UNIVERSITÄT DUISBURG ESSEN

*Offen im Denken*

PALUNO
The Ruhr Institute for Software Technology

Institut für Informatik und Wirtschaftsinformatik (ICB)
Lehrstuhl für Software Engineering, insb. mobile Anwendungen
Prof. Dr. Volker Gruhn

# Techniken der Computerlinguistik zur Verbesserung von Suchfunktionen in der Software-Entwicklung

Bachelorarbeit

vorgelegt der Fakultät für Wirtschaftswissenschaften
der Universität Duisburg-Essen (Campus Essen) von

Leon Zimmermann
Laddringsweg 8
45219 Essen
Matrikelnummer: 3080384

Essen, den 29. Juni 2023

| | |
|---|---|
| **Betreuung:** | Wilhelm Koop, Sascha Feldmann |
| **Erstgutachter:** | Prof. Dr. Volker Gruhn |
| **Zweitgutachter:** | ??? |
| **Studiengang:** | Angewandte Informatik - Systems Engineering (B. Sc.) |
| **Semester:** | 10 |

## Abstract

TODO: Zusammenfassung auf Englisch

## Zusammenfassung

Softwareentwickler müssen sich bei ihrer Arbeit Informationen zusammensuchen, welche sie für die weitere Arbeit benötigen. So muss ein Softwareentwickler bei der Implementierung eines Features die Intention des Features kennen. Solche Informationen können in Wissensdatenbanken hinterlegt sein. Um dort die gewünschten Informationen zu finden können Suchfunktionen verwendet werden. Insbesondere, wenn dem Softwareentwickler im Vorfeld nicht klar ist, wo er die gewünschten Informationen in der Wissensdatenbank finden kann. Aber nicht immer liefert diese Suchfunktion die gewünschten Informationen.

In dieser Arbeit wird eine semantische Suchfunktion entwickelt. Diese soll eine Verbesserung zu bestehenden Suchfunktionen bieten, wie beispielsweise die Suchfunktion von Confluence. Um festzustellen, ob die neue Suchfunktion *besser* ist als die bestehende, werden Methoden und Kriterien zur Evaluierung von Suchfunktionen erörtert. Anhand dieser Kriterien werden die neu implementierte Suchfunktion und die bestehende Suchfunktion von Confluence in einer Studie verglichen. In der Studie werden beispielhafte Sucheingaben definiert, sowie die erwarteten Ergebnisse. Die Definition der Sucheingaben erfolgt auf Basis von Anwendungsfällen. Die Anwendungsfälle beschreiben Situationen, in denen es realistisch ist, dass ein Softwareentwickler die Suchfunktion einer Wissensdatenbank verwendet. Anhand von Argumentationen werden entsprechend realistische Sucheingaben erstellt.

TODO: Ergebnis der Studie darstellen

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Abkürzungsverzeichnis

**NLP** Natural Language Processing

# 1. Einleitung

Für die tägliche Arbeit benötigt ein Softwareentwickler Informationen, welche über den Code, an dem er arbeitet, hinausgehen. Um an diese Informationen zu kommen kann der Softwareentwickler eine Person suchen, welche ihm die gewünschte Information geben kann. Diese Person kann er im Projekt finden, oder über Websites, wie StackOverflow. Darüber hinaus wird in vielen Projekten eine Wissensdatenbank angelegt, welche Informationen enthält, welche spezifisch für das Projekt sind. Eine solche Wissensdatenbank ist Confluence. Sie bietet eine Suchfunktion, welche es dem Softwareentwickler erleichtern soll, die gewünschten Informationen zu finden. Beispiele für Informationen sind die Spezifikationen oder Dokumentationen eines Teiles der Software, mit welcher der Softwareentwickler gerade arbeitet. Oder aber auch Best-Practices, Guides, oder Informationen darüber, wie die Software gestartet oder ausgeliefert wird. Auch Informationen über den Projektplan sind für einen Softwareentwickler von Bedeutung. Aber nicht immer finden Softwareentwickler die gewünschten Informationen mithilfe der Suchfunktion. Ziel dieser Arbeit soll es sein, mithilfe von Wissen über Suchalgorithmen und Algorithmen aus dem Natural Language Processing zu zeigen, wie sich bestehende Suchfunktionen verbessern lassen.

Software, wie chatGPT[1] zeigt, dass Large Language Models eine valide Möglichkeit zur Information Extraction sind. Es ist denkbar, dass Suchfunktionen für Softwareentwickler durch Large Language Models verbessert werden können. Auch die Verwendung von vorgefertigten semantischen Netzen, welche in Knowledge Bases, wie DBPedia[2] zu finden sind, sind als Lösung für dieses Problem denkbar.[3] Sowohl die Verwendung von semantischen Netzen als auch die Verwendung von Vektordatenbanken kann als semantische Suche verstanden werden. Später soll noch einmal auf den genauen Unterschied zwischen den beiden Methoden eingegangen werden. In dieser Arbeit soll es lediglich um die Verwendung von Vektordatenbanken gehen.

## 1.1. Vorgehensweise

Die Gründe, warum eine gewünschte Information schwierig zu finden ist, sind vielzählig. Manchmal kennt der Softwareentwickler nicht das genaue Wording, um die gewünschten Informationen zu finden. Manchmal ist das Abstraktionslevel der gefundenen Informationen nicht das, welches sich der Softwareentwickler gewünscht hat. Beispielsweise, wenn eine allgemeine Definition von Domänenobjekten ge-

[1] https://openai.com/blog/chatgpt
[2] https://www.dbpedia.org/

sucht wird, aber eine Spezifikation eines Anwendungsfalls gefunden wird, in welchem das Domänenobjekt lediglich erwähnt wird. Es werden folgende Teilschritte durchlaufen, um die Suchfunktion von Wissensdatenbanken, wie Confluence, zu verbessern:

- **Definition von Anwendungsfällen:** Es werden zuerst Anwendungsfälle definiert. Damit wird das Problem der Qualität einer Suchfunktion heruntergebrochen in Teilprobleme. Die Anwendungsfälle beschreiben die konkreten Situationen, in welchen ein Softwareentwickler eine Suche nutzen könnte. Das hilft später dabei mehrere Suchfunktionen miteinander vergleichen zu können. Denn anhand der Anwendungsfälle können realistische Suchanfragen definiert werden. Diese Suchanfragen können an zwei verschiedene Suchfunktionen übergeben werden. Anschließend können die gefundenen Ergebnisse verglichen werden. Die genaue Vorgehensweise für diesen Vergleich wird in Kapitel 3 und Kapitel 6 erklärt. Außerdem bietet die Aufteilung in Anwendungsfälle bereits Aufschluss über die möglichen Verbesserungen, welche gemacht werden können. Hierauf wird in den Teilschritten "Erklärung des theoretischen Hintergrunds" und "Implementierung" weiter eingegangen.

- **Herausarbeitung von Evaluationsmethoden und -Kriterien:** Es werden Methoden für die Bewertung herausgearbeitet. Damit wird die Frage beantwortet, wann eine Suchfunktion *gut* ist. Das hier erläuterte Wissen wird für die Durchführung der Studie benötigt.

- **Erklärung des theoretischen Hintergrunds:** Es wird die Theorie für die Implementierung einer Suchfunktion erläutert. Hier wird erklärt, welche Suchfunktionen es gibt. Außerdem werden Verfahren zur Indizierung von Dokumenten erklärt. Darüber hinaus werden NLP-Techniken erläutert, mit welchen Informationen aus gefundenen Dokumenten extrahiert werden können.

- **Implementierung:** Es wird eine neue Suchfunktion anhand der Informationen des theoretischen Hintergrunds implementiert. Es wird die Weaviate[3] Vektordatenbank verwendet, um Dokumente zu indizieren.

- **Durchführung einer Studie:** Um festzustellen, ob die Implementierung eine Verbesserung darstellt, muss eine Studie durchgeführt werden. Aufgrund des Scopes der Arbeit wird nur eine rudimentäre Studie durchgeführt. Die Ergebnisse werden dargestellt und diskutiert. Dabei wird auch darauf eingegangen, an welchen Stellen die Studie weiter ausgearbeitet werden muss, um präzise Ergebnisse liefern zu können. Außerdem wird der Versuchsaufbau beschrieben und die gemessenen Daten. Die Ergebnisse werden interpretiert und es wird ein Schluss gezogen.

[3] https://weaviate.io/

## 1.2. Verwandte Arbeiten

Es gibt einige Arbeiten, welche die gleichen oder sehr ähnliche Probleme adressieren.

So wird in *Automatic Query Reformulations for Text Retrieval in Software Engineering* von Haiduc et. al. ein System zur Verbesserung von Suchanfragen vorgeschlagen. Ausgangspunkt für das Paper ist das Problem der Traceability zwischen Code und anderen Softwareentwicklungs-Artefakten. Traceability bedeutet, dass sich von einer Stelle im Code, auf die entsprechenden Stellen in anderen Artefakten zurückschließen lässt. Ein Anwendungsfall für eine solche Traceability-Funktionalität ist "Feature Location", also das finden der Spezifikation eines Features, wenn nur der Code vorhanden ist. Das System, welches von Haiduc et. al. vorgeschlagen wird, verwendet Query Reformulations, um die Traceability herzustellen. Query Reformulation bedeutet, dass das System den Softwareentwickler bei der Eingabe einer Suchanfrage zur Suche nach den passenden Artefakten unterstützt. Dazu gibt der Softwareentwickler zunächst eine Suchanfrage ein, und markiert diejenigen Ergebnisse, welche am relevantesten für ihn sind. Auf Grundlage der gewählten Ergebnisse und mithilfe eines Machine Learning Algorithmus werden nun Vorschläge für eine verbesserte Suchanfrage gemacht. Dabei gibt es verschiedene Strategien. Wenn der Softwareentwickler zu Beginn eine sehr lange Suchanfrage eingegeben hat, dann kann das System eine Reduktion der Suchanfrage vorschlagen. Hat der Softwareentwickler dagegen lediglich einen Suchbegriff angegeben, so kann das System eine Erweiterung der Suchbegriffe vorschlagen. Dazu greift das System auf Synonyme des eingegebenen Suchbegriffes zurück.[2]

In dem Paper *From Word Embeddings To Document Similarities for Improved Information Retrieval in Software Engineering* von Ye et. al. wird beschrieben, wie Word Embeddings dazu verwendet werden können, um Traceability zwischen Code und anderen Softwareentwicklungs-Artefakten herzustellen. Word Embeddings sind eine Datenstruktur, welche einem Wort einen Vektor in einem n-dimensionalen Raum zuweist. Anhand dieses Vektors kann die Ähnlichkeit zwischen Wörter beschrieben werden. Ähnliche Wörter haben eine geringe Distanz im n-dimensionalen Raum. Unähnliche Wörter haben eine hohe Distanz. Der Algorithmus, welcher die Ähnlichkeit der Wörter bestimmt, macht Gebrauch von der Distributional Hypthesis. Dieser besagt, dass Wörter, welche im gleichen Kontext verwendet werden, eine ähnliche Semantik besitzen. Hermit wird also die Ähnlichkeit der Wörter bestimmt. Dieses Verfahren wird nun sowohl auf den Code angewendet als auch auf die Softwareentwicklungs-Artefakte.[6]

In dem Paper *Information Retrieval Models for Recovering Traceability Links between Code and Documentation* verwenden Antoniol et. al. einen ähnlichen Ansatz, wie Ye et. al. Auch hier werden Word Embeddings verwendet um Softwareentwicklung Artefakte gegen den Code zu matchen. Hier durchlaufen die Artefakte und der Code zwei verschiedene Pipelines. Die Wörter der Artefakte in natürlicher Spra-

che werden in lowercase umgewandelt. Anschließend werden Stoppwörter entfernt. Zuletzt werden Flexionen entfernt. Aus dem Code werden zunächst Identifier extrahiert. Identifier, welche mehrere Wörter unter Verwendung von CamelCase oder snake_case beinhalten, werden in die einzelnen Wörter aufgeteilt. Anschließend werden die Identifier auf die gleiche Art und Weise normalisiert, wie die Wörter der Softwareentwicklungs-Artefakte. Dann erfolgt sowohl für die Identifier als auch für die Wörter aus den Artefakten die Indizierung, also die Umwandlung in Word Embeddings.[1]

In dem Paper *TaskNav: Task-based Navigation of Software Documentation* von Treude et. al. geht es um die Entwicklung einer Oberfläche, welche die Suche von *Tasks* ermöglicht. Dabei ist unter Task eine Operation im Code zu verstehen. Das Paper beschreibt einen Task als Verben, welche mit einem direkten Objekt oder einer Präposition in Verbindung stehen. Die Autoren nennen die Phrasen *get iterator* und *get iterator for collection* als Beispiele. Die Software analysiert nun die gesamte Dokumentation und extrahiert Tasks. Die Tasks werden in einen Index geschrieben, sodass der Softwareentwickler nach ihnen suchen kann. So wie die vorherigen Paper soll auch dieses Paper eine Brücke zwischen Dokumentation und Code schaffen.[5]

# 2. Definition von Anwendungsfällen

In diesem Kapitel sollen Anwendungsfälle ausgewählt werden, für welche später Lösungsansätze entwickelt werden. Die Anwendungsfälle beschreiben die Situationen, in denen Softwareentwickler die Suchfunktionen von Wissensdatenbanken verwenden könnten. Zur Identifikation von Anwendungsfällen wurde zunächst Literatur herangezogen. Die Literatur ist bereits unter den verwandten Arbeiten aufgeführt. In den Verwandten Arbeiten wurden Arbeiten genannt, welche ähnliche Probleme lösen sollen. Diese Fokussieren sich vorallem auf die *Feature Location*, *Bug Localization* und die Traceability zwischen Code und anderen Artefakten. Mit anderen Worten: Die Suchfunktion soll eine Brücke zwischen Code und Dokumentation herstellen.

## 2.1. Onboarding im Projekt

Wenn ein neuer Softwareentwickler in einem Softwareprojekt startet, dann muss er sich zunächst einmal mit dem Projekt vertraut machen. Das bedeutet, dass er verstehen muss, was das Projekt eigentlich ist. Er muss verstehen, was das eigentliche Problem des Kunden ist. Außerdem muss er verstehen wie die Software dieses Problem löst. Dazu muss der Softwareentwickler sehr allgemeine Informationen über das Projekt finden können. Er könnte Dinge suchen, wie einen Projektüberblick oder ein Glossar.

Neben diesen allgemeinen Informationen muss sich der neue Softwareentwickler mit dem Code vertraut machen. Er muss verstehen, welche Technologien verwendet werden, welche Best-Practices, Code-Styles, Guidelines, Prozesse und Quality Gates eingehalten werden müssen. Und er muss verstehen, wie die Software lokal oder in einer Testumgebung ausgeführt werden kann.

## 2.2. Feature Location

Bei der Entwicklung eines neuen Features greift der Softwareentwickler also auf die entsprechende Spezifikation zurück. Dazu muss ihm bekannt sein, wo die Spezifikation zu finden ist. Nun muss er bei der Entwicklung darauf achten, dass er Best-Practices und Konventionen einhält, sowie die Qualitätsanforderungen. Eine Qualitätsanforderungen könnte dabei eine vereinbarte Testabdeckung der Software sein. Der Softwareentwickler muss also bei der Entwicklung eines neuen Features auch diese Informationen einfach finden können. Und ihm muss klar

sein, an welcher Stelle im Code er den neuen Code einbauen sollte. Das ist Feature Location.

## 2.3. Bug Localization

Wenn der Softwareentwickler gerade kein neues Feature implementiert, dann korrigiert er gerade möglicherweise einen Fehler in der Software. Um einen Fehler überhaupt zu identifizieren, muss aber zuerst wieder die Spezifikation herangezogen werden. Denn in der Spezifikation wird, wie bereits erwähnt, die gewünschte Funktionsweise der Software beschrieben. Damit wird auch definiert, was ein fehlerhaftes Verhalten ist, und was ein korrektes Verhalten ist. Wenn der Softwareentwickler nun ein Fehlerticket erhält, dann muss er die entsprechende Spezifikation zu diesem Fehlerticket finden können. Und idealerweise wird ihm durch die Suche sogar gleich die betroffene Stelle im Code angezeigt. Das ist Bug Localization.

## 2.4. Informationen über das Projektmanagement finden

Softwareentwickler sollten darüber Bescheid wissen, was in dem Projekt, in dem sie arbeiten, vor sich geht. Sie sollten über organisatorische Dinge Bescheid wissen, wie die Teamaufteilung und die Zuständigkeiten in den einzelnen Teams und in dem gesamten Projekt. Das ist wichtig für eine gute Kommunikation und entsprechenden Wissensaustausch zwischen Kollegen. Außerdem sollten Softwareentwickler darüber Bescheid wissen, wann das nächste Release der Software ansteht. Diesen Termin müssen sie bei der Planung ihrer Arbeit berücksichtigen, damit sie auch rechtzeitig alle relevanten Features implementiert haben und alle kritischen Fehler behoben haben. Neben der Einhaltung von Terminen ist die Planung wichtig für die Motivation der Softwareentwickler. Denn Motivation entsteht durch die Wahrnehmung, sich einem Ziel zu nähern (Quelle: Flow - mihaly csikszentmihaly). Aus diesem Grund müssen die Softwareentwickler auch die Vision der Software verstehen und auch die übergreifende Vision des Unternehmens. Sie müssen verstehen, warum die Arbeit, welche sie erledigen so wichtig ist. Und sie müssen verstehen, für wen sie diese Arbeit tun. Auch diese Faktoren sind wichtig für eine hohe Motivation bei der Arbeit. Daher ist es so wichtig diese Informationen einfach zugänglich zu machen, also einfach auffindbar zu machen.

## 2.5. Implementierung nach Spezifikation

Bei der Implementierung von neuen Anforderungen ist es wichtig, dass sich der Softwareentwickler an die Spezifikation hält. Nur so bekommt der Kunde die Software, die er sich gewünscht hat. Dazu sollte der Softwareentwickler es schaffen alle relevanten Dokumente zu finden, die zu der Spezifikation dazugehören. Zuerst sollte er die Spezifikation selbst finden können. Er sollte die Dokumentation der

damit einhergehendenden Prozesse finden, und auch die Domänenobjekte, welche bei der Implementierung relevant sein werden. Er sollte Diagramme finden können, welche zu dem Anwendungsfall gehören, und auch die weiteren Dokumente, welche den Kontext der Anforderung erläutern. Außerdem wäre es hilfreich für den Softwareentwickler auch gleich die relevanten Stellen im Code angezeigt zu bekommen.

## 2.6. Abgleich mit Spezifikation

Der Abgleich mit einer Spezifikation ist notwendig, um Testfälle zu schreiben, und zu prüfen, ob das Verhalten der Anwendung korrekt ist. Natürlich gibt es Arten von Fehlern, welche erkennbar sind, ohne dafür die Spezifikation heranzuziehen. Wenn in einem Online-Shop der Preis für ein Produkt in Amerika bei 5$ liegt, aber in Deutschland der Preis bei 1.000€ liegt, dann braucht es nicht die Spezifikation, um festzustellen, dass es bei der Umrechnung von Dollar zu Euro einen Fehler gegeben hat. Aber nicht alle Fehler sind so offensichtlich. Die Spezifikation zum Abgleich mit dem Verhalten der Software heranzuziehen ist besonders in Randbedingungen hilfreich. Angenommen ein Online-Shop bietet einen kostenlosen Versand ab einem Mindestbestellwert an. Sobald der Preis den Wert von 25$ übersteigt ist der Versand gratis. Der Versand kostet im Normalfall 5$. Nun muss definiert werden, ob der Versand in dem Mindestbestelltwert miteinbezogen wird oder nicht. Wird er miteinbezogen, dann sorgt das dafür, dass ein Produkt, welches 20$ kostet einen kostenlosen Versand hat, weil der Mindestbestellwert zuzüglich dem Versandpreis 25$ beträgt. Ob dieses Verhalten gewünscht ist oder nicht, muss in der Spezifikation festgehalten werden.

Wenn nun ein Bugticket dieser Art bei einem Softwareentwickler landet, dann muss er, wie auch bei der Implementierung, alle relevanten Dokumente heranziehen.

## 2.7. Deployment

TODO: Ergänzen

## 2.8. Dokumentation zum besseren Verständnis heranziehen

TODO: Glossar, API-Dokumentation etc.

Im nachfolgenden Kapitel "Evaluationsmethoden und -Kriterien" wird aufgezeigt, wie die Anforderungen der Anwendungsfälle quantifizierbar gemacht werden können. Das wird dabei helfen nachzuvollziehen, inwieweit die Anforderungen der Anwendungsfälle erreicht wurden, welche genannt wurden.

# 3. Evaluationsmethoden und -Kriterien

Um zu verifizieren, dass die Implementierung ihre gewünschte Wirkung erzielt, müssen zuerst Methoden und Kriterien zur Messung herangezogen werden. Karl Popper gilt als Begründer des kritischen Rationalismus. Es war der Anfang von wissenschaftlich durchgeführten Experimenten. Bei einem Experiment wird zunächst eine Hypothese aufgestellt. Diese wird anschließend durch das Experiment untersucht. Ein solches Experiment soll später für die genannten Anwendungsfälle erstellt werden. Dazu sollen für die Anwendungsfälle Sucheingaben erstellt werden. Für jede Sucheingabe wird definiert, welche Dokumente als Ergebnis erwartet werden. Die Hypothese: Mit der implementierten Suchfunktion werden die erwarteten Dokumente *besser* gefunden als mit der bisherigen Suchfunktion. Diese Hypothese muss nun quantifizierbar und messbar gemacht werden. Die subjektive Wahrnehmung reicht nicht für eine wissenschaftliche Arbeit aus.

Die Formulierung der Hypothese ist für sich genommen zu unspezifisch. Es muss geklärt werden, wann eine Information *besser* zu finden ist. Im Folgenden werden Precision, Recall und F-Maß, sowie Qualitätskriterien gemäß ISO/IEC 9126 herangezogen, um Suchfunktionen anhand dieser Eigenschaften vergleichbar zu machen. Anhand dessen wird hergeleitet, was in diesem Kontext *besser* bedeutet. Im Anschluss werden diese Eigenschaften in einem Versuchsaufbau verwendet.

## 3.1. Precision, Recall und F-Maß

Zur Evaluation der Suchfunktionen werden später die statistischen Messwerte Precision und Recall verwendet. Die Messwerte beschreiben, inwieweit eine Hypothese zutrifft. Wenn also die Hypothese ist, dass ein bestimmtes Dokument gefunden wird, dann ist der Precision-Wert das Verhältnis zwischen allen gefundenen Dokumenten und den gefundenen Dokumenten, die tatsächlich relevant sind. Der Wert lässt sich im Kontext der Suche nach Dokumenten wie folgt definieren[4]:

$$Precision = P = \frac{gefundenerelevanteDokumente}{gesamteAnzahlgefundenerDokumente}$$

Der Recall-Wert gibt wiederum an, wie viele von den tatsächlich relevanten Dokumenten auch gefunden wurden. Er lässt sich in diesem Kontext wie folgt definieren[4]:

$$Recall = R = \frac{gefundenerelevanteDokumente}{gesamteAnzahlrelevanterDokumente}$$

Es ist schwierig beide Werte zu optimieren, da der Precision-Wert versucht die Anzahl der gefundenen Dokumente einzugrenzen und der Recall-Wert versucht die

Anzahl der gefundenen Dokumente zu erweitern. Das F1-Maß fasst beide Werte zu einem neuen Wert zusammen[4]:

$$F_1 = 2\frac{PR}{P+R}$$

Neben der Verwendung des F1-Maß ist es wichtig sich Gedanken darüber zu machen, welcher der beiden Messwerte wichtiger ist. In diesem Fall ist es sinnvoll eher die Precision zu optimieren. Denn, wenn ein Dokument gesucht wird, aber überhaupt nicht gefunden werden kann, dann erfüllt die Suchfunktion nicht ihren Zweck. Wenn die Suchfunktion irrelevante Dokumente darstellt, kann sie trotzdem ihren Zweck erfüllen, solange die relevantesten Dokumente zuerst in der Liste der Ergebnisse dargestellt wird. Dieser Faktor gilt auch bei der Implementierung zu berücksichtigen.

## 3.2. ISO/IEC 9126

Die ISO/IEC 9126 legen Qualitätskriterien für die Entwicklung von Software fest. Es gibt sechs verschiedene Qualitätskriterien, welche sich wiederum in verschiedene Facetten einteilen lassen. Doch nicht alle der sechs Qualitätskriterien sind in dieser Arbeit von Relevanz. So werden Die Qualitätskriterien Wartbarkeit, Effizienz, Übertragbarkeit und Zuverlässigkeit nicht beachtet. Die Wartbarkeit bezieht sich auf die Fähigkeit von Software sich zu verändern. Diese Fähigkeit hat allerdings nichts mit der Qualität einer Suchfunktion zu tun, wie sie hier gemessen werden soll. Die Effizienz ist im Kontext von Suchfunktionen als gegeben anzunehmen. Der Author unterstellt hier, dass jede Suchfunktion, welche untersucht werden soll, auch effizient ist. Denn im Jahr 2023 ist anzunehmen, dass eine Suchfunktion auch schnell die Suchergebnisse darstellen kann. Die Übertragbarkeit beschreibt die Fähigkeit einer Software, auf verschiedenen Umgebungen lauffähig zu sein. Die Zuverlässigkeit beschreibt, ob die Software auch bei unterschiedlichen Rahmenbedingungen in der gewünschten Form funktioniert. Die beiden Kriterien liegen ebenfalls außerhalb des Scopes der Arbeit, weil auch sie sich nicht auf die Qualität der Suchfunktion als solches beziehen.

Die zu betrachtenden Qualitätskriterien sind Benutzbarkeit und Funktionalität. Die beiden Qualitätskriterien werden in den nächsten Kapiteln im Detail erläutert. Das schließt die verschiedenen Facetten der Qualitätskriterien mit ein.

### 3.2.1. Benutzbarkeit

Die Benutzbarkeit von Software teilt sich in die Facetten Attraktivität, Konformität, Erlernbarkeit, Verständlichkeit und Bedienbarkeit ein. Die Attraktivität soll hier nicht weiter betrachtet werden, weil die reine Optik einer Suchfunktion im Kontext dieser Arbeit keine Relevanz besitzt. Die Facetten Erlernbarkeit und Verständlichkeit ergeben sich durch die Konformität der Suche. Damit eine Suchfunktion konform ist, muss sie alle gängigen Arten von Suchen unterstützen. Also

Keyword Search, Phrase Search, Boolean Search, Phrase Search. Die unterschiedlichen Arten von Suchen werden in dem Kapitel "Theoretischer Hintergrund" näher erläutert.

Die übrige Facette ist die Bedienbarkeit. Eine Autovervollständigung von Wörtern kann es dem Softwareentwickler vereinfachen, passende Sucheingaben zu machen. Sie verhindert zum einen, dass er Tippfehler macht. Zum anderen hilft sie dem Softwareentwickler auch bei der Wortwahl. Sie sorgt also für eine bessere Bedienbarkeit.

### 3.2.2. Funktionalität

Die Funktionalität von Software teilt sich in die Facetten Angemessenheit, Sicherheit, Interoperabilität, Konformität, Ordnungsmäßigkeit und Richtigkeit auf.
TODO: Weiter ausführen

## 3.3. Versuchsaufbau

Für den Versuchsaufbau werden Dokumente benötigt, welche durchsucht werden können. Dazu sollen Daten aus einem echten Projekt verwendet werden. Da die Informationen zu dem Projekt nicht veröffentlicht werden dürfen, müssen die Sucheingaben und Ergebnisse so eingeschränkt werden, dass keine projektspezifischen Informationen preisgegeben werden. Die Daten sind bereits in einem Confluence Space vorhanden, dessen Suche als Benchmark für die neu implementierte Suchfunktion verwendet wird. Die Implementierung der neuen Suchfunktion und die Befüllung dessen Datenbank mit den gleichen Daten, wie die Confluence Suche, wird in dem Kapitel *Implementierung* näher erklärt.

Wie zuvor erwähnt, müssen nun Sucheingaben erstellt werden, welche in beiden Suchen eingegeben werden können. Zum Vergleich der beiden Suchfunktionen werden für jede Sucheingabe die obersten fünf Ergebnisse betrachtet. Für jedes Dokument in diesen Ergebnissen, welches laut Experimentaufbau als Ergebnis erwartet wird, erhält die Suchfunktionen einen Hit. Die Hits werden für beide Suchfunktionen zusammengezählt und verglichen. Anschließend wird eine einfaktorielle Varianzanalyse mithilfe von SPSS durchgeführt. Die Analyse wird zeigen, ob die neu implementierte Suchfunktion signifikant mehr Hits generiert als die bestehende Confluence Suche.

## 3.4. Einfaktorielle Varianzanalyse

Die Varianzanalyse überprüft, ob ein signifikant unterschiedlicher Wert zwischen den beiden Suchfunktionen besteht. Wenn die neu implementierte Suchfunktion signifikant mehr Dokumente findet als die Confluence-Suche, dann bestätigt dies

die Hypothese. Die Hypothese ist dabei, dass die neu implementierte Suchfunktion eine Verbesserung zur bestehenden Confluence-Suche darstellt.

Zur Durchführung der einfaktoriellen Varianzanalyse muss die abhängige Variable und die unabhängige Variable definiert werden. Die unabhängige Variable ist die verwendete Suchfunktion. Diese muss numerisch repräsentiert werden. So steht die Zahl 0 für die Confluence-Suche, und die Zahl 1 für die neu implementierte Suchfunktion. Die abhängige Variable ist die Anzahl der gefundenen erwarteten Ergebnisse. Sie ist die abhängige Variable, weil die Anzahl von der verwendeten Suchfunktion abhängt.

# 4. Theoretischer Hintergrund

TODO: Ergänzen

## 4.1. Suchalgorithmen

In dem Kapitel *Evaluationsmethoden und -Kriterien* wurde bereits beschrieben, dass eine Suche das Qualitätskriterium der Konformität erfüllen sollte. Das bedeutet, dass eine Suchfunktion die gängigen Arten von Suchanfragen unterstützen muss. Dieses Kapitel soll die Arten von Suchanfragen vorstellen. Es ist wichtig diese zu kennen, um das Qualitätskriterium später bei der Implementierung erfüllen zu können.

### 4.1.1. Keyword Search

Eine Keyword Search durchsucht Dokumente nach der Sucheingabe des Nutzers. Die Eingabe wird dabei nicht als ganzes betrachtet, sondern jedes Wort einzeln. Für jedes Keyword werden Dokumente als Ergebnis angezeigt, wenn dieses in dem Dokument vorhanden ist. Wenn ein Dokument mehrere der Keywords beinhaltet, wird dessen Relevanz höher eingeschätzt als für Dokumente, welche weniger Keywords enthalten. Dokumente mit höherer Relevanz werden weiter oben in der Ergebnisliste angezeigt.

### 4.1.2. Phrase Search

Eine Phrase Search ist die Suche nach Textausschnitten in Dokumenten. Hier werden nicht mehrere Keywords einzeln betrachtet, sondern die gesamte Eingabe in das Suchfeld als eine Einheit. Es reicht also nicht mehr aus, dass ein Dokument eines der Wörter enthält. Es muss die gesamte Sucheingabe als ein String enthalten sein.

### 4.1.3. Boolean Search

Die Boolean Search bietet die Möglichkeit einen boolschen Ausdruck als Sucheingabe zu machen. Ein Beispiel dafür ist die Sucheingabe *Dokumentation AND Angular*. Die Sucheingabe bedeutet, dass die Suchmaschine nur Dokumente als Ergebnis darstellen soll, welche beide Keywords Dokumentation und Angular enthalten. Die Boolean Search kann auch eine Phrase, wie bei der Phrase Search, beinhalten: *"Dokumentation von Software" AND Angular*. In diesem Beispiel werden nur Dokumente als Ergebnis nur angezeigt, wenn sie den gesamten String *Dokumentation von Software* enthalten, sowie das Keyword *Angular*. Bei einer

Boolean Search können die boolschen Operatoren *AND*, *OR*, *NOT* beliebig kombiniert werden.

### 4.1.4. Field Search

Eine Field Search sucht Dokumente anhand von Attributen. Der Nutzer kann diese Attribute auswählen. Wenn der Nutzer beispielsweise ein Dokument sucht, welches am 01.01.2005 erstellt wurde, dann kann die Eingabe der Suche so aussehen: *erstelldatum: 01.01.2005*. Es können beliebig viele Attribute verwendet werden, um die Suche einzugrenzen.

Neben der Verwendung der Attribute für die Suche selbst, können die Attribute komplementär zu einer anderen Art von Suche verwendet werden. So kann eine Suchfunktion Buttons bereitstellen, über welche Filter festgelegt werden. Nun kann eine Keyword Search durchgeführt werden, aber die gefundenen Dokumente werden mithilfe der Filter weiter eingeschränkt.

### 4.1.5. Structured Search

Eine klassische Suchfunktion verwendet Keywords, um relevante Dokumente für eine Sucheingabe zu ermitteln. Der Vorteil dieser Art von Suche ist, dass die technische Struktur, in der die Daten vorliegen und gespeichert sind, nicht bekannt sein müssen. Der Nachteil ist auf der anderen Seite, dass die Suchergebnisse unpräzise sein können. Wenn beispielsweise der Suchbegriff Kamera eingegeben wird, dann werden Kameras als Ergebnisse zurückgegeben. Soll die Kamera nun bestimmte Eigenschaften besitzen, dann müssen diese Eigenschaften ebenfalls als Keywords angegeben werden. Nun wird aber nicht die Suche auf Ergebnisse eingegrenzt, bei denen eine Kamera diese bestimmten Eigenschaften besitzt. Stattdessen werden Suchergebnisse angezeigt, bei denen einige dieser Keywords vorkommen. Demgegenüber stehen Datenbankabfragen, beispielsweise mithilfe von SQL. Bei einer Datenbankabfrage können Objekte abgefragt werden, dessen Eigenschaften ganz bestimmte Werte haben. Die Ergebnisse, die eine solche Abfrage zurückgibt, sind dabei vollkommen genau. Es werden keine Objekte zurückgegeben, welche diese Kriterien nicht erfüllen. Voraussetzung für eine solche Suchabfrage ist allerdings, dass die Struktur der Datenbank a priori bekannt ist. Dem Nutzer muss der Name der Datenbank, der relevanten Tabellen, sowie der relevanten Properties bekannt sein, damit er das passende SQL für die Datenbankabfrage schreiben kann. Strukturierte Suchen sollen die Vorteile beider Vorgehensweisen kombinieren. Die Struktur der Daten soll a priori nicht bekannt sