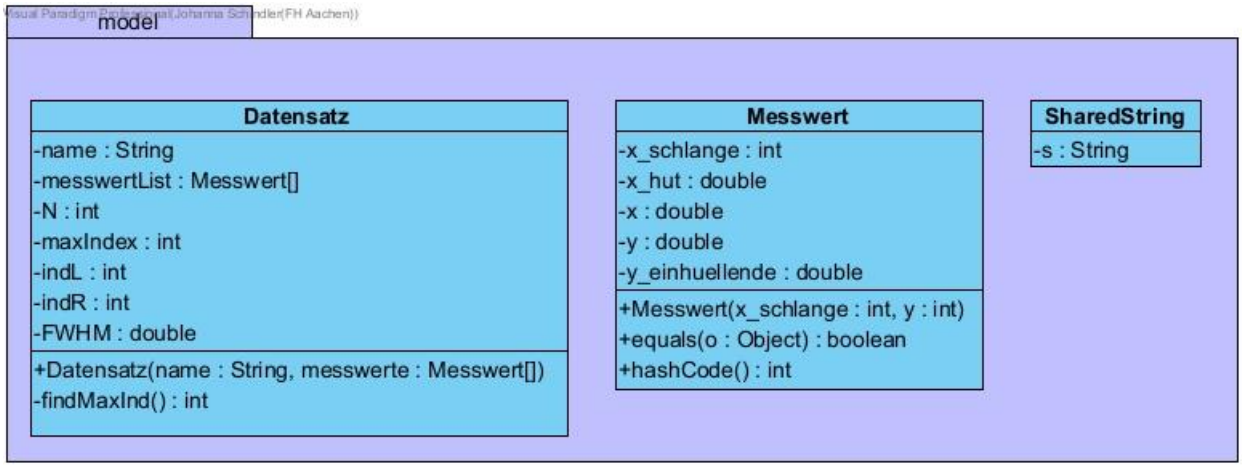

Abbildung 6: Klassendiagramm von Model

Das Model beinhaltet die Klasse Datensatz, die neben einem Feld von Objekten der Klasse Messwert auch den Eingabedateinamen und weitere für die Auswertung wichtige oder aus der Auswertung entstandene Werte speichert. Initial enthält der ein Objekt der Klasse nur die Messwerte und den Dateinamen der Eingabedatei. Daraus wird die Anzahl an Messwerten N und der Index des maximalen y-Wertes im Array gespeichert. Nach der Auswertung durch die Methode *werteAus()* der Klasse Auswertung, wurde FWHM (die Pulsbreite) sowie indL und

indR (die Indizes der Punkte L und R im Array) gesetzt. Ein Objekt von Messwert enthält initial nur x_schlange und y und wird im Laufe der Auswertung modifiziert und erweitert um x_hut und x. Der normierte Wert von y wird in derselben Variable gespeichert, weswegen es sich um ein double handelt, trotz, dass x_schlange und y initial Ganzzahlen sind.

Die Klasse SharedString ist eine Wrapper Klasse für einen String, die für die Synchronisierung der Threads von Bedeutung ist.

3.1.4 Controller

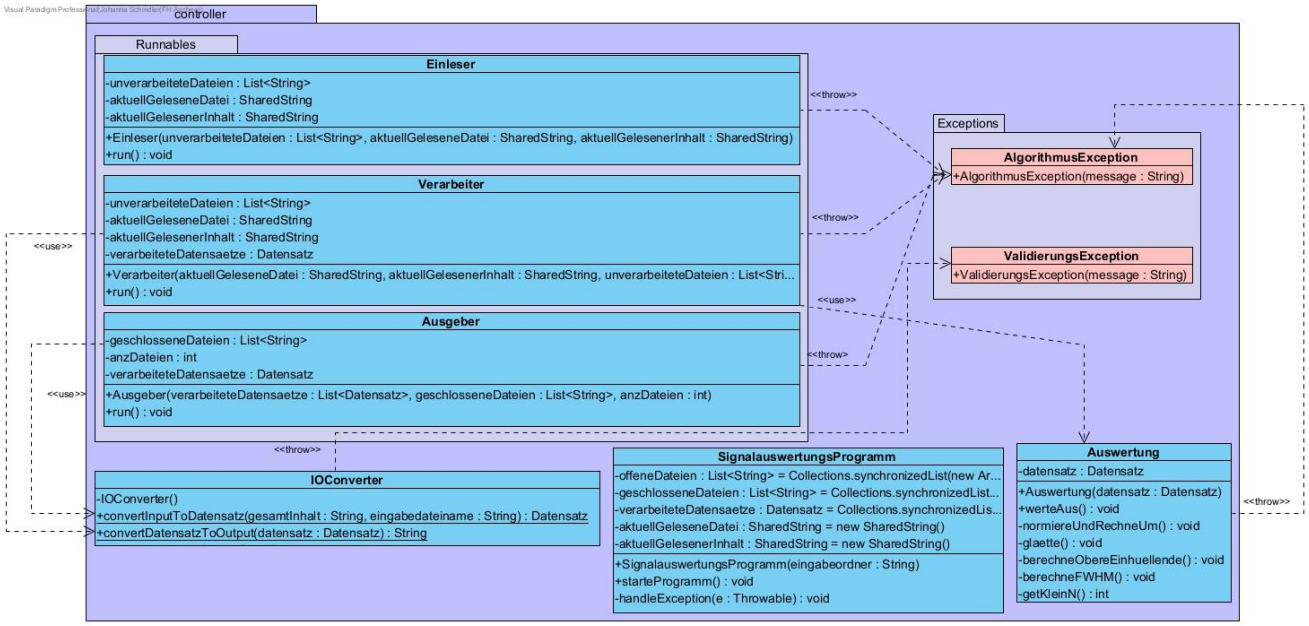

Abbildung 7: Klassendiagramm von Controller

Die Klasse SignalauswertungsProgramm übernimmt den kompletten Programmablauf, das heißt sie steuert nach Aufruf der Methode *starteProgramm()* die Threads für Eingabe, Vearbeitung und Ausgabe. Kommt es zu einem Fehler während der Auswertung oder wirft ein Thread eine Exception, so wird eine Exception der Klasse AlgorithmusException erstellt. Diese liegt wie die aufrufende Klasse im Package Controller und erbt von java.lang.RuntimeException, da sie Fehler behandelt, die zur Laufzeit auftreten können.

3.2 Ablauf

Der Ablauf des Programms lässt sich wie folgt als UML-Sequenzdiagramm beschreiben. Die Klasse Main übernimmt dabei die Aufgabe, das eigentliche Programm mit dem

Eingabedateinamen zu starten und ist an sich trivial, sodass auf ein Klassendiagramm verzichtet wurde.

Zunächst wird der Gesamtablauf der Programmausführung vereinfacht dargestellt. Die Vorgänge innerhalb der Threads sind komplexer als hier dargestellt. Diese werden daher anschließend noch mittels Aktivitätsdiagrammen präzisiert, um das Sequenzdiagramm möglichst übersichtlich und verständlich zu halten.

