UNIVERSITÄT SIEGEN

Fachbereich 12 – Elektrotechnik und Informatik
Institut für Werkstoffe der Elektrotechnik und Diagnostik
Prof. Dr. rer. nat. Rainer Patsch

Entwicklung einer Software zur Puls-Sequenz-Analyse (PSA)

STUDIENARBEIT

vorgelegt von: Samer Hijazi
Matr.-Nr.: 539306
Eingereicht am: 15.07.2018

Betreuer: Prof. Dr. rer. nat. Rainer Patsch
Betreuender Assistent: Dipl.-Phys. Djamal Benzerouk

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Am Institut für Werkstoff und Diagnostik (WED) an der Universität Siegen ist die Teilentladungsdiagnose der Schwerpunkt der Forschung. Das Verfahren Puls-Sequenz-Analyse (PSA), das zur Teilentladungsdiagnose eingesetzt wird, ermöglicht eine detaillierte Analyse der Teilentladungs-Messdaten.

Die vorliegende Arbeit stellt die Entwicklung einer Software, die in der Programmiersprache JAVA geschrieben ist, zur PSA-Auswertung dar.

Der Teilentladungsprozess und der Begriff Teilentladung sowie dessen Entstehung werden in Kapitel 2 diskutiert. Das PSA-Verfahren wird ebenso erläutert.

In Kapitel 3 werden die Anforderungen an die Software dargestellt und im Detail beschrieben. Vorschläge zur Lösung der Anforderung werden diskutiert. Prof. Patsch, der über langjährige Erfahrung in Programmiersprachen und Masken verfügt, hat mehrere Lösungen vorgestellt, die in der Implementierung der Software eingesetzt wurden.

Die Realisierung der Anforderungen, die im Kapitel 3 beschrieben sind, wird in Kapitel 4 implementiert. Die Implementierung wird als erstes in Form eines Flussdiagramms dargestellt. In der zweiten Stufe der Implementierung werden die Screen-Spots und deren Erklärungen dargestellt. Als letztes werden die Kodes der Implementierung, die in der Programmiersprache JAVA geschrieben sind, dargestellt.

2 Grundlagen der TE-Diagnostik

In den letzten Jahren ist das Interesse an der Feststellung von Lebensdauerprognosen der Isolationssysteme von elektronischen und elektrischen Produkte gestiegen. Hochspannungstechnische Anlagen und Geräte werden umfangreichen Qualitätsprüfungen unterzogen. Anhand der Ergebnisse der vorgenommenen Messung werden Rückschlüsse auf die Lebensdauer und die Langzeitstabilität der Geräte gezogen [1].

Teilentladungsdiagnostik ist ein quasizerstörungsfreies Verfahren zur Beurteilung des aktuellen Zustandes von Isolationssystemen und elektrischen Anlagen wie z.B. Hochspannungsgeneratoren, -motoren, -transformatoren, -kondensatoren, -isolatoren, -schaltungen, und -kabeln. Durch die Teilentladungsdiagnostik wird nicht nur eine Aussage über das Vorhandensein von Teilentladungen getroffen, sondern es werden auch die möglichen Ursachen, Quellen und Entstehungsorte bestimmt [1, 2].

Im folgenden Kapitel 2.1 wird der Begriff Teilentladung erklärt. Die Puls-Sequenz-Analyse (PSA) als Verfahren zur Teilentladungsdiagnostik und ihr Prinzip werden in Kapitel 2.2 vorgestellt.

2.1 Teilentladungen

Teilentladungen sind physikalisch komplexe Prozesse, die dann entstehen, wenn aufgrund lokal unterschiedlicher Feldstärken (bei homogenen Dielektrika) oder unterschiedlicher lokaler Festigkeiten (bei Isolieranordnungen von Dielektrika mit unterschiedlicher elektrischer Festigkeit) die Durchschlagfeldstärke des Dielektrikums lokal überschritten wird und ein partieller Zusammenbruch des Isolationssystems erfolgt. Die Ursache von Teilentladungen kann im Vorhandensein von Fehlstellen im Isolationssystem, einer Alterung des Isolationssystems oder an einer starken Inhomogenität des elektrischen Feldes bedingt durch die Anordnung liegen [2].

Nach der VDE Norm 0434/05.83 ist eine Teilentladung definiert als "eine elektrische Entladung, welche die Isolierung zwischen zwei Elektroden nur teilweise überbrückt. Sie kann, muss aber nicht, an der Elektrodenoberfläche selbst auftreten".

Die Typen der Teilentladungen können in vier Anordnungen genannt werden. Diese Anordnungen sind Spitze-Platte (Koronaentladung), Spitze-Dielektrikum-Platte (Oberflächenentladung), Hohlraum im Dielektrikum bzw. an eine Elektrode angrenzend (Hohlraumentladung) und Nadel im Dielektrikum (Tree Entladung) [2].

2.2 PSA-Verfahren

Das PSA ist ein Verfahren zur TE-Diagnostik. Das Verfahren basiert auf der Auswertung der Korrelationsbeziehungen aufeinander folgenden TE-Impulse [3].

Die folgenden Analysemöglichkeiten werden in dem PSA-Verfahren eingesetzt um Teilentladungen zu charakterisieren [4].

Analysemöglichkeiten:

- n **über** t **-Analyse**

„*Durch die Auftragung der Impulsnummer über der Zeit wird eine Aussage über zeitlichen Konstanz des Teilentladungsverhaltens bzw. über eventuell auftretende Änderungen erhalten. Die Steigung dieser Kurve entspricht der mittleren Impulsrate.“*

- U **über** t **-Analyse**

„*Auftragung der Momentanspannungen U bei denen Teilentladungen auftreten über Beanspruchungszeit auf. Diese Auftragung ist besonders dann interessant, wenn die Spannung zeitlinear verändert wird. Wenn die Teilentladungen nur jeweils in den Scheitelwerten auftauchen, so ergibt sich bei dieser Darstellung im Wesentlichen die Änderung der Spannung mit der Beanspruchungszeit. Z.T. treten hierbei auch interessante Effekte auf, dass z.B. trotz Steigerung der Spannung Entladungen bei kleineren Spannungen, d.h. schon während der Anstiegsphase der Spannung vor Erreichen des Spannungsmaximums auftreten. Ein solches Verhalten lässt Rückschlüsse auf das Raumladungsgeschehen in der Fehlstelle zu.*“

- Δt **über** t **-Analyse**

„*Die Analyse der Zeitabstände Δt zwischen aufeinanderfolgenden Teilentladung ist eine wichtige, elementare Analysemöglichkeit. Oft führt – bei zwei Teilentladungen pro Periode – eine Teilentladungen zum Aufbau einer Raumladung, die durch nachfolgende Teilentladung abgebaut oder überkompensiert wird. Z.T. baut sich bei diesem Prozeß auch eine unipolare Raumladung auf, die zu einer systematischen Gleichspannungsverlagerung der durch die externe Wechselspannungsbelastung hervorgerufenen internen symmetrischen lokalen elektrischen Feldstärke führt. In Ergänzung zur Δt-Analyse ermöglicht die Δt über t-Analyse*

eine erste Charakterisierung der zeitlichen Änderungen des TE-Verhaltens sowie die Feststellung, ob eine oder mehrere TE-Quellen aktiv sind oder aktiv warden.“

Nachfolgend ein Auswahl zusätzlicher Analysemöglichkeiten:

- I_1 über t-Analyse
- ΔU über t-Analyse
- $\Delta U / \Delta t$ über t-Analyse
- t_1-Histogramm
- t_2-Histogramm
- $t_1 - t_2$-Histogramm
- I_1-Histogramm
- U-Histogramm
- ΔU-Histogramm
- Δt-Histogramm
- $\Delta U / \Delta t$-Histogramm
- …

3 Anforderungen an der Software

In diesem Kapitel werden die Anforderungen an die zu entwerfende Software aufgelistet und erklärt. Zuerst wird eine abstrakte Problembeschreibung dargestellt, die die Grundanforderungen an der Software festlegt.

3.1 Problembeschreibung

Das Institut für Werkstoffe der Elektrotechnik und Diagnostik (WED) an der Universität Siegen beschäftigt sich mit TE-Diagnostik und setzt das PSA-Verfahren zur TE-Charakterisierung bei den Untersuchungen elektrischer Betriebsmittel ein.

An dem zu prüfenden Betriebsmittel wird im entsprechenden Prüfkreis (meist mit 50 Hz Wechselspannung) eine Teilentladungsmessung durchgeführt. Üblicherweise werden jeweils 4096 Teilentladungereignisse registriert. Die jeweils pro Ereignisse aufgenommene Daten sind $[I_1, I_2, U, t, t_1, t_2]$. Die TE-Messdaten werden in einer binä-

ren Datei gespeichert, die in Form einer sechsspaltigen Datenmatrix in den ASCII-Code umgewandelt wird. Die TE-Messdaten werden den Anwender zur PSA bereitgestellt [1].

Bevor man mit der TE-Charakterisierung anfängt und die Analysemöglichkeiten einsetzt, die im Kapitel 2.2 erklärt wurden, werden die TE-Messdaten bearbeitet. Die TE-Messdaten können, nach ihrer Bearbeitung, bestimmten Filteroperationen unterzogen werden. Sind die Bearbeitung bzw. die Filteroperationen vollzogen werden, so kann man die Analysemöglichkeiten bzw. die Korrelationsdarstellungen einsetzen bzw. darstellen.

Um den Einsatz des PSA-Verfahrens effizienter zu machen, wird eine Entwicklung einer Software gefordert, die die oben benannten Operationen - die Bearbeitung und Filterung der TE-Messdaten und anschließend die Darstellung der Analysemöglichkeiten – realisiert.

Die Hauptanforderungen an der Software sind grundsätzlich das Einlesen, die Bearbeitung und Filterung des Datensatzes, anschließend die Erstellung von Korrelationsdarstellungen, die anschließend auf Papier ausgedruckt werden können. Diese Anforderungen werden in den folgenden Kapiteln ausführlich erklärt.

3.1.1 Einlesen der TE-Messdaten

Die TE-Messdaten sind in einer Datei, die aus 4096 Zeilen besteht und in Form einer sechsspaltigen Datenmatrix im ASCII-Code gespeichert ist.

Die Software soll in der Lage sein, diese Datei zu lesen. Nach dem Lesen, soll die Software den Inhalt der Datei in einer sechsspaltigen Tabelle darstellen. Die zu lesende Datei ist in der Abbildung 1 zu sehen. Die in der Tabelle anzuzeigenden Zahlen müssen in einem bestimmten Format dargestellt werden, so dass der Benutzer sie besser einlesen kann. Die Datensätze können auch Störimpulse enthalten, welche vor der eigentlichen Analyse entfernt werden sollen.

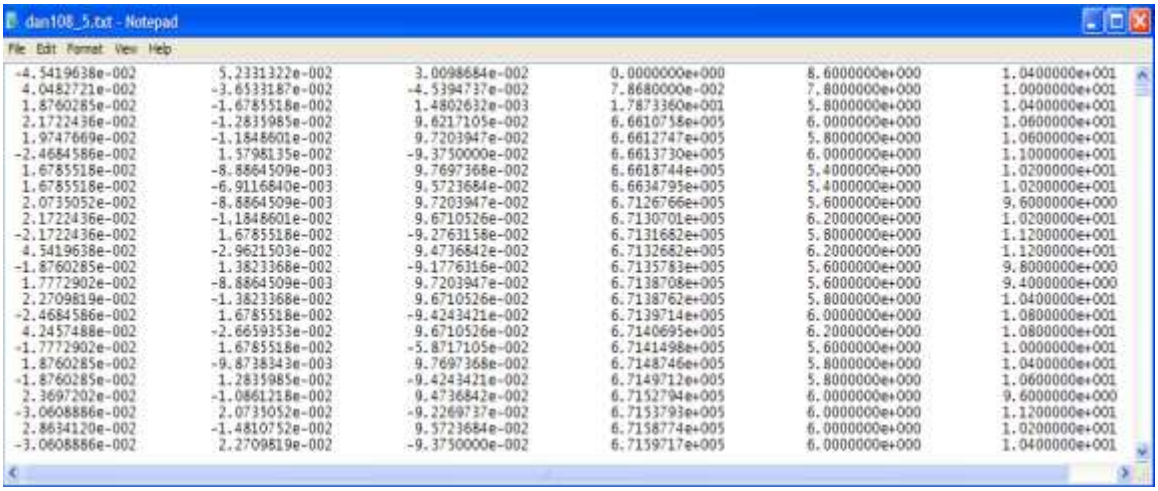

Abbildung 1 : TE-Messdaten in ASCII-Format

3.1.2 Datensatzbearbeitung

Bevor mit der Auswertung (Einsatz der Analysemöglichkeiten) der TE-Daten anfangen werden kann, müssen bestimmte Operationen an den TE-Daten durchgeführt werden.

- Zuerst muss die Software in der Lage sein, Startimpulse, die bei der Messwerterfassung mitregistriert werden, zu entfernen

- In der zweiten Stufe muss der Anwender bei Messungen mit zeitlinear erhöhter Spannung den Wert für die Hochfahrgeschwindigkeit eingeben können, die in [V/min] angegeben werden muss

- Dann muss die Frequenz der Prüfspannung in Hz eingegeben werden

- Die Umrechnungsfaktoren müssen eingegeben werden, dabei muss die Tabelle, die die Datensätze darstellt, aktualisiert werden

- Als letzte Stufe müssen die Störimpulse im Datensatz entfernt werden, dabei werden z.B. Impulse mit dem Wert t_1 kleiner Null gelöscht

- Nach der Durchführung der Operationen an dem Datensatz kann er, muss aber nicht, in einer neuen Datei gespeichert werden. Die neue erzeugte Datei soll den gleiche Dateinamen und einen neuen Datentyp „PSA“ besitzen. Die Speicherung der Datensätze hat das Ziel, dass die durchgeführten Operationen nicht noch mal bei einer neuen Auswertung der TE-Messdaten durchgeführt werden müssen

Der Speicherungsprozess darf nicht stattfinden, bevor alle oben beschriebenen Operationen außer der Löschung der Störimpulse durchgeführt werden sind. Für Auswertung der TE-Messdaten müssen auch die oben beschriebenen Operationen durchgeführt werden wie z.B. Löschen der Störimpulse, Einsetzen von der Hochfahrgeschwindigkeit, der Frequenz und der Umrechnungsfaktoren und der Löschung von Impulsen mit t_1 kleiner Null.

