

Institut für Parallele und Verteilte Systeme
Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart
Germany

Fachstudie Softwaretechnik B.Sc.

**Verbesserung der Textqualität in Digitalen Log-
bucheinträgen in der Instandhaltung**

Katja Seber
Jan Fuhrmann
Gamze Uysal

Studiengang:	Softwaretechnik
Prüfer/in:	PD Dr. rer. nat. habil. Holger Schwarz
Betreuer/in:	Yannick Wilhelm, M.Sc., Christian Weber, M.Sc.
Beginnt am:	23. April 2019
Endet am:	23. Oktober 2019

Kurzfassung

Im Rahmen dieser Fachstudie sollten die digitalen Logbucheinträge der Firma Festo, welche unter anderem Fehlerausgangssituationen aber auch die darauffolgenden Handlungen zur Lösung dieser Probleme beschreiben, in Hinblick auf die Textqualität verbessert und eine effiziente Suche nach semantisch gleichen Einträgen ermöglicht werden. Dadurch soll das gegebene Logbuch als Wissensbasis nutzbar gemacht werden, sodass Lösungen auf bereits analysierte Fehler nicht noch einmal gefunden werden müssen und so ein effizienteres Arbeiten erzielt wird.

Um geeignet auf dieses domänenspezifische digitale Logbuch einzugehen wurde zunächst eine umfangreiche Analyse durchgeführt. Das Hauptaugenmerk hierbei lag auf den Fehlerausgangssituationen und Handlungsbeschreibungen, welche in Form von Freitext verfasst wurden. Im Zuge dessen wurde festgestellt, dass die Texte durch eine Vielzahl von Rechtschreibfehlern, Eigenworten, fachspezifischen Abkürzungen und semantisch gleichen Einträgen gekennzeichnet sind, was eine effiziente Schlüsselwortsuche verhindert. Aufgrund dieser Erkenntnisse wurden zunächst die Rechtschreibkorrektur und die Auflösung der fachspezifischen Abkürzungen in Bezug auf die spezielle Domäne vorgenommen. Auf dieser aufgebesserten Wissensbasis wurde im Anschluss die effiziente Suche nach semantisch gleichen Einträgen durch die Query Expansion mit Einbindung von Synonymen durchgeführt. Diesbezüglich wurde SolR für die Umsetzung verwendet.

Die Ergebnismenge der Suche auf den digitalen Logbucheinträgen mit verbesserter Textqualität liefert nun ein exakteres Ergebnis, welches alle relevanten semantisch gleichen Einträge enthält.

Inhaltsverzeichnis

Kurzfassung.....	3
1 Einleitung.....	13
2 Analyse der bestehenden Daten.....	15
2.1 Ergebnisse	15
3 Stand der Wissenschaft und Technik.....	19
3.1 Textqualitätsverbesserung.....	19
3.1.1 Literaturergebnisse.....	19
3.1.2 Methoden.....	19
3.1.3 Existierende Tools	20
3.1.4 Vergleich und Entscheidung.....	21
3.2 Effiziente Suche nach semantisch gleichen Einträgen	22
3.2.1 Literaturergebnisse.....	22
3.2.2 Existierende Tools	25
3.2.3 Vergleich und Entscheidung.....	35
4 Konzept und Umsetzung	38
4.1 Allgemeines Konzept.....	38
4.2 Textqualitätsverbesserung.....	38
4.2.1 Konzept der Textqualitätsverbesserung.....	38
4.2.2 Umsetzung der Textqualitätsverbesserung.....	39
4.2.3 Code-Einblick.....	40
4.3 Suche nach semantisch gleichen Einträgen.....	41
5 Evaluation.....	43
5.1 Ergebnisse	43
5.1.1 Textqualitätsverbesserung	43
5.1.2 Semantische Suche	44
5.2 Bewertung	50
6 Zusammenfassung und Ausblick.....	52
7 Literaturverzeichnis.....	53
Erklärung.....	57

Abbildungsverzeichnis

Abbildung 1. Beispiel für semantisch gleiche Einträge [1]	14
Abbildung 2. Prozess zum Generieren der Query Expansion [11]	23
Abbildung 3. Visualisierung Synonymzuweisung zur Abfragezeit.....	32
Abbildung 4. Gesamtarchitektur	38
Abbildung 5. Konzept der Textqualitätsverbesserung.....	39
Abbildung 6. Code-Einblick Umsetzung “pyspellchecker”	40
Abbildung 7. Code-Einblick Auflösung der fachspezifischen Abkürzungen.....	41
Abbildung 8. Architektur von SolR [42].....	41
Abbildung 9. Rechtschreibfehler und Abkürzungen der Samples Datei.....	44
Abbildung 10. FN, TN, TP, FP der Funde aus der Samples Datei.....	44

Tabellenverzeichnis

Tabelle 1. Beispiel Rechtschreibfehler DigiLog Einträge	15
Tabelle 2. Beispiele der fachspezifischen Abkürzungen aus den DigiLog Einträgen.....	16
Tabelle 3. Beispiel identische Ausgangssituation fast identische Handlungen.....	17
Tabelle 4. Beispiel semantisch gleich Einträge mit unterschiedlichen Beschreibungen	18
Tabelle 5. Beispiel Levenshtein-Distanz "kallibiret"	20
Tabelle 6. Beispieleinträge aus dem DigiLog für die Erstellung des invertierten Index Tabelle...	29
Tabelle 7. Invertierter Index der Beispieleinträge aus Tabelle 5.....	30
Tabelle 8. Akzeptierte Syntax der Synonymlisten.....	31
Tabelle 9. Ausschnitt Synonymdatei der digitalen Logbucheinträge	31
Tabelle 10. Beschreibung Scoring Bewertungsfaktoren.....	32
Tabelle 11. Boots zur Abfragezeit	34
Tabelle 12. Vergleich der Suchmaschinen [41]	35
Tabelle 13. Beispiel 1: Soll-Werte suche nach semantisch gleichen Einträgen.....	45
Tabelle 14. Beispiel 2: Soll-Werte suche nach semantisch gleichen Einträgen.....	47

Formelverzeichnis

Formel 1. praktische Bewertungsfunktion Apache Lucene.....	32
Formel 2. Formel zur Berechnung des Bewertungsparameters Precision.....	43
Formel 3. Formel zur Berechnung des Bewertungsparameters Recall.....	43
Formel 4. Formel zur Berechnung des Bewertungsparameters Accuracy	43

1 Einleitung

Die Digitale Revolution, in welcher wir uns heute befinden, hat nicht nur Auswirkungen auf das Privatleben, sondern auch auf die Wirtschafts- und Arbeitswelt. Der digitale Umschwung in der industriellen Produktion wird auch als 4. Industrielle Revolution bezeichnet. Alle Arbeitsschritte und Prozesse werden digitalisiert, um effizienteres Arbeiten zu ermöglichen. Im Zuge der Digitalisierung werden somit in einigen produzierenden Unternehmen die Wartungs- und Instandhaltungsmaßnahmen der Fertigung und Montage, statt wie in früheren Tagen händisch mit Stift und Papier, digital in Logbüchern erfasst. Ein typischer Logbucheintrag umfasst die Beschreibung der Fehlerausgangssituation (Fehler, Fehlersymptomatik), eine Beschreibung der Fehlerursache, eine Beschreibung, wie der Fehler gelöst beziehungsweise dabei vorgegangen wurde (Handlung), sowie einen Zeitstempel und eine Maschinen- beziehungsweise Arbeitsplatzzuordnung. Werden die digitalen Logbucheinträge gebündelt in einer Datenbank gespeichert, und der Zugriff auf diese ermöglicht, so stellen diese eine Wissensbasis für die Wartung und Instandhaltung dar. Werkern wird es ermöglicht ihre Arbeitserfahrungen zu teilen und auszutauschen, was im Gegenzug eine Zeitersparnis im Zuge der Fehlerursachenfindung und Lösungsfindung einbringt, da Fehler, welche bereits analysiert und gelöst wurden nicht aufs Neue ins Detail analysiert werden müssen. [1]

Die digitalen Logbucheinträge liegen, in unstrukturierter Form der natürlichen Sprache vor. Dies stellt ein großes Problem für die Wissensrückgewinnung dar. Die Fehlerausgangssituationen und die darauf folgenden Handlungsbeschreibungen zur Lösung dieser, werden in Form von Freitexten von den Werkern verfasst. Auf Grund dessen leidet die Textqualität enorm und die Freitexte sind durch eine Vielzahl von Rechtschreibfehlern, Abkürzungen, Synonyme etc. gekennzeichnet. Viele der Datenqualität mindernden Ursachen entstehen durch mangelnde Zeit, die Einträge ordentlich zu verfassen, oder durch mangelnde Konzentration bei der Texteingabe mittels der Tastatur. Die Freitexte werden von den unterschiedlichen Werkern in verschiedenen Schreibstilen verfasst, was bedeutet, dass Einträge beispielsweise in Fließtexten, stichwortartigen Fließtexten und in Form von Stichworten geschrieben werden. Doch nicht nur die Verwendung verschiedener Schreibstile führt zu uneinheitlichen Einträgen, sondern auch die wechselnde Verwendung von Umgangs- und Fachsprache. Des Weiteren werden Fehlerbilder und Ursachenbehandlungen welche technisch gesehen äquivalent sind unterschiedlich beschrieben, da diese von unterschiedlichen Werkern festgehalten und ausformuliert werden. So haben semantisch gleiche Einträge synonyme Beschreibungen, welche aufgrund der Uneinheitlichkeit der Einträge und fehlenden Form- und Stilvorschriften entstehen. Aufgrund dieser Probleme wird eine traditionelle Schlüsselwortsuche, wie sie bei vielen Suchmaschinen verwendet wird ineffizient. Denn wird bei einem derartigen Datensatz diese Art von Suche angewandt, so können aufgrund der vielen Synonyme und Abkürzungen nicht alle relevanten Einträge des Logbuches für eine Suchanfrage ausgegeben werden. Die Ergebnismenge ist nicht vollständig, weshalb wichtige Informationen untergehen könnten. Zwei Beispiele für die semantisch gleichen Einträge werden in Abbildung 1. gezeigt. Es sind vier Beispieleinträge des digitalen Logbuchs aus der Instandhaltung zu sehen. Die ersten zwei Einträge (lila), sowie die unteren bedienten Einträge (grün) beschreiben jeweils dieselbe Ausgangssituation mit semantisch ähnlichen Handlungsmaßnahmen. Eine schnelle zielführende Abfrage der Wissensbasis wird durch solche Fälle kombiniert mit der schlechten Textqualität erschwert. Dies verhindert es den Werkern schneller passende Fehlerursachen und Lösungen zu Fehlern zu finden, welche bereits untersucht und erfolgreich abgeschlossen wurden. [1]

Abbildung 1. Beispiel für semantisch gleiche Einträge [1]

Ausgangssituation	Handlung
Klimagerät Störung	War wahrscheinlich die Schaltschranktür nicht richtig zu -> Meldung "Klimagerät Störung"
Klimagerät Störung PMC	Schaltschranktür nicht richtig geschlossen, falsche Fehlermeldung! Lichtschanke meldet falsch - justiert.
Türschalter von Werkstückplatz verriegelt nicht.	Schaltzunge geht nicht weit genug in den Schalter. Schaltzunge neu eingestellt.
Türverriegelung schließt nicht richtig.	Betätigungszunge an Tür war verstellt. Zunge wieder ausgerichtet.

Ziel dieser Fachstudie ist es, zum einen die Textqualität der Freitexte der digitalen Logbucheinträge der Montage der Festo AG Co. KG zu erhöhen. Zum anderen das Ermöglichen der Nutzung des digitalen Logbuches als Wissensbasis für die Werker, indem die zielführendsten Handlungsempfehlungen zu einem vorliegenden Fehler (zum Beispiel: „Türverriegelung schließt nicht“) durch die Suche nach semantisch gleichen Einträgen zurückgegeben wird. Das bedeutet, dass Monteure mit der Wissensbasis des digitalen Logbuches nach dem entdeckten Fehler suchen können und alle bereits dazu verfassten Einträge, die ihm bei Fehlerlösung helfen könnten, ausgegeben werden. Mehrfach vorkommende Fehler müssen somit nicht bei jedem auftreten erneut untersucht und eine passende Fehlerbehandlung gesucht werden. Das erspart den Monteuren Zeit und effizienteres arbeiten wird möglich.

Um das Ziel dieser Fachstudie zu erreichen, muss zunächst eine Analyse der realen digitalen Logbucheinträge der Festo AG Co. KG durchgeführt werden, welche die verschiedenen existenten Fehlerarten und Auffälligkeiten der Freitexte aufzeigt. Die Ergebnisse der Analyse werden nun als Basis für das weitere Verfahren verwendet. Um in Zukunft die Suche nach semantisch gleichen Einträgen der digitalen Logbucheinträge zur Generierung zielführenden Handlungsempfehlungen zu ermöglichen, muss zunächst die Textqualität der Freitextfelder erhöht werden. Dies beinhaltet unter anderem die Korrektur von Rechtsschreibfehlern und die Auflösung von fachspezifischen Abkürzungen. Hierfür wird die grundlegende Literatur, die Methoden und die bereits existierenden Software-Tools in Bezug auf die Textqualitätsverbesserung und der Suche nach semantisch gleichen Einträgen analysiert. Im Anschluss werden die Ergebnisse der Recherche und der Analyse der digitalen Logbucheinträge dafür verwendet ein Konzept für die Lösung des vorliegenden Problems aufzustellen. Es folgt die Beschreibung der Methodik bzw. der Implementation des erstellten Konzeptes, sowie die Evaluation und Bewertung der entstandenen Ergebnisse.

2 Analyse der bestehenden Daten

Um die Textqualität der Freitextfelder in den digitalen Logbüchern zu erhöhen und die nächsten Schritte für die Verbesserung dieser definieren zu können, müssen vorerst die bestehenden DigiLog Einträge, welche für diese Fachstudie von der Festo AG Co. KG zur Verfügung gestellt wurden, auf ihre Fehlerarten analysiert werden. Erst wenn die häufigsten Fehlerarten bekannt sind, kann entschieden werden, wie weiter Verfahren wird und welche Methoden und Techniken auf den Einträgen anzuwenden sind.

Das Hauptaugenmerk bei der Analyse wurde auf die Ausgangssituationen und die entsprechenden Handlungen gelegt. Im Gegensatz zu den Feldern des Zeitstempels, in denen das Datum und die genaue Uhrzeit des Eintrages hinterlegt ist (Beispiel Zeitstempel aus den DigiLog Einträgen: „2019-03-21T13:19:47Z“) und der Arbeitsplatzbeschreibungen, welche jeweils aus einer alphanumerischen Kombination, gefolgt von einem kurzen Text bestehend aus 1-3 Wörtern (Beispiel Arbeitsplatzbeschreibung aus den DigiLog Einträgen: „XYXH 1000 alle Zellen“), bestehen die Einträge der Felder „Ausgangssituation“ und „Handlung“ lediglich aus Freitext, welcher von den zuständigen Monteuren eingetragen wurde. Die Einträge wurden händisch in Bezug auf Synonyme und die einzelnen Einträge jeweils auf Rechtschreibfehler und weitere Ausfälligkeiten analysiert.

2.1 Ergebnisse

Die Freitexte der Ausgangssituationen und Handlungen sind meist in Form von Fließtexten beziehungsweise stichwortartigen Fließtexten verfasst. Bei dem Schreiben von Fließtexten treten schnell Fehler auf, was auch bei diesen Einträgen der Fall war. Buchstabenvertauschungen, Groß- und Kleinschreibung, fehlende oder falsch gesetzte Buchstaben und die Verwendung der alten Grammatik waren die am häufigsten auftretenden Rechtschreibfehler. In Tabelle 1. Beispiel Rechtschreibfehler DigiLog Einträge sind Beispieleinträgen der DigiLog Daten zu finden, an denen die aufgezählten Fehler gezeigt werden.