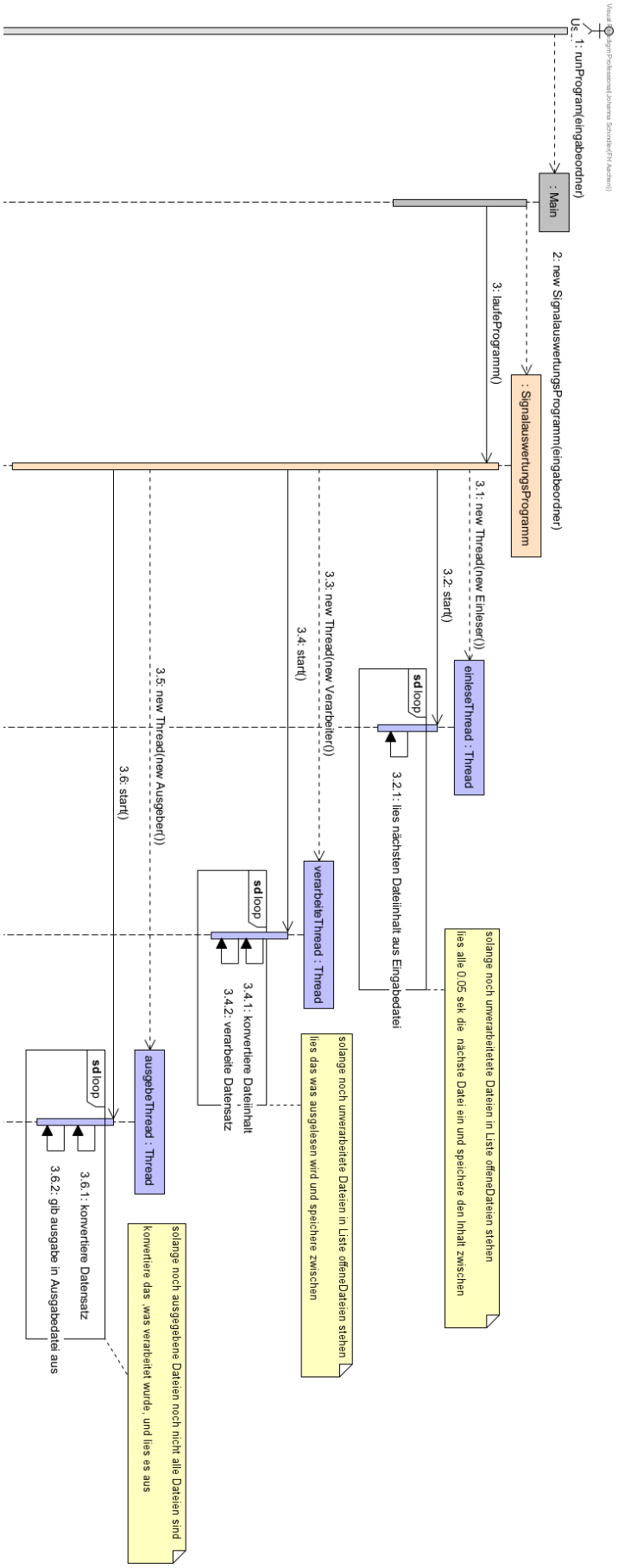

Abbildung 8: Sequenzdiagramm für Gesamtablauf

3.3 Auswertungsmethoden

3.3.1 Normieren und Umrechnen

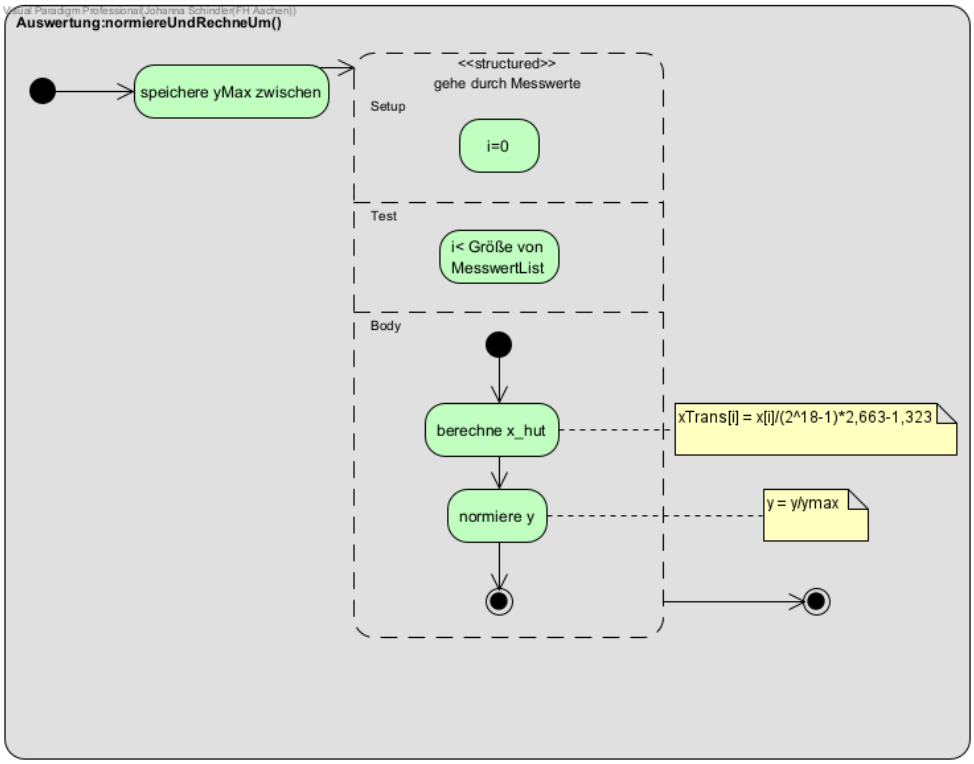

Abbildung 9: Aktivitätsdiagramm für Auswertung::normieren()