3.1.3 Auswertung der TE-Messdaten

Zur Auswertung (Einsatz der Analysemöglichkeiten) der TE-Messdaten werden unterschiedliche Haupt- bzw. Nebenparameter ausgewählt, um unterschiedliche Impuls-Korrelationsmöglichkeiten darzustellen. Die Software muss in der Lage sein, dem Anwender die Auswahl aus unterschiedlichen Parametern zur Darstellung einer Impuls-Korrelation zu ermöglichen. Als Hauptparameter sind $[I_1, I_2, U, t, t_1, t_2]$ zu betrachten. $[t_1 - t_2, I_2 / I_1, \Delta U, \Delta t, \Delta U / \Delta t, U_{eff}, t \bmod 20,$ *Phasenverschiebung*$]$ werden als Nebenparameter gezählt. Generell gilt $\Delta U_{(n)} = U_{(n+1)} - U_{(n)}$ und $\Delta t_{(n)} = t_{(n+1)} - t_{(n)}$. Scatter-Plots und Histogramme werden als mögliche Darstellung der Impuls-Korrelation betrachtet.

3.1.4 Filterung der TE-Messdaten

Die Software muss die Messdaten nach bestimmten Parametern, die von dem Anwender bestimmt werden, filtern und sie zur weiteren Analysemöglichkeiten zur Verfügung stellen. Der Anwender soll in der Lage sein, die schon gefilterten Messdaten noch einmal zu filtern oder sie in den vorherigen Zustand zurück zu setzen. Es soll dem Anwender möglich sein, den aktuellen Datensatz bzw. den gefilterten Datensatz einer Nachberechnung der Δ-Werte zu unterziehen.

3.1.5 Ausdrucken der Auswertungen

Die ausgewerteten Datensätze sollen auf DIN A4 Papier ausgedruckt werden. Es ist erwünscht, dass auf einem A4 Papier maximal 6 Darstellungen abgebildet werden sollen.

3.2 Lösungsstrategien

Der Schwerpunkt der Software liegt bei der Bearbeitung von Datensätzen, Auswertungsmöglichkeiten, Filteroperationen und Ausdrucken. Daher wird das Programm in vier Subsysteme unterteilt:

- Datensatzbearbeitung
- Auswertungsverwaltung
- Filterverwaltung
- Ausdruckverwaltung

Nach dem Laden eines Datensatzes muss er zuerst bearbeitet werden. Eine Filteroperation steht dem Anwender, vor oder nach dem Durchführen einer Auswertung, zur Verfügung. Letztlich können die Auswertungen über die Ausdruckverwaltung ausgedruckt werden.

4 Realisierung der Software

Im diesem Kapitel werden die Anforderungen der Software anhand der Flussdiagramme, des grafischen Benutzerinterfaces, der UML-Diagramme und anschließend des Kodes implementiert. Im Kapitel 4.1 wird die Sprache Java vorgestellt, in der die Software programmiert wurde. Die technischen Voraussetzungen für einen einwandfreien Betrieb der Software werden in Kapitel 4.2 festgelegt.

4.1 Die Programmiersprache JAVA

Java ist eine plattformunabhängige Objektorientierte Programmiersprache, deren Grundlagen in der ersten Hälfte der 1990er Jahre von Sun Microsystems entwickelt worden sind. Die Syntax von Java lehnt sich stark an C++ an, obwohl die Sprache selbst sich in vielen Punkten unterscheidet.

Bis zur Einführung von Java 1995 wurden von den meisten Programmierern hauptsächlich Sprachen wie Eiffel und Smalltalk verwendet, um Objektorientiere Programme zu erstellen. Beide sind durch Java stark zurückgedrängt worden. Im kom-

merziellen Bereich dominierte C++ die objektorientierte Entwicklung, was aber ebenfalls in hohem Maße durch Java ersetzt worden ist. Für Echtzeitanwendungen im technischen Bereich ist allerdings C++ vorzuziehen.

Java nimmt in der klassischen Unterscheidung zwischen interpretierten und kompilierten Programmiersprachen eine Sonderstellung ein. Der Quelltext wird zwar kompiliert, aber nicht für ein spezielles Zielsystem. Stattdessen entsteht ein Zwischencode (Bytecode), der von eine Java Virtual Machine (JVM) ausgeführt wird. Deshalb muss auf jedem Zielsystem neben dem eigentlichen Java-Programm auch die Laufzeitumgebung JRE (Java Runtime Environment) installiert werden [5].

Neben den für Programmiersprachen üblichen Sprachelementen (reservierte Wörter, Variablen, elementare Datentypen und Kontrollstrukturen) unterstützt Java nahezu alle Techniken der Objektorientierung: Geheimnisprinzip, Botschaften, Vererbung, Polymorphie und Überladung. Explizit nicht unterstützt wird Mehrfachvererbung, die als übermäßig komplex und in der Praxis unbedeutend angesehen wird. Stattdessen wird auf das Entwurfsmuster der Interfaces verwiesen [5, 6].

Java deckt viele Themenbereiche ab und wird durch eine Vielzahl von Bibliotheken (APIs) ergänzt. Deshalb benennt "Java" heute oft nicht nur die Sprache selbst, sondern dient als Oberbegriff für alle zugrunde liegenden Technologien. Diese sind durch die Plattformunabhängigkeit und das breite Verwendungsspektrum der Sprache geprägt [5, 6].

4.2 Technische Voraussetzungen

Damit die Software einwandfrei laufen kann, muss das System bestimmte Voraussetzungen erfüllen. Die Geschwindigkeit des Prozessors soll mindestens 2 GHz betragen. Es ist zu empfehlen, dass der Arbeitspeicher größer als 1G ist. Die Software ist in der Lage, sowohl auf Linux als auch auf Windows Betriebssystemen zu laufen. Das System muss über eine Laufzeitumgebung JRE v5.0 verfügen.

4.3 Flussdiagramm

Das gesamte System wird zuerst anhand eines Flussdiagramms implementiert. Das System ist in fünf Stufen unterteilt. Diese sind Laden, Bearbeitung, Auswertung, Filtern des Datensatzes und Ausdruck der Auswertung.

Das gesamte System

Als erstes werden die Datensätze geladen. In der zweiten Stufe werden sie bearbeitet dann geht es zur deren Auswertung. Der Anwender hat als nächstes die Möglichkeit, den Datensatz zu filtern. Nach dem der Anwender die gewünschten Darstellungen der Auswertungen bereitgestellt hat, kann er diese auf A4 Papier-Format ausdrucken.

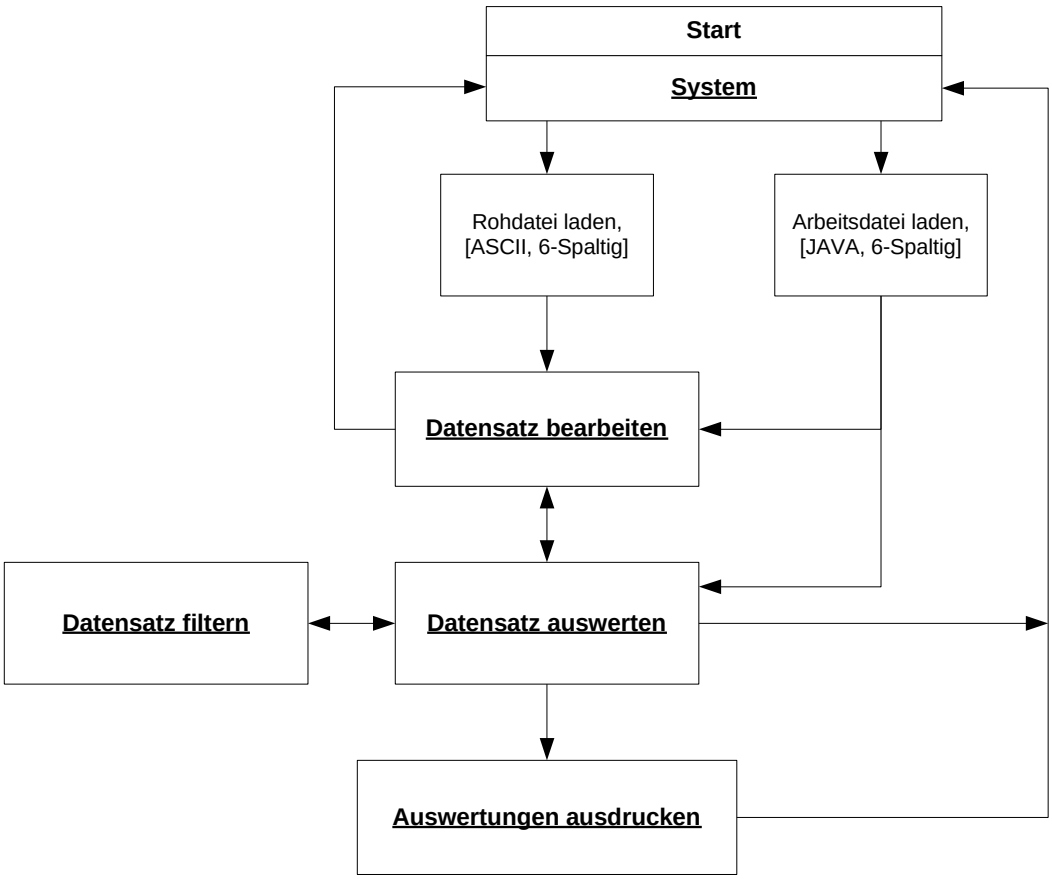

Abbildung 2 : Gesamtsystem Flussdiagramm

Datensatz bearbeiten

Die Abbildung 3 stellt den Ablauf der Verarbeitung der TE-Messdaten dar. Nachdem eine Datei geladen wird, wird das Dateiformat überprüft.

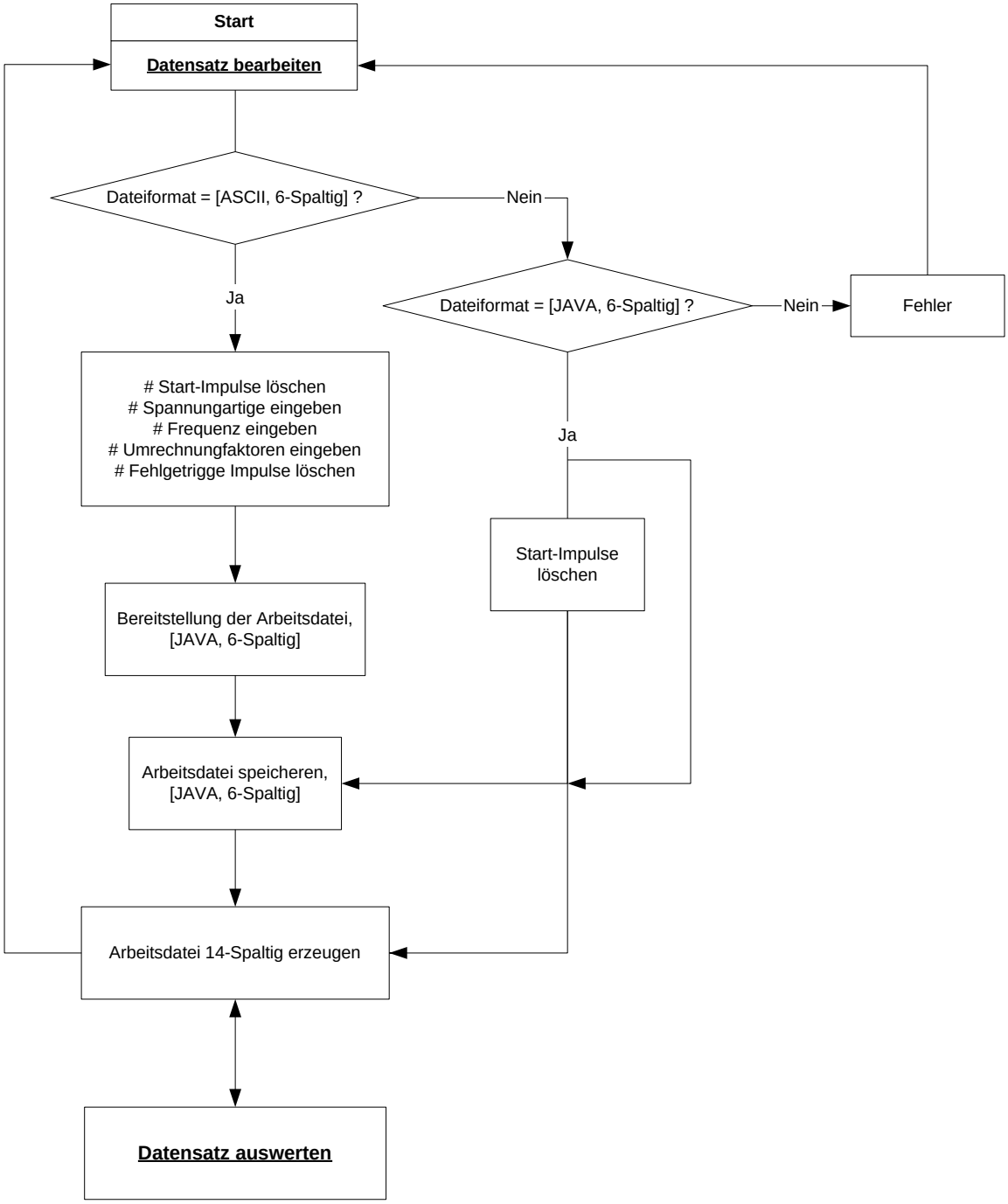

Abbildung 3 : Datensatz Bearbeitung Flussdiagramm

Liegt die Datei im ASCII-Format vor, so wird sie bestimmten Operationen, wie z.B. Löschung der Start-Impulse, Bearbeitung der eingegebenen Spannungsart, der eingegebenen Frequenz, der eingegebenen Umrechnungsfaktoren und der Löschung von Fehlerimpulse, unterzogen. Sind alle Operationen vollzogen, so wird aus der 6-spaltigen Datei eine 14-spaltige Datei erzeugt, die zum weiteren Einsatz verwendet wird.

Datensatz auswerten

Zur Erstellung einer Auswertung wird zwischen einem Scatter-Plot und einem Histogramm differenziert. Für jede Auswahl werden bestimmten Parameter festgesetzt, um eine Darstellung zu erstellen. Es gibt die Möglichkeit, die erwünschten Auswertungen zu einer Auftragsliste hinzufügen, die in einer späteren Phase importiert werden kann. Der aktuelle Datensatz kann in dieser Phase gefiltert werden.

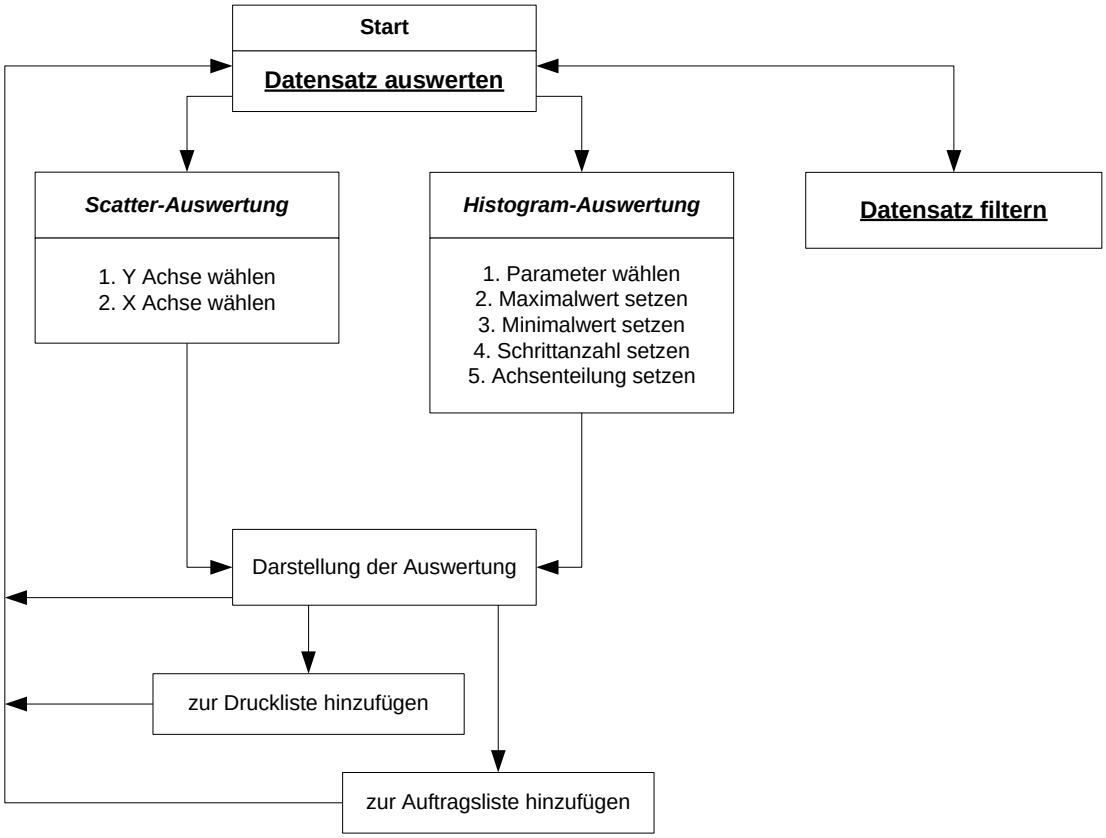

Abbildung 4 : Datensatz Auswertung Flussdiagramm

Datensatz filtern

Der Datensatz kann anhand der angegebenen Parameter und deren Grenzen gefiltert werden. Es gibt eine zusätzliche Option, die die Delta-Werte im Datensatz neu berechnet.

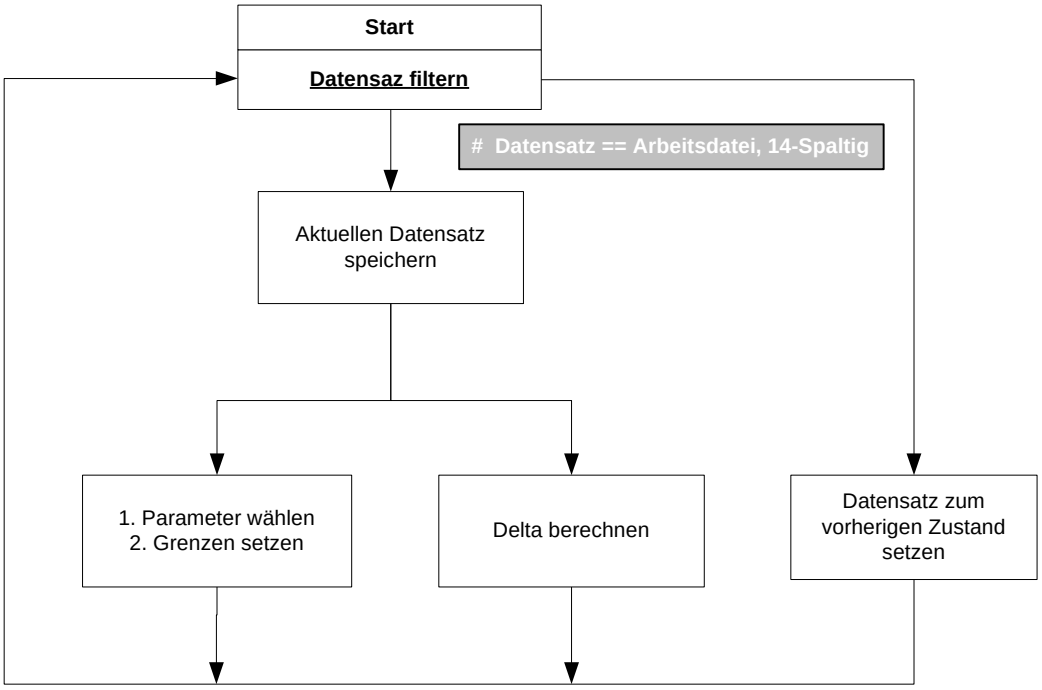

Abbildung 5 : Datensatz Filtern Flussdiagramm

Auswertungen ausdrucken

Die Auswertungen, die ein Scatter-Plot oder Histogramm sein können, werden entweder aus der Druckliste oder aus der Auftragsliste ausgedruckt. Es liegt eine zusätzliche Option vor, die die Achsen der Darstellungen einstellen kann.

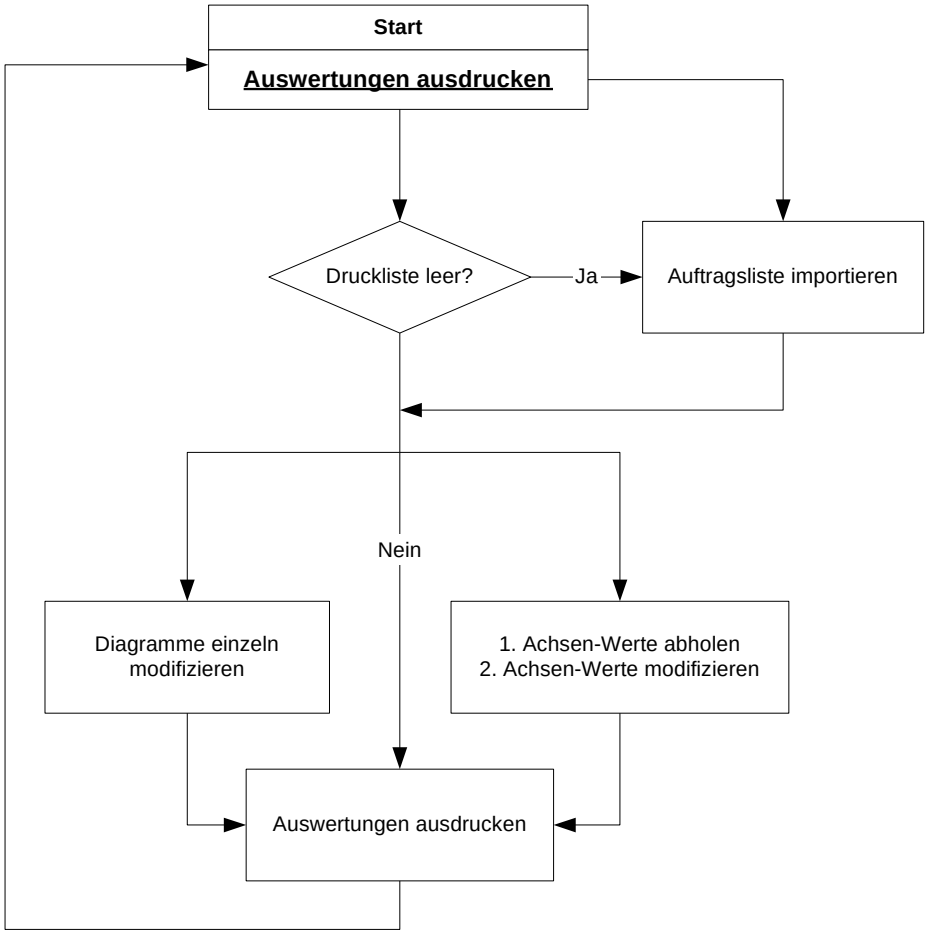

Abbildung 6 : Ausdrucken der Auswertungen - Flussdiagramm

4.4 GUI Realisierung

Die Implementierung des grafischen Benutzerinterfaces „*Graphical User Interface - GUI*“ der Software wird in diesem Kapitel behandelt. Beim Starten des Programms erscheint das Fenster, das in der Abbildung 7 ersichtlich ist.

Abbildung 7 : Haupt-Fenster

Unter den Menüpunkt *Datei* hat der Anwender die Möglichkeit eine Datei zu lesen, zu laden oder das Programm zu beenden. Die Abbildung 8 zeigt den Menüpunkt *Datei*.

Abbildung 8 : Menupunkt Datei

Im Menüpunkt *Bearbeitung* kann der Anwender eine Datei-Info aufrufen, die Informationen über eine geöffnete Datei gibt. Auch kann der Anwender die Auswertungsverwaltung bzw. Ausdruckverwaltung aufrufen.

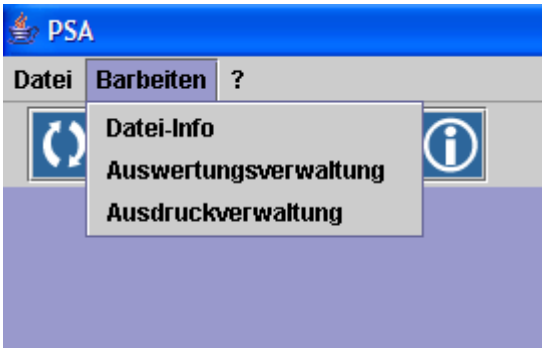

Abbildung 9 : Menupunkt Bearbeitung

Die im Start-Fenster befindliche Symbolleiste, die in der Abbildung 10 ersichtlicht wird, besitzt folgende Symbole:

Abbildung 10 : Hauptfenster Symbolleiste

4.4.1 Datei lesen

Die Messdaten, die in Form einer sechsspaltigen Datenmatrix im ASCII– Code vorliegen, können durch die Aufrufung des Menüpunktes *Datei* > *Datei lesen* oder den Button *Datei lesen* geladen werden. Beim Laden der Datei wird eine Tabelle, in der die sechsspaltige Datenmatrix dargestellt ist, und ein Dialogfenster, das so genannte Datensatzbearbeitungs-Fenster, erscheinen. Diese ist in der Abbildung 11 ersichtlich.

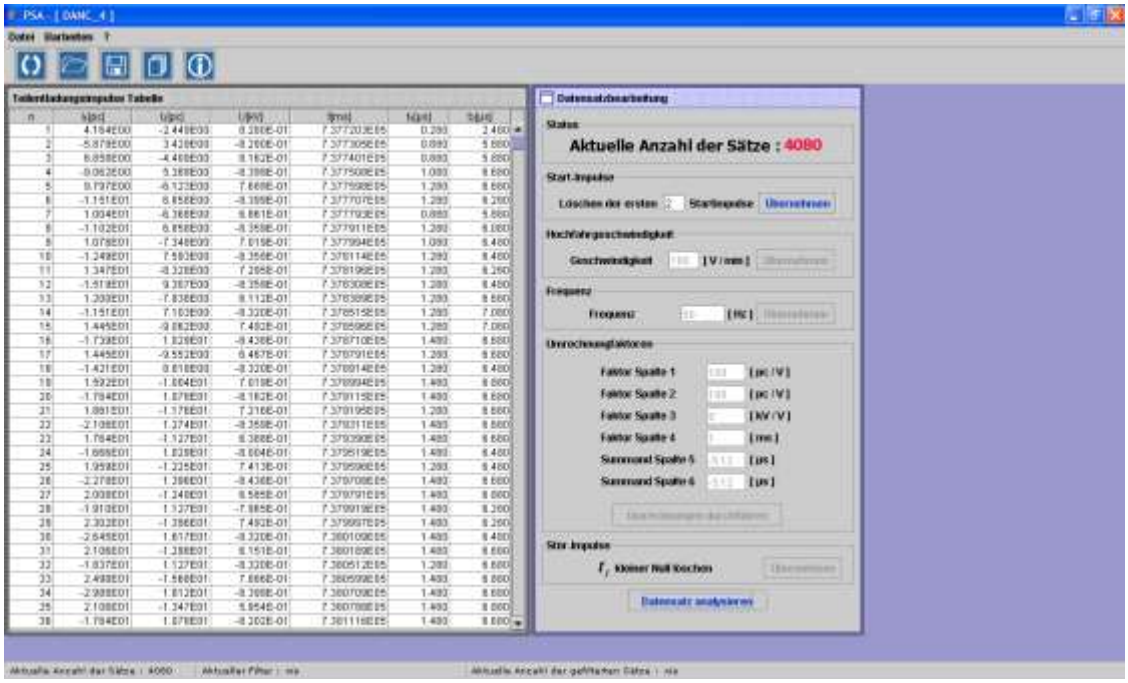

Abbildung 11 : Datei laden

4.4.2 Datensatzbearbeitung

Nachdem eine Datei geladen wurde, kann der Anwender den Datensatz bearbeiten, dabei können folgende Operationen im Datensatzbearbeitungs-Fenster durchgeführt werden;

- Start-Impulse löschen
- Hochfahrgeschwindigkeit einsetzen
- Frequenz einsetzen
- Umrechnungsfaktoren einsetzen
- Stör-Impulse mit z.B. t_1 kleiner Null löschen

Die Abbildung 12 zeigt das Datensatzbearbeitungs-Fenster. Diese vorherigen Operationen außer der Löschung der Start-Impulse können nacheinander durchgeführt werden. Die Abbildung 12 zeigt auch wie die Schaltfläche deaktiviert sind, solange die vorherige Operation nicht durchgeführt wurde.

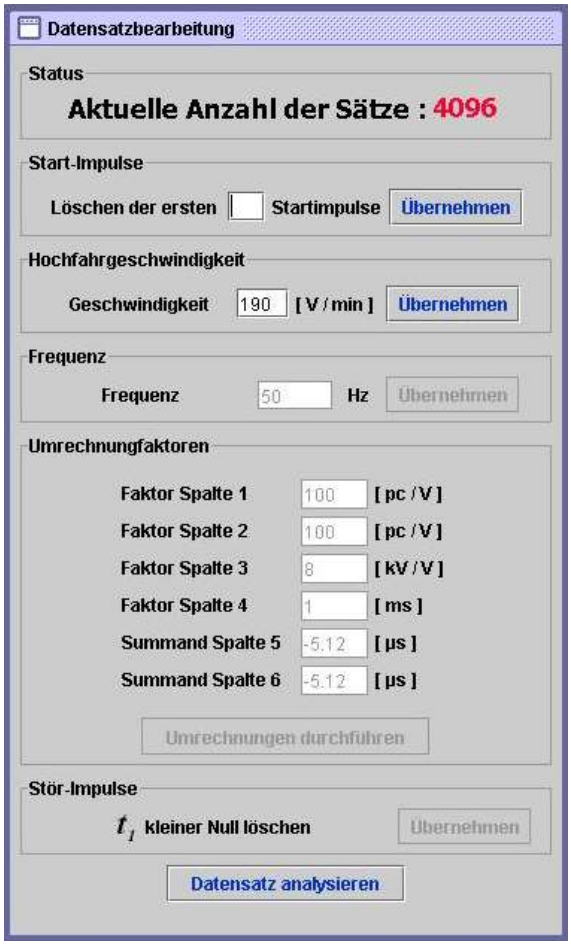

Abbildung 12 : Datensatzbearbeitung

4.4.3 Auswertungsverwaltung

Sind alle Operationen an dem Datensatz vollzogen worden, tritt die nächste Stufe der Software in Aktion. In der Auswertungsverwaltung wird aus der 6-spaltigen Datei eine 15-spaltige Datei erzeugt, mit der die Auswertung beginnen kann. Die Auswertungsverwaltung unterteilt sich in fünf Registerkarten (Auftragliste, Chart, Scatter, Histogramm und Funktionstabelle), die in der Abbildung 13 ersichtlich sind.

Ein Chart ist ein Oberbegriff für Scatter und Histogramm. Bei einer Auswertung hat der Anwender die Möglichkeit zwischen einem Scatter-Plot und einem Histogramm zu wählen, was am Ende die gewünschte Auswertung in Form eines Charts darstellt. Die Registerkarte-Scatter gibt dem Anwender die Auswahlmöglichkeit zwischen 15-Parametern zu jeder Achse zu wählen.

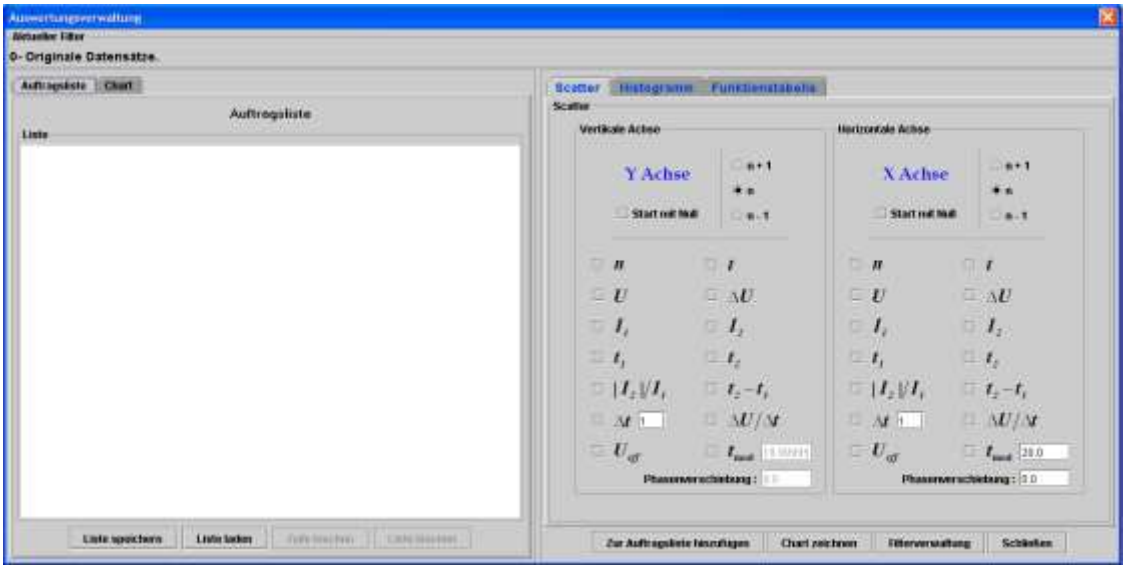

Abbildung 13 : Auswertungsverwaltung

Die Registerkarte-Histogramm, die in der Abbildung 14 ersichtlich ist, stellt dem Anwender eine Auswahlmöglichkeit zwischen 10-Parametern in Form eines Kontrollfeldes zur Verfügung. Zusätzliche Optionen und Einstellungen der Achsen bzw. Darstellung des Histogramms stehen in Form eines Eingabefeldes zur Verfügung. Maximalwert bzw. Minimalwert und Schrittanzahl bzw. Achsenteilung der Achsen können seitens des Anwenders eingegeben werden.

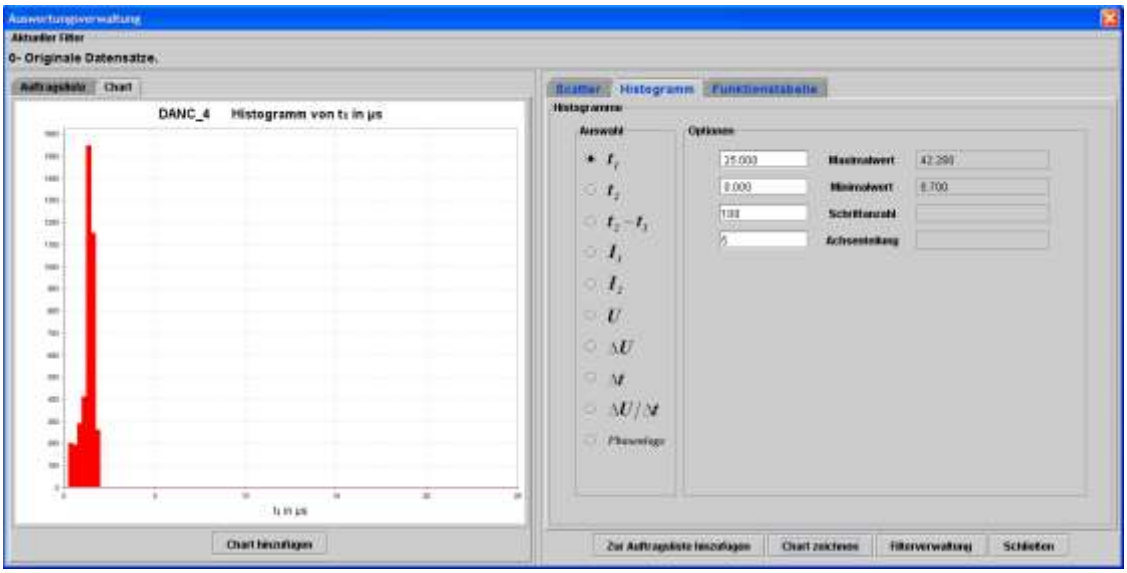

Abbildung 14 : Auswertungsverwaltung - Histogramm

Sind die Parameter für eine Auswertung aus einem Scatter-Plot bzw. Histogramm ausgewählt und wird dann auf die Schaltfläche *Chart* zeichnen geklickt, erscheint das

Auswertungs-Chart in der Registerkarte Chart. Die Abbildung 15 zeigt eine Auswertung mit Auswahl-Parametern (Index über t).

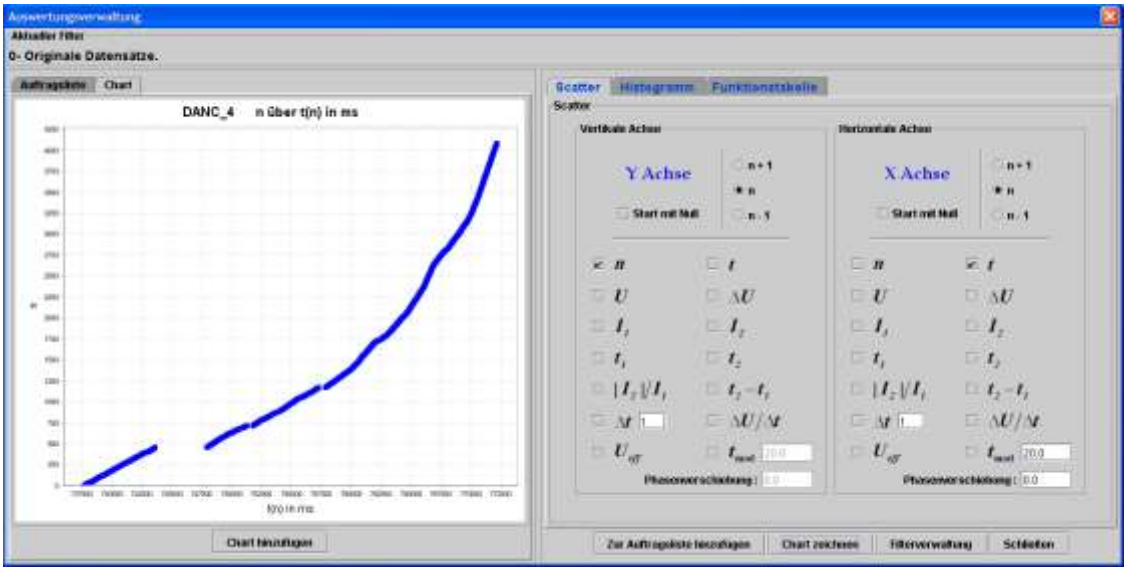

Abbildung 15 : Auswertungsverwaltung - Chart

Der Anwender hat die Möglichkeit, mehrere Auswertungen in einer Liste zu erstellen, um den Aufwand der wiederholten Parameter-Auswahl der unterschiedlichen Messdaten zu vermeiden. Die Liste kann zu jeder Zeit gespeichert und zu einem späteren Zeitpunkt geladen werden. Die Abbildung 16 zeigt, wie bei einem Anklicken der Schaltfläche „Liste speichern“ eine Auftragsliste gespeichert wird. Wird die Schaltfläche „Liste laden“ angeklickt, so wird die Auftragsliste geladen. Die Schaltflächen „Zeile löschen“ und „Liste löschen“ haben die Funktion, die ausgewählte Auswertung bzw. alle Auswertungen aus der Liste zu löschen.

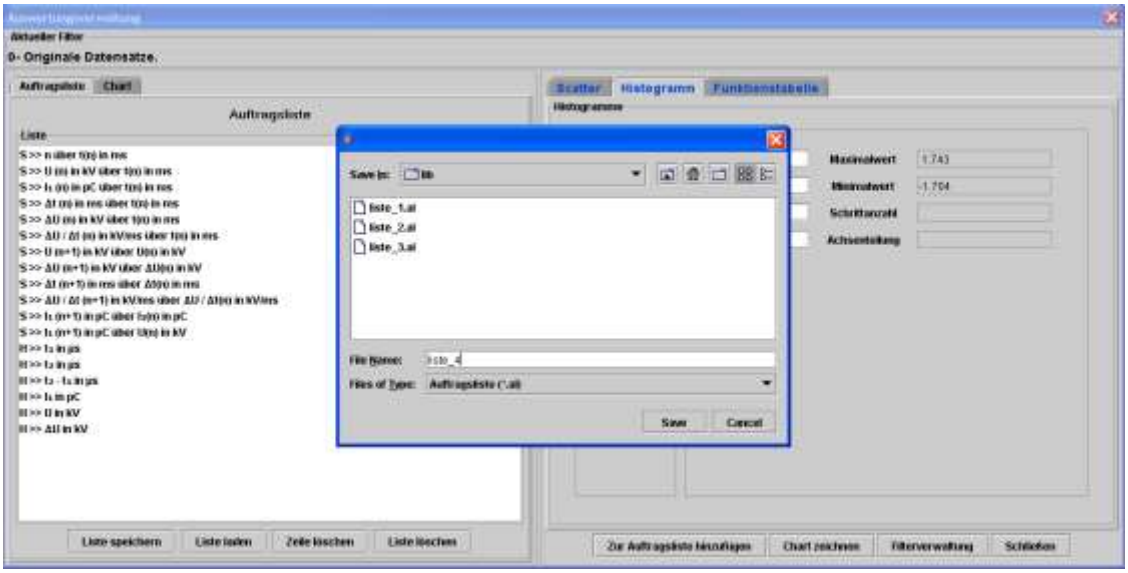

Abbildung 16 : Auswertungsverwaltung - Auftragsliste

Die Registerkarte-Funktionstabelle stellt die 15-spaltige Datei anhand einer Tabelle dar, diese wird durch die Abbildung 17 ersichtlich.

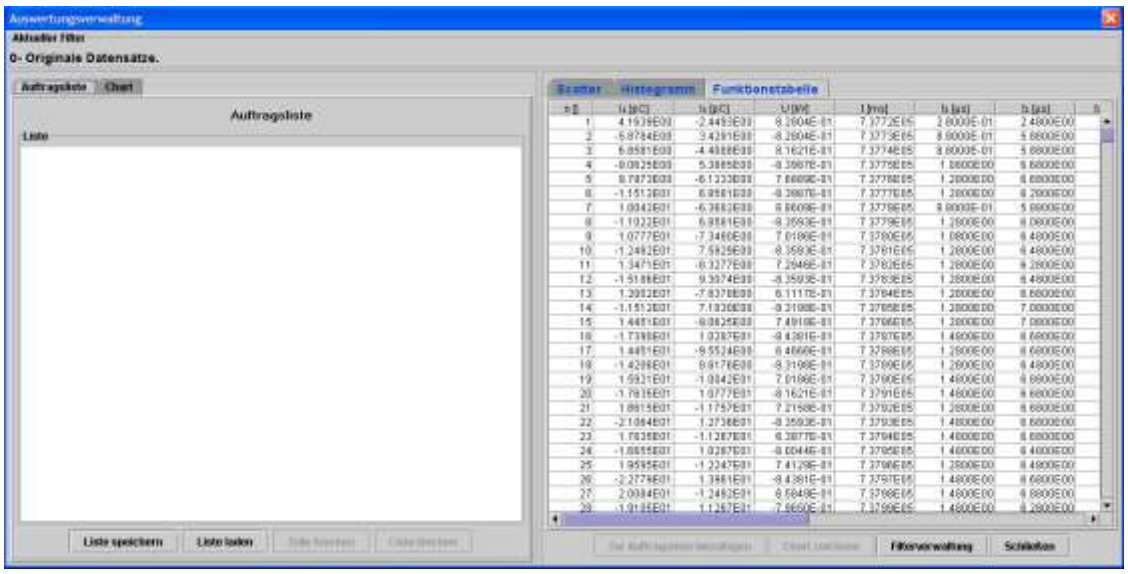

Abbildung 17 : Auswertungsverwaltung - Funktionstabelle

Um die Datensätze zu filtern, wird die Filterverwaltung durch Anklicken der Schaltfläche *Filterverwaltung* aufgerufen. Diese wird im nächsten Kapitel diskutiert.