Tabelle 1. Beispiel Rechtschreibfehler DigiLog Einträge

Eintrag	Fehler - korrektes Wort	Fehlerart
Keine aktuelle Datensicherung der Deprag Schraubprogramme Zuordnung der Schraubprogramm passt nicht	- „Zuordnung“ - „Zuordnung“	Buchstabenvertauschung
Achse meldet Robotikfehler 1. CMXR reseten, zelle neu starten und Anlage neu starten hat keinn erfolg gebracht. An Portal Z-Achse Geberkabel getauscht. Ohne Erfolg.. An Portal Z-Achse Motor getauscht. Und Referiert.	- „reseten“ - „resetten“ - „zelle“ – „Zelle“ - „keinn“ – „keinen“ - „erfolg“ – „Erfolg“	- Fehlender Buchstabe - Groß- und Kleinschreibung - Fehlender Buchstabe - Groß- und Kleinschreibung

Grundstellungsfahrt durchgeführt, Testlauf i. O.		
<p>Externes Meßsystem von X-Achse und Y-Achse getauscht.</p> <p>In der S7 im FB152 NW14 den DB156.DBX0.6 wieder gesetzt. Fehler, allg. Robotikfehler, X-Encoder, Y-Encoder, weg.</p> <p>Meßsystem TeileNr. 1428315</p> <p>Testlauf i.O.</p> <p>Beide Meßsysteme defekt, da bei Fehlersuche falsche Geberleitung benutzt wurde. Bei Verwendung der Geberleitung 1599107 ist zwingend ein Adapterkabel zu verwenden. Adapterkabel liegt bei.</p>	- „Meßsystem“ – „Messsystem“	- Alte Rechtschreibung
<p>Funktion und richtiger Ablauf ist gegeben. Saugdüse ist schräg montiert, dadurch kann die Saugdüse das Ventil nicht richtig ansaugen, sobald der Greifer öffnet wackelt das Ventil und fällt in die Kiste. Saugdüse ausgetauscht--> Ablauf stabiler.</p> <p>Prozess muß optimiert werden</p>	- „muß“ – „muss“	- Alte Rechtschreibung

Da die Logbucheinträge im fachspezifischen Bereich geschrieben wurden, sind auch viele fachspezifische Wörter zu finden, welche auch oft abgekürzt aufzufinden sind. Da es sich bei den Einträgen um Probleme bei bestimmten Teilen, wie Sensoren, Zylinder usw. handelt, wird aus Zeitgründen und der Genauigkeit halber oft nur die Teilenummer oder alphanumerische Abkürzungen zur Beschreibung dieser verwendet. Gleiches gilt ebenso für Prozesse und Arbeitsschritte. Diese Abkürzungen wurden bei der Analyse gesammelt und anschließend teilweise von den Fachleuten übersetzt. In Tabelle 1 Tabelle 2 sind einige Abkürzungen mit Übersetzungen aus den DigiLog Einträgen zu finden.

Tabelle 2. Beispiele der fachspezifischen Abkürzungen aus den DigiLog Einträgen

Abkürzung	Übersetzung
KSS	Kühlschmierstoff
GS-Fahrt	Grundstellungsfahrt
FCT	Festo Configuration Tool

CMXR	Achsensteuerung von Festo
WGL	Walter Logik Grenzwertmodul
FB235	Funktionsbaustein in einem S7 Programm
E153.0	Eingangsadresse eines Sensors
FB336, FB340 NW5+/-, FB350 NW45-	Funktionsbaustein 336 Netzwerk 5+ und - vertauscht
MM25; Z10; MM11; ...	Zylinder
Schritt 16	„Datensatz killen“

Treten Fehler auf, dann oftmals mit einer gewissen Häufigkeit. Um die gleichen Fehler beziehungsweise Ausgangssituation innerhalb der Einträge zu finden, und zu schauen, ob immer mit derselben Handlung vorgegangen wurde, wurden die Einträge auf Synonyme untersucht. Die Analyse ergab, dass sowohl Einträge mit exakt denselben Beschreibungen existieren (Tabelle 3), als auch semantisch gleiche Einträge mit einer unterschiedlichen Beschreibung.

Tabelle 3. Beispiel identische Ausgangssituation fast identische Handlungen

Ausgangssituation	Handlung
X-Achse steht schräg.	Achse an Ref.-Punkt gerade gestellt.
X-Achse steht schräg.	X-Achse gerade gestellt. Referenzpunkte neu eingestellt (H.Smirnov/H.Tönis).
X-Achse steht schräg.	X-Achse gerade gestellt. Referenzpunkte neu eingestellt (H.Smirnov/H.Tönis).

Im Falle der Einträge aus Tabelle 3, sind drei identische Ausgangssituationen zu finden. Die Erste Handlung beschreibt das gleiche Vorgehen, wie die beiden identischen Beschreibungen in den beiden darunterliegenden Zellen, jedoch auf andere Weise ausformuliert. Bei der ersten Beschreibung der Handlung wurden die Worte „X-Achse“ und „Referenzpunkt“ durch die Abkürzungen „Achse“ und „Ref.-Punkt“ synonym gesetzt, sowie die ausführliche Beschreibung darüber, dass X-Achse und Referenzpunkt geradegestellt und neu eingestellt wurden durch eine kürzere Beschreibung zusammengefasst. In allen Einträgen gleich ist die Beschreibung, dass die Achsen „gerade gestellt“ wurden.

Tabelle 4. Beispiel semantisch gleich Einträge mit unterschiedlichen Beschreibungen

Ausgangssituation	Handlung
Druckregler bläst ab.	Neuen Druckregler DR450 gewechselt und kalibriert.
St. 50 AD-Regler 550 bläst ab.	Regler getauscht und neu kalibriert. -> Testlauf i.O.
Druckregler B450 bläst ab.	Neuen Regler eingebaut und eingestellt.

Die Beschreibungen in der Ausgangssituation in Tabelle 4 beschreiben jeweils dasselbe Problem, auch wenn diese eventuell an verschiedenen Reglern aufgetreten ist. Dies semantische Gleichheit ist dadurch zu erkennen, dass in allen Beschreibungen die Wörter „Druckregler“ oder „Regler“ und „bläst ab“ enthalten sind. „Regler“ und „Druckregler“ kann hier als Synonym angenommen werden. Bei den semantisch gleichen Handlungsbeschreibungen werden folgende synonyme Beschreibung für das Austauschen des Druckreglers verwendet:

„Neuen Druckregler... gewechselt...“ – „Regler getauscht...“ – „Neuen Regler eingebaut...“

Das erneute kalibrieren des gewechselten Druckreglers wird ebenfalls bei allen Einträgen vorgenommen, jedoch werden die Worte:

„kalibriert“ – „eingestellt“

als Synonyme verwendet. Auf diese Weise existieren innerhalb der DigiLog Einträge viele weitere semantisch gleiche Einträge mit unterschiedlichen Beschreibungen.

Zusammenfassend wurden bei der Analyse der Daten festgestellt, dass die Einträge in Form von Fließtexten bzw. stichwortartigen Fließtexten festgehalten wurden. Die Einträge Rechtschreibfehler enthalten, bei denen hauptsächlich die alte Rechtschreibung angewendet, die Groß- und Kleinschreibung missachtet sowie Buchstaben vertauscht und vergessen wurden. Zudem wurde analysiert, dass häufig fachspezifische Wörter und Abkürzungen aber auch numerische und alphanumerische Abkürzungen für beispielsweise Teilenummern, Prozesse und Arbeitsschritte Anwendung gefunden haben. Zuletzt wurde gezeigt, dass viele semantisch gleiche Beschreibungen der Probleme bzw. Ausgangssituationen und ihre Handlungen öfters im Datensatz vorkommen, jedoch auf verschiedene Arten und Weisen mit Synonymen beschrieben sind.

3 Stand der Wissenschaft und Technik

3.1 Textqualitätsverbesserung

3.1.1 Literaturergebnisse

Wie bereits erwähnt, fielen im Zuge der ausführlichen Analyse der Daten, eine Vielzahl an fachspezifischen Worten, welche auch in Abgekürzter Form auftraten, auf. Diese fachspezifischen Abkürzungen wurden von den Werkern sowohl in Form von Buchstabenkombinationen als auch in Form von alphanumerischen Kombinationen beschrieben. Aus der Analyse in Bezug auf die Rechtschreibung der Logbucheinträge, ergaben sich drei Kategorien: Verwendung der alten Rechtschreibung, Buchstabenvertauschungen und die falsche Verwendung der Groß- und Kleinschreibung.

Um das digitale Logbuch als Wissensbasis für eine effiziente und korrekte Suche nutzen zu können, muss somit die Textqualität der Daten verbessert werden. Dazu werden zunächst die Daten von Rechtschreibfehlern behoben sowie fachspezifische Abkürzungen aufgelöst. Dabei können verschiedene Methoden und Techniken angewandt werden, um eine Bereinigung der Daten zu ermöglichen. Diese Methoden werden in den folgenden Kapiteln beschrieben, verglichen und die ausgewählte Methode umgesetzt.

3.1.2 Methoden

Beim Schreiben auftretende Rechtschreibfehler und Grammatikfehler können in folgende Klassen unterteilt werden:

Orthografische Fehler

Unter Orthografie versteht man eine abweichende Schreibung von der allgemein üblichen Schreibweise der Wörter einer Sprache in der verwendeten Schrift. Darunter gehören beispielsweise Tippfehler. Hier werden die Buchstabenfolgen (strings) in Betracht gezogen, die nicht in dem Wörterbuch der aktuellen Sprache gefunden werden können. Als Lösung bietet sich die Methode der Levenshtein-Distanz an. Dabei werden die Buchstabenfolgen mit den Wörterbucheinträgen verglichen und das zur fehlerhaften Buchstabenfolgen ähnlichste Wort als Korrektur vorgeschlagen. [2]

3.1.2.1 Levenshtein-Distanz

Bei dieser Methode wird die Ähnlichkeit zwischen zwei Wörtern gemessen. Das Ergebnis nennt sich Distanz, die sich aus der Anzahl der Ersetzungen, Einfügungen oder Löschungen, die erforderlich sind, um das falsche Wort in das richtige umzuwandeln, zusammensetzt. Je größer die Levenshtein-Distanz ist, desto unterschiedlicher sind die Wörter.

Die Levenshtein-Distanzalgorithmus wird bei der Rechtschreibprüfung, in der Spracherkennung, bei der DNA-Analyse und bei der Plagiat Erkennung verwendet. [3]

In der folgenden Tabelle 5, wird ein Beispielwort aus den Logbucheinträgen, welches falsch geschrieben ist, mit der Levenshtein-Distanz berechnet.

Tabelle 5. Beispiel Levenshtein-Distanz "kallibiret"

		k	a	l	l	i	b	i	r	e	t
	0	1	2	3	4	5	6	7	8	9	10
k	1	0	1	2	3	4	5	6	7	8	9
a	2	1	0	1	2	3	4	5	6	7	8
l	3	2	1	0	1	2	3	4	5	6	7
i	4	3	2	1	2	1	2	3	4	5	6
b	5	4	3	2	3	2	1	2	3	4	5
r	6	5	4	3	4	3	2	3	2	3	4
i	7	6	5	4	5	4	3	2	3	4	5
e	8	7	6	5	6	5	4	3	4	3	4
r	9	8	7	6	7	6	5	4	3	4	5
t	10	9	8	7	8	7	6	5	4	5	4

Die Levenshtein-Distanz befindet sich in der untersten rechten Ecke der Matrix, in diesem Fall beträgt sie vier. Dies entspricht unserer intuitiven Erkenntnis, dass "kallibiret" durch zwei Löschungen und zwei Einfügungen in "kalibriert" umgewandelt werden kann.

Grammatikalische Fehler

Wenn Wörter, die in einem Wörterbuch vorkommen, aber in der Reihenfolge oder in der Form, wie sie geschrieben sind, in einem Satz zu einem Fehler führen, spricht man von grammatikalischen Fehlern. Hier muss der Satz auf grammatikalische Korrektheit geprüft werden, da eine reine, wortbezogene Rechtschreibprüfung den Fehler nicht finden würde. Die Implementierung einer Grammatikprüfung nutzt die Verarbeitung natürlicher Sprache (NLP, Natural Language Processing). [2]

Komposita

In vielen Sprachen werden Wörter häufig zu neuen Wortkombinationen verkettet. Insbesondere in der deutschen Sprache kommen zusammengesetzte Wörter häufig vor. Um Rechtschreib- sowie Grammatikfehler auch in zusammengesetzten Wörtern zu finden ist ein Wortaufteilungsalgorithmus notwendig. Nach der Wortaufteilung muss der Wortstamm jedes Wortes gefunden werden. Diese Methode nennt sich Stemming.

3.1.2.2 Stemming

Bei dieser Methode werden also die Worte auf ihre Wortstämme zurückgeführt. Auch in der Suche wird diese Methode angewandt, da sie dadurch den benötigten Speicherplatz verringert und die Suche insgesamt schneller verläuft. Jedoch können bei dieser Methode auch semantisch unsinnige Zusammensetzungen nicht als Fehler erkannt werden, wenn beide Wortstämme korrekt geschrieben sind (beispielweise „Hautnachrichten“). [4]

3.1.3 Existierende Tools

Folgende Tools wurden den eben vorgestellten, vorhandenen Methoden und Klassen zugeordnet.

3.1.3.1 Orthografische Fehler

Für die Methode mit der Levenshtein-Distanz gibt es zwei bekannte Python-Bibliotheken.

- **Abydos:** Diese Bibliothek bietet phonetische Algorithmen, String-Distanz-Maß-Algorithmen und -Metriken, Stemmer und String-Fingerabdrücke an. Unter den Zeichenfolgenabstandsmetriken ist auch die Levenshtein-Distanz zu finden. Die Algorithmen unterstützen auch die deutsche Sprache. [5]
- **pyspellchecker:** Diese Bibliothek unterstützt ebenfalls mehrere Sprachen, darunter auch die deutsche Sprache. Die Wörterbücher wurden mit dem WordFrequency-Projekt auf GitHub erstellt. Der Vorteil hier ist, dass man die eingebundene Wortliste nach seinen eigenen Wünschen anpassen kann. [6]

3.1.3.2 Grammatikalische Fehler

Für grammatikalische Fehler gibt es eine Vielzahl von Tools und Bibliotheken. Die drei bekanntesten Bibliotheken, auch in der Programmiersprache Python, werden nun vorgestellt.

- **pyLanguageTool:** LanguageTool ist eine Open Source-Plattform für die Rechtschreibprüfung. Es unterstützt eine Vielzahl von Sprachen und bietet erweiterte Grammatikunterstützung. [7]
- **grammar-check:** Das ist ein Python-Wrapper für das obige LanguageTool. [8]
- **language-check:** Das ist ebenfalls ein Python-Wrapper für das obige LanguageTool. [9]

3.1.3.3 Komposita

Für das Problem mit den zusammengesetzten Wörtern gibt es ebenfalls zwei Python Bibliotheken, die das lösen.

- **Abydos:** Da diese Bibliothek sehr viele Algorithmen enthält, wird sie hier ebenfalls als Tool für das Problem mit zusammengesetzten Wörtern betrachtet. Die deutsche Sprache wird hier auch unterstützt, beispielsweise der „CLEF German“, „Snowball stemmers for German“ und „Caumann’s German Stemmer“. [5]
- **cwsplit:** Ist eine Bibliothek die zusammengesetzten Worte stemmt, unterstützt von „enchant“, ein API mit Wörterbüchern vielzähliger Sprachen. [10]

3.1.4 Vergleich und Entscheidung

Im Anschluss an die umfangreiche Recherche nach Methoden und existierenden Tools zur Textqualitätsverbesserung der digitalen Logbucheinträge wurde folgende Entscheidung für Umsetzung dieser getroffen.

Da im Rahmen der Analyse lediglich eine sehr geringe Anzahl an grammatischen Fehlern gefunden wurden, wurde sich dazu entschieden den Hauptfokus bei der Textqualitätsverbesserung auf die anderen Fehlerarten zu legen.

Hierfür fiel die Entscheidung gegen eine Stemmer, aus dem Grund, dass eine Vielzahl an fachspezifischen Wörtern in den Einträgen des Logbuches existieren. Die fachspezifischen Worte sind in den gängigen Wörterbüchern, welche von Stemmern verwendet werden, nicht existent, weshalb für diese kein Wortstamm für die Korrektur von zusammengesetzten Worten gefunden werden

kann. Bei dem Einsatz eines Stemmers für die Verbesserung der Rechtschreibung werden auf diesem Datensatz somit viele Fehler außer Acht gelassen und die Textqualität kann hiermit nicht gesteigert werden.

Aus diesem Grund fiel die Entscheidung auf die Methode der Editierdistanz wobei mit dem „pyspellchecker“ gearbeitet wurde. Der entscheidende Vorteil des „pyspellcheckers“ liegt für den Fall der Fachstudie mit der speziellen Domäne darin, dass ein benutzereigenes Wörterbuch statt einem gängigen deutschen Wörterbuch miteingebunden werden kann. Dadurch kann eine verbesserte Wortliste, welche alle Worte des digitalen Logbuchen beinhaltet eingebunden werden, wodurch unter anderem auch auf die Eigenworte eingegangen werden kann. Mit Hilfe dieser Wortliste und der Levenshtein-Distanz verbessert der „pyspellchecker“ Rechtschreibfehler, geht auf die spezifische Domäne ein und erhöht die Textqualität.

Für die Ersetzung der fachspezifischen Abkürzungen fiel die Entscheidung auf eine eigens implementierte Methode in Python. Auf Grund der fachspezifischen Abkürzungen ist die Möglichkeit ein gängiges Dictionary von Abkürzungen zu verwenden ausgeschlossen. Es wurde somit eine Liste an Abkürzungen aufgestellt, welche durch „Experten“ von Festo in die ausgeschriebene Form übersetzt wurde. Mit Hilfe von regulären Ausdrücken wird nun innerhalb der Methode nach diesen Abkürzungen gesucht und durch eine Ausgeschriebene Form dieser ersetzt.

Mit Hilfe dieser Auswahl an Tools und Methoden kann die Textqualität erhöht und das digitale Logbuch als Wissensbasis für die Suche nach semantisch gleichen Einträgen genutzt werden.

3.2 Effiziente Suche nach semantisch gleichen Einträgen

3.2.1 Literaturergebnisse

Um das digitale Logbuch als Wissensbasis für die Fehlerursachenfindung und für die Lösungsfindung von Fehlern, welche bereits analysiert und behoben wurden, nutzen zu können ist eine passende Suchfunktion, welche alle relevanten Einträge an den Benutzer zurückliefert von Bedeutung. Aufgrund dessen, dass einige Fehler- und Lösungsbeschreibungen mehrfach, aber anders formuliert im digitalen Logbuch auftreten, sollte eine Suchmethode verwendet werden, welche auf die Spezifische Domäne eingehen kann. In Zuge dessen wurden die Schlüsselwortsuche, die Query Expansion und die semantische Suche unter die Lupe genommen.

3.2.1.1 Schlüsselwortsuche

Bei dieser Suche handelt es sich um eine „einfache Suche“. Hier werden lediglich nach einzelnen Suchbegriffen in den einzelnen Dokumenten gesucht. Es wird jedes einzelne Token in einem Dokument mit dem Suchwort verglichen. Dabei wird nicht auf die semantische Bedeutung eingegangen. Eine solche Suchmethode ist bei der Datengröße unserer Logbuch-Einträge sehr ineffizient, da sie viel Zeit in Anspruch nimmt und keine semantisch ähnlichen Einträge findet. [10]

3.2.1.2 Query Expansion

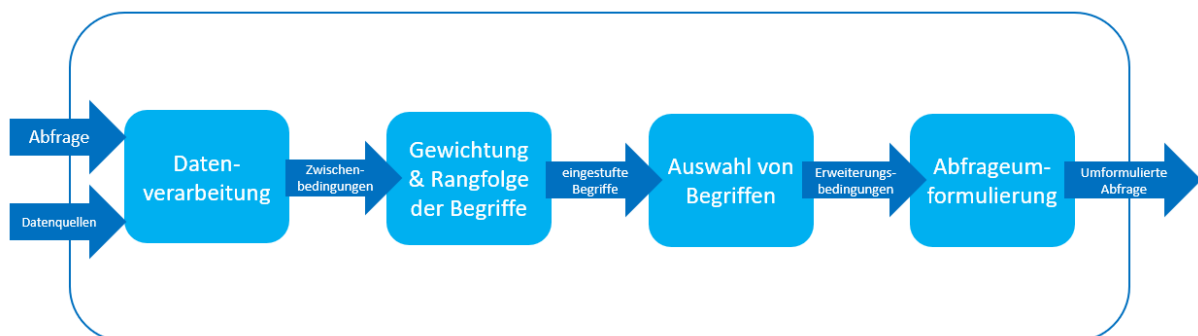
Mit der Größe des Internets und der unzähligen Daten im Web, ist eine relevante Informationsgewinnung mit einer Suchanfrage, bestehend aus wenigen Schlüsselwörtern, fast unmöglich. Hierbei spielt die Query Expansion eine entscheidende Rolle, um die Suche im Internet zu verbessern. Bei der Query Expansion werden zur umformulierten, ursprünglichen Abfrage des Benutzers, zusätzlich aussagekräftige Begriffe mit semantischer Bedeutung hinzugefügt. Es handelt sich also um eine Erweiterung einer bestimmten Suchanfrage/-abfrage, um die An-/Abfrageleistung bei Informationsabrufvorgängen zu verbessern, indem man die Nichtübereinstimmung von Abfragedokumenten minimiert. [11]

Es werden eine Reihe von Methoden und Techniken verwendet, um diese erweiterte Suche zu ermöglichen. Zum einen kann man nach Synonymen suchen und Synonyme von Wörtern finden, indem man die Synonyme auf Wörter der Suchanfrage abbildet. Dazu gehören auch semantisch verwandte Wörter, wie beispielsweise Antonyme, Meronyme, Hyponyme und Hyperonyme. Eine weitere Methode ist das Stemmen, also das Finden von bestimmten Wörtern durch den Stamm jedes Wortes in der Suchanfrage. Rechtschreibfehler werden direkt bei der Suchanfrage behoben. Außerdem werden die Begriffe in der ursprünglichen Abfrage neu gewichtet, indem ein Score genutzt wird.

Diese QE Techniken können in 3 Klassen unterteilt werden: [12]

- Manuelle QE: Hier formuliert der Benutzer die Abfrage manuell neu.
- Automatische QE: Hier formuliert das System die Abfrage automatisch neu, ohne dass der Benutzer eingreifen muss.
- Interaktive QE: Hier erfolgt die Neuformulierung von Abfragen als Ergebnis einer gemeinsamen Zusammenarbeit zwischen dem System und dem Benutzer. Hierbei gibt das System Suchergebnisse auf einer automatisch umformulierten Abfrage zurück und die Benutzer wählen aussagekräftige Ergebnisse aus. Basierend auf den Vorlieben des Benutzers formuliert das System die Abfrage weiter um und ruft die Ergebnisse ab. Der Vorgang wird fortgesetzt, bis der Benutzer mit den Suchergebnissen zufrieden ist.

Abbildung 2. Prozess zum Generieren der Query Expansion [11]



Der Prozess zum Generieren der Abfrageerweiterung besteht hauptsächlich aus vier Schritten: Vorverarbeitung von Datenquellen, Gewichtung und Rangfolge von Begriffen, Auswahl von Begriffen und Neuformulierung von Abfragen (siehe Abbildung 2). [11]

Der Hauptfokus bei einer Query Expansion liegt in der erweiterten Abfrage, die für den Benutzer generiert wird. Die häufigste Form der Abfrageerweiterung ist die globale Analyse unter Verwendung eines Thesaurus. Für jeden Begriff in einer Abfrage kann die Abfrage automatisch um Synonyme und verwandte Wörter aus dem Thesaurus erweitert werden.

Die Verwendung eines Thesaurus kann mit einer Begriffsgewichtung kombiniert werden: Beispielsweise kann man hinzugefügte Begriffe weniger als die ursprünglichen Abfragebegriffe gewichten. Bei der Thesaurus-basierten Abfrageerweiterung sind keine Benutzereingaben erforderlich, was klar von Vorteil ist. [13]

3.2.1.3 Semantische Suche

Um eine effiziente Suche ermöglichen zu können wurde unter anderem auch die semantische Suche genauer in Betracht gezogen. Die Semantische Suche ist eine Suche, welche die Bedeutung

hinter der Suchanfrage versteht und nicht lediglich nach den Suchbegriffen in den einzelnen Einträgen des Logbuches sucht [14]. Eine solche Suchmethode wird „Semantische Suche“ (englisch: Semantic Search) genannt.

Im Gegensatz zur Semantischen Suche steht die klassische Volltextsuche, welche von vielen Suchmaschinen verwendet wird. Hierbei wird nach dem Vorkommen der eingegebenen Keywords in einer gegebenen Menge von Texten gesucht und nicht auf die Bedeutung der Suchanfragen eingegangen. Damit ist es schwer effizient Informationen abzurufen („search with meanings“) und semantisch gleiche Einträge mit verschiedenen Beschreibungen im digitalen Logbuch würden nicht gefunden werden. Die Semantische Suche liefert somit das Potential, die Abrufleistung von Suchen zu Verbessern [14] indem die Suchanfragen präziser erfasst und mit den inhaltlich relevanten Texten in Verbindung gebracht wird [15]. Der Grundpfeiler für eine Semantische Suche ist eine gute Wissensbasis und Ontologien, die auf der Grundlage der Menge an Dokumenten, Texten, Dateien etc. entsteht, welche bei der Suche beachtet werden sollen [16]. Die Wissensbasis stellt die logische Darstellung dieser Ressourcen in Form von beispielsweise Knowledge Graphen oder Wortnetzen (Thesauren) dar [16].

Die Methode des Knowledge Graphen als Wissensbasis für die semantische Suche stellt einen gerichteten, gewichtetet Graph dar, welcher komplexe Beziehungen und Ontologien zwischen Ressourcen darstellt, entdeckt und interpretiert [16]. Dabei werden die Beziehungen zwischen den Ressourcen als Pfade im Graph dargestellt und für die Untersuchung und Navigation genutzt [16]. Das semantische Suchsystem hat die Möglichkeit aussagekräftige Ergebnisse ab zu rufen, in dem es auf solche Wissensbasen zurückgreift [16]. Um eine Wissensbasis auf zu bauen, muss jedoch zuerst die natürliche Sprache verstanden werden. Techniken wie POS tagging, chunking, entity recognition und disambiguation, sentence parsing und word vectors kommen an dieser Stelle zum Einsatz [17]. Bevor der Benutzer die Ergebnisse seiner Suchanfrage zurückbekommt, kommen Ranking-Algorithmen zum Einsatz die die am besten passenden Ergebnisse aus der Wissensbasis bestimmen [16].

Die Semantische Suche steckt noch in ihren Anfängen, weswegen noch einige Probleme in diesem Bereich auftreten: Ein Problem stellt der manuelle Annotationsprozess der Daten für den Aufbau einer Wissensbasis dar. Zwar existieren bereits automatisierte Annotatoren, jedoch sind diese noch nicht ausgereift genug, weswegen des Öfteren händisch nachgebessert werden [16]. Oftmals können die Wissensstrukturen, die bei den semantischen Suchmaschinen hinterlegt sind, nicht aktualisiert und an die neuen Anforderungen der Nutzer angepasst werden [14]. Die Suche bringt nicht mehr alle Ergebnisse, die sich der Benutzer wünscht und bleibt auf dem „alten Wissensstand“. Damit stellt die statische Wissensstruktur der Wissensbasis ein weiteres Problem der semantischen Suche dar. Da die semantische Suche noch in ihren Anfängen ist, existieren viele Methoden erstmals nur konzeptionell, weshalb bisher eine große Zahl der semantischen Suchen noch nicht experimentell getestet werden konnte [14]. Durch das Fehlen dieser experimentellen Tests kann nicht belegt werden, dass die semantische Suche besser als eine traditionelle Suche funktioniert [14]. Da die semantische Suche versucht, die Bedeutung der Suchanfrage zu verstehen, kann das gelieferte Ergebnis nur so gut wie die gestellte Frage sein [18]. Je mehr Informationen durch die Suchanfrage an die semantische Suchmaschine übermittelt werden, desto besser sind die Ergebnisse, die geliefert werden und dass die Anfrage korrekt verstanden wird [18].

Im Falle der Fachstudie müsste zuerst eine solche Wissensbasis aus den digitalen Logbucheinträgen aufgebaut werden, da keine bestehenden Ontologien bzw. Knowledge Graphen für diese fachspezifische Domäne existieren. Im Anschluss könnte dann die Suche nach semantisch gleichen Einträgen realisiert werden.

3.2.1.4 Entscheidung ein Suchverfahren

Aufgrund der spezifischen Domäne und der Menge an semantisch gleichen, aber unterschiedlichen Beschreibungen der Fehlersituationen und Handlungen fiel die Entscheidung auf die Query Expansion mit Einbindung von Synonymen. Auf Grundlage der ausführlichen Analyse zu Beginn der Fachstudie konnte eine Sammlung von Synonymen fachspezifischen Worten und Abkürzungen mit ihren Auflösungen erstellt werden. Um dieses spezifische Dictionary zu erweitern, besteht die Möglichkeit existente, umfangreiche Synonymlisten wie von „OpenThesaurus“ zusätzlich mit ein zu binden. Die detaillierte Umsetzung ist im Punkt Konzept und Umsetzung zu finden.

3.2.2 Existierende Tools

3.2.2.1 Open Semantic Search (Wissensbasis: Knowledge Graph)

Open Semantic Search, kurz OSDS, ist eine aus dem Bereich des Journalismus stammende Open Source Software zur Implementierung einer eigenen Suchmaschine wahlweise auf Basis von Apache SolR oder der Elasticsearch Enterprise-Search [25]. Es werden auf der Webseite www.opensemanticsearch.org selbst mehrere Varianten abhängig von den Interessen der Nutzer zur Verfügung gestellt, darunter auch die Open Semantic Search Appliance für die Nutzung auf öffentlichen Servern und das Open Semantic Search Server Package für die Installation auf Linux Servern. Im Folgenden wird die standardisierte Verwendung der speziellen Browseroberfläche innerhalb einer virtuellen Maschine gezeigt, die eigentliche Open Semantic Search. Diese kann über das bereitgestellte OVA-Image in einer Virtualisierungs-Maschine wie VirtualBox importiert und gestartet werden. Man erhält nun ein vollständiges auf Debian basierendes Betriebssystem mit modifizierter Oberfläche, das aus mehreren unterschiedlichen Komponenten besteht. Der zentrale Baustein findet sich dabei im User Interface, das im Browser der VM integriert ist und direkt beim Start der VM geöffnet wird. Über diverse Schaltflächen in der Menüleiste kann eine simple Suche erstellt und angewendet werden. Zunächst muss dafür ein Datensatz importiert werden, auf dessen Grundlage im Folgenden gesucht werden kann. Dabei unterstützt Open Semantic Search mehrere Dateiformate, darunter auch das CSV-Format oder Texte von Webseiten. Außerdem bietet OSDS die Möglichkeit, eine eigene Synonymliste hochzuladen, die automatisch vom System erkannt wird und dann in der erweiterten Suche genutzt werden kann. [19]

Nun soll die Architektur von Open Semantic Search aufgezeigt werden. Die zentralen Komponenten sind:

- Das gerade vorgestellte **User Interface** dient einem einfachen, übersichtlichen Zugang zum System und nimmt Benutzeranfragen direkt entgegen. Solche wären u.a. die Volltextsuche innerhalb einer oder mehrerer Quellen, das Erstellen unterschiedlicher Filter, die Navigation in der Benutzeroberfläche selbst oder das sogenannte Tagging, um Metadaten zu strukturieren.
- Der **Index Server** und der **Suchserver**, wahlweise von Elasticsearch oder SolR, um den bereits kreierten Index zu verwalten und um Suchanfragen entgegenzunehmen und zu bearbeiten.
- Das **Open Semantic ETL Framework**, das den Import der Datenquellen und deren Integration in das System durch Analyse übernimmt. Dabei steht ETL für „Extract, transform, load“ der Pipelines, ein spezieller ETL Explorer sorgt dafür, dass die Daten zum Suchindex zugewiesen werden.

- Über **Web Services (REST-API)** können Befehle in der Konsole direkt an die Suchmaschine übermittelt werden, diese Befehle könnten z.B. der Zugriff auf bestimmte Verzeichnisse oder das Suchen über einen definierten Index sein.
- Der **Queue Manager** verwaltet die Warteschlange der Prozesse und das Starten der Textextraktion, Analyse, Datenanreicherung und Indizierung von Jobs.

Nachdem die Komponenten des Systems genannt wurden, soll nun betrachtet werden, wie neue Daten innerhalb dieser Komponenten verarbeitet werden. Entweder der Nutzer selbst oder ein vorgefertigtes Skript startet immer wieder Befehle, die dann entgegengenommen werden können. Dies geschieht über eine spezielle Web API, die dann das bereits erwähnte ETL Framework und die Analyse der Daten startet, um diese einzulesen, zu analysieren und zu indexieren. Dabei kümmert sich ein sogenannter Connector darum, die Dateien von der ursprünglichen Quelle einzulesen, um deren Inhalt anschließend mithilfe eines Apache Tika Parsers oder Converters zu extrahieren. Das ETL Framework ruft dabei zusätzlich alle Tools für die Auswertung der Daten auf, um weitere Analysen der Daten durchzuführen. Ein sogenanntes Output Storage Plugin indexiert dann den Text sowie weitere Metadaten für den SolR/ElasticSearch Index, um Benutzeroberflächen eine Suche hiermit zu ermöglichen. Zudem bietet OSDS eine automatische Annotation (Aufbau der Wissensbasis) der Texte durch spezielle integrierte Text-Mining- und Textanalysetechnologien, die jedoch nicht für jeden Anwendungsfall korrekt funktionieren und dadurch händisch ergänzt werden können. Auf Basis dieser Annotationen wird dann ein Knowledge Graph erstellt, der als Wissensbasis genutzt werden kann. Anschließend kann der Benutzer mithilfe einer Oberfläche (Search User Interface) oder vergleichbaren Tools eine Suche erstellen. [19]

3.2.2.2 ElasticSearch

ElasticSearch stellt eine eigene Such- und Analysemaschine dar, welche auf Basis von Apache Lucene erstellt wurde und hauptsächlich der Volltextsuche dient [20]. Diese Suchmaschine bietet außerdem mehrere Möglichkeiten, um die Suche zu spezifizieren und kann mit Hilfe von „Kibana“ visualisiert werden [21]. Unter anderem ist somit die Suche nach semantisch gleichen Einträgen über die Query Expansion möglich, wie es im Rahmen der Fachstudie benötigt wird. Hierzu werden zunächst Synonymlisten der fachspezifischen Domäne eingebunden, welche schließlich in der Ergebnisausgabe berücksichtigt werden. Wie ElasticSearch bei der Verarbeitung vorgeht, kann in folgende Schritte eingeteilt werden:

1. **Benutzer:** Synonymdatei der spezifischen Domäne erstellen und einbinden
2. **Benutzer:** Dokumente/Datei hinzufügen, auf welchen am Ende gesucht & analysiert werden soll
3. **ElasticSearch:** einfache Textverarbeitungsschritte
4. Apache Lucene Standard Analyzer: Indexierung der Dokumente
5. **Benutzer:** Suchanfrage stellen
6. ElasticSearch/Lucene: Query Expansion
7. **Apache Lucene:** Scoring der Ergebnisse
8. Ergebnisausgabe

Da Suchanfragen bei dieser Suchmaschine nicht direkt auf die Inhalte selbst, sondern auf die Indexe bezogen werden, müssen die Dokumente zuerst durchsuchbar gemacht werden [22]. Hierzu werden die Inhalte der Dokumente, welche am Ende durchsucht und analysiert werden sollen, durch einfache Textverarbeitungsschritte aufbereitet worauf schließlich die Indexierung mit Hilfe des Apache Lucene Standard Analyzer durchgeführt wird. Die grundlegende Struktur des Index bei dieser Suchmaschine ist gleich wie bei SolR der invertierte Index [22]. Die genaue Beschreibung der Verarbeitungsschritte von einem Dokument zu einem invertierten Index sind in Kapitel 3.2.2.4 zu finden.

Nach der Aufbereitung der Daten und der Indexierung ist es möglich Suchanfragen zu stellen. Die hier verwendete Query DSL verwendet für die Ausführung und Verarbeitung dieser Anfragen die Lucene TermQuery [23]. Für die Suche nach den semantisch gleichen Einträgen innerhalb des digitalen Logbuches wurde in der Fachstudie die Zuweisung der Synonyme zur Abfragezeit verwendet (Query Expansion). Hier werden die Synonyme der eingebundenen Synonymliste auf die einzelnen Worte der Abfrage abgebildet und ebenfalls in der anschließenden Suche im invertierten Index berücksichtigt. Da diese Verarbeitungsschritte erneut äquivalent zu den Schritten in SolR sind, wurden diese ebenfalls in einem gesonderten Kapitel 3.2.2.5 für beide Suchmaschinen beschrieben.

Wurden für die Suchanfrage relevante Einträge gefunden, so werden diese schließlich nach Relevanz sortiert und ausgegeben. Elasticsearch und SolR verwenden auch hier gleichermaßen eine Scoring-Funktion von Apache Lucene, das so genannte tf.idf-Modell [24]. Aus diesem Grund wird in Kapitel 3.2.2.6 das Scoring für beide Suchmaschinen erläutert.

Durch die oben beschriebenen Schritte wird die Suche nach semantisch gleichen Einträgen, wie sie im Rahmen der Fachstudie auf den digitalen Logbucheinträgen von Festo benötigt wird, durch Elasticsearch möglich.

3.2.2.3 Apache SolR

SolR, ausgesprochen „Solar“, ist eine in Java geschriebene Open-Source-Suchplattform auf Unternehmensniveau aus dem Apache Lucene Projekt. Die Suchplattform lässt sich für Anwendungen im Big-Data-Umfeld einsetzen. Außerdem ist sie eines der beliebtesten Werkzeuge für die Integration vertikaler Suchmaschinen, die sich auf eine bestimmte Domain, ein bestimmtes Thema oder eine Zielgruppe beschränkt. Netflix und Ebay sind bekannte Beispiele, die als Grundlage für ihre vertikale Suchmaschinen ihrer Webseite, SolR nutzen, um spezifische Texte oder Angebote auf ihrer Webpräsenz für ihre Benutzer fündig machen zu können.

SolR ist bekannt für ihre Beständigkeit, Skalierbarkeit und Zuverlässigkeit. Zu den weiteren Vorteilen gehört unter anderem der große Funktionsumfang und die beschleunigte Indexierung. Außerdem bietet SolR eine Schnittstelle für Plugins zur Anpassung von spezifischen Anforderungen. Befehle werden in HTTP (Hypertext Transfer Protocol) geschrieben und für zu speichernde Dateien wird XML (Extensible Markup Language) genutzt. SolR bietet eine Suche neben der Suche nach Textfeldern (auch mehrere gleichzeitig), eine Sortierung nach Ähnlichkeit oder Relevanz der Suchergebnisse. Unter anderem ist auch die Funktion der Synonym-Suche vorhanden, die eine Suche nach semantisch gleichen Einträgen ermöglicht. Dies wird durch eine Query Expansion unterstützt, wie es in dem Anwendungsfall von Festo im Rahmen der Fachstudie benötigt wird. Hierzu werden zunächst Synonymlisten der fachspezifischen Domäne eingebunden, welche schließlich in der Ergebnisausgabe berücksichtigt werden. [25]

Parallel zu Elasticsearch kann man die Verarbeitung bei SolR in fast identische Schritte einteilen, da beide Suchmaschinen auf demselben Projekt, Apache Lucene, basieren.

1. Aufbereitung der Daten
2. Indizierung der Daten
3. Abfragen
4. Mapping
5. Darstellung und Sortierung der Ergebnisse

Genau wie bei Elasticsearch müssen die Dokumente zunächst durch einfache Textverarbeitungsschritte aufbereitet werden, um sie durchsuchbar zu machen. Daraufhin findet die Indexierung mit Hilfe von Apache Lucene Standard Analyzer statt, dessen grundlegende Struktur dem von Elasticsearch gleich ist. Hierfür werden sie in ein maschinenlesbares Format, dem Index, übersetzt.

Nachdem die ersten zwei Schritte abgeschlossen sind, können nun Suchanfragen gestellt werden. Diese kann über einen User erfolgen oder über eine Anwendung, der die Sucheingaben und Suchbegriffe übersetzt. Diese Suchbegriffe werden im Mapping auf die Dokumente des Index angewandt, um die gewünschten Ergebnisse zu finden. Durch die Query Expansion wurde die Suche nach semantisch gleichen Einträgen innerhalb des digitalen Logbuches in der Fachstudie ermöglicht, indem man die Zuweisung der Synonyme zur Abfragezeit verwendet hat. Dabei werden die Synonyme der eingebundenen Synonymliste auf die einzelnen Worte der Abfrage abgebildet und ebenfalls in der anschließenden Suche im invertierten Index berücksichtigt. Da diese Verarbeitungsschritte erneut äquivalent zu den Schritten in Elasticsearch sind, wurden diese ebenfalls in einem gesonderten Kapitel 3.2.2.5 für beide Suchmaschinen beschrieben.

Die von der Suchmaschine gelieferten Ergebnisse werden anschließend nach bestimmten Kriterien wie Relevanz sortiert und in der SolR-Weboberfläche oder in der genutzten Anwendung aufgelistet. Für die Relevanz nutzt SolR auch die Scoring-Funktion von Apache Lucene (tf-idf-Modell). Dies wird im Kapitel 3.2.2.6 näher erläutert.

Durch SolR und ihren oben beschriebenen Schritten, wird die Suche nach semantisch gleichen Einträgen, wie sie im Rahmen der Fachstudie auf den digitalen Logbucheinträgen von Festo benötigt wird, möglich. [26]

3.2.2.4 Indexierungsvorgang in SolR und Elasticsearch

Die Indexierung der Daten wird sowohl bei Elasticsearch als auch bei SolR [27] mit Hilfe von Apache Lucene durchgeführt. Hierbei werden die Daten durch die Struktur des „invertierten Index“ für Maschinen lesbar gemacht [28]. Der Prozess der Indexierung wird auch als Analyse bezeichnet und beinhaltet neben der Tokenisierung aber auch die Filterung [28]. Dadurch, dass die Daten bereits bei der Indexierung aufbereitet werden, wird bei der anschließenden Verarbeitung der Suchanfrage Zeit gespart [29].

Der erste Schritt in der Verarbeitung von einem Dokument zu einem invertierten Index ist die Tokenisierung. Bei der Tokenisierung wird der eingegebenen Text in separate Worte „aufgesplittet“ [28]. Diese Worte/Indexe werden anschließend, in sortierter Form in einem Dictionary gespeichert [30]. Liegt ein invertierter Index vor, so werden die Begriffe des Dictionary auf das jeweilige Dokument abgebildet, in dem diese existent sind, statt die Dokumente auf die Worte des Dictionary ab zu bilden [30]. Ein solches Dictionary besteht aus [30]

- dem Token – Wort, welches erkannt wurde (Index)

- der Frequenz – Häufigkeit des Auftretens
- dem Dokument – Aufzählung der Dokumente in dem das Bestimmte Wort enthalten ist
- der Position – An welcher Stelle im Dokument das Wort/Index zu finden ist.

Die Information der Frequenz fließt hierbei in die Berechnung der Relevanz der Suchergebnisse auf eine bestimmte Suchanfrage mit ein, wobei die Angaben zur Position der Indexe im Dokument spielt eine wichtige Rolle, wenn bei den Suchanfragen des Nutzers nach Phrasen gesucht werden soll [31]. Dabei kann sichergestellt werden, dass die einzelnen Worte der gesuchten Phase in den Daten, die durchsucht werden, tatsächlich in der gewünschten Reihenfolge nacheinander auftauchen. So werden im Ranking für die Ergebnisausgabe die Dokumente, in denen die Suchbegriffe näher beieinander liegen höher gewichtet als andere.

Nachdem die Tokenisierung durchgeführt wurde und die jeweiligen Begriffe in einem Dictionary abgespeichert wurden, werden einige Textverarbeitungsschritte bzw. Filter auf den Worten des Dictionary durchgeführt (Standard-Analysator). Dazu gehörten standardmäßig [30]:

- Lowercasing – konvertiert alle Großbuchstaben in Kleinbuchstaben [32]
- Removing punctuation – entfernt alle enthaltenen Satzzeichen

Neben den Standard Filtern, die hier auf den Indexen angewendet werden, kann der Benutzer noch weitere Verarbeitungsschritte hinzufügen. Zu diesen gehören beispielsweise:

- Stemming – Term wird auf „Wortstamm“ reduziert [33]
- Synonyme – Worte, die die gleiche Bedeutung haben wie ein anderes Wort

Um die einzelnen Schritte zu verdeutlichen folgt ein Beispiel, welches Daten aus dem digitalen Logbuch enthält.

Tabelle 6. Beispieleinträge aus dem DigiLog für die Erstellung des invertierten Index Tabelle

Z-Achse Fehler Winkelgeber
Fehler Nachschieben trotz funktionierenden Sensoren.

1. Tokenisierung der Einträge

Z-Achse	Fehler	Winkelgeber	
Fehler	Nachschieben	trotz	funktionierenden Sensoren.

2. Lowercasing

z-achse	fehler	winkelgeber	
fehler	nachschieben	trotz	funktionierenden sensoren.

3. Removing punctuation

z-achse	fehler	winkelgeber	
fehler	nachschieben	trotz	funktionierenden sensoren

Das entstehende Dictionary lässt sich im Anschluss wie folgt darstellen:

Tabelle 7. Invertierter Index der Beispieleinträge aus Tabelle 6

Term	Frequenz	Dokument: Position
fehler	2	1:2, 2:1
funktionierenden	1	2:4
nachschieben	1	2:2
sensoren	1	2:5
trotz	1	2:3
winkelgeber	1	1:3
z-achse	1	1:1

Nach der Indexierung ist es möglich Suchanfragen zu stellen. Die Worte der Suchanfrage müssen nun nur noch mit denen des Dictionary abgeglichen werden, um heraus zu finden, welche Dokumente für die Ergebnismenge relevant sind.

3.2.2.5 Synonymverarbeitung in SolR und ElasticSearch

Um mit Hilfe der Query Expansion auf den digitalen Logbüchern eine effiziente Suche durchführen zu können, welche alle relevanten, semantisch gleichen Einträge auf eine Suchanfrage ausgibt, wurden sowohl in SolR, als auch in ElasticSearch Synonyme der spezifischen Domäne eingebunden. Da beide Suchmaschinen auf Apache Lucene basieren und die Verarbeitung der Query jeweils durch Apache Lucene TermQuery [23] erfolgt, kann die Verarbeitung der Synonyme durch die Query Expansion gemeinsam beschrieben werden.

Neben der Möglichkeit die Synonyme in der Query Expansion zu verarbeiten, können diese alternativ direkt auf den Index abgebildet werden. Werden die Synonyme jedoch auf die Indexe abgebildet, so steigen die Indexierungszeit und -größe an [34], was die Index-Performance sinken lässt und einen höheren Speicher für die diese erfordert. Werden die Synonyme auf der anderen Seite zur Abfragezeit auf die Query abgebildet (Query Expansion), so trennen die Performance der Abfrage zwischen den beiden Möglichkeiten laut Marquiss und Kluwer [34] etwa 10%. Jedoch liegt die Query Performance, bei der die Synonyme auf die Query abgebildet werden, immer noch im guten Bereich [34]. Aus diesen Gründen ist die Synonymverarbeitung zur Abfragezeit zu empfehlen [34].

Wie bereits erwähnt wird zunächst eine Liste von Synonymen in die beiden Suchmaschinen eingebunden. Hier können bereits existierende umfangreiche Synonymlisten verwendet werden, aber auch eigens erstellte. Eine solche bestehende umfangreiche Liste an Synonymen wird Beispielsweise von „OpenThesaurus“ bereitgestellt, welche auf dessen Webseite, zum Download bereit zu finden ist [35]. Da es sich im Falle der digitalen Logbucheinträge um eine fachspezifische Domäne handelt, wofür noch keine vorgefertigte Synonymliste bereitsteht, wurde eine eigene Liste dieser erstellt, welche für diesen Anwendungsfall einsetzbar ist. Von hoher Wichtigkeit ist es, dass die Liste, welche als Textdokument (.txt), eingebunden werden soll in der Richtigen Form vorliegt. Welche Syntax akzeptiert werden ist in der nachfolgenden Tabelle zu finden [36]:

Tabelle 8. Akzeptierte Syntax der Synonymlisten

Beschreibung	Beispiel
Eine durch Kommata getrennte Liste von Synonymen Worten in einer Zeile. Wenn ein Token äquivalent zu einem Wort der Liste ist, so werden alle synonymen Worte der Liste für die Weiterverarbeitung verwendet.	ausgetauscht, ersetzt, getauscht, neu eingebaut, gewechselt
Zwei durch Kommata getrennte Wortlisten, zwischen denen ein Pfeil-Symbol „=>“ existiert (Explizite Zuordnung). Falls ein Token zu einem Wort, welches auf der linken Seite steht, äquivalent ist, werden lediglich die Worte der rechten Seite zur Weiterverarbeitung verwendet.	RT => RT, Rundschalttisch

Sind die Synonyme in der richtigen Form eingebunden und es wurde ausgewählt, dass diese zur Abfragezeit verarbeitet werden sollen, so werden die Einzelnen Worte der Abfrage mit der Liste der Synonymen abgeglichen. Um den Prozess zu verdeutlichen wurde das nachfolgende Beispiel erstellt. Die erste Aufgabe ist es, die Synonymliste für die fachspezifische Domäne anzulegen. In Tabelle 9 Tabelle 9. Ausschnitt Synonymdatei der digitalen Logbucheinträge ist eine verkürzte Form mit einigen Beispielen, der angefertigten Liste über die ganze Domäne zu finden.

Tabelle 9. Ausschnitt Synonymdatei der digitalen Logbucheinträge

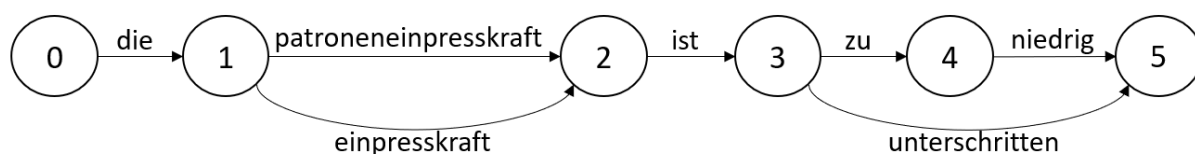
ausgetauscht, ersetzt, getauscht, neu eingebaut, gewechselt
unterschriften, zu niedrig
Einpresskraft, Patroneneinpresskraft
entfernt, gelöscht
RT => RT, Rundschalttisch

Sind sowohl die Synonym-Datei als auch die Daten des digitalen Logbuches eingebunden und indexiert, so kann eine Suchanfrage gestellt werden. Zunächst wird die Suchanfrage verarbeitet, was bedeutet, dass diese zuerst in einzelne Tokens eingeteilt wird, worüber dann, wie bei der Verarbeitung zum Index, verschiedene Filter bzw. Analysatoren laufen können [36]. Einer dieser Filter stellt der Synonym-Filter dar. Die Tokenisierung und die Zuweisung der Synonyme dieser Verarbeitungsschritte lassen sich über einen Graphen veranschaulichen [37]. Hier stellt jeder Knoten eine Position in der Query und jeder Pfeil das zugehörige Token dar [37]. Stellt man die Abfrage:

„Die Patroneneinpresskraft ist zu niedrig.“

So ergibt sich nach den Verarbeitungsschritten also folgender Graph:

Abbildung 3. Visualisierung Synonymzuweisung zur Abfragezeit



Im invertierten Index werden somit nach den Worten der Query, aber auch nach dessen Synonymen gesucht. Somit ist eine effizientere Suche garantiert, da sich durch die zusätzliche Suche nach Synonymen im Index eine präzisere Ergebnismenge ergibt.

3.2.2.6 Scoring der Ergebnisse in SolR und Elasticsearch

Der letzte Schritt in der Verarbeitungsschritte, welche für eine Suchanfrage des Benutzers ausgeführt wird, ist das Scoring der Dokumente nach Relevanz. Elasticsearch und SolR verwenden hierzu gleichermaßen die Scoring-Funktion von Apache Lucene, welche auch als tf.idf- Modell bekannt ist [24]. Hier werden die Dokumente, welche mit der Suchanfrage übereinstimmen durch einen bestimmten Algorithmus nach Relevanz geordnet und bestimmen die Reihenfolge, in der sie dem Nutzer ausgegeben werden [38].

Die folgende Beschreibung der Bewertungsfunktion basiert hauptsächlich auf Quelle [39] und wird als praktische Bewertungsfunktion bezeichnet. Die Formel (Formel 1) der Bewertungsfunktion hat folgenden Aufbau:

Formel 1. praktische Bewertungsfunktion Apache Lucene

$$score(q, d) = coord(q, d) \times queryNorm(q) \times \sum_{t \in q} (tf(t \text{ in } d) \times idf(t)^2 \times t.getBoost() \times norm(t, d))$$

Die Bedeutung der einzelnen Komponenten der Formel sind in der Tabelle 10 nach zu lesen.

Tabelle 10. Beschreibung Scoring Bewertungsfaktoren

Bewertungsfaktoren	Beschreibung	Berechnung
tf(t in d) – Termhäufigkeit	Häufigkeit, mit der ein Begriff in einem Dokument vorkommt. Je häufiger der Begriff im Dokument vorkommt, desto höher ist die Bewertung	$\sqrt{Termhäufigkeit}$
idf(t) – Inverse Dokumentfrequenz	Je höher das Vorkommen eines Terms in verschiedenen Dokumenten ist, desto niedriger wird er gescored	$\log\left(\frac{numDocs}{docFrequency + 1}\right) + 1$
coord(q,d) – Koordinierungsfaktor	Je mehr Suchbegriffe aus der Anfrage in einem Dokument gefunden	$\frac{Overlap}{maxOverlap}$

	werden, desto höher ist das Scoring	
fieldNorm – Feldlänge	ein Begriff, der in Feldern mit weniger Begriffen übereinstimmt, hat eine höheres Scoring	$\frac{1}{\sqrt{\#Terme}}$
queryNorm(q)	Normalisierungsfaktor, damit Abfragen verglichen werden können. Bewertungen liegen nach der Normalisierung zwischen 0 und 1,0	$\frac{1}{\sqrt{sumOfSquaredWeights}}$
t.getBoost()	Suchzeitverlängerung des Begriffs t in der Abfrage q. Es gibt keine direkte API für den Zugriff auf einen Boost von einem Term in einer Multi-Term-Abfrage	
norm(t,d)	Index-Zeit-Boost-Faktor, der ausschließlich von der Anzahl der Token dieses Feldes im Dokument abhängt, so dass kürzere Felder mehr zum Ergebnis beitragen	$doc.getBoost() * lengthNorm * \prod_{field\ in\ d\ named\ as\ t} f.getBoost()$

Basierend auf der Formel lassen sich folgende Aussagen über die Relevanz von Einträgen/Dokumenten treffen:

- Dokumente/Einträge, welche alle Suchbegriffe beinhalten haben eine hohe Relevanz
- Die Übereinstimmung mit selteneren Worten in den Dokumenten/Einträgen bringt eine höhere Relevanz ein, als die Übereinstimmung mit häufig vorkommenden Worten
- Kurze Dokumente/Einträge haben eine höhere Relevanz als Lange Dokumente
- Dokumente/Einträge, welche die Suchbegriffe der Anfrage häufig enthalten haben eine hohe Relevanz

Die Anpassung der Relevanz der Dokumente können von Nutzern durch so genannte Boost Befehle durchgeführt werden. Diese Boosts werden dann in der Berechnung der Relevanz unter dem Punkt `t.getBoost()` einberechnet. Diese Boosts sind sowohl zur Abfragezeit als auch zur Indexzeit möglich [24].

Boosts zur Indexzeit werden angewendet, wenn Dokumente zur Indexierung hinzugefügt werden und werden auf das gesamte Dokument oder nur auf einzelne Felder wirksam [24]. Finden Boosts zur Abfragezeit statt, so werden diese hinzugefügt, wenn eine Suchanfrage konstruiert wird und

gelten für einzelne Felder [24]. Jedoch wird die Anpassung der Relevant durch Boosts zur Indexzeit nicht empfohlen [40].

In Tabelle 11 wird gezeigt, wie ein solcher Boost zur Abfragezeit in SolR und in ElasticSearch aussehen kann:

Tabelle 11. Boots zur Abfragezeit

SolR	ElasticSearch
Ausgangssituation:"Einpresskraft zu niedrig" && Art:"Rezeptur") (Ausgangssituation:"Einpresskraft zu niedrig" && Art:"Mechanisch")^2	POST _search { "query": { "match": { "Ausgangssituation": { "query": "Die Patroneneinpresskraft ist Zu niedrig" "boost": "2" } } }

3.2.3 Vergleich und Entscheidung

Tabelle 12. Vergleich der Suchmaschinen [41]

	ElasticSearch	SolR	Open Semantic Search
Kurzbeschreibung	Eine moderne Such- und Analyseplattform basierend auf Apache Lucene (2010)	Häufig verwendete Suchmaschinentechologie basierend auf Apache Lucene (2004)	eine aus dem Bereich des Journalismus stammende Open Source Software
APIs und andere Zugriffskonzepte	Java API RESTful HTTP/JSON API	Java API RESTful HTTP API	
Dokumentformat	JSON	XML, CSV, JSON	
Datenschema	Schemafrei (Vorteil) → Zu Projektbeginn wird keine Datenstruktur angegeben und übernimmt das Mapping von Daten auf einen spezifischen Datentyp automatisch.	Besitzt ein Schema (managed-schema) Bei SolR hingegen muss man erst umständlich die Datei schema.xml pflegen, bevor man loslegen kann.	
Dokumentation	Mangelt an Dokumentation und konkreten Anwendungsbeispielen	verfügt über eine außerordentlich gute Dokumentation → stabilere, erfahrenere Entwickler-Fundament	Mangelnde Dokumentation. 1-2 Anwendungsbeispiele
Suchen	analytischen Abfragen, Filtern und Gruppieren	Auf Textsuche ausgerichtet	
Hauptunterschied	Architektur zum Erstellen moderner Echtzeit-Suchanwendungen (mit Kibana) → ist einfacher skalierbar, schnell und lässt sich problemlos integrieren	<ul style="list-style-type: none"> • Standardsuchanwendungen erstellen. • Hat eine größere, ausgereifere Community für Benutzer, Entwickler und Mitwirkende. • SolR ist reifer und schneller beim Indizieren von Dokumenten. 	
Tools & Funktionen	Übernimmt Führung → lange Liste von Werkzeugen, die mit ES verwendet werden können (z.B. Kibana, Logstash, ...)	Hat auch sehr viele Funktionen, wobei durch die neu kommenden Updates, alte Tools nicht mehr so wie davor funktionieren	

Wie man der Tabelle 12 entnehmen kann, ist die Spalte zu Open Semantic Search nicht vollständig. Das liegt an der mangelnden Dokumentation. Auch das Nutzen des Tools war kein Erfolg. Den Datensatz konnte man hochladen, jedoch gab es Probleme bei der Synonymliste. Diese konnte nicht eingebunden und folglich auch keine Query Expansion abgebildet werden.

Bei den meisten Menschen mit Erfahrung und technischem Hintergrund (auch bei uns), ist SolR ein klarer Favorit unter den drei vorgestellten Tools. SolR ist weit verbreitet und viel reifer. Es bietet eine detailliertere Funktionsumgebung, zahlreiche Plug-Ins und eine große Community der Entwickler.

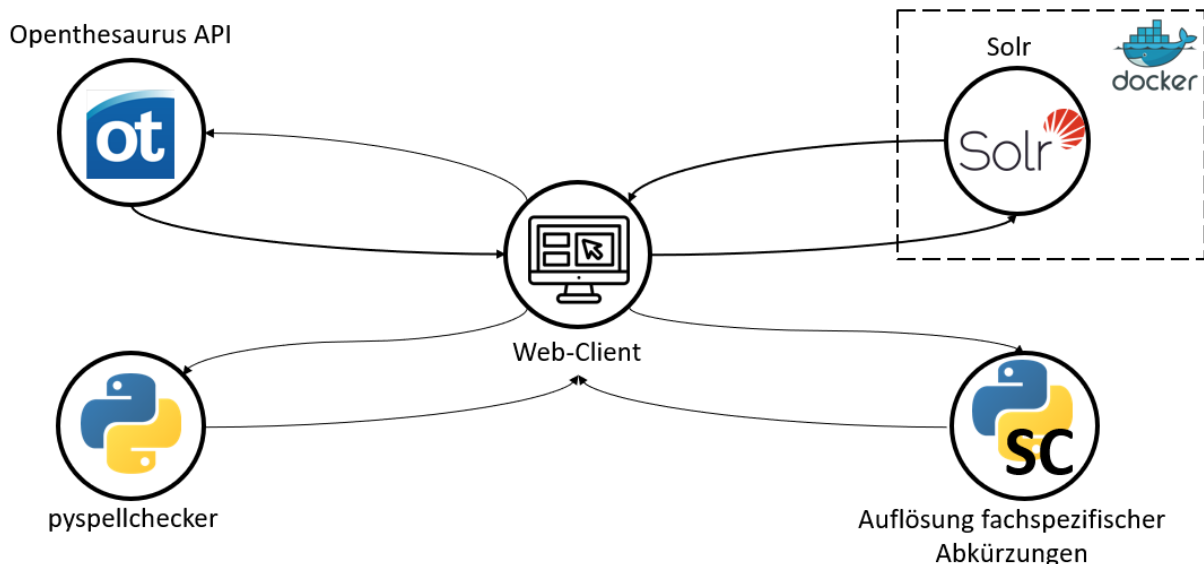
ElasticSearch ist zwar neuer, aktueller, leichter und punktet mit Kibana in der Visualisierung der Daten, jedoch ist sie bei unserer Inbetriebnahme nicht einwandfrei gelaufen. Nachdem die CSV-Datei mit dem vorliegenden Datensatz mit Logstash in ElasticSearch hochgeladen und indexiert wurde, konnten die Synonymliste nicht auf den Datensatz angewendet werden. Dadurch, dass ElasticSearch ein relativ junges Projekt ist, gab es auch in der Community keine Lösung, die dieses Problem gelöst hat.

Aus diesem Grund hat man sich letztendlich für das Tool SolR entschieden.

4 Konzept und Umsetzung

4.1 Allgemeines Konzept

Abbildung 4. Gesamtarchitektur



In Abbildung 4 wird die Gesamtarchitektur des Konzepts visuell dargestellt. Zentral befindet sich der Web-Client von dem aus, alle Programme, Tools und Algorithmen laufen. Nach der Analyse des Datensatzes wurde ein Algorithmus in Python geschrieben, welcher die fachspezifischen Abkürzungen auflöst (unten rechts, „sc“ = shortcuts). Anhand des von Abkürzungen befreiten Datensatzes, konnte der „pypellchecker“ von Python angewendet werden, der die Rechtschreibfehler im Datensatz in Bezug auf die spezifische Domäne korrigiert hat (unten links). Nun ist die Wissensbasis für eine korrekte und effiziente semantische Suche geschaffen.

Als nächstes wird ein Thesaurus für die Query Expansion bestimmt, sowie eine eigene List an Synonymen in Bezug auf die spezifische Domäne aufgestellt. Hierbei wurde zusätzliche „OpenThesaurus“ genutzt, welches eine ausführliche Liste an gängigen Synonymen bereitstellt. Dieser Thesaurus wurde in Form einer Textdatei heruntergeladen und mit den fachspezifischen Abkürzungen, sowie fachspezifischen Synonyme der Textdatei ergänzt, um bei der Query Expansion eine Suche nach Abkürzungen und semantisch gleichen Einträgen zu ermöglichen (oben links).

Zuletzt wurde auf dem Web-Client SolR aufgesetzt. Hierzu wurde mit Docker Containern gearbeitet, um den Server zu starten und den bezüglich der Textqualität verbesserten Datensatz in SolR hochzuladen (oben rechts). Auf der Weboberfläche von SolR können nun Suchanfragen auf den hochgeladenen Datensatz gestellt werden, welche eine präzise Ergebnismenge zurückliefern.

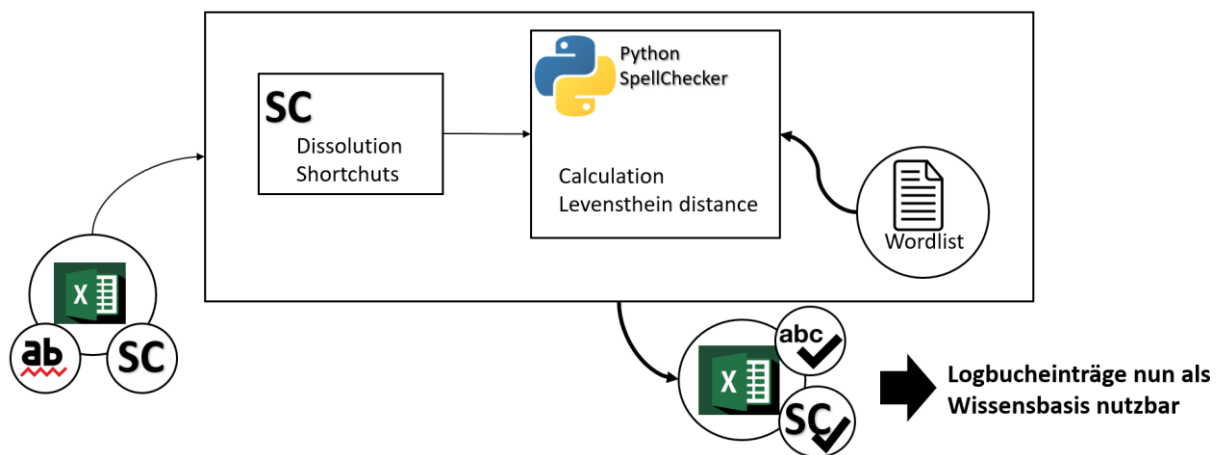
4.2 Textqualitätsverbesserung

4.2.1 Konzept der Textqualitätsverbesserung

Für die beiden wesentlichen Verarbeitungsschritte der Textqualitätsverbesserung, nämlich die Auflösung der fachspezifischen Abkürzungen (Shortcuts = SC) und die Korrektur der Rechtschreibfehler, soll im Folgenden das zugrunde liegende Konzept näher erläutert werden. Betrachtet werden hierbei die vorliegenden Rohdaten im Ausgangsformat als Excel-Tabelle der digitalen

Logbucheinträge von Festo. Zunächst wird diese Datei im UTF-8-Format gespeichert und mit Hilfe des selbst geschriebenen Python-Programms alle domänenspezifischen Abkürzungen und Fehlernummern aufgelöst. Anschließend gelangt die nun bereits bearbeitete Version in den „pyspellchecker“, der jedes einzelne Wort mit dem modifizierten Wörterbuch inklusive integrierter fachspezifischer Wortliste abgleicht und alle gefundenen Rechtschreibfehler innerhalb des Datensatzes durch die bestmögliche Korrektur ersetzt. Hierzu berechnet der „pyspellchecker“ die Levenshtein-Distanz und verwendet das Zielwort mit der geringsten Distanz für die Ersetzung. Das gesamte Verfahren ist in Abbildung 5 Fehler! Verweisquelle konnte nicht gefunden werden. anschaulich beschrieben und dort in drei Teile gegliedert. Der erste Teil zeigt die vorliegenden unkorrigierten Rohdaten, der zweite und zentrale Baustein des Konzepts beschreibt die Bearbeitung und Berechnung der Schritte der Textqualitätsverbesserung in Python und der letzte Teil liefert dann die korrigierte Version der Daten wiederum als Excel-Tabelle. Nachdem diese Schritte abgearbeitet wurden, liegt nun der gesamte Datensatz als strukturierte vollständige Wissensbasis vor, die als Grundlage für die Suche nach semantisch gleichen Einträgen genutzt werden kann.

Abbildung 5. Konzept der Textqualitätsverbesserung



4.2.2 Umsetzung der Textqualitätsverbesserung

Der erste Schritt der Implementierung sieht die Verbesserung der Textqualität durch Rechtschreibkorrektur und durch die Auflösung von fachspezifischen Abkürzungen vor. Durch die Verwendung des „pyspellcheckers“ als Grundlage für die Rechtschreibkorrektur wird die Programmiersprache Python für die gesamte Textqualitätsverbesserung genutzt. Zur Integration der Excel-Tabelle des vorliegenden Datensatzes wird die Bibliothek „Openpyxl“ verwendet. Die Sprache des „pyspellcheckers“ muss innerhalb des Programmcodes auf Deutsch gestellt werden, anschließend wird eine selbst angelegte Wortliste integriert, die speziell für die vorgegebene Domäne korrekt geschriebene Wörter enthält. Diese wurde zuvor basierend auf der vorangehenden Analyse des Datensatzes händisch angelegt und vervollständigt damit die Grundwörter der deutschen Sprache um die Fachwörter der spezifischen Domäne.

Im ersten Schritt werden die relevanten Spalten „Ausgangssituation“ und „Handlung“ der Excel-Tabelle mit Hilfe von Openpyxl korrekt eingelesen und im Anschluss die zahlreichen domänenspezifischen Abkürzungen substituiert. Dafür wurde der gesamte fachspezifische Datensatz zuvor händisch auf Abkürzungen und Fehlernummern analysiert und diese anschließend von den fachspezifischen Experten der Firma Festo aufgelöst und in einer Liste detailliert beschrieben. Auf Basis dieser Paare aus Abkürzung und Beschreibung wird nun jede vorkommende Abkürzung in-

nerhalb des eingelesenen Textes erkannt und mit Hilfe von regulären Ausdrücken durch die dazugehörige Beschreibung ersetzt. Beispielsweise enthielt der Datensatz die Abkürzung „Fehler A125“, der die fehlende Kommunikation der Maschine mit einem Roboter darstellt. Um diesen speziellen Fehler später bei der Suche nach dem Schlagwort „Roboter“ zu finden, wird diese Abkürzung automatisch durch „Fehlernummer A125: Keine Kommunikation zum Roboter“ ersetzt. Somit beinhaltet der Programmcode die komplette Liste aller fachspezifischen Abkürzungen, welche für die Anwendung innerhalb eines anderen Fachbereichs vollständig ersetzt werden müsste.

Im zweiten Schritt wird nun der bearbeitete Text einer Rechtschreibkorrektur unterzogen, indem der Volltext vom „pyspellchecker“ in jedes Wort gesplittet und dann auf Rechtschreibung geprüft wird. Durch die Methode „unknown“ (Liste aller Wörter) gleicht der „pyspellchecker“ jedes einzelne Wort mit seinem eigenen Wörterbuch, das auch die anfangs integrierten Fachwörter der Domäne enthält, ab. Danach werden die gefundenen Wörter unter Verwendung der Methode `correction(Wort)` mit der bestmöglichen Korrektur des Begriffs substituiert. Hierfür verwendet der „pyspellchecker“ die Levenshtein-Distanz und ersetzt das jeweilige Wort durch den Begriff des Wörterbuchs, der die geringste Distanz zum ursprünglichen Wort aufweist.

4.2.3 Code-Einblick

In der folgenden Abbildung (Abbildung 6) wird nun ein Ausschnitt des Python-Codes aufgezeigt, welcher die Rechtschreibkorrektur durchführt. Die gezeigte Methode „`spellcheck(String)`“ erhält einen String, der die gesamten Rohdaten beinhaltet und teilt diesen über die Hilfsmethode „`split_words(String)`“ der „pyspellchecker“-Bibliothek in einzelne Wörter auf. Anschließend prüft die bereits erwähnte „`unknown()`“-Methode alle Wörter auf Rechtschreibfehler, sodass danach über jeden gefundenen Fehler iteriert werden kann. Die zuvor genannte „`correction()`“-Methode findet nun die Korrektur mit der geringsten Levenshtein-Distanz aus der eingebundenen Wortliste und ersetzt das Wort demnach im Zieltext.

Abbildung 6. Code-Einblick Umsetzung „pyspellchecker“

```
def spellcheck(text):
    correctedText = text
    # Split sentence in list of words
    words = spell.split_words(text)
    # Spellchecker detects misspelled words
    misspelled = spell.unknown(words)
    # Iterate over misspelled words
    for word in misspelled:
        cor = spell.correction(word)
        replaced = correctedText.replace(word, cor)
        correctedText = replaced
```

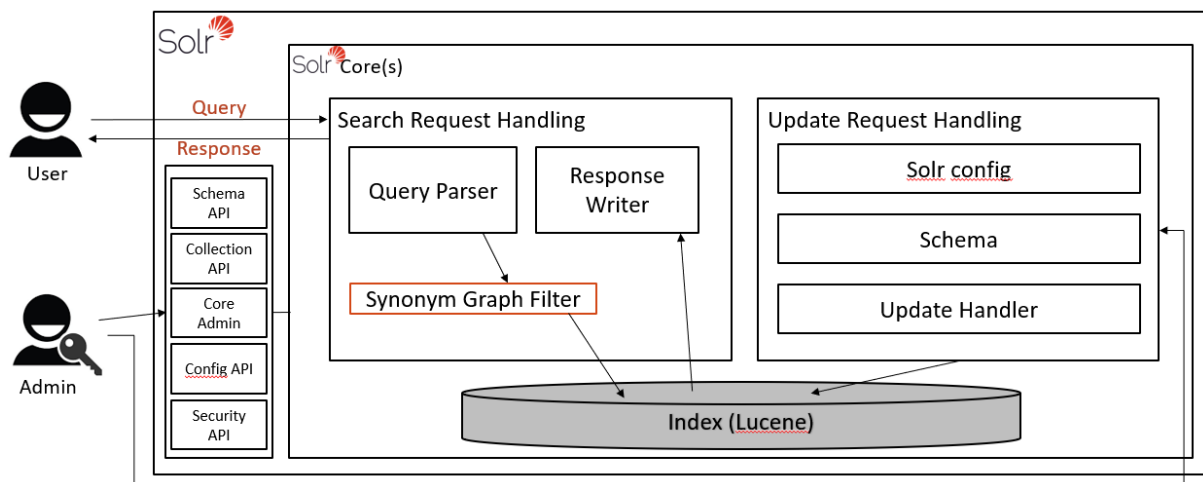
Für die Auflösung der fachspezifischen Abkürzungen wird die Methode „`replaceShortcuts(String)`“ genutzt. Hier sind alle Fehlernummern der vorliegenden Daten mitsamt der zugehörigen Substitution als „ErrorNumbers“ aufgelistet. Zudem sind alle beschriebenen Abkürzungen, deren Bedeutung ohne Auflösung unklar wäre, unter „Abbreviations“ beschrieben, wie in Abbildung 7 zu sehen. Hier werden zur besseren Veranschaulichung nur die ersten sieben domänen-spezifischen Abkürzungen angezeigt, die mit Hilfe von regulären Ausdrücken geschildert werden.

Abbildung 7. Code-Einblick Auflösung der fachspezifischen Abkürzungen

```
def replaceShortCuts(textWithShortCuts):
    replacedText = textWithShortCuts
    # abbreviations and error numbers to be replaced
    shortcuts = {"ErrorNumbers": [
        {"shortcut": "(F|f)ehler(nummer)?\\s?:?\\s?A125", "substitution": "Fehlernummer: A125 (Keine Kommunikation zum Roboter)"},
        {"shortcut": "(F|f)ehler\\s?:?\\s?A125", "substitution": "Fehler: A125 (Keine Kommunikation zum Roboter)"},
        {"shortcut": "(F|f)ehler(nummer)?\\s?:?\\s?E512", "substitution": "Fehlernummer: E512 (Interne Spannungsversorgung Ext.1 Ext.2)"},
        {"shortcut": "((F|f)ehler\\s?:?\\s?E512)", "substitution": "Fehler: E512 (Interne Spannungsversorgung Ext.1 Ext.2)"},
        {"shortcut": "CMXR\\s?STÖRUNG\\s", "substitution": "CMXR Störung (Störung von Festo Achsensteuerung)"}
    ],
    "Abbreviations": [
        {"shortcut": "(R|r)obi\\s", "substitution": "Roboter"},
        {"shortcut": "(S|s)ervice\\sDoc\\.\\s", "substitution": "Service Documentation"},
        {"shortcut": "(S|s)PS\\s", "substitution": "SPS (Speicherprogrammierbare Steuerung)"},
        {"shortcut": "(F|f)CT\\s", "substitution": "FCT (Festo Configuration Tool)"},
        {"shortcut": "(S|s)W-(E|e)ndscha\\s", "substitution": "Software-Endschalter"},
        {"shortcut": "(R|r)T\\s", "substitution": "RT (Rundschtaltisch)"},
        {"shortcut": "(I|i)ni\\s", "substitution": "Initiator"}
    ]
    return replacedText
```

4.3 Suche nach semantisch gleichen Einträgen

Abbildung 8. Architektur von SolR [42]



In der Abbildung 8 kann man die Architektur von SolR sehen, die wie folgt beschrieben wird: Nachdem ein User auf der Weboberfläche von SolR eine Suchanfrage gestellt hat werden zunächst alle Cores durchlaufen. Darunter werden zwei Handler unterschieden. Zum einen der Search Request Handler und zum anderen der Update Request Handler. Zunächst gelangt die Suchanfrage in den Search Request Handler. Dort verarbeitet der Query Parser die Anfrage und bereitet die Daten vor. Hier werden die Filter, die im Mapping von SolR festgelegt sind, beispielsweise Tokenizer, Stoppwortentferner, Lowercase parser, Stemmer usw., bei der Suchanfrage angewandt, um den benötigten Speicherplatz zu verringern und die Suche insgesamt zu beschleunigen.

Der Synonym-Graph-Filter ermöglicht dem User bei der Suchanfrage, eine Suche nach semantisch gleichen Einträgen. Hier werden die Synonyme und Abkürzungen dem Index abgebildet und nach allen semantisch ähnlichen Einträgen in der Datei, die dem Index unterliegt, gesucht.

Nachdem die Einträge im Index gefunden worden sind, werden diese dem Response Writer übergeben. Dieser verarbeitet gefundenen Daten und verpackt sie in einem von dem User ausgewählten Format. Diese werden dann in der Web-Oberfläche abgebildet.

Als Admin (in unserem Fall) kann man Einstellungen an der SolR-Konfiguration vornehmen. Diese Einstellungen laufen über den Update Request Handler. Da er bestimmte Änderungen aktualisiert,

muss dieser auch über den Index laufen, damit die nächste Suchanfrage auch mit der aktualisierten Konfiguration funktioniert. In der SolR-Konfiguration kann man beispielsweise weitere Filtermöglichkeiten hinzufügen oder neue Thesauri oder eine Stoppwortliste hinzufügen.

5 Evaluation

5.1 Ergebnisse

5.1.1 Textqualitätsverbesserung

Um die Qualität der bereits durchgeführten Textqualitätsverbesserung in Form der Rechtschreibkorrektur mit dem „pyspellchecker“ sowie der Auflösung der Abkürzungen angemessen bewerten zu können, wird das „Precision and Recall“-Bewertungsverfahren verwendet [43]. Hierfür wurden zunächst zufällig 20 Einträge aus den ursprünglichen Rohdaten in eine spezielle Samples-Datei herauskopiert und von Hand auf enthaltene Fehler analysiert. Insgesamt waren 18 Fehler vorhanden, davon 6 Rechtschreibfehler, 2 grammatikalische Fehler und 10 vorkommende Abkürzungen (Siehe Abbildung 9).

Anschließend wird das Python-Programm, welches bereits vorgestellt wurde, über die neue Samples-Datei gestartet, sodass sowohl die Auflösung der vorhandenen fachspezifischen Abkürzungen als auch die Rechtschreibkorrektur durchgeführt und ausgewertet werden können. Das Programm findet insgesamt 20 Fehler, davon 11 Rechtschreibfehler und 9 Abkürzungen. Daraus lässt sich nun die Anzahl der gefundenen relevanten Fehler (True Positive - TP), der nicht gefundenen relevanten Fehler (False Negative - FN), der gefundenen nicht relevanten Fehler (False Positive - FP) und der nicht gefundenen und zugleich nicht relevanten Fehler (True Negative - TN) bestimmen. In Abbildung 10 sind diese Werte dargestellt. Anhand dieser Werte lassen sich nun die Bewertungsparameter Precision, Recall und Accuracy berechnen. Die Precision, Recall und die Accuracy lassen der Korrektur lassen sich über die Formel folgenden Formeln berechnen:

Formel 2. Formel zur Berechnung des Bewertungsparameters Precision

$$Precision = \frac{TP}{TP + FP}$$

Formel 3. Formel zur Berechnung des Bewertungsparameters Recall

$$Recall = \frac{TP}{TP + FN}$$

Formel 4. Formel zur Berechnung des Bewertungsparameters Accuracy

$$Accuracy = \frac{TP + TN}{\text{Gesamte Wörter}}$$

Somit ergeben sich für die Bewertung der selbst implementierten Rechtschreibkorrektur folgende Ergebnisse: Die Precision der Rechtschreibkorrektur beläuft sich auf 0.75, der Recall liegt bei 0.8334 und die Accuracy bei 0.9839.

Abbildung 9. Rechtschreibfehler und Abkürzungen der Samples Datei

Samples Datei enthält folgende Fehler:

Zeile 2: resetet → resettet & 2x SPS → Speicherprogrammierbare Steuerung & das → dass

Zeile 3: Zuordnung → Zuordnung

Zeile 6: seien → sein

Zeile 8: GS → Grundstellung & popsitiv → positiv

Zeile 9: QS → Qualitätssicherung

Zeile 14: Grungstellung → Grundstellung

Zeile 16: SPS → Speicherprogrammierbare Steuerung

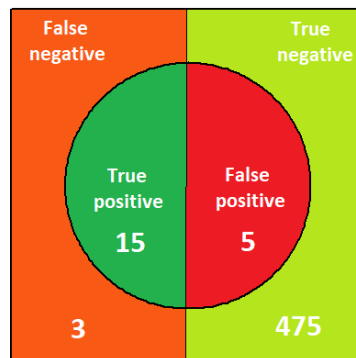
Zeile 17: nachgestellt → nachgestellt & abgenutzt → abgenutzt

Zeile 18: FCT → Festo Configuration Tool

Zeile 19: 4x Robi → Roboter

→ Insgesamt 18 Fehler gefunden, davon 6 Rechtschreibfehler, 2 grammatikalischer Fehler und 10 Abkürzungen aufgelöst

Abbildung 10. FN, TN, TP, FP der Funde aus der Samples Datei



5.1.2 Semantische Suche

Um die in SolR implementierte Suche nach semantisch gleichen Einträgen angemessen beurteilen zu können, wurde mit Hilfe eines zweiten Indexes in SolR direkt ein Vergleich der Ergebnisse der erweiterten Query-Expansion und einer einfachen Schlüsselwortsuche durchgeführt. Hierfür wurden unterschiedliche Suchbegriffe sowie ganze Sätze zunächst mit Einbindung der erstellten Synonymdatei in Hinblick auf die spezifische Domäne analysiert, um die gefundenen Ergebnisse anschließend direkt mit den Ergebnissen der einfachen Schlüsselwortsuche zu vergleichen. Im Folgenden werden nun drei unterschiedliche Beispiele aufgezeigt, anhand derer eine Analyse der Qualität der implementierten Query-Expansion durchgeführt und beschrieben wird.

Beispiel 1: Ausgangssituation:"Einpresskraft zu niedrig"

Im ersten Beispiel werden alle Ausgangssituationen gesucht, in denen die „Einpresskraft zu niedrig“ war. Hierbei sollen die Synonyme „zu niedrig/unterschritten“ erkannt werden und damit alle relevanten Ergebnisse für diese Suchanfrage gelistet werden. Die gestellte Suchanfrage hierfür lautet folgendermaßen:

Ausgangssituation:"Einpresskraft zu niedrig"

Die in SolR implementierte Query-Expansion erkennt hierbei alle fünf zu findenden Suchergebnisse, während die einfache Schlüsselwortsuche nur die zwei exakt korrekt geschriebenen Ergebnisse findet.

Folgende fünf Einträge sollen gefunden werden:

Tabelle 13. Beispiel 1: Soll-Werte suche nach semantisch gleichen Einträgen

Ausgangssituation	Handlung
Einpresskraft unterschritten	Zeichnungsänderung auf 45N--> Rezeptur angepasst
Patronen Einpresskraft UTG unterschritten	Fehler trat nach umrüsten von T.Nr. 736435 auf 1417753 auf. Einpresskraft 14er Seite wird unterschritten unter 50 Nm. Die 3te Patrone zeigt immer mal wieder 0,0 Nm an, das passiert bei ca. 50 % der Teile. Bereits in der Tagschicht wurde bei T.Nr.736435 etwas geändert. Problem muss in der Tagschicht geklärt werden. Anlage wurde erneut umgerüstet ,damit die Anlage läuft.
Einpresskraft unterschritten	Zeichnung auf UTG 45N geändert--> Rezepturen angepasst
Einpresskraft zu niedrig	Einpressgeschwindigkeit von 50 auf 40 geändert (Rezeptur)
Einpresskraft zu niedrig	SF:Toleranz kurzfristig auf 50N Einpresskräfte reduziert, Prozess validiert. Bei Teilenummern:1069756,1000500,1000504,1000506,1000507,757626

Ergebnismenge in SolR auf die Suchanfrage mit Hilfe der Query-Expansion:

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 76,
    "params": {
      "q": "Ausgangssituation:\"Einpresskraft zu niedrig\"",
      "_": "1571668456705"}},
}
```

```

"response": {"numFound": 5, "start": 0, "docs": [
  {
    "Arbeitsplatzbezeichnung": ["XYXH10 (Gesamtanlage)>>>"],
    "Art": ["Rezeptur"],
    "Ausgangssituation": ["Einpresskraft unterschritten"],
    "Handlung": ["Zeichnungsänderung auf 45N--> Rezeptur angepasst"],
    "Created": ["2018-11-21T08:15:02Z"],
    "id": "878d2ab7-6603-4e47-ab57-1b21147aa6af",
    "_version_": 1647471915629019137,
  },
  {
    "Arbeitsplatzbezeichnung": ["XYXH 140 (Gesamtanlage)>>>"],
    "Art": ["Rezeptur"],
    "Ausgangssituation": ["Einpresskraft unterschritten"],
    "Handlung": ["Zeichnung auf UTG 45N geändert--> Rezepturen angepasst"],
    "Created": ["2018-11-21T08:14:09Z"],
    "id": "c1ba495a-473e-4a69-a3cd-5892c3435bad",
    "_version_": 1647471915842928640,
  },
  {
    "Arbeitsplatzbezeichnung": ["XYXH 140 (Gesamtanlage)>>>"],
    "Art": ["Rezeptur"],
    "Ausgangssituation": ["Einpresskraft zu niedrig"],
    "Handlung": ["Einpressgeschwindigkeit von 50 auf 40 geändert (Rezeptur)"],
    "Created": ["2018-11-22T10:30:44Z"],
    "id": "2e731827-32fb-44e6-9f13-8189462dbd52",
    "_version_": 1647471915842928641,
  },
  {
    "Arbeitsplatzbezeichnung": ["XYXH 140 Zelle 2 Patroneneinpressen"],
    "Art": ["Einstellungen"],
    "Ausgangssituation": ["Einpresskraft zu niedrig"],
    "Handlung": ["SF:Toleranz kurzfristig auf 50N Einpresskräfte reduziert, Prozess_x000D_\r\nvalidiert._x000D_\r\nBei Teilenummern:1069756,1000500,1000504,1000506,1000507,757626"],
    "Created": ["2018-07-30T11:21:13Z"],
    "id": "0f734928-bfff-4efb-96aa-7ae37d96d4cb",
    "_version_": 1647471915966660608,
  },
  {
    "Arbeitsplatzbezeichnung": ["XYXH10 Zelle 2 Patroneneinpressen"],
    "Art": ["Mechanisch"],
    "Ausgangssituation": ["Patronen Einpresskraft UTG unterschritten"],
    "Handlung": ["Fehler trat nach umrüsten von T.Nr. 736435 auf 1417753 auf._x000D_\r\nEinpresskraft 14er Seite wird unterschritten unter 50 Nm._x000D_\r\nDie 3te Patrone zeigt immer mal wieder 0,0 Nm an, das passiert_x000D_\r\nbei ca. 50 % der Teile._x000D_\r\nBereits in der Tagschicht wurde bei T.Nr.736435 etwas geändert._x000D_\r\nProblem muss in der Tagschicht geklärt werden._x000D_\r\nAnlage wurde erneut umgerüstet ,damit die Anlage läuft"],
    "Created": ["2017-12-14T22:13:10Z"],
    "id": "42d40c9f-b834-4d58-b579-4651abe3c462",
    "_version_": 1647471915710808064,
  }
]}

```

Ergebnismenge in SolR auf die Suchanfrage mit Hilfe der einfachen Schlüsselwortsuche:

```

{
  "responseHeader": {
    "status": 0,
    "QTime": 11,
    "params": {
      "q": "Ausgangssituation:\\"Einpresskraft zu niedrig\\"",
      "_": "1571668456705"
    }
  },
  "response": { "numFound": 2, "start": 0, "docs": [
    {
      "Arbeitsplatzbezeichnung": ["XYXH 140 (Gesamtanlage)>>>"],
      "Art": ["Rezeptur"],
      "Ausgangssituation": ["Einpresskraft zu niedrig"],
      "Handlung": ["Einpressgeschwindigkeit von 50 auf 40 geändert (Rezeptur)"],
      "Created": ["2018-11-22T10:30:44Z"],
      "id": "2b43aa16-f5be-42c0-bafc-afb6a0632193",
      "_version_": 1647473971258458114,
    }
  ]
}

```

```

    "Arbeitsplatzbezeichnung":["XYXH 140 Zelle 2 Patroneneinpressen"],
    "Art":["Einstellungen"],
    "Ausgangssituation":["Einpresskraft zu niedrig"],
    "Handlung":["SF:Toleranz kurzfristig auf 50N Einpresskräfte reduziert, Prozess_x000D_\r\n
validiert._x000D_\r\nBei Teilenummern:1069756,1000500,1000504,1000506,1000507,757626"],
    "Created":["2018-07-30T11:21:13Z"],
    "id":["4fdb2360-3afe-4d07-ad7c-050164e83037"],
    "_version_":1647473971317178371}}
  }}

```

Beispiel 2: Ausgangssituation:"zu hohe Schaltzeit"

Im zweiten Beispiel werden alle Ausgangssituationen gesucht, in denen eine „zu hohe Schaltzeit“ vorkommt. Hierbei sollen die Synonyme „hohe Schaltzeit/Schaltzeit zu schnell“ erkannt werden. Die gestellte Suchanfrage hierfür lautet folgendermaßen:

Ausgangssituation:“zu hohe Schaltzeit“

SolR implementierte Query-Expansion erkennt hierbei alle drei zu findenden Suchergebnisse, während die einfache Schlüsselwortsuche nur ein Ergebnis findet.

Folgende drei Einträge sollen gefunden werden:

Tabelle 14. Beispiel 2: Soll-Werte suche nach semantisch gleichen Einträgen

Ausgangssituation	Handlung
Testlauf Schaltzeit zu schnell St. 40	"Regler und Druckaufnehmer nachkalibriert. Leckage an einer Steckverbindung behoben. Schalldämpfer fehlten am Fettauffangbehälter. Diese eingeschraubt"
Testlauf an der PZ ST.50 geht nicht durch Schaltzeit zu schnell bei T.Nr. 1000500	"Drücke überprüft —> in Ordnung. Mit neuen TV geht der Testlauf ohne Probleme durch. Werte werden von Der Prüftechnik überprüft. "
An Station 40 zu hohe Schaltzeit.	"Ventilstecker an Ventil Ky462.2 festgeschraubt, Schlauch an Ausgang 4 richtig eingesteckt. Testlauf in Ordnung. Schaltzeit genau in der Mitte (6s). "

Ergebnismenge in SolR auf die Suchanfrage mit Hilfe der Query-Expansion:

```
{  "responseHeader":{
    "status":0,
    "QTime":3,
    "params":{
        "q":"Ausgangssituation:\\"zu hohe Schaltzeit\\"",
        "_":"1571669016593"}},
  "response":{"numFound":3,"start":0,"docs":[
    {
      "Arbeitsplatzbezeichnung":["XYXH10 Prüfzelle"],
      "Art":["Elektrisch"],
      "Ausgangssituation":["An Station 40 zu hohe Schaltzeit. "],
      "Handlung":["Ventilstecker an Ventil Ky462.2 festgeschraubt, Schlauch an Ausgang 4 richtig
reingesteckt. _x000D_\r\nTestlauf in Ordnung. Schaltzeit genau in der Mitte (6s). "],
      "Created":["2017-09-11T10:46:38Z"],
      "id":["504a4742-9e81-49c5-a259-c88755418945"],
      "_version_":1647471915632164865},
    {
      "Arbeitsplatzbezeichnung":["XYXH 140 Prüfzelle"],
      "Art":["Mechanisch"],
      "Ausgangssituation":["Testlauf Schaltzeit zu schnell St. 40"],
      "Handlung":["Regler und Druckaufnehmer nachkalibriert. _x000D_\r\nLeckage an einer Steckverbindung
behoben. _x000D_\r\nSchalldämpfer fehlten am Fettauffangbehälter. _x000D_\r\nDiese eingeschraubt"],
      "Created":["2017-12-19T13:31:01Z"],
      "id":["8b361179-b69a-445a-8ccb-38036d177591"],
      "_version_":1647471915861803010},
    {
      "Arbeitsplatzbezeichnung":["XYXH 140 Prüfzelle>>>"],
      "Art":["Elektrisch"],
      "Ausgangssituation":["Testlauf an der PZ ST.50 geht nicht durch Schaltzeit zu schnell bei T.Nr.
1000500 _x000D_\r\n"],
      "Handlung":["Drücke überprüft -> in Ordnung. _x000D_\r\nMit neuen TV geht der Testlauf ohne Probleme
durch. _x000D_\r\nWerte werden von Der Prüftechnik überprüft. "],
      "Created":["2019-03-21T08:52:51Z"],
      "id":["14cd2f9d-1b76-459f-8f3c-8b3d6ff1f103"],
      "_version_":1647471915917377536}]
  ]}
```

Ergebnismenge in SolR auf die Suchanfrage mit Hilfe der einfachen Schlüsselwortsuche:

```
{
  "responseHeader":{
    "status":0,
    "QTime":1,
    "params":{
        "q":"Ausgangssituation:\\"zu hohe Schaltzeit\\"",
        "_":"1571668456705"}},
  "response":{"numFound":1,"start":0,"docs":[
    {
      "Arbeitsplatzbezeichnung":["XYXH10 Prüfzelle"],
      "Art":["Elektrisch"],
      "Ausgangssituation":["An Station 40 zu hohe Schaltzeit. "],
      "Handlung":["Ventilstecker an Ventil Ky462.2 festgeschraubt, Schlauch an Ausgang 4 richtig
reingesteckt. _x000D_\r\nTestlauf in Ordnung. Schaltzeit genau in der Mitte (6s). "],
      "Created":["2017-09-11T10:46:38Z"],
      "id":["9efed5fa-a0c2-485f-bf03-6ddae0b2435a"],
      "_version_":1647473971153600512}]
  ]}
```

Beispiel 3: Ausgangssituation:"WLG nicht ready"

Im letzten Beispiel werden alle Ausgangssituationen gesucht, in denen das „WLG nicht ready“ war. Hierbei sollen die Synonyme „nicht ready/nicht bereit/kein betriebsbereit“ erkannt werden. Die gestellte Suchanfrage hierfür lautet folgendermaßen:

Ausgangssituation: "WLG nicht ready"

Die in SolR implementierte Query-Expansion erkennt hierbei alle drei zu findenden Suchergebnisse, während die einfache Schlüsselwortsuche nur ein Ergebnis findet.

Folgende vier Einträge sollen gefunden werden:

Ausgangssituation	Handlung
WLG 1 nicht bereit.	WLG1 (am hinteren Schaltschrank) neu gestartet. -> Testlauf i.O.
WLG meldet kein Betriebsbereit	WLG ausgetauscht und Daten von Sicherung geladen
- Greifer 2 Kolben geht nicht auf - Zusatzheizung zeigt zu hohe Temperatur - WLG 1 kein Betriebsbereit	"- an Greifer 2 Kolben Schläuche richtig angeschlossen, waren nach einen Tausch des Zylinders nicht richtig angeschlossen - Zusatzheizung 2 am Ring zu hohe Temperatur, RTD aus Steuerung gezogen und wieder eingesteckt - WLG 1 24V abgezogen um neu zu starten"
WLG1 nicht ready	Am Walther Grenzwertmodul leuchten die AnzeigeLED vom Grenzdruck nicht. Spannungsversorgung am Modul =Z3=P2-AF1 aus und wieder eingesteckt

Ergebnismenge in SolR auf die Suchanfrage mit Hilfe der Query-Expansion:

```
{ "responseHeader": {
  "status": 0,
  "QTime": 0,
  "params": {
    "q": "Ausgangssituation: \" nicht ready\"",
    " _": "1571669016593"
  },
  "response": { "numFound": 4, "start": 0, "docs": [
    {
      "Arbeitsplatzbezeichnung": ["XYXH 140 Zelle 3 Kolben"],
      "Art": ["Elektrisch"],
      "Ausgangssituation": ["WLG1 nicht ready"],
      "Handlung": ["Am Walther Grenzwertmodul leuchten die Anzeige-LED vom Grenzdruck nicht._x000D_\r\nSpannungsversorgung am Modul =Z3=P2-AF1 aus und wieder eingesteckt"],
      "Created": ["2017-11-24T05:43:02Z"],
      "id": "b4bbfa81-6212-4a05-997f-dacbc72f05c8",
      "_version_": 1647471915996020736,
    },
    {
      "Arbeitsplatzbezeichnung": ["XYXH 140 Zelle 2 Patroneneinpressen>>>"],
      "Art": ["Elektrisch"],
      "Ausgangssituation": ["WLG meldet kein Betriebsbereit"],
      "Handlung": ["WLG ausgetauscht und Daten von Sicherung geladen"],
      "Created": ["2019-03-29T06:16:42Z"],
      "id": "50c8ab56-3a19-4999-a6a1-e6771e53827c",
      "_version_": 1647471915989729281,
    },
  ]
}
```

```
{
  "Arbeitsplatzbezeichnung":["XYXH 100 Zelle 3 Kolben"],
  "Art":["Elektrisch"],
  "Ausgangssituation":["WLG 1 nicht bereit_x000D_\r\n"],
  "Handlung":["WLG1 (am hinteren Schaltschrank) neu gestartet_x000D_\r\n-> Testlauf in
  Ordnung_x000D_\r\n"],
  "Created":["2017-06-12T05:45:04Z"],
  "id":["3a0cd95f-eddd-478f-b704-dbcc144ebd36"],
  "_version_":1647471915582881793},
{
  "Arbeitsplatzbezeichnung":["XYXH 140 Zelle 3 Kolben"],
  "Art":["Elektrisch"],
  "Ausgangssituation":["- Greifer 2 Kolben geht nicht auf_x000D_\r\n- Zusatzheizung zeigt zu hohe
  Temperatur_x000D_\r\n- WLG 1 kein Betriebsbereit"],
  "Handlung":["- an Greifer 2 Kolben Schläuche richtig angeschlossen, waren nach einen Tausch des
  Zylinders nicht richtig angeschlossen_x000D_\r\n- Zusatzheizung 2 am Ring zu hohe Temperatur, RTD
  aus Steuerung gezogen und wieder eingesteckt_x000D_\r\n- WLG 1 24V abgezogen um neu zu starten"],
  "Created":["2017-09-13T12:36:11Z"],
  "id":["a995865f-efaa-4bc8-b91c-e46094d298bc"],
  "_version_":1647471915993923584]}
}}
```

Ergebnismenge in SolR auf die Suchanfrage mit Hilfe der einfache Schlüsselwortsuche:

```
{ "responseHeader":{
  "status":0,
  "QTime":0,
  "params":{
    "q":"Ausgangssituation:\n\nnicht ready\n",
    "_":"1571668456705"}},
  "response":{"numFound":1,"start":0,"docs":[
    {
      "Arbeitsplatzbezeichnung":["XYXH 140 Zelle 3 Kolben"],
      "Art":["Elektrisch"],
      "Ausgangssituation":["WLG1 nicht ready"],
      "Handlung":["Am Walther Grenzwertmodul leuchten die Anzeige-LED vom Grenzdruck
      nicht_x000D_\r\nSpannungsversorgung am Modul =Z3=P2-AF1 aus und wieder eingesteckt"],
      "Created":["2017-11-24T05:43:02Z"],
      "id":["dc0b25e7-71e6-4b7b-88a9-b37fd6eda75f"],
      "_version_":1647473971326615556}]
  }}
```

5.2 Bewertung

Zusammenfassend lässt sich sagen, dass sowohl bei der Textqualitätsverbesserung als auch bei der Implementation der Suche nach semantisch gleichen Ergebnissen hervorragende Ergebnisse erzielt wurden.

Bei der Textqualitätsverbesserung, in dem vorliegenden Datensatz liegt der Fokus primär darin, die Textqualität bestmöglich zu erhöhen, indem möglichst alle vorhandenen Fehler erkannt und ersetzt werden. Zudem sollen die zahlreichen fachspezifischen Abkürzungen und Fehlernummern erkannt und aufgelöst werden, um den korrigierten Datensatz anschließend als Wissensbasis für die Suche nach semantisch gleichen Einträgen verwenden zu können. Hierfür ist der Wert des Recalls relevant, insgesamt wurden 15 von 18 möglichen Fehlern erkannt. Von den verbleibenden drei Fehlern waren zudem zwei grammatikalische Fehler, die somit vom "pyspellchecker" ohne hinzugehörigen Gramarchecker nicht erkannt werden müssen. Daraus resultiert ein sehr guter Wert von 0.8334 für die Precision, der Recall ist hierbei zu vernachlässigen, da in vier der fünf Fälle der vermeintlich gefundene durch dasselbe Wort ersetzt wurde und somit keine Korrektur stattfand. Weshalb das Programm das Wort trotzdem als Fehler erkannte, konnte nicht festgestellt werden. Der äußerst hohe Wert der Accuracy (0.9839) bestätigt die These, dass die genutzte Rechtschreibkorrektur überdurchschnittlich gute Ergebnisse erzielen konnte.

Im Falle der effizienten Suche mit Hilfe der Query-Expansion, welche die eingebundene fachspezifische Liste an Synonymen verarbeitet, ist deutlich zu erkennen, dass das Ziel, alle semantisch gleichen, relevanten Einträge auf eine Suchanfrage zu erhalten erreicht wurde. Es wurden jeweils alle Einträge des digitalen Logbuches gefunden, welche den Soll-Werten entsprachen. Im Kontrast hierzu die Ergebnisse der einfachen Schlüsselwortsuche. Diese hat sich für diesen Anwendungsfall wie vorhergesagt als äußerst ineffizient erwiesen, da die Ergebnismenge, welche gefunden wurde nicht vollständig den Soll-Werten entspricht. Die Suche mit dem Suchverfahren der Query-Expansion bringt somit im Vergleich zur einfachen Schlüsselwortsuche einen echten Mehrwert mit sich und definierte eine präzisere und effizientere Ergebnismenge.

Anhand der hervorragenden Ergebnismenge der Evaluation der Textqualitätsverbesserung und der Suche nach semantisch gleichen Einträgen mit Hilfe der Query-Expansion, lässt sich sagen, dass das Ziel der Fachstudie erreicht wurde. Die das digitale Logbuch, welche eine deutlich bessere Textqualität besitzt als zuvor, kann nun als effiziente Wissensbasis für die Werker dienen. Alle relevanten Logbucheinträge werden auf eine bestimmte Suche gefunden, was unter anderem ausschließt, dass relevante Informationen für einen vorliegenden Fehlerfall verloren gehen, aber auch die Fehler- und Ursachenbekämpfung erleichtern und verschnellern kann.

6 Zusammenfassung und Ausblick

Zunächst bestand die Aufgabe darin, die Textqualität der vorliegenden digitalen Logbucheinträge soweit zu verbessern, dass die korrigierte Version als Wissensbasis für eine anschließende Suche nach semantisch gleichen Einträgen verwendet werden kann. Hierfür wird ein modifiziertes Python-Programm basierend auf dem "pyspellchecker" genutzt, das zum einen eine Rechtschreibkorrektur über alle Wörter des Datensatzes mit Hilfe einer fachspezifischen selbst definierten Wortliste durchführt und zum anderen alle Fachabkürzungen dieser Domäne auflöst. Nachdem die Wissensbasis als Grundlage für die weitere Suche aufgebaut und korrigiert wurde, kann diese nun effizient genutzt werden. Da eine effiziente Suche mit Hilfe der einfachen Schlüsselwortsuche auf dieser Wissensbasis, mit vielen semantisch gleichen Einträgen nicht möglich ist, wurde nach weiteren, effizienteren Suchverfahren recherchiert. Die fiel auf die Query Expansion mit der Einbindung von Synonymen der fachspezifischen Domäne. Diese Idee wurde mit Hilfe von SolR umgesetzt.

Mit Hilfe dieses Konzeptes und der praktischen Umsetzung in SolR ist es nun für die Werksarbeiter möglich, Lösungsvorschläge für bestehende bereits gelöste Probleme wiederzuverwenden und damit effektiv Zeit einzusparen, die sonst in die oftmals lang andauernde händische Suche fallen würde. Auch das Wissen erfahrener Mitarbeiter kann hierdurch rekonstruiert und erneut genutzt werden, um Probleme an Maschinen und Geräten schneller zu beheben. Das beschriebene Konzept bietet somit eine effiziente Nutzung der Wissensbasis und eine daraus resultierende hohe Zeit- und Kosteneinsparung.

Um das erläuterte Prinzip benutzerfreundlicher zu gestalten und den Mitarbeitern eine übersichtliche Oberfläche zur Verfügung zu stellen, könnte die Erstellung und Nutzung einer Client-Anwendung hilfreich sein. Da sowohl SolR als auch Elasticsearch über REST-Anfragen angesprochen werden können, sollte die Applikation diese unterstützen und sinnvoll einbinden, sodass die Suche effizient genutzt werden kann. Außerdem wäre es aufgrund der sehr spezifischen Domäne von Nutzen, wenn große fachspezifische Wörterbücher vom Fachbereich selbst zur Verfügung gestellt werden, um damit zunächst die Textqualitätsverbesserung zu ermöglichen, aber auch um die vielen Fachbegriffe korrekt auflösen zu können. Leider werden bisher zahlreiche Wörter nicht richtig erkannt, da sie nur in einer speziellen Branche vorkommen und dadurch in den gängigen Wortlisten nicht enthalten sind. Zudem gibt es zwar durchaus viele englischsprachige Wortlisten, für die deutsche Sprache jedoch sind jene knapp bemessen. Mit Hilfe ausführlicher fachspezifischer Wörterbücher für den deutschen Sprachgebrauch könnte somit der Großteil der Textqualitätsverbesserung schneller durchgeführt werden, was demnach auch die Implementierung vereinfachen und den Gesamtaufwand erheblich verringern würde.

7 Literaturverzeichnis

- [1] Y. Wilhelm, *Verbesserung der Textqualität in Digitalen Logbucheinträgen in der Instandhaltung*, April 2019.
- [2] Rechtschreibprüfung, „Wikipedia,“ [Online]. Available: <https://de.wikipedia.org/wiki/Rechtschreibpr%C3%BCfung>. [Zugriff am 20 Oktober 2019].
- [3] i. T. F. Levenshtein Distance, „by Michael Gilleland, Merriam Park Software,“ [Online]. Available: <https://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring2006/assignments/editdistance/Levenshtein%20Distance.htm>. [Zugriff am 20 Oktober 2019].
- [4] Stemming, „netzmarketing,“ [Online]. Available: <https://www.netzmarketing.ch/gratis-ratgeber/glossar/stemming>. [Zugriff am 20 Oktober 2019].
- [5] a. 0.4.0, „pypi,“ Python Software Foundation, 2019. [Online]. Available: <https://pypi.org/project/abydos/>. [Zugriff am 20 Oktober 2019].
- [6] p. 0.5.2, „pypi,“ Python Software Foundation, 2019. [Online]. Available: <https://pypi.org/project/pyspellchecker/>. [Zugriff am 20 Oktober 2019].
- [7] p. 0.8.3, „pypi,“ Python Software Foundation, 2019. [Online]. Available: <https://pypi.org/project/pyLanguagetool/>. [Zugriff am 20 Oktober 2019].
- [8] g.-c. 1.3.1, „pypi,“ Python Software Foundation, 2019. [Online]. Available: <https://pypi.org/project/grammar-check/>. [Zugriff am 20 Oktober 2019].
- [9] l.-c. 1.1, „pypi,“ Python Software Foundation, 2019. [Online]. Available: <https://pypi.org/project/language-check/>. [Zugriff am 20 Oktober 2019].
- [10] c. 0.4.1, „pypi,“ Python Software Foundation, 2019. [Online]. Available: <https://pypi.org/project/cwsplit/>. [Zugriff am 20 Oktober 2019].
- [11] A. D. Hiteshwar Kumar Azad, „Query Expansion Techniques for Information Retrieval: a Survey,“ Elsevier Inc, Cornell University, 2017.
- [12] Q. expansion, „Wikipedia,“ 5 April 2019. [Online]. Available: https://en.wikipedia.org/wiki/Query_expansion. [Zugriff am 21 Oktober 2019].
- [13] Q. Expansion, „nlp.stanford.edu,“ Cambridge University Press, 2008. [Online]. Available: <https://nlp.stanford.edu/IR-book/html/htmledition/query-expansion-1.html>. [Zugriff am 21 Oktober 2019].
- [14] F. K. H. a. E. C. H. Dong, „A survey in semantic search technologies,“ in *2nd IEEE International Conference on Digital Ecosystems and Technologies*, Phitsanuloke, Thailand, Februar 2008, pp. 403-408.

- [15] „Semantische Suche,“ Wikipedia, [Online]. Available: https://de.wikipedia.org/wiki/Semantische_Suche. [Zugriff am 18 Oktober 2019].
- [16] P. M. B. A. B. Wang Wei, „Search with Meanings: An Overview of Semantic Search Systems,“ School of Computer Science, University of Nottingham Malaysia Campus Jalan Broga, 43500 Semenyih, Selangor, Malaysia, Selan, Januar 2008.
- [17] B. B. a. E. H. H. Bast, „Semantic Search on Text and Knowledge Bases,“ in *FNT in Information Retrival*, 2016, pp. 119-271.
- [18] „Semantic Search,“ RYTE, 2019. [Online]. Available: https://de.ryte.com/wiki/Semantic_Search. [Zugriff am 18 Oktober 2019].
- [19] „Opensearch Search Engine Modules and Architecture,“ [Online]. Available: <https://www.opensearch.org/doc/modules>. [Zugriff am 10 9 2019].
- [20] M. Aslo, „Volltextsuche mit Elasticsearch im Geodaten-Umfeld,“ Hochschule für angewandte Wissenschaften, München, 09.April 2018.
- [21] „Kibana,“ elastic, [Online]. Available: <https://www.elastic.co/de/products/kibana>. [Zugriff am 18 Oktober 2019].
- [22] „Elasticsearch: Die flexible Search Engine,“ Digital Guide IONOS, 26 September 2019. [Online]. Available: <https://www.ionos.de/digitalguide/server/konfiguration/elasticsearch/>. [Zugriff am 18 Oktober 2019].
- [23] M. Makadia, „What Is Elasticsearch? (And Why You Need to Be Using It),“ 17 Oktober 2018. [Online]. Available: <https://dzone.com/articles/what-is-elasticsearch-and-how-it-can-be-useful>. [Zugriff am 18 Oktober 2019].
- [24] K. Tan, „Solr Search Relevancy,“ SolrTutaoral, 2019. [Online]. Available: <http://www.solrtutorial.com/solr-search-relevancy.html>. [Zugriff am 18 Oktober 2019].
- [25] S. D. I. S.-S. v. Apache, „Digital Guide ionos,“ 13 März 2019. [Online]. Available: <https://www.ionos.de/digitalguide/server/konfiguration/solr/>. [Zugriff am 22 Oktober 2019].
- [26] W. i. SolR, „BigData Insider,“ Stefan Luber, Nico Litzel, 27 06 2018. [Online]. Available: <https://www.bigdata-insider.de/was-ist-solr-a-728279/>. [Zugriff am 22 Oktober 2019].
- [27] „Solr: Der leistungsstarke Such-Server von Apache,“ 17 Oktober 2019. [Online]. Available: <https://www.ionos.de/digitalguide/server/konfiguration/solr/>.
- [28] „Elasticsearch Inverted index(Analysis): How to create inverted index and how inverted index is stored in segments of Shards (Elasticsearch and Kibana Devtool),“ devinline , [Online]. Available: <http://www.devinline.com/2018/09/elasticsearch-inverted-index-and-its-storage.html>. [Zugriff am 17 Oktober 2019].
- [29] „Elasticsearch: Die flexible Search Engine,“ Digital Guide IONOS, 2019 September 2019. [Online]. Available:

<https://www.ionos.de/digitalguide/server/konfiguration/elasticsearch/>. [Zugriff am 18 Oktober 2019].

- [30] „Elasticsearch from the Bottom Up, Part 1,“ elastic, [Online]. Available: <https://www.elastic.co/de/blog/found-elasticsearch-from-the-bottom-up>. [Zugriff am 17 Oktober 2019].
- [31] „Indexing for Beginners, Part 3,“ elastic, [Online]. Available: <https://www.elastic.co/de/blog/found-indexing-for-beginners-part3>. [Zugriff am 17 Oktober 2019].
- [32] „Lower-, Upper- und Proper-Funktionen in PowerApps,“ microsoft, [Online]. Available: <https://docs.microsoft.com/de-de/powerapps/maker/canvas-apps/functions/function-lower-upper-proper>. [Zugriff am 17 Oktober 2019].
- [33] R. K. Johannes Lang, „Stemming,“ Universität Heidelberg, 2002.
- [34] W. K. John Marquiss, „Doning Synonyms Right,“ 12-15 September 2017. [Online]. Available: <https://de.slideshare.net/lucidworks/doing-synonyms-right-john-marquiss-wolters-kluwer>. [Zugriff am 18 Oktober 2019].
- [35] „OpenThesaurus.de - Synonyme und Assoziationen,“ openthesaurus, [Online]. Available: <https://www.openthesaurus.de/about/download>. [Zugriff am 18 Oktober 2019].
- [36] „Filter descriptions,“ SolR Ref Guide 6.6, [Online]. Available: https://lucene.apache.org/solr/guide/6_6/filter-descriptions.html. [Zugriff am 18 Oktober 2019].
- [37] M. McCandless, „Lucene's TokenStreams sind eigentlich Grafiken!“, 30 April 2012. [Online]. Available: <http://blog.mikemccandless.com/2012/04/lucenes-tokenstreams-are-actually.html>. [Zugriff am 18 Oktober 2019].
- [38] „How scoring works in Elasticsearch,“ COMPOSE, 18 Februar 2016. [Online]. Available: <https://www.compose.com/articles/how-scoring-works-in-elasticsearch/>. [Zugriff am 18 Oktober 2019].
- [39] „Class Similirty,“ Apache Lucene, 2000-2011. [Online]. Available: http://lucene.apache.org/core/3_5_0/api/core/org/apache/lucene/search/Similarity.html. [Zugriff am 18 Oktober 2019].
- [40] „Lucene's Practical Scoring Function,“ elastic, [Online]. Available: <https://www.elastic.co/guide/en/elasticsearch/guide/current/practical-scoring-function.html>. [Zugriff am 18 Oktober 2019].
- [41] A. S. u. E. i. Streitgespräch, „heise Developer,“ Markus Klose, Daniel Wrigley, 10 12 2013. [Online]. Available: <https://www.heise.de/developer/artikel/Apache-Solr-und-ElasticSearch-im-Streitgesprach-2062107.html?seite=all>. [Zugriff am 21 Oktober 2019].
- [42] V. m. L. u. Solr, „slideshare,“ Thomas Koch, 26 08 2012. [Online]. Available: <https://de.slideshare.net/tomykoch/lucene-solr-froscontko20120826>. [Zugriff am 21 Oktober 2019].

- [43] „Google Developers,“ 10 September 2019. [Online]. Available: <https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall>.
- [44] D. Stichwortsuche, „onpulson,“ onpulson.de - Das Business-Magazin für den Mittelstand, 2019. [Online]. Available: <https://www.onpulson.de/lexikon/stichwortsuche/>. [Zugriff am 21 Oktober 2019].

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Stuttgart, 23.10.2019

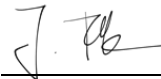
Ort, Datum



Unterschrift

Stuttgart, 23.10.2019

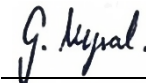
Ort, Datum



Unterschrift

Stuttgart, 23.10.2019

Ort, Datum



Unterschrift