3.3.2 Glätten

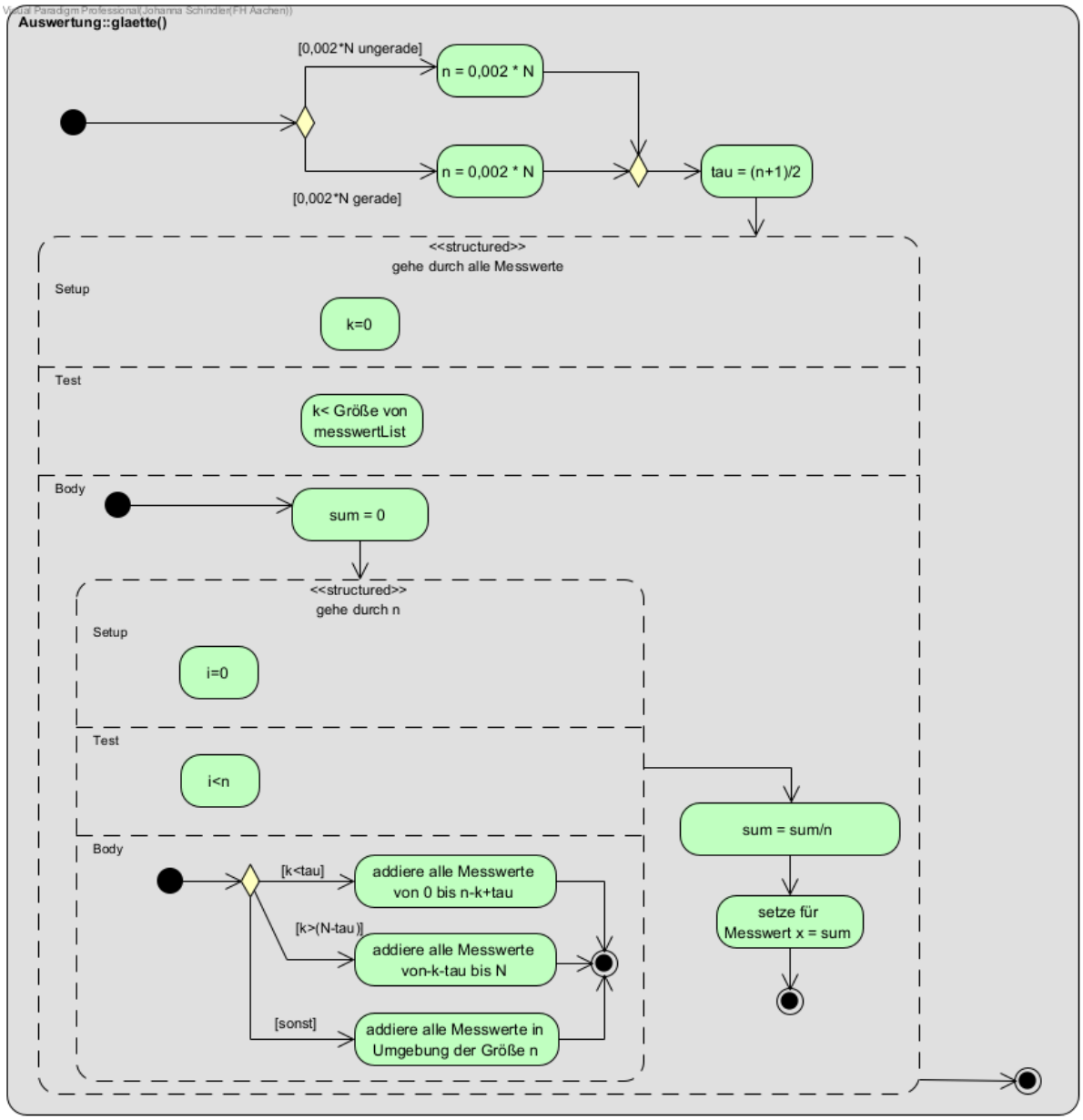

Abbildung 10: Aktivitätsdiagramm für Auswertung::glaetten()

3.3.3 Obere Einhüllende berechnen

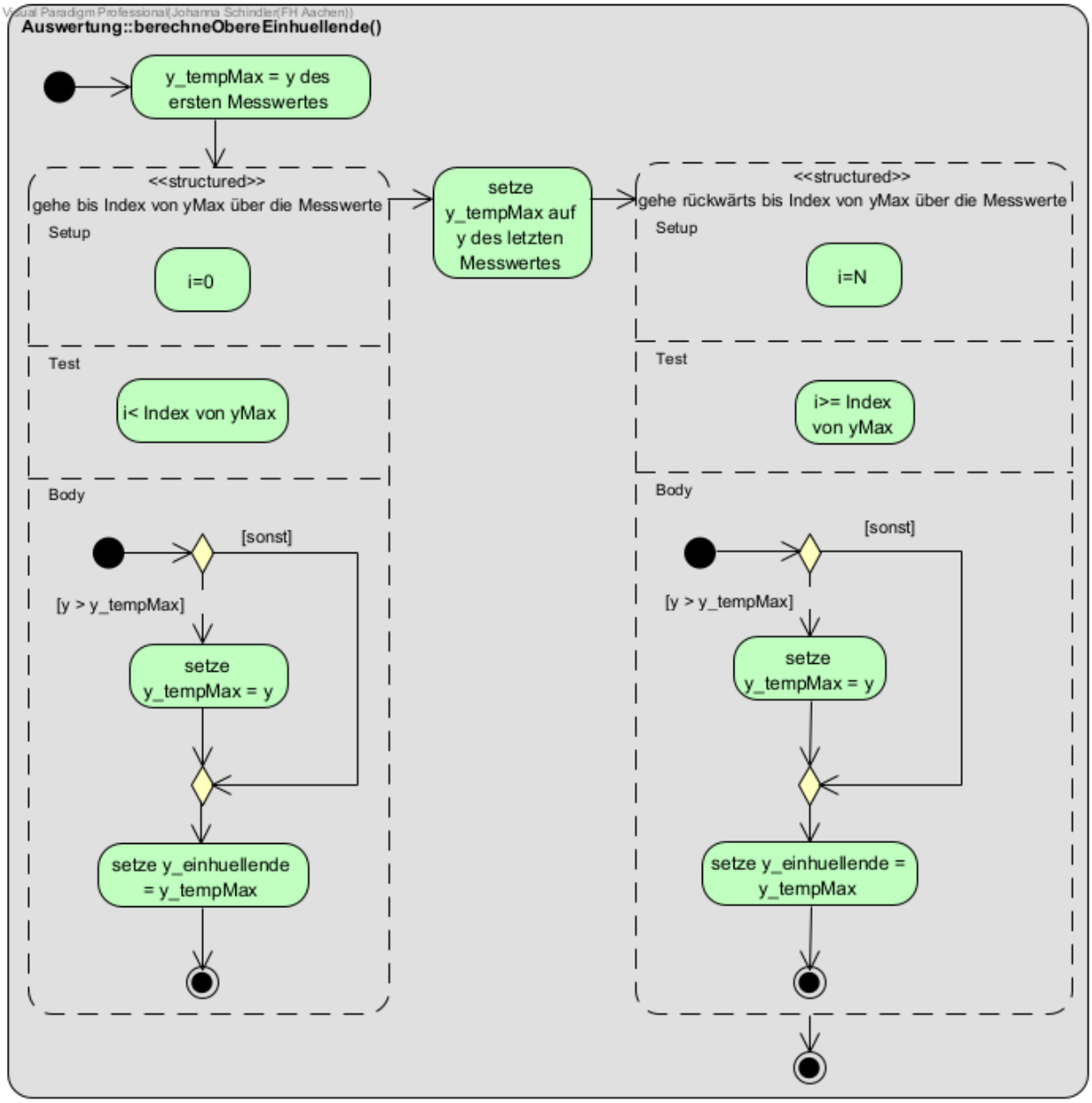

Abbildung 11: Aktivitätsdiagramm für Auswertung::berechneObereEinhuellende()

3.3.4 Pulsbreite berechnen

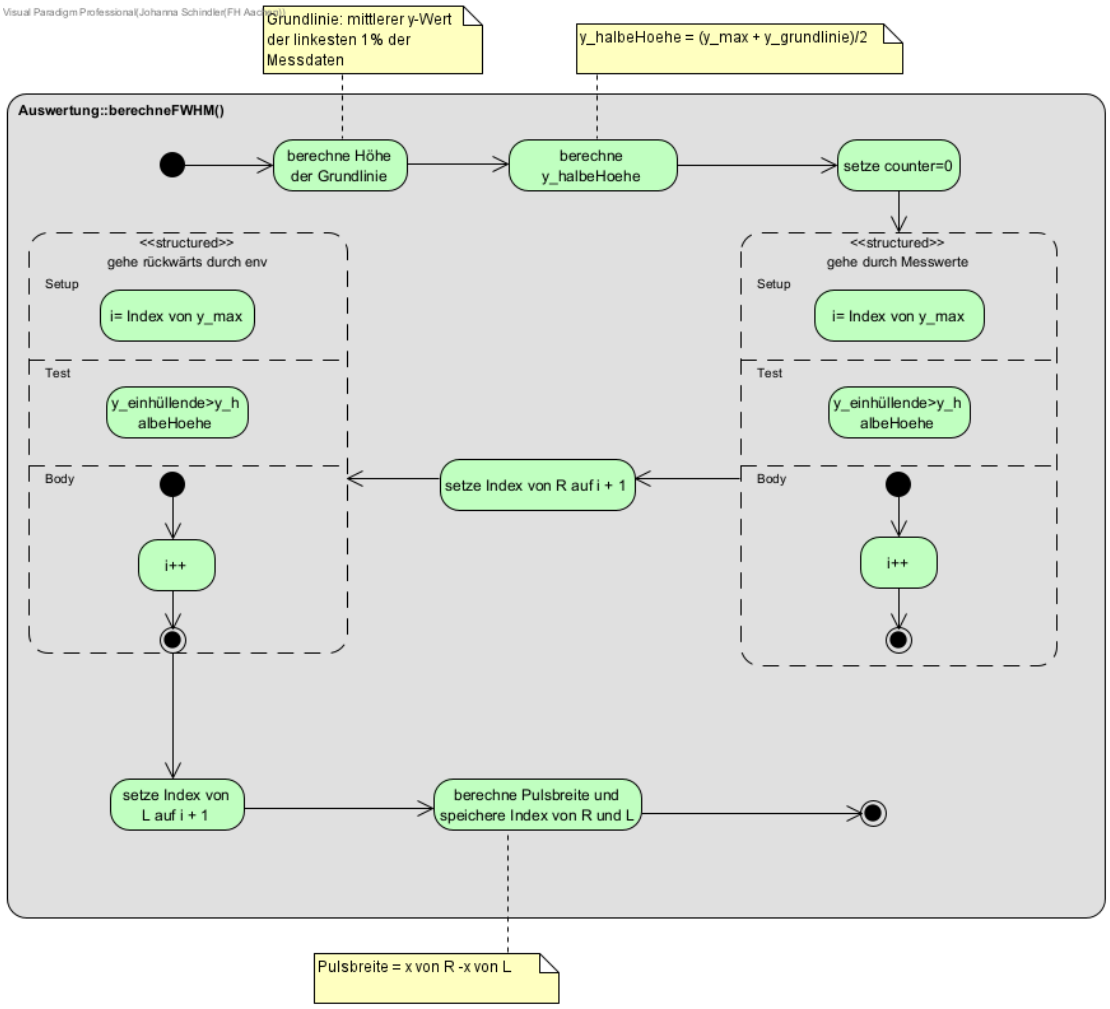

Abbildung 12: Aktivitätsdiagramm von Auswertung::berechneFWHM()

3.4 Threads

3.4.1 Einleser

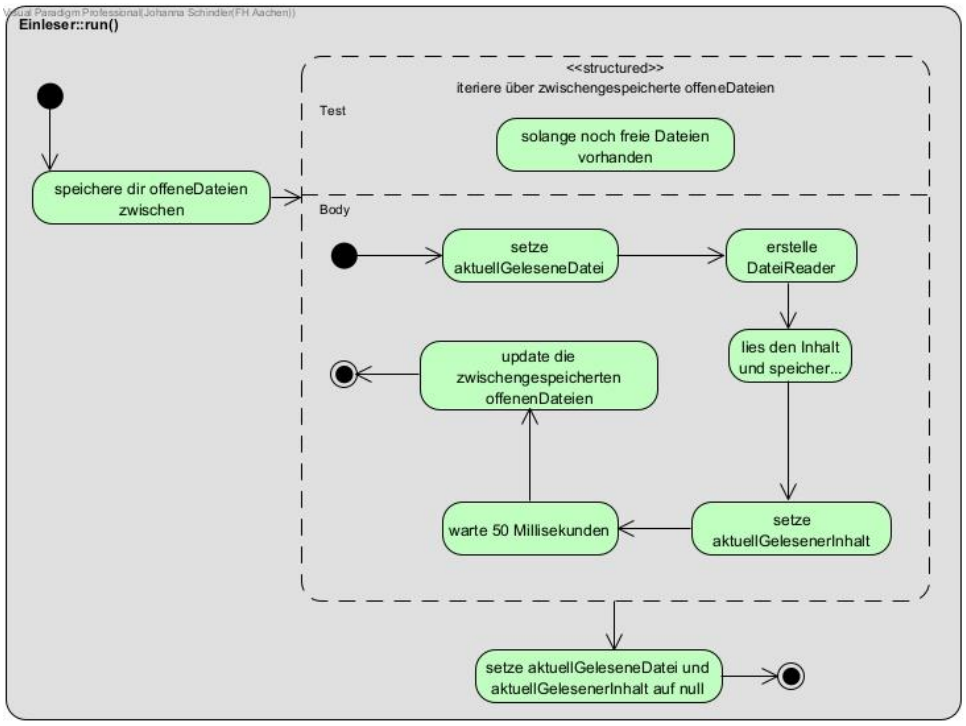

Abbildung 13: Aktivitätsdiagramm für Einleser::run()

3.4.2 Verarbeiter

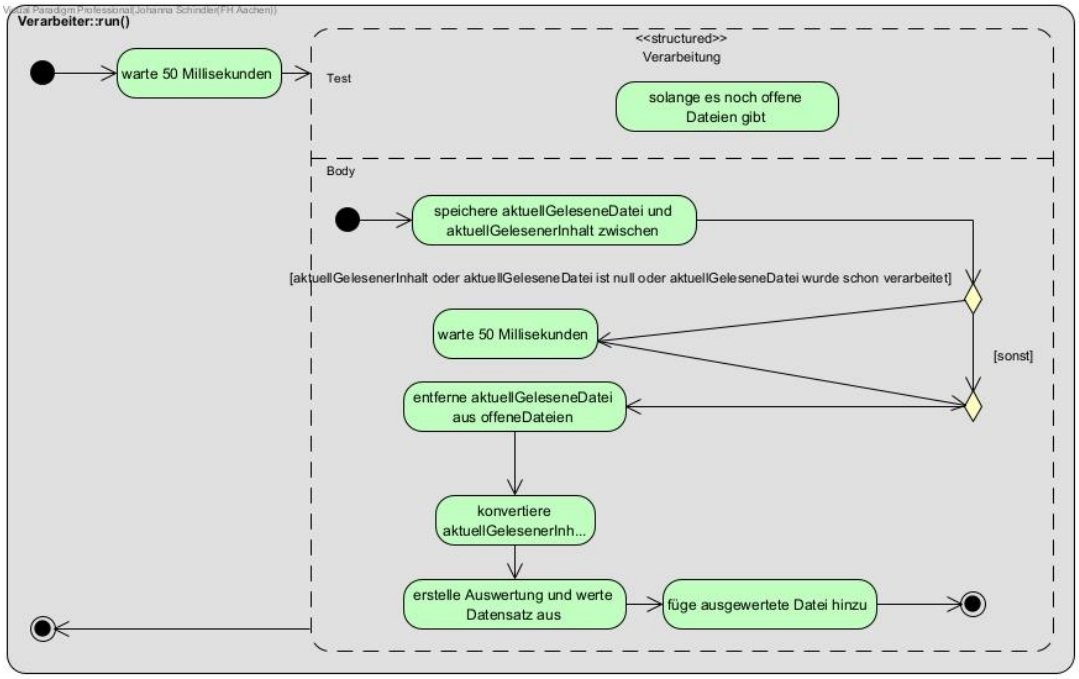

Abbildung 14: Aktivitätsdiagramm für Verarbeiter::run()

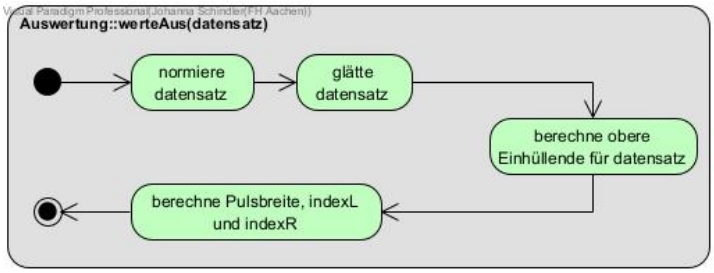

Abbildung 15: Aktivitätsdiagramm für Auswertung::werteAus()

3.4.3 Ausgeber

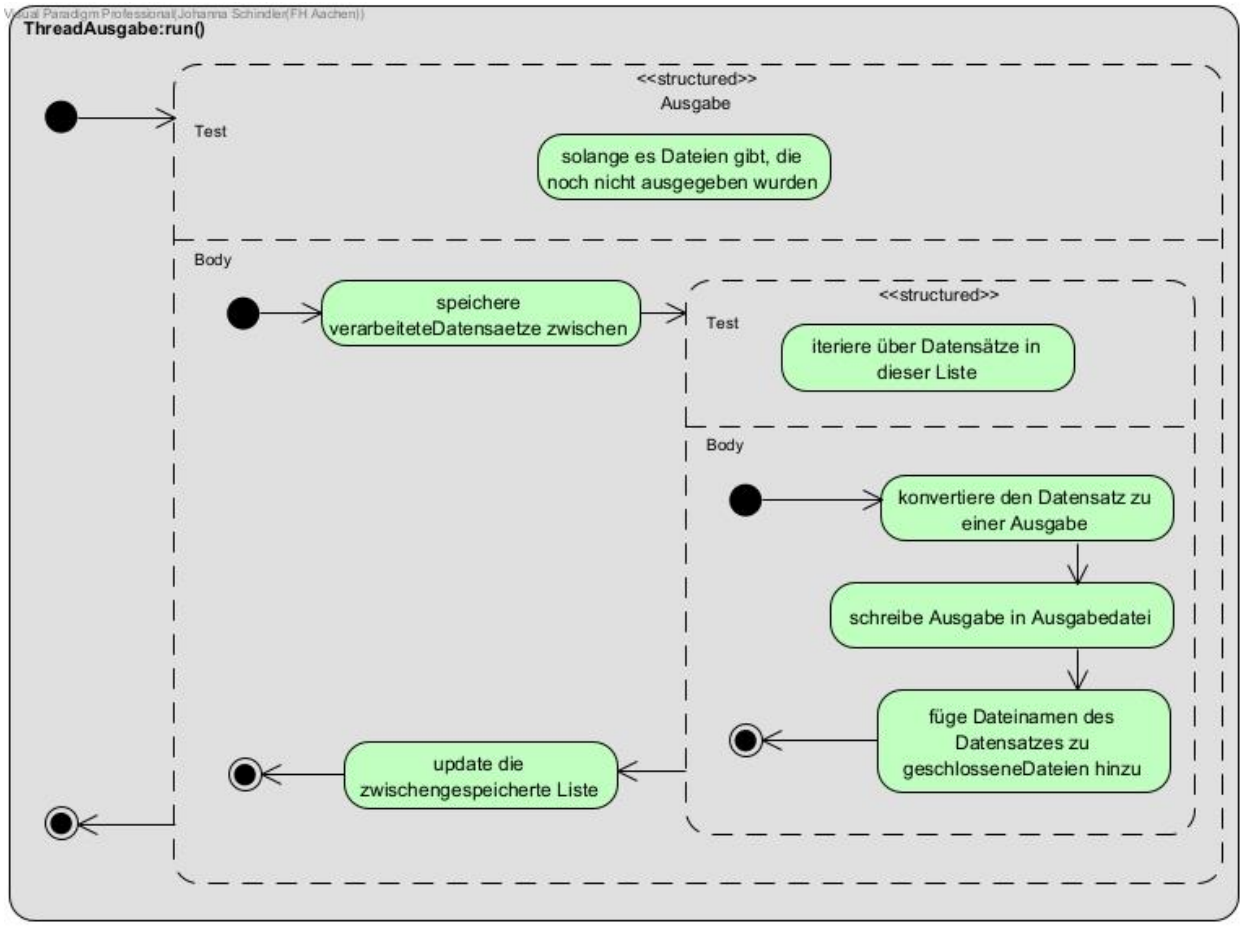

Abbildung 16: Aktivitätsdiagramm für Ausgeber::run()

3.5 Ein- und Ausgabekonvertierung

3.5.1 Eingabe

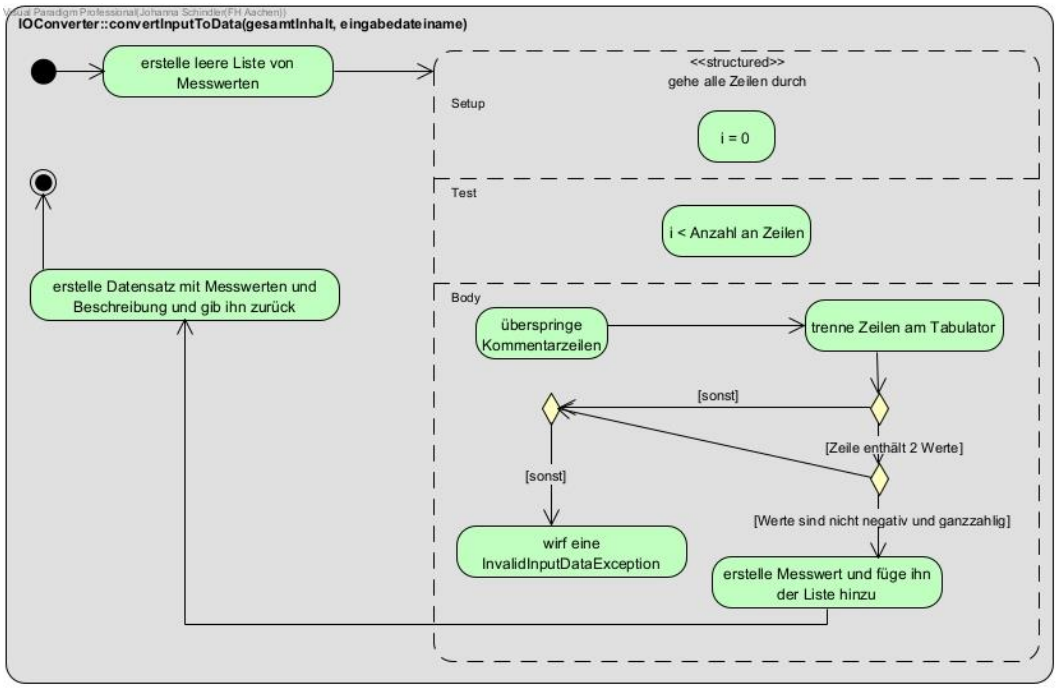

Abbildung 17:Aktivitätsdiagramm für IOConverter::convertInputToData()

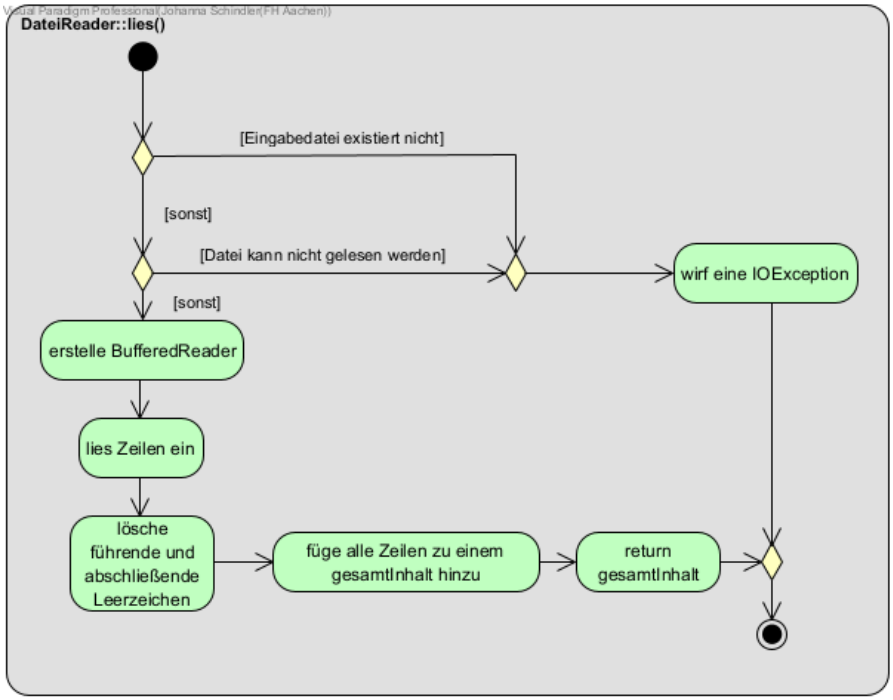

Abbildung 18: Aktivitätsdiagramm für DateiReader::lies()

3.5.2 Ausgabe

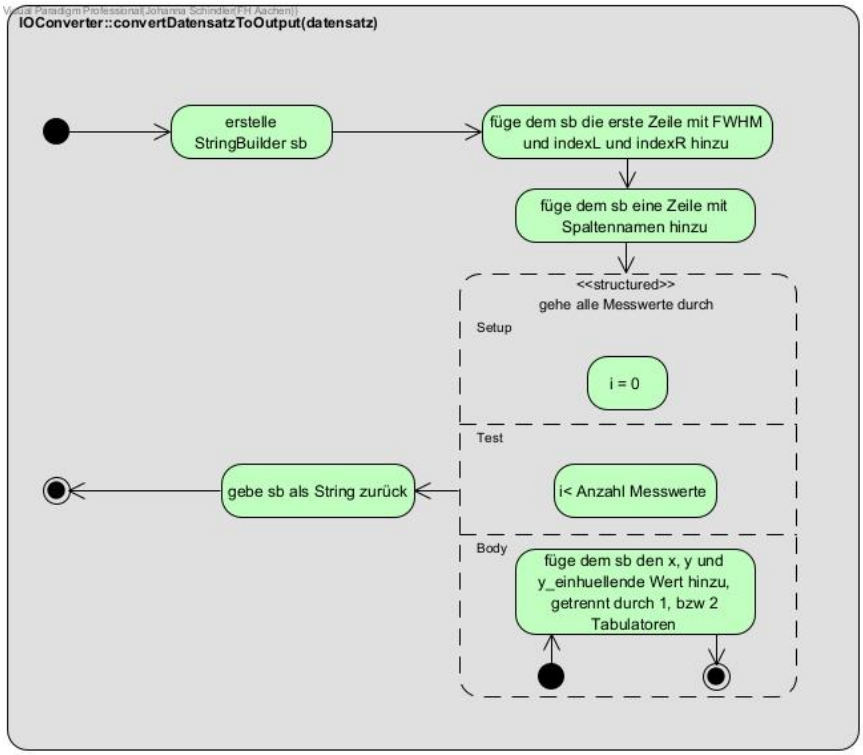

Abbildung 19: Aktivitätsdiagramm für IOConverter::convertDatensatzToOutput()

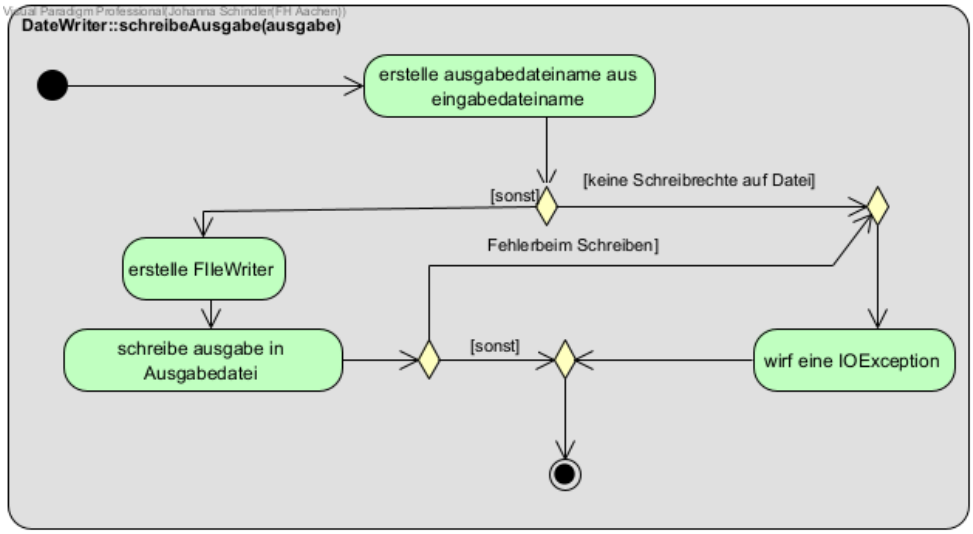

Abbildung 20: Aktivitätsdiagramm für DateiWriter::schreibeAusgabe()

4 Testdokumentation

Da diese in der Aufgabenstellung nicht explizit gefordert war, wird hier lediglich darauf eingegangen, wie die in Abschnitt 1.6.3.1 beschriebenen technischen Fehlerfälle bei Eingabe und Ausgabe getestet werden können und wie sie behandelt werden. Für etwaige syntaktische und semantische Fehlerfälle wurden auch Fehlermeldungen spezifiziert, sodass beispielsweise im Falle einer leeren Datei ein Nutzer über diesen Fehler aufgeklärt würde. Dieser Fehlermeldung wird auch der Name der Datei, bei der der Fehler aufgetreten ist, beigefügt.

4.1 Technische Fehler

Technischer Fehler 1 Der Test soll zeigen, dass das Programm das Fehlen einer Eingabedatei erkennt und den Benutzer mit der folgenden Fehlermeldung in der Konsole informiert.

```
"Eingabe/Ausgabe Fehler: Eingabedatei existiert nicht"
```

```
java -jar ./run/ SignalauswertungsProgramm.jar
```

Technischer Fehler 2 Der Test soll zeigen, dass das Programm erkennt, dass die angegebene Eingabedatei nicht existiert und den Benutzer mit einer passenden Fehlermeldung informiert.

```
"Eingabe/Ausgabe Fehler: Eingabedatei existiert nicht"
```

```
java -jar ./run/ SignalauswertungsProgramm.jar ¬
existiertNicht.txt
```

Technischer Fehler 3 Der Test soll zeigen, dass das Programm erkennt, dass die Leserechte für die angegebene Eingabedatei fehlen und den Benutzer mit einer passenden Fehlermeldung informiert.

```
"Eingabe/Ausgabe Fehler: beim Lesen der Eingabedatei"
```

```
java -jar ./run/ SignalauswertungsProgramm.jar ¬
nichtLesbar.txt
```

Technischer Fehler 4 Der Test soll zeigen, dass das Programm erkennt, dass die Schreiberechte zur passenden Ausgabedatei zur Eingabedatei fehlen und die Ausgabe nicht geschrieben werden kann. Darüber soll der Benutzer mit einer passenden Fehlermeldung informiert werden.

```
"Eingabe/Ausgabe Fehler: keine Schreibrechte auf
Ausgabedatei"
```

```
java -jar ./run/Programm.jar nichtSchreibbar.txt
```

Bei sonstigen unerwarteten Fehlern beim Lesen oder Schreiben einer Datei wird die folgende Fehlermeldung in die Konsole geschrieben:

```
"Eingabe/Ausgabe Fehler: beim Lesen/Schreiben der Eingabedatei"
```

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung

Das Programm erfüllt die in der Aufgabenstellung definierten Anforderungen und wurde in Bezug auf Funktionalität und Fehlererkennung mit den gegebenen Datensätzen getestet.

Bisher bricht das Programm bei einem Syntax- oder Semantikfehler in einem Datensatz das ganze Programm ab. Da die Daten in der Zukunft von einem Autokorrelator stammen werden, macht es Sinn, dieses Verhalten zu unterdrücken und die fehlerhafte Datei bei der Verarbeitung zu überspringen.

Die Anzahl an Dateien, sowie die Anzahl an Messdaten pro Datei sollten außerdem nach oben beschränkt werden, um zu hohe Lauf- und Verarbeitungszeiten zu vermeiden. Die bisherige maximale Größe von 2147483647 (maximaler Integerwert) ist nur technisch bedingt und sollte geringer sein. Dies ließ sich jedoch aufgrund der Datenlage nicht testen.

5.2 Ausblick

Für das Programm gibt es diverse Erweiterungsmöglichkeiten in verschiedenen Bereichen.

5.2.1 Ein- und Ausgabe

Die bisherige Ein- und Ausgabe über Textdateien im vorgegebenen Format lässt sich durch die Verwendung des Strategy-Entwicklungsmusters und die Trennung von IO und Konvertierung leicht um weitere Formate erweitern. Dies ist durch zusätzliche Implementierungen der Interfaces IReader und IWriter möglich.

Denkbar wären:

Eingabe als Stream (für Echtzeitdaten)	Dazu die Implementierung von IReader den Stream einlesen und im Einleser muss beachtet werden, dass das Einlesen deutlich schneller als das Verarbeiten von statten geht. (siehe Abschnitt 5.2.2)

Eingabe aus Datenbank (für historische Messdaten)	Dazu müsste die Implementierung von IReader den Eingabestring mittels SQL-Query von einer Datenbank abfragen. Die Methode zur Umwandlung der Daten aus der Datenbank muss dafür dem IOConverter hinzugefügt werden.
Ausgabe in eine Datenbank	Dazu müsste die Implementierung von IWriter einen String mittels SQL-Query an eine Datenbank absetzen. Die Methode zur Erzeugung des ein SQL-Queries muss dafür dem IOConverter hinzugefügt werden.

5.2.2 Nebenläufigkeiten

Im Echtzeitbetrieb wird die Eingabe wahrscheinlich nicht wie im simulierten Fall über bestehende Input-Dateien geschehen, sondern so, dass die Datensätze, die zur Verfügung stehen nach der Zeit von 0.05 Sekunden verloren gehen, wenn sie nicht verarbeitet wurden. In diesem Fall macht es Sinn, die ausgelesenen Daten in ein Array zu speichern, welches der Verarbeiter abarbeiten kann.

Das Einlesen passiert jedoch deutlich schneller als das Verarbeiten. Damit nicht die maximale Größe des Array erreicht oder sehr viele Datensätze gespeichert werden und der Speicher unter Umständen überlastet wird, sollte zum Beispiel wie folgt verfahren werden:

- Setze eine feste Arraygröße
- Lies einkommende Datensätze ein und füge sie als String ins Array ein
- Der String den letzten freien Platz im Array belegen, lösche jeden zweiten Wert im Array

Dadurch verliert man zwar jeweils die Hälfte der Messungen, jedoch bleibt die Entwicklung der Messreihen noch nachvollziehbar.

5.2.3 Graphische Oberfläche

Um die erfassten und berechneten Werte nach Auswertung graphisch darzustellen, könnte der Architektur eine View des MVCs hinzugefügt werden. So könnte man die aktuell zuletzt verarbeitete Messung als Plot darstellen und in der Graphik auch die berechneten Werte Pulsbreite sowie die Punkte R und L visualisieren.

5.2.4 Auswertung

Neben der Implementierung der Auswertung der Daten in Form von Normierung, Umrechnung, Glättung und Berechnung von oberer Einhüllender und Pulsbreite, wäre es denkbar, in der Zukunft noch weitere Merkmale zu berechnen, wie zum Beispiel die Pulsform. (Steinmeyer, 2005)

Neben der verwendeten Auswertungs-Strategie wäre noch eine Vielzahl weiterer möglich. Diese müssten ebenfalls das Interface IAuswertung implementieren. Diese leichte Erweiterbarkeit wurde auch hier durch das Verwenden des Strategy-Entwicklungsmusters erreicht.

Außerdem wäre es möglich, nicht nur die Daten eines optischen Autokorrelators zu verarbeiten, sondern auch beispielsweise die eines elektronischen Autokorrelators. Die Verfahrensweise bliebe bis auf das verwendete Model (und somit die Konvertierung) sowie die Auswertung identisch.

6 Abbildungsverzeichnis

Abbildung 1: Beispiel einer gültigen Eingabedatei („0.txt“) 2
Abbildung 2: Beispiel einer Ausgabe in einer Textdatei („out0.txt“) 5
Abbildung 3: Module im Paketdiagramm 17
Abbildung 4: Klassendiagramm inklusive Pakete 18
Abbildung 5: Klassendiagramm von IO 19
Abbildung 6: Klassendiagramm von Model 20
Abbildung 7: Klassendiagramm von Controller 21
Abbildung 8: Sequenzdiagramm für Gesamtablauf 23
Abbildung 9: Aktivitätsdiagramm für Auswertung::normieren() 24
Abbildung 10: Aktivitätsdiagramm für Auswertung::glaetten() 25
Abbildung 11: Aktivitätsdiagramm für Auswertung::berechneObereEinhuellende() 26
Abbildung 12: Aktivitätsdiagramm von Auswertung::berechneFWHM() 27
Abbildung 13: Aktivitätsdiagramm für Einleser::run() 28
Abbildung 14: Aktivitätsdiagramm für Verarbeiter::run() 28
Abbildung 15: Aktivitätsdiagramm für Auswertung::werteAus() 29
Abbildung 16: Aktivitätsdiagramm für Ausgeber::run() 29
Abbildung 17:Aktivitätsdiagramm für IOConverter::convertInputToData() 30
Abbildung 18: Aktivitätsdiagramm für DateiReader::lies() 30
Abbildung 19: Aktivitätsdiagramm für IOConverter::convertDatensatzToOutput() 31
Abbildung 20: Aktivitätsdiagramm für DateiWriter::schreibeAusgabe() 31

7 Literaturverzeichnis

Steinmeyer, G. (November 2005). *Femtonik*. Von https://application.wiley-vch.de/berlin/journals/ltj/05-04/LTJ0504_34_39.pdf abgerufen

A Abweichungen und Ergänzungen zum Vorentwurf

Der prinzipielle Ablauf und das Handling der Nebenläufigkeiten wurde nach dem Vorentwurf wie folgt verändert: es existieren, wie in der Aufgabenstellung gefordert, drei Nebenläufigkeiten, statt 4 wie im Vorentwurf behauptet. Es wurden die Klassen Einleser, Verarbeiter und Ausgeber hinzugefügt, die die Parallelisierung ermöglichen und die Abbruchbedingungen und die Zugriffe auf die Listen realisieren. Der synchrone Zugriff und die genauen Bedingungen, unter denen ein Thread arbeitet, wurden korrigiert. Die Threads werden aus der Klasse SignalauswertungsProgramm erstellt und stoßen dann die Verarbeitungen an.

Weiterhin wurden die mathematischen Methoden aus dem Datensatz entfernt und der Klasse Auswertung hinzugefügt, da diese Teil des Controllers sein sollten. Sie wurden *private* gemacht, damit die Reihenfolge der Abarbeitung nur die in der Methode *werteAus()* gekapselten sein kann. Das musste geschehen, damit es nicht möglich ist, von außen erst *berechneFWHM()* und dann *berechneObereEinhuellende()* aufzurufen, da zur Berechnung der Pulsbreite die Werte der oberen Einhüllenden verwendet werden und es sonst zu einem Fehler kommen könnte.

Die Messwerte, die ein Datensatz speichert, werden jetzt außerdem statt in einer Liste in einem Array fester Größe gespeichert, da sich die Anzahl an Messwerten nach der Initialisierung nicht mehr verändert.

B Benutzeranleitung

a. Verzeichnisstruktur

Im angegebenen Archiv befinden sich folgende Verzeichnisse:

run	beinhaltet Skripte zum Kompilieren des Programms und zum Ausführen der Testfälle, sowie das kompilierte Programm
src	enthält den Quellcode des Programms
doc	enthält diese Dokumentation, sowie die Java API-Dokumentation, auf die in Anhang C verwiesen wird
testfaelle	beinhaltet die ausgeführten Testdateien als Ein- und Ausgabedateien

b. Systemvorraussetzungen

Um das Programm verwenden zu können, wird ein Microsoft Betriebssystem in der Version 10 oder höher, sowie eine Java Laufzeitumgebung (JRE) in der Version 16 benötigt. Für das Kompilieren des Programms muss zusätzlich das Java Development Kit (JDK) in der Version 16 oder höher installiert sein.

Sollten diese nicht oder in veralteter Version installiert sein, kann ein Download der JRE und JDK in gewünschter Version unter der folgenden Adresse geschehen:

https://www.oracle.com/java/technologies/downloads/

Um das Programm, wie in Abschnitt d beschrieben, zu kompilieren, also für alle „.java“-Dateien zunächst „.class“-Dateien und schließlich die Projekt „.jar“-Datei zu erstellen, muss das Tool Maven installiert sein. Nachdem die Java SDK bereits installiert wurde, kann Maven unter der folgenden Adresse in der Version 3.8.1 als „.zip“ heruntergeladen werden. Dafür müssen 10MB Speicher für die Installation und ungefähr 500MB für das lokale Repository frei sein.

https://maven.apache.org/download.cgi

Die „.zip“-Datei muss danach in einen geeigneten Ordner entpackt werden und der Speicherort des Unterordners `/bin` in den PATH-Variablen des Systems hinterlegt werden.

c. Installation

Der gesamte Inhalt der vorliegenden „.zip“-Datei ist in ein beliebiges, beschreibbares Verzeichnis zu kopieren. Danach ist das Programm betriebsbereit. Das ausführbare Skript, das alle Testfälle nacheinander ausführt, liegt im Verzeichnis run, genauso wie die „.jar“-Datei zur Ausführung einzelner Testbeispiele.

d. Kompilieren

Um den Quellcode des Programms in eine ausführbare Datei zu kompilieren, muss im Terminal auf Wurzelebene des Projektes der Befehl mvn clean install durchgeführt werden. Dadurch wird ein eventuell bestehender „target“-Ordner gelöscht und die benötigten Dateien inklusive der „.jar“ -Datei im Ordner „target“ hinterlegt. Diese muss im Anschluss in das Verzeichnis /run kopiert oder verschoben werden und zu „SignalauswertungsProgramm.jar“ umbenannt werden. Damit ist das Programm kompiliert.

e. Programmausführung

Um Testdatensätze zu verarbeiten, muss das Programm über die Kommandozeile (cmd) ausgeführt werden und der Ordner der Testdatensätze übergeben werden. Der Aufruf aus dem Wurzelverzeichnis des Projektes erfolgt über den Befehl:

```
java -jar ./run/SignalauswertungsProgramm.jar <pfad/zu/inputOrdner/>
```

f. Testausführung

Um die in testfaelle/input vorliegenden Testdateien zu verarbeiten, kann das Skript run/RunProgram.bat genutzt werden. Mittels Doppelklick auf die Datei im Windows Explorer startet die Ausführung.

Zum Öffnen des Ordners /run hier klicken.

Zum Testen von bestimmten weiteren, bereits im Verzeichnis testfaelle/input hinterlegten Beispielen muss das Programm wie in Absatz B.e beschrieben ausgeführt werden.

C Entwicklerdokumentation

a. Verfügbare Klassen und Schnittstellen

Die Dokumentation der verfügbaren Klassen und Schnittstellen wurde mit Hilfe des Tools „javadoc“ aus dem Quellcode erzeugt und im mitgelieferten Archiv im Verzeichnis doc/javadoc abgelegt. Zur Einsicht muss die Datei doc/javadoc/index.html in einem Webbrowser geöffnet werden.

Zum Öffnen der index.html hier klicken.

b. Nutzungshinweise

Im Folgenden wird exemplarisch erklärt, wie das Programm angepasst und erweitert werden kann.

- **Auswertungsstrategie**

  Die Auswertung ist im Paket controller zu finden. Soll die bisherige Strategie durch weitere Methoden ergänzt werden, so geschieht das durch eine Änderung der Methode Auswertung::*werteAus()* oder durch eine zweite Implementierung des Interfaces IStrategie. Der Aufruf der Auswertungsmethode geschieht im Verarbeiter.

- **Ein- und Ausgabe**

  Die Funktionalität für Ein- und Ausgabe befindet sich im Paket io. Die Programmiererschnittstelle wird durch die Interfaces IReader fürs Lesen und IWriter für das Schreiben des Dateiinhaltes realisiert. Bisherige Implementierungen sind DateiReader, DateiWriter und ConsoleWriter. Die Erweiterung des Programms für andere Ein- und Ausgabeformate wird über eine weitere Implementierung dieser Interfaces realisiert. Der Aufruf der Lesemethode geschieht im Einleser und die Schreibemethode wird im Ausgeber aufgerufen.

Wichtig: Die Implementierung von IWriter legt nur das Ausgabemedium, also zum Beispiel Datei oder Konsole fest. Die Konvertierung ins Ausgabeformat wird im Folgenden erklärt.

- **Ausgabeformat**

  Das Ausgabeformat wird durch die Methode IOConverter:*convertResultToOutput()* bestimmt. Soll dieses verändert werden oder weitere Formate ergänzt werden, muss das also in der Klasse IOConverter geschehen. Der Aufruf der Konvertierungsmethode geschieht im Einleser.

- **Model**

  Die Klassen für die in der Auswertung benötigten Datenstrukturen wurden im Paket model abgelegt. Diese müssen für eigene Implementierungen von IStrategie, beziehungsweise IReader und IWriter importiert werden.

c. Fehlerbehandlung

Die Fehlerbehandlung beim Initialisieren des Programms ist im Falle von technischen Fehlern Aufgabe der Implementierung von IReader und im Falle von syntaktischen oder semantischen Fehlern von der Klasse IOConverter. Die Klassen werfen im Fehlerfall eine spezielle Exception: java.io.IOException bei klassischen Lese- und Schreibefehlern und ValidierungsException bei syntaktischen oder semantischen Fehlern. Die Klasse ValidierungsException erbt dabei von java.lang.RuntimeException, da es sich um Laufzeitfehler handelt. Sollte der Algorithmus zu keinem Ergebnis kommen, wirft die Implementierung von IStrategie eine Exception der Klasse AlgorithmusException, welche ebenfalls von java.lang.RuntimeException erbt.

Die Fehlermeldung, welche als String im ersten Parameter des Konstruktors in die Exception gegeben wird und über die Methode *getMessage()* ausgelesen werden kann, wird bei ValidierungsException und AlgorithmusException dann in die Konsole und die Ausgabedatei geschrieben, bei einer java.io.IOException nur in die Konsole. Diese

Konvention muss von zusätzlichen Implementierungen des Interfaces IStrategie und bei einer Veränderung von der Klasse SignalauswertungsProgramm beibehalten werden.

D Entwicklungsumgebung

Programmiersprache	:	Java
Compiler	:	OpenJDK javac 16
Rechner	:	Dell Latitude 5500
Betriebssystem	:	Windows 11 Pro

E Verwendete Hilfsmittel

- IntelliJ 2021.1
  Entwicklungsumgebung für Java
  https://www.jetbrains.com/de-de/idea/
- Maven 3.8.1
  Projektmanagement-Tool
  https://maven.apache.org/download.cgi
- Visual Paradigm Professional 16.3
  UML-Tool
  https://www.visual-paradigm.com/
- Microsoft Word
  Texteditor
  https://www.microsoft.com/de-de/download/office.aspx

F Erklärung

Erklärung des Prüfungsteilnehmers:

Ich erkläre verbindlich, dass das vorliegende Prüfprodukt von mir selbstständig erstellt wurde. Die als Arbeitshilfe genutzten Unterlagen sind in der Arbeit vollständig aufgeführt. Ich versichere, dass der vorgelegte Ausdruck mit dem Inhalt der von mir erstellten digitalen Version identisch ist. Weder ganz noch in Teilen wurde die Arbeit bereits als Prüfungsleistung vorgelegt. Mir ist bewusst, dass jedes Zuwiderhandeln als Täuschungsversuch zu gelten hat, der die Anerkennung des Prüfprodukts als Prüfungsleistung ausschließt.

Köln, 13.05.2022

Ort, Datum

Unterschrift des Prüfungsteilnehmers