4.4.4 Filterverwaltung

Die Messdaten können nach bestimmten Parametern gefiltert werden. Nach der Filterung der Messdaten stehen sie zur weiteren Auswertung zur Verfügung. Die Abbildung 18 stellt die Filterverwaltung vor, welche aus drei Bereichen - *Filter Status*, *Filter*

Auswahl und *Filter Liste* - besteht. Der Bereich *Filter Status* zeigt den aktuellen Filter an. Aus dem Bereich *Filter Auswahl* werden die Parameter und die Werte der Filter eingesetzt. Die Schaltfläche „Δ *berechnen*“ führt eine Delta-Berechnung an dem aktuellen Datensatz durch. Der Bereich *Filter Liste* listet die an den Messdaten angeführten bzw. eingesetzten Filter auf. Beim Anklicken der Schaltfläche „*Filter“* wird die Filter-Operation durchgeführt. Die Schaltfläche „*Letzte Operation rückgängig machen“* setzt den aktuellen Datensatz auf die vorherige Filter-Operation zurück.

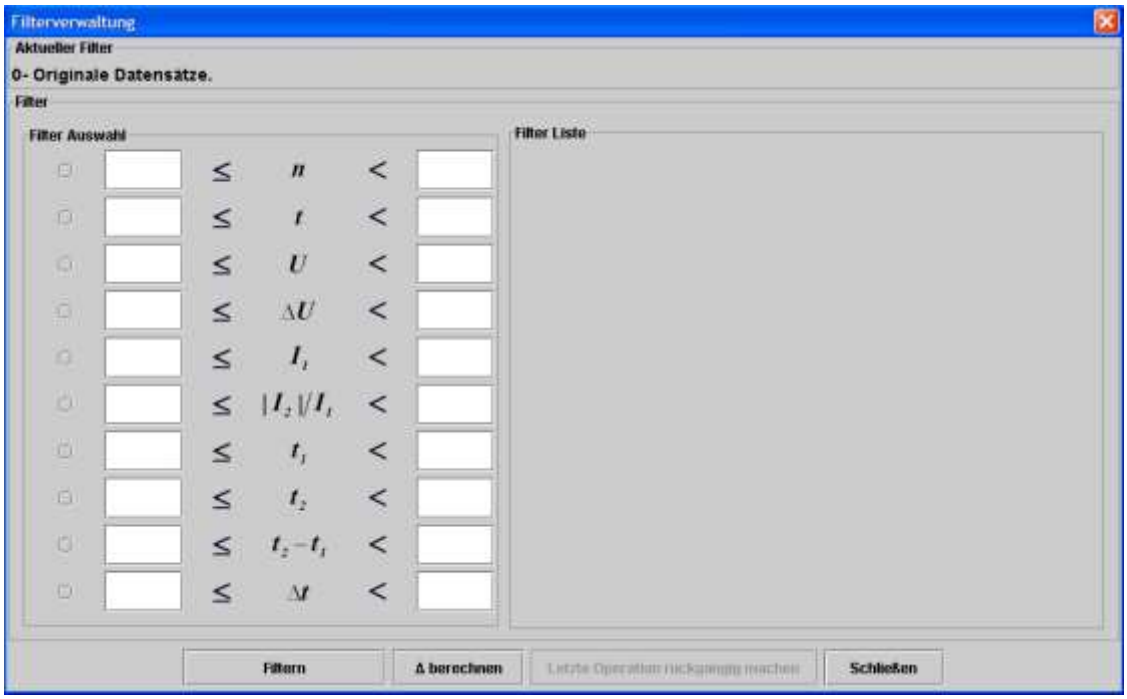

Abbildung 18 : Filterverwaltung

4.4.5 Ausdruckverwaltung

Die Ausdruckverwaltung verwaltet die auszudruckenden Auswertungen, die durch das Anklicken, entweder auf die Schaltfläche „*Chart zeichnen“* oder „*zur Auftragsliste hinzufügen*“ in der Auswertungsverwaltung, entstanden sind. Die durch das Anklicken auf die Schaltfläche „*Chart zeichnen*“ entstehenden Auswertungen werden Automatisch in die Ausdruckverwaltung importiert. Hingegen müssen die durch das Anklicken der Schaltfläche „*zur Auftragsliste hinzufügen*“ entstehenden Auswertungen bzw. die Auftragsliste durch den Anwender importiert werden.

Die Ausdruckverwaltung besteht hauptsächlich aus den folgenden Funktionen; Seiten blättern, Auftragsliste importieren, Achsen definieren und Seiten ausdrucken. Die Ab-

bildung 19 zeigt das Ausdrucksverwaltungs-Fenster. Die Symbolleiste in der Ausdruckverwaltung besteht aus fünf Schaltflächen. Im Textfeld wird der Text eingegeben, der auf den zu druckenden Seiten der Auswertungen erscheinen soll.

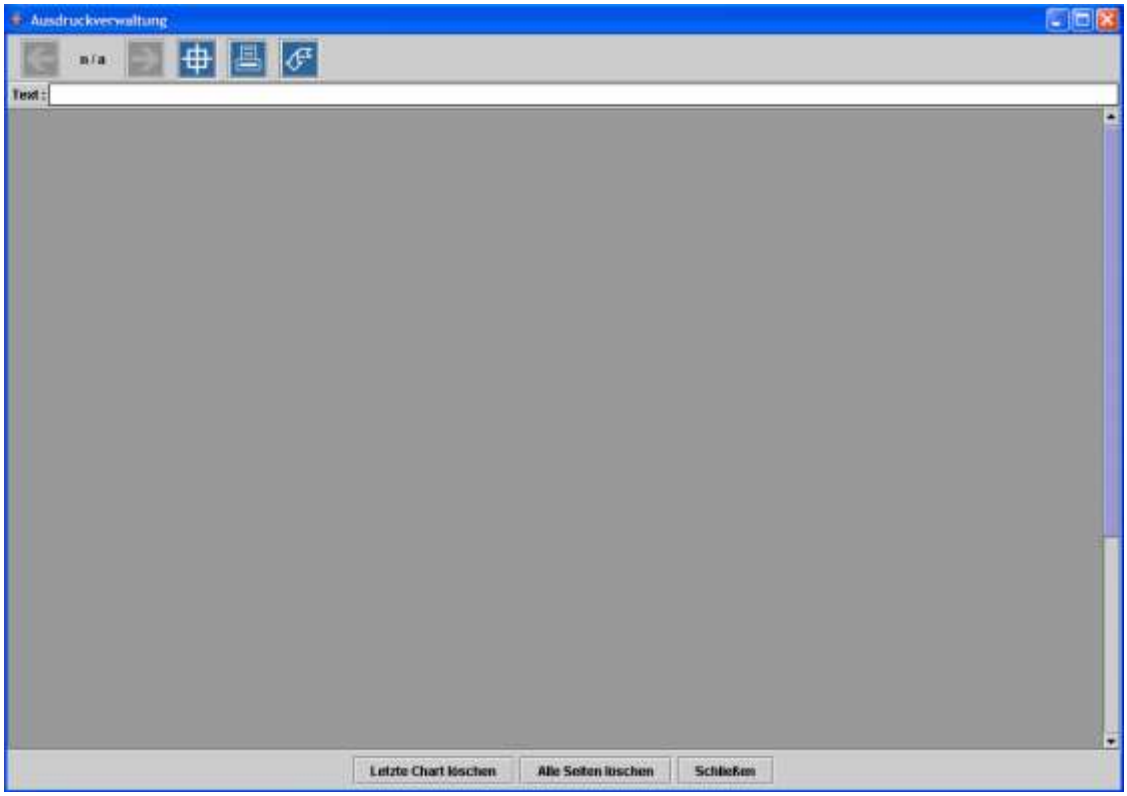

Abbildung 19 : Ausdruckverwaltung

Die unten aufgelisteten Symbole dienen den folgenden Funktionen:

-  Seiten blättern
-  Aufrufen der Achsen-Eigenschaft Fenster
-  „Auswertung ausdrucken“ zum Ausdrucken der Auswertungen
-  Importieren der Auftragsliste

Wird das Symbol *Auftragsliste-importieren* angeklickt, so werden die in der Austragsliste gespeicherten Auswertungen in der Ausdruckveraltung dargestellt. Abbildung 20 zeigt die Ausdruckverwaltung nach Importieren der Auswertungen. Bei der Eingabe eines Texts im Textfeld wird er automatisch auf allen Seiten abgebildet. Ist die Schalfläche „*Letzte Chart löschen“* geklickt, wird der letzte Chart auf der letzten Seite ge-

löscht. Hat der Anwender den Wunsch, alle Seiten zu löschen, wird dieses durch das Anklicken der Schaltfläche *„Alle Seiten löschen“* durchgeführt.

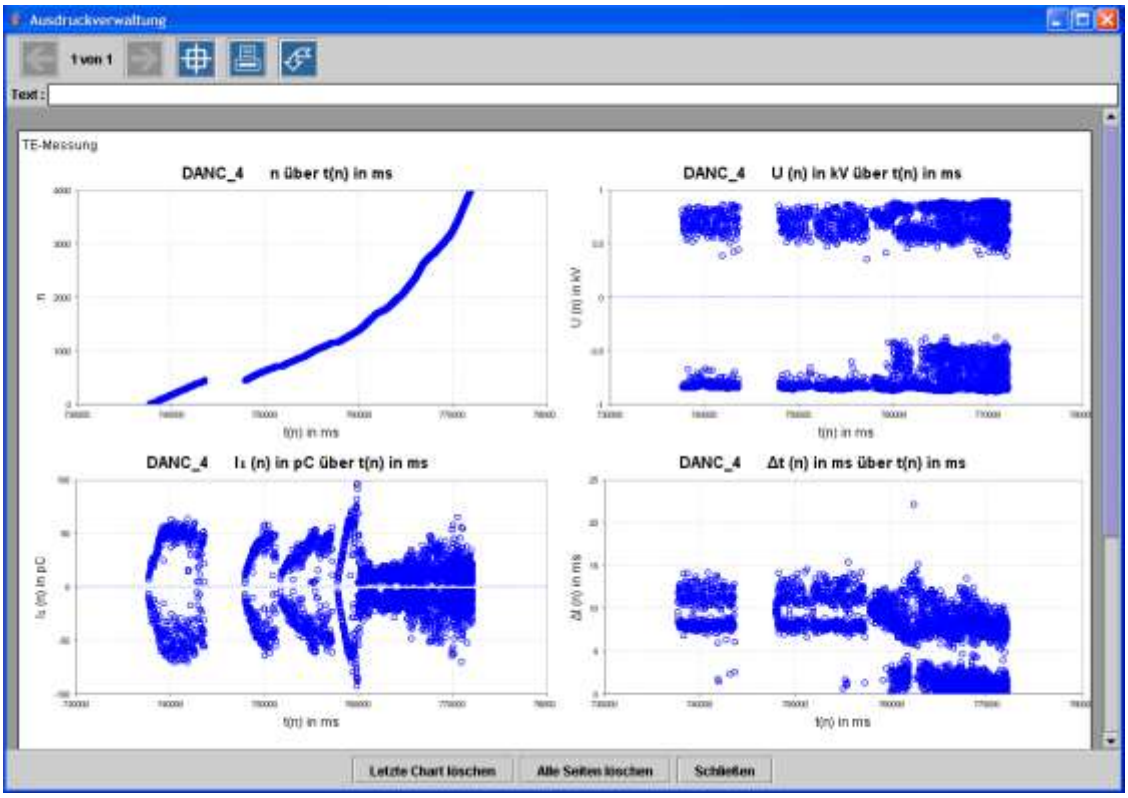

Abbildung 20 : Auswertungen-importieren

Wünscht der Anwender die Achsen-Einstellungen einer Chart zu ändern, wird dieses durch das Klicken der rechten Maustaste erreicht, wobei ein Fenster erscheint, in dem die Achsen-Einstellung eines Charts festgelegt werden kann. Die Abbildung 21 zeigt das Fenster zur Einstellung eines Charts.

Abbildung 21 : Achsen-Einstellung eines Charts

Die oben beschriebene Methode zur Einstellung von Achsen ist nur für wenige Charts praktisch. Liegen mehrere Charts in der Ausdruckverwaltung vor, ist eine universale Achsen-Einstellung praktischer als jeden einzelnen Chart einzustellen. Indem der An-

wender das Symbol Achsen-Einstellung in der Symbol-Leiste anklickt, erscheint ein Fenster, das in der Abbildung 22 ist. Da die Charts entweder aus Scatter-Plots oder Histogrammen bestehen können, besteht die universale Achsen-Einstellung aus allen möglichen Parametern, aus denen ein Scatter-Plot bzw. ein Histogramm dargestellt wird. Die Abbildungen 22 und 23 zeigen alle möglichen Parameter für ein Scatter-Plot bzw. ein Histogramm. Die Werte der zahlreichen Parameter können abgerufen werden, indem der Anwender die Schaltfläche „*Werte holen*“ anklickt. So müssen die Werte der zahlreichen Parameter nicht einzeln eingegeben werden, sondern werden auf einmal abgerufen. Danach kann der Anwender die gewünschten Werte der Parameter beliebig ändern und anschließend in den Charts einsetzen, indem er die Schaltfläche „*Werte setzen*“ anklickt.

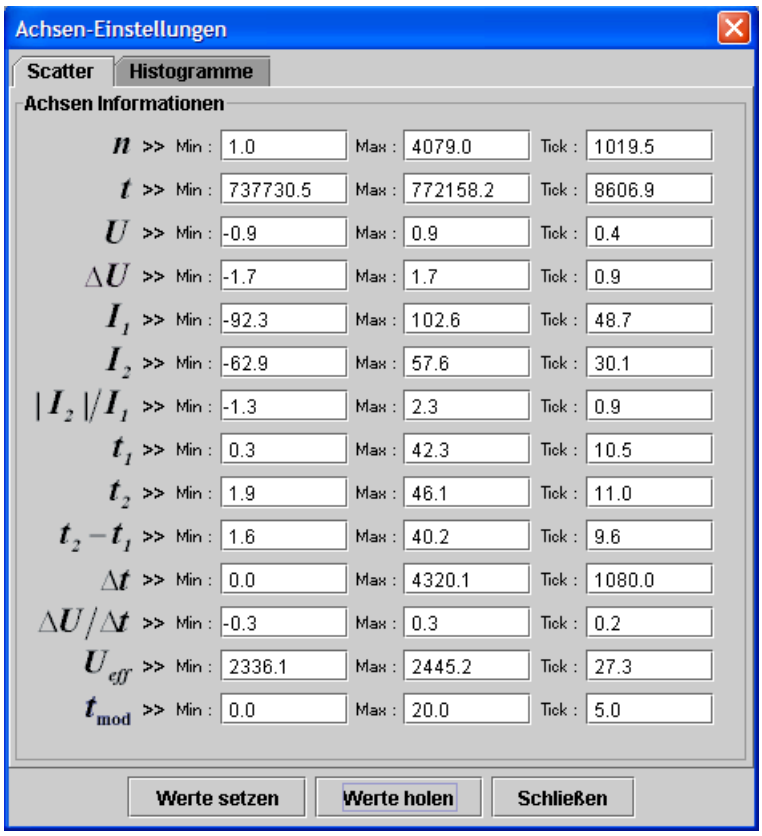

Abbildung 22 : Achsen-Einstellung/Scatter

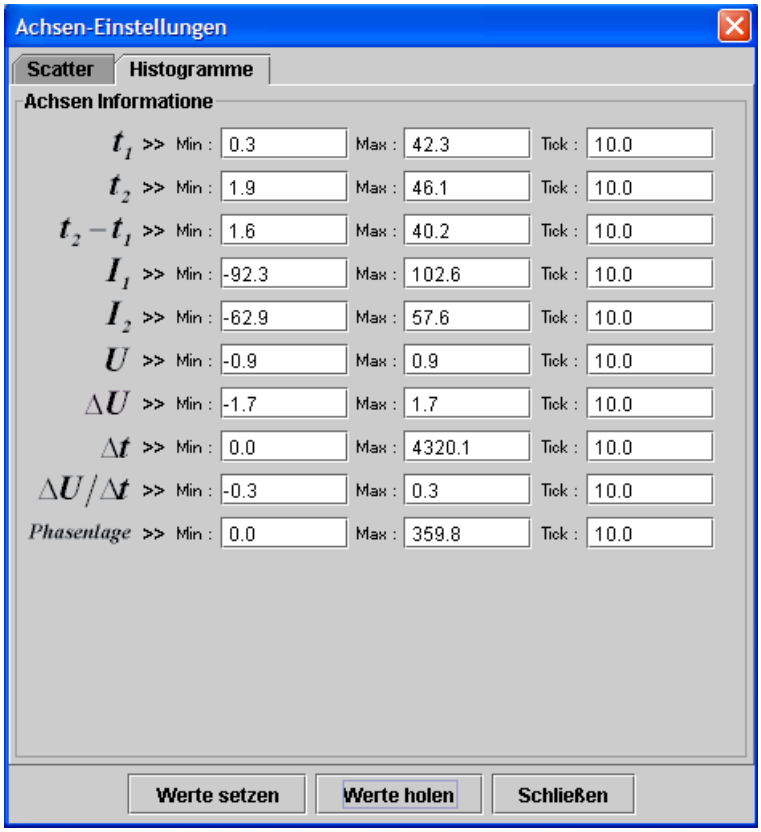

Abbildung 23 : Achsen-Einstellung/Histogramm

Zum Ausdrucken der Auswertungen ist das Symbol „*Auswertungen ausdrucken*“ zu klicken. Demzufolge werden die in der Ausdruckverwaltung dargestellten Auswertungen auf A4-Format Papier ausgedruckt.

Die Abbildungen 24, 25, 26 zeigen die üblichen Auswertungsmöglichkeiten, die im PSA-Verfahren eingesetzt werden. Dabei wird zwischen TE-Sequenz, TE-Korrelation und TE-Histogramm unterschieden. Die Software druckt diese Auswertungen aus und stellt sie zur weiteren Analyse zur Verfügung. Ein Ausdruck dieser Abbildung wird als Anhang eingefügt.

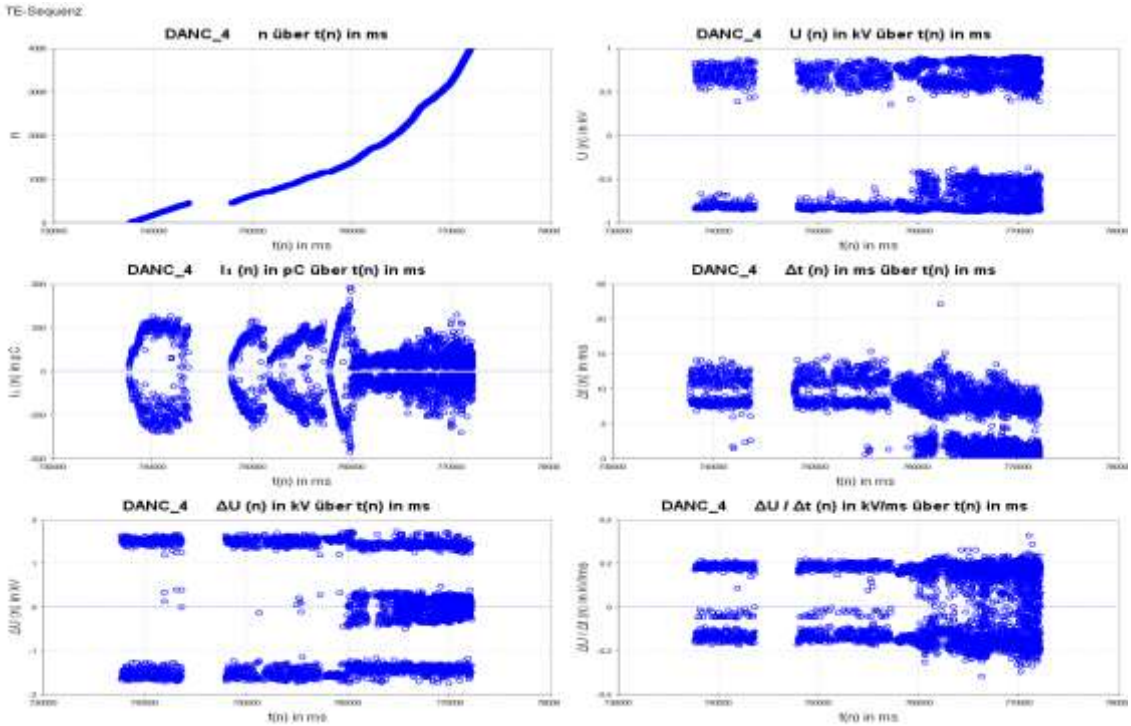

Abbildung 24 : TE-Sequenz

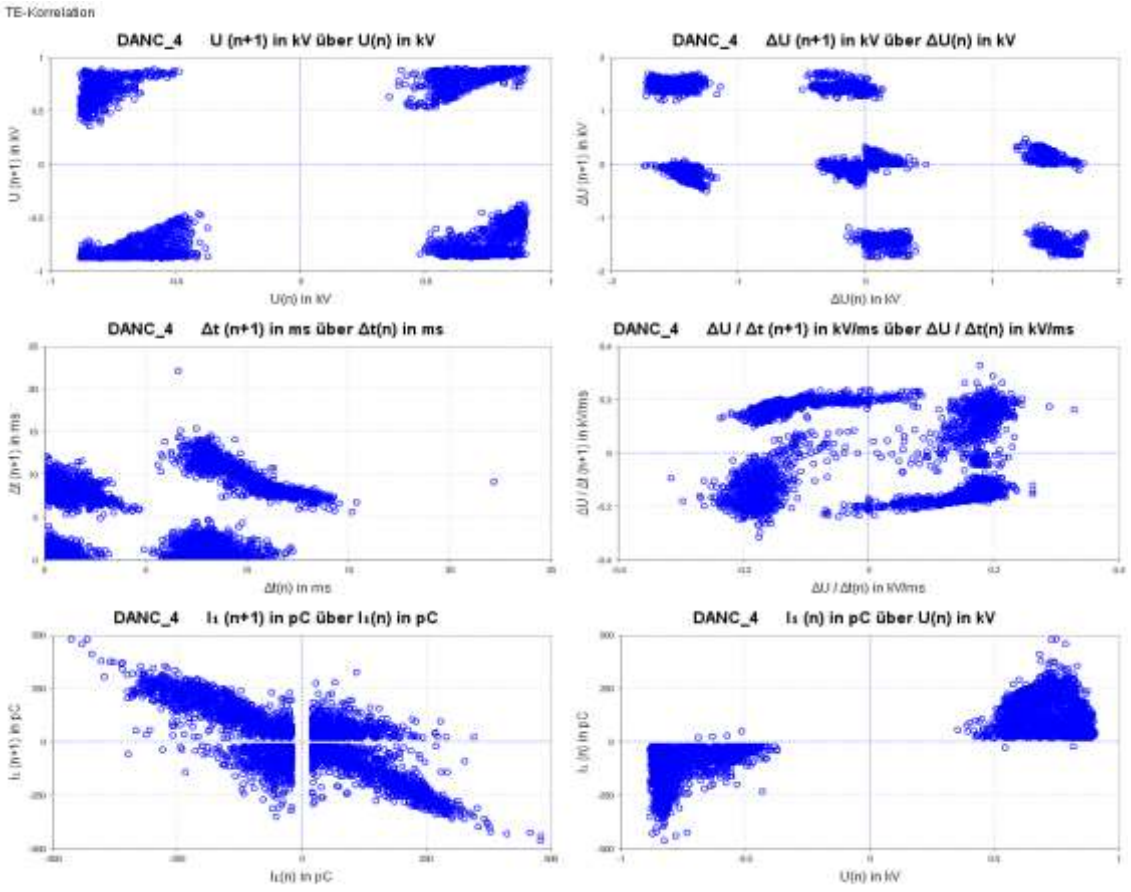

Abbildung 25 : TE-Korrelation

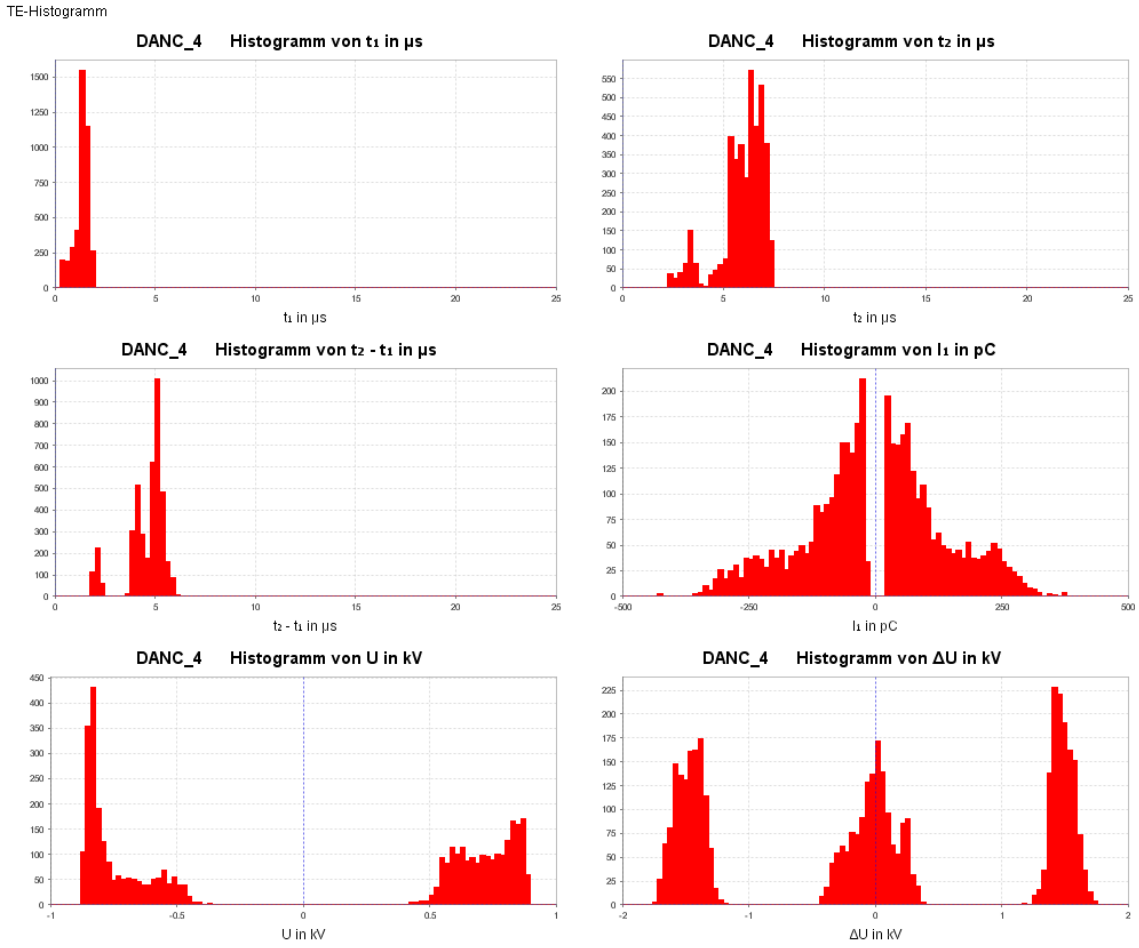

Abbildung 26 : TE-Histogramm

4.5 UML-Diagramm

4.5.1 Was ist UML

UML ist die Abkürzung für "Unified Modelling Language". UML ist eine durch die *Object Management Group* (OMG) standardisierte graphische Sprache zur Beschreibung objektorientierter Modelle. Mit UML werden Softwaresysteme spezifiziert, visualisiert, konstruiert und dokumentiert; Folglich ist UML eine Sprache, die zur Konstruktion und Dokumentation eines Systems verwendet wird. Ein UML-Diagramm besteht aus Symbolen, die durch Pfeile miteinander verbunden sind. Die UML-Sprache spielt in der Entwicklung objektorientierter Software und beim Software-Entwicklungsprozess eine wichtige Rolle. Die Konstruktion von Softwareprojekten wird hauptsächlich durch grafische Notationen vorgenommen. Mit Hilfe von UML kommunizieren Projektgruppen miteinander, mögliche Architekturen werden erkundet und Softwaredesigns werden validiert [7, 8].

4.5.2 UML-Diagramm des Systems

Es wird ein UML-Diagramm für die wichtigsten Aspekte des Systems erstellt, da die anderen Aspekte wie z.B. die Masken Java-Standard sind und für diese Arbeit irrelevant sind. In der Abbildung 27 werden die wichtigsten Klassen des Programms gezeigt.

Die Klasse *Impuls* stellt die Struktur eines Impulses und dessen Parameter $[I_1, I_2, U, t, t_1, t_2]$ dar. Die Klasse „*ImpulsDaten"* stellt die TE-Messdaten dar, die aus 4096 Impulsen besteht. Die Klasse „*SystemVerwaltung"* stellt das gesamte System dar, und obliegt daher die Klasse „*DatenManagment"*, die für das Einlesen von den TE-Messdaten zuständig ist. Die Klasse „*ImpulsDatenVerwaltung*" dient der Verwaltung der TE-Messdaten. Die Klasse „*Funktionen"* stellt die Arbeitsdatei mit den Parametern $[t_1 - t_2,\ I_2 / I_1,\ \Delta U,\ \Delta t,\ \Delta U / \Delta t,\ U_{eff},\ t \bmod 20,\ Phasenverschiebung]$ dar. Die Klasse „*FunktionenVerwaltung"* ist für die Erstellung der Arbeitsdatei bzw. die *Funktionen*-Klasse und für die Filterung der Arbeitsdatei zuständig. Die Klasse „*Filter"* stellt die Informationen der Filterung bereit, die für den Filterungsprozess der Arbeitsdatei notwendig sind.

Die Klasse Impuls und „*ImpulsDatenVerwaltung"* haben im aktuellen Programm keine große Verwendung. Sie wurden programmiert, so dass sie für zukünftige Erweiterungen des Programms einen Einsatz finden könnten. Mit Hilfe der Klasse Impuls würde es möglich sein einzelne Impulse zu Betrachten. Mit Hilfe der Klasse „*ImpulsDaten-Verwaltung"* könnten mehrere TE-Messdaten gleichzeitig bearbeitet werden.

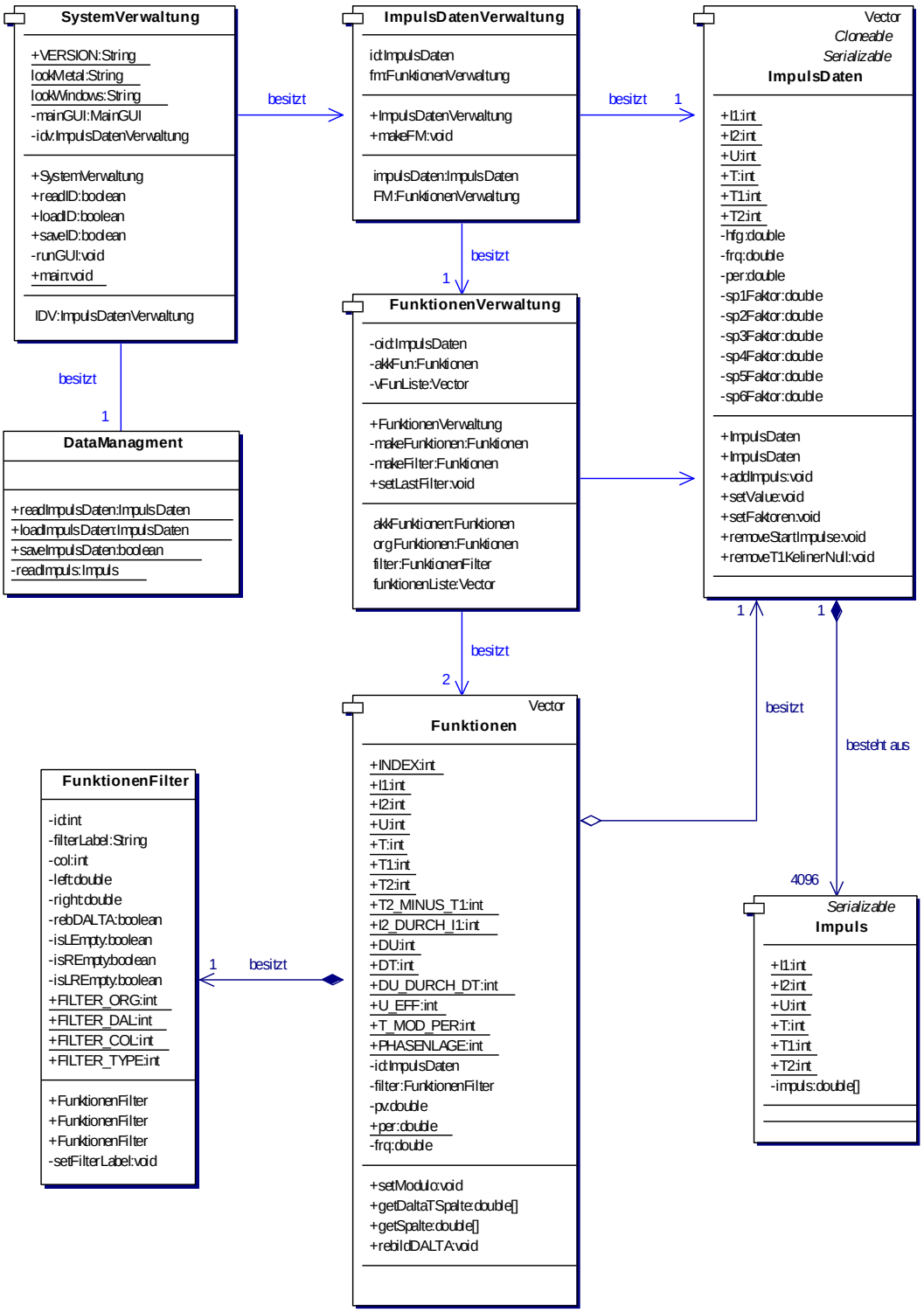

Abbildung 27 : UML-Diagramm des Systems

4.6 Kode Realisierung

In diesem Kapitel werden die Klassen, die in Kapitel 4.5.2 erklärt wurden, in Kode-Form dargestellt. Es wurden nur die wichtigsten Teile des Kodes eingefügt. Der restliche Kode wird wegen Größe und Irrelevanz weggelassen.

4.6.1 Impuls

```
import java.io.Serializable;

public class Impuls
        implements Serializable {

    public static final int I1 = 0;
    public static final int I2 = 1;
    public static final int U = 2;
    public static final int T = 3;
    public static final int T1 = 4;
    public static final int T2 = 5;

    private double[] impuls;

    public Impuls() {
        impuls = new double[6];
    }

    public Impuls(double I1, double I2,
                    double U, double T, double T1,
                double T2) {
        impuls = new double[6];
        impuls[Impuls.I1] = I1;
        impuls[Impuls.I2] = I2;
        impuls[Impuls.U] = U;
        impuls[Impuls.T] = T;
        impuls[Impuls.T1] = T1;
        impuls[Impuls.T2] = T2;
    }

    public double getValue(int index) {
        return impuls[index];
    }

    public void setValue(int index, double value) {
        impuls[index] = value;
    }
}
```

4.6.2 ImpulsDaten

```
public class ImpulsDaten extends Vector implements Cloneable, Serial-
izable {

    public static final int I1 = Impuls.I1;
    public static final int I2 = Impuls.I2;
    public static final int U = Impuls.U;
    public static final int T = Impuls.T;
```

```
    public static final int T1 = Impuls.T1;
    public static final int T2 = Impuls.T2;

    private double hfg; // Hochfahrgeschwindigkeit
    private double frq; // frequenz
    private double per; // periode

    private double sp1Faktor;
    private double sp2Faktor;
    private double sp3Faktor;
    private double sp4Faktor;
    private double sp5Faktor;
    private double sp6Faktor;


    public static final String[] colNames = new String[] {"n",
"I\u2081",
            "I\u2082", "U", "t", "t\u2081", "t\u2082"};
    public static final String[] colEin_1 = new String[] {"", "[V]",
"[V]",
            "[V]", "[ms]", "[\u00B5s]", "[\u00B5s]"};
    public static final String[] colEin_2 = new String[] {"", "[pc]",
"[pc]",
            "[kV]", "[ms]", "[\u00B5s]", "[\u00B5s]"};

    public ImpulsDaten(Vector impulse) {
        super(impulse);
    }

    public void addImpuls(Impuls impuls) {
        super.add(impuls);
    }

    public Impuls getImpuls(int index) {
        Impuls impuls = (Impuls) get(index);
        return impuls;
    }

    public double getValue(int row, int col) {
        Impuls impuls = this.getImpuls(row);
        return impuls.getValue(col);
    }

    public void setValue(int row, int column, double value) {
        Impuls impuls = (Impuls)this.get(row);
        impuls.setValue(column, value);
    }

    public void setHFG(double hfg) {
        this.hfg = hfg;
        isHFG = true;
    }

    public void setFRQ(double frq){
        this.frq = frq;
        this.per = 1000/frq;
        isFRQ = true;
    }

    public void setFaktoren(double fk1, double fk2,
                            double fk3, double fk4,
```

```
                            double fk5, double fk6) {

        this.sp1Faktor = fk1;
        this.sp2Faktor = fk2;
        this.sp3Faktor = fk3;
        this.sp4Faktor = fk4;
        this.sp5Faktor = fk5;
        this.sp6Faktor = fk6;

        for (int i = 0; i < this.size(); i++) {
            Impuls impuls = this.getImpuls(i);
            impuls.setValue(impuls.I1, impuls.getValue(impuls.I1) *
sp1Faktor);
            impuls.setValue(impuls.I2, impuls.getValue(impuls.I2) *
sp2Faktor);
            impuls.setValue(impuls.U, impuls.getValue(impuls.U) *
sp3Faktor);
            impuls.setValue(impuls.T, impuls.getValue(impuls.T) *
sp4Faktor);
            impuls.setValue(impuls.T1, impuls.getValue(impuls.T1) +
sp5Faktor);
            impuls.setValue(impuls.T2, impuls.getValue(impuls.T2) +
sp6Faktor);
        }
        isFaktoren = true;
    }

    public void removeStartImpulse(int anzahl) {
        for (int i = 0; i < anzahl; i++) {
            this.remove(0);
        }
    }

    public void removeT1KelinerNull() {
        Vector temp = new Vector();
        for (int i = 0; i < this.size(); i++) {
            Impuls impuls = this.getImpuls(i);
            if (impuls.getValue(Impuls.T1) < 0) {
                temp.add(impuls);
            }
        }
        this.removeAll(temp);
        isT1KelinerNull = true;
    }

}
```

4.6.3 Funktionen

```
public class Funktionen extends Vector {

    public static final int INDEX = 0;
    public static final int I1 = 1;
    public static final int I2 = 2;
    public static final int U = 3;
    public static final int T = 4;
    public static final int T1 = 5;
    public static final int T2 = 6;
    public static final int T2_MINUS_T1 = 7;
    public static final int I2_DURCH_I1 = 8;
    public static final int DU = 9;
```

```
    public static final int DT = 10;
    public static final int DU_DURCH_DT = 11;
    public static final int U_EFF = 12;
    public static final int T_MOD_PER = 13;
    public static final int PHASENLAGE = 14;

    private FunktionenFilter filter;
    private double pv = 0; // PhasenVerschiebung
    public static double per = 20.0; // periode
    private double frq = 50; // frquenz
    private int dtc = 1; // den Wert von dalta T Count
    private int duc = 1; // den Wert von dalta U Count

    // Columns Namen
    public static final String[] colNames = new String[] {
            "n", // Spalte 0 = Index
            "I\u2081", // Spalte 1 = I1
            "I\u2082", // Spalte 2 = I2
            "U", // Spalte 3 = U
            "t", // Spalte 4 = T
            "t\u2081", // Spalte 5 = T1
            "t\u2082", // Spalte 6 = T2
            "t\u2082 - t\u2081", // Spalte 7 = T2-T1
            "|I\u2082| / I\u2081", // Spalte 8 = I2/I1
            "\u0394U", // Spalte 9 = dU
            "\u0394t", // Spalte 10 = dT
            "\u0394U / \u0394t", // Spalte 11 = dU / dT
            "Ueff", // Spalte 12 = U_eff
            "t modulo ", // Spalte 13 = T_mod_20
            "Phasenlage" // Spalte 14 = Phasenlage
    };

    // Columns Einheiten
    public static final String[] colEinheit = new String[] {
            " ", // Spalte 0 = Index
            "pC", // Spalte 1 = I1
            "pC", // Spalte 2 = I2
            "kV", // Spalte 3 = U
            "ms", // Spalte 4 = T
            "\u00B5s", // Spalte 5 = T1
            "\u00B5s", // Spalte 6 = T2
            "\u00B5s", // Spalte 7 = T2-T1
            "pC", // Spalte 8 = I2/I1
            "kV", // Spalte 9 = dU
            "ms", // Spalte 10 = dT
            "kV/ms", // Spalte 11 = dU / dT
            "V", // Spalte 12 = U_eff
            "ms", // Spalte 13 = T_mod_20
            "Grad" // Spalte 14 = Phasenlage
    };

    public Funktionen(String fileName, int column, FunktionenFilter
filter) {

        this.fileName = fileName;
        final int SPALTEN = column;
        this.filter = filter;
    }

    public void setModulo(double per, double pv) {
```

```
        this.pv = pv;
        this.per = per;

        for (int i = 0; i < size(); i++) {

            double wert = 0;
            FunktionenRow row = getRow(i);

/*////////////////////////////////////////////////////////////////
            *     t_modulo_20
            */
           wert = (getValue(i, Funktionen.T) + pv) % per;
           row.setValue(Funktionen.T_MOD_PER, wert);
        }

        for (int i = 0; i < size(); i++) {
            double wert = 0;
            FunktionenRow row = getRow(i);


/*////////////////////////////////////////////////////////////////
            *     Phasenlage
            */
           wert = getValue(i, Funktionen.T_MOD_PER) * (360/per);
            row.setValue(Funktionen.PHASENLAGE, wert);
        }
    }

    public void setFRG(double frq){
        this.frq = frq;
        this.per = 1000/frq;
    }

    public void setPeriode(double per){
        this.per = per;
        this.frq = 1000/per;
    }

    public String getColName(int index) {
        String value = "";
        if (index == Funktionen.T_MOD_PER) {
            value = colNames[index] + per;
        }
        else {
            value = colNames[index];
        }
        return value;
    }

    public String getColEinheit(int index) {
        String value = "";
        if (index == 0) {
            value = "";
        }
        else {
            value = colEinheit[index];
        }
        return value;
    }

    public double[] getDaltaTSpalte(int dtc){
```

```
        // dass funktioniert nur für Dalta T und nicht für Dalta U
        int last = size() - 1;
        double[] dtSpalte = new double[size()];
        for (int i = 0; i < size(); i++) {

/*////////////////////////////////////////////////////////////////
             *     jeder reihe wird normal
             */
            if (i < last && i + dtc < last) {
                dtSpalte[i] = (this.getValue(i + dtc, Funktionen.T) -
this.getValue(i, Funktionen.T));
            }
/*////////////////////////////////////////////////////////////////
             *     die letzte reihe wird mit null ausgefühlt.
             */
            if (i == last && i + dtc > last) {
                dtSpalte[i] = 0;
            }
        }
        return dtSpalte;
    }

    public double[] getSpalte(int col){
        double[] spalte = new double[size()];
        for(int i=0; i< size(); i++){
            spalte[i] = getValue(i,  col);
        }
        return spalte;
    }

    public void rebildDALTA(){

        int last = size() - 1;
        for (int i = 0; i < size(); i++) {

/*////////////////////////////////////////////////////////////////
             *     jeder reihe wird normal
             */
            if (i < last) {
                // set dU
                this.setValue(i, Funktionen.DU,
                              (this.getValue(i + 1, Funktionen.U) -
                               this.getValue(i, Funktionen.U)));

                // set dT
                this.setValue(i, Funktionen.DT,
                              (this.getValue(i + dtc, Funktionen.T) -
                               this.getValue(i, Funktionen.T)));

                // set dU/dT
                this.setValue(i, Funktionen.DU_DURCH_DT,
                              (this.getValue(i, Funktionen.DU) /
                               this.getValue(i, Funktionen.DT)));
            }


/*////////////////////////////////////////////////////////////////
             *     die letzte reihe wird mit null ausgefühlt.
             */
            if (i == last) {
```

```
                // return dU
                this.setValue(i, Funktionen.DU, 0);

                // return dT
                this.setValue(i, Funktionen.DT, 0);

                // return dU/dT
                this.setValue(i, Funktionen.DU_DURCH_DT, 0);
            }
        }
    }
}
```

4.6.4 FunktionenFilter

```
public class FunktionenFilter {

    private int id = 0;
    private String filterLabel = "";
    private int col = 1;
    private double left = 0;
    private double right = 0;
    private boolean rebDALTA = true;

    private boolean isLEmpty = true;
    private boolean isREmpty = true;
    private boolean isLREmpty = true;

    public static int FILTER_ORG = 0;
    public static int FILTER_DAL = 1;
    public static int FILTER_COL = 2;
    public int FILTER_TYPE = 0;


    public FunktionenFilter(){
        id = 0;
        filterLabel = "0- Originale Datensätze.";
        FILTER_TYPE = FILTER_ORG;
    }

    public FunktionenFilter(int idd){
        id = idd;
        filterLabel = idd + "- " + "Dalta wird nachgebildet.";
        FILTER_TYPE = FILTER_DAL;
    }

    public FunktionenFilter(int idd, int col, String left, String
right) {
        id = idd + 1;
        FILTER_TYPE = FILTER_COL;

        this.col = col;
        if (!left.equals("")) {
            this.left = Double.parseDouble(left);
            isLEmpty = false;
        }
        if (!right.equals("")) {
            this.right = Double.parseDouble(right);
            isREmpty = false;
        }
        if (!left.equals("") && !right.equals("")) {
```

```
            isLREmpty = false;
        }
        setFilterLabel();
    }

    private void setFilterLabel(){
        String label = "";
        if (id == 0) {
            label = "Origenal Datensätz";
        }
        else {
            if (!isLREmpty) {
                label = "(" + left + " \u2264 " + Funktio-
nen.colNames[col] +" < " + right + ")";
            }
            else {
                if (isLEmpty) {
                    label = "(" + Funktionen.colNames[col] + " < " +
right + ")";

                }
                if (isREmpty) {
                    label = "(" + left + " \u2264 " + Funktio-
nen.colNames[col] + ")";
                }
            }
        }
        this.filterLabel = id + "- " + label;
    }
}
```

4.6.5 FunktionenVerwaltung

```
import java.util.Vector;
import java.util.Collection;


public class FunktionenVerwaltung {

    private ImpulsDaten oid;

    /**
     * @label besitzt
     * @supplierCardinality 2
     */
    private Funktionen akkFun;
    private Vector vFunListe;

    public FunktionenVerwaltung(ImpulsDaten oid)throws Exception{
        if(!oid.isReady ()){
            throw new Exception("Überprüfen Sie, ob Hochfahrgeschwin-
digkeit, Frequenz, Umrechnungsfaktoren\nund Stör-Impulse eingesetzt
sind.");
        }else{
            this.oid = oid;
            this.orgFun = makeFunktionen(oid);
            this.akkFun = orgFun;
            this.vFunListe = new Vector();
        }
    }
```

```
    private Funktionen makeFunktionen(ImpulsDaten id)  {
        int last = id.size() - 1;
        FunktionenFilter ff = new FunktionenFilter();
        Funktionen fun = new Funktionen(id.getFileName(), Funktio-
nen.SPALTEN, ff);
        fun.setFRG(id.getFRG());

        for (int i = 0; i < id.size(); i++) {
            FunktionenRow row = new FunktionenRow();
            double wert = 0;

/*////////////////////////////////////////////////////////////////
             *     @return INDEX
             */
           wert = i + 1;
            row.setValue(Funktionen.INDEX, wert);

/*////////////////////////////////////////////////////////////////
             *     @return I1
             */
           wert = id.getValue(i, id.I1);
            row.setValue(Funktionen.I1, wert);


/*////////////////////////////////////////////////////////////////
             *     @return I2
             */
           wert = id.getValue(i, id.I2);
            row.setValue(Funktionen.I2, wert);


/*////////////////////////////////////////////////////////////////
             *     @return U
             */
           wert = id.getValue(i, id.U);
            row.setValue(Funktionen.U, wert);


/*////////////////////////////////////////////////////////////////
             *     return t
             */
           wert = id.getValue(i, id.T);
            row.setValue(Funktionen.T, wert);


/*////////////////////////////////////////////////////////////////
             *     return t1
             */
           wert = id.getValue(i, id.T1);
            row.setValue(Funktionen.T1, wert);


/*////////////////////////////////////////////////////////////////
             *     return t2
             */
           wert = id.getValue(i, id.T2);
            row.setValue(Funktionen.T2, wert);


/*////////////////////////////////////////////////////////////////
             *     return t2 - t1
```

```
             */
            wert = id.getValue(i, id.T2) - id.getValue(i, id.T1);
             row.setValue(Funktionen.T2_MINUS_T1, wert);


/*////////////////////////////////////////////////////////////////
             *     return |I2| / I1
             */
            wert = Math.abs(id.getValue(i, id.I2)) / id.getValue(i,
id.I1);
             row.setValue(Funktionen.I2_DURCH_I1, wert);


/*////////////////////////////////////////////////////////////////
             *     jeder reihe wird normal
             */
            if (i < last) {
                // set dU
                wert = id.getValue(i + 1, id.U) - id.getValue(i,
id.U);
                row.setValue(Funktionen.DU, wert);

                // set dT
                wert = id.getValue(i + 1, id.T) - id.getValue(i,
id.T);
                row.setValue(Funktionen.DT, wert);

                // set dU/dT
                wert = row.getValue(Funktionen.DU) /
row.getValue(Funktionen.DT);
                row.setValue(Funktionen.DU_DURCH_DT, wert);
            }


/*////////////////////////////////////////////////////////////////
             *     die letzte reihe wird mit null ausgefühlt.
             */
            if (i == last) {
                // return dU
                wert = 0;
                row.setValue(Funktionen.DU, wert);

                // return dT
                wert = 0;
                row.setValue(Funktionen.DT, wert);

                // return dU/dT
                wert = 0;
                row.setValue(Funktionen.DU_DURCH_DT, wert);
            }


/*////////////////////////////////////////////////////////////////
             *     U_eff
             */
            wert = id.getValue(i, id.T) * (id.getHFG() / 60000);
             row.setValue(Funktionen.U_EFF, wert);


/*////////////////////////////////////////////////////////////////
             *     t_modulo_20
```

```
            */
           wert = ((id.getValue(i, id.T) + fun.getPV()) %
fun.getPeriode());
            row.setValue(Funktionen.T_MOD_PER, wert);


/*////////////////////////////////////////////////////////////
            *     Phasenlage
            */
           wert = ((id.getValue(i, id.T) + fun.getPV()) %
fun.getPeriode()) * (360/fun.getPeriode());
            row.setValue(Funktionen.PHASENLAGE, wert);


/*////////////////////////////////////////////////////////////
            */
           fun.addRow(i, row);
        }
        return fun;
    }

    public void setAkkFunktionen(Funktionen fun){
        this.akkFun = fun;
    }

    public Funktionen getAkkFunktionen(){
        return akkFun;
    }

    public Funktionen getOrgFunktionen(){
        return orgFun;
    }

    public void setFilter(FunktionenFilter filter)throws Exception{
        if(getAkkFunktionen() != null){
            vFunListe.addElement(getAkkFunktionen().clone());
            switch(filter.FILTER_TYPE){
                case 2 : {///setFilter
                    Funktionen fun = makeFilter(getAkkFunktionen(),
filter);
                    setAkkFunktionen(fun);
                    break;
                }
                case 1 : {//Daltanachbilden
                    Funktionen fun = getAkkFunktionen();
                    fun.rebildDALTA();
                    fun.setFilter(filter);
                    setAkkFunktionen(fun);
                    break;
                }
            }
            Funktionen fun = makeFilter(getAkkFunktionen(), filter);
            setAkkFunktionen(fun);
            System.out.println(fun.toString() + vFunListe.size());
        }
    }

    private Funktionen makeFilter(Funktionen aFun, FunktionenFilter
filter){
        Funktionen nFun = new Funktionen(aFun.getFileName(), Funktio-
nen.SPALTEN, filter);
```

```
        int col = filter.getCol();

////////////////////////////////////////////////////////////////////
//
        /// A <= m < B
        if(!filter.IsLREmpty()){
            double ls = filter.getLeft();
            double rs = filter.getRight();

            // _________<=###############<_________
            if(ls < rs){
                for(int i = 0; i<aFun.size(); i++){
                        double value = aFun.getValue(i, col);


                        if(ls <= value && value < rs){
                                nFun.addElement(aFun.get(i));

                        }
                }
            }
            // ###############<=_________________<###############
            if(ls > rs){
                    for(int i = 0; i<aFun.size(); i++){
                            double value = aFun.getValue(i, col);

                            if(ls <= value || value < rs){
                                    nFun.addElement(aFun.get(i));

                            }
                    }
            }
        }

////////////////////////////////////////////////////////////////////
//
        /// ############### <= ############## <_________
        if(filter.IsLEmpty()){
                double rs = filter.getRight();
                for(int i = 0; i<aFun.size(); i++){
                    double value = aFun.getValue(i, col);
                    if(value < rs){
                            nFun.addElement(aFun.get(i));

                    }
            }
        }

////////////////////////////////////////////////////////////////////
//
        /// _________________ <= ############## < ##############
        if(filter.IsREmpty()){
            double ls = filter.getLeft();
            for(int i = 0; i<aFun.size(); i++){
                double value = aFun.getValue(i, col);
                if(ls <= value){
                        nFun.addElement(aFun.get(i));

                }
            }
        }
```

```
////////////////////////////////////////////////////////////////////
//
        nFun.setFilter(filter);
        return nFun;
    }

    public void setLastFilter(){
        Funktionen lastFilter = (Funktio-
nen)vFunListe.remove(vFunListe.size()-1);
        setAkkFunktionen(lastFilter);
    }

    public Vector getFunktionenListe(){
        return vFunListe;
    }
}
```

4.6.6 DataManagment

```
public class DataManagment {

    public static ImpulsDaten readImpulsDaten(File file) throws Ex-
ception {

        ImpulsDaten id = new ImpulsDaten();
        ////////////////////////////////////////// Set File Name
        String[] fileToken = file.getName().split(".txt");
        id.setFileName(fileToken[0].toUpperCase());
        ////////////////////////////////////////// Set File Name

        String line;
        FileInputStream fis = new FileInputStream(file);
        InputStreamReader isr = new InputStreamReader(fis);
        BufferedReader br = new BufferedReader(isr);

        line = br.readLine();
        StringTokenizer row = new StringTokenizer(line);

        if (row.countTokens() == 6) {
            id.addImpuls(readImpuls(line));
            while ( (line = br.readLine()) != null) {
                id.addImpuls(readImpuls(line));
            }
            br.close();
            System.out.println("OK_# [" + file.getName() +
                               "] Datei ist gelesen.");
        }
        else {
            throw new Exception("Die ausgewählte Datei ist kein
ASCII-Datei oder besitzt keine sechsspaltige Matrix.");
        }
        return id;
    }

    public static ImpulsDaten loadImpulsDaten(File file) throws Ex-
ception {
        ImpulsDaten id = new ImpulsDaten();

        FileInputStream fis = new FileInputStream(file);
        ObjectInputStream ois = new ObjectInputStream(fis);
```

```
        id = (ImpulsDaten) ois.readObject();
        ois.close();
        System.out.println("OK_# [" + file.getName() +
                           "] Datei ist geladen.");
        return id;
    }

    public static boolean saveImpulsDaten(ImpulsDaten id, File file)
throws Exception {

        file.createNewFile();
        FileOutputStream fos = new FileOutputStream(file);
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(id);
        oos.close();
        System.out.println("OK_# [" + file.getName() +
                           "] Datei ist gespeichert.");

        return true;
    }

    private static Impuls readImpuls(String line) {

        StringTokenizer row = new StringTokenizer(line);
        Impuls impuls = new Impuls();
        impuls.setValue(Impuls.I1,
Double.parseDouble(row.nextToken()));
        impuls.setValue(Impuls.I2,
Double.parseDouble(row.nextToken()));
        impuls.setValue(Impuls.U,
Double.parseDouble(row.nextToken()));
        impuls.setValue(Impuls.T,
Double.parseDouble(row.nextToken()));
        impuls.setValue(Impuls.T1,
Double.parseDouble(row.nextToken()));
        impuls.setValue(Impuls.T2,
Double.parseDouble(row.nextToken()));

        return impuls;
    }
}
```

5 Zusammenfassung

Das Institut für Werkstoff und Diagnostik (WED) an der Universität Siegen setzt das Verfahren PSA, um Teilentladungen zu charakterisieren. Die angeforderte Software zur TE-Charakterisierung und zum Einsatz in PSA-Verfahren wurde in der Programmiersprache JAVA implementiert.

Die entwickelte Software ist geeignet, die 6-spaltige ASCII-Datei zu lesen und zu bearbeiten. Mit Hilfe der entwickelten Software wird die PSA-Verfahren leichter und effizienter durchzuführen.

Literaturverzeichnis

[1] Farhad Berton: Entwicklung und Anwendung eines DSP-gesteuerten TE-Meßgerätes zur Teilentladungsdiagnotik von Isolationssystemen; Shaker Verlag, Deutschland (2003)

[2] Dieter König, Y. Narayana Rao; Teilentladungen in Betriebsmitteln der Energietechnik; VDE Verlag; Deutschland (1993)

[3] Rainer Patsch; Patial Discharges with a spezial Emphasis on the Pulse Sequence Anaylsis (PSA); Publications between 1993-1999 from WED;

[4] Rainer Patsch; Patial Discharges with a spezial Emphasis on the Pulse Sequence Anaylsis (PSA); Publications between 2000-2004 from WED;

[5] Guido Krüger; GoTo Java 2; Addison Wesley Verlag, Deutschland, (2000)

[6] Bernd Oestereich; Objektorientierte Softwareentwicklung; Oldenbourg Verlag, Deutschland (2001)

[7] Berhard Rumpe; Modellierung mit UML; Springer Verlag, Deutschland (2004)

[8] Borrmann, Komnick, Landgrebe, Matrne, Sauer; Rational und UML; Galileo Computing Verlag, Deutschland (2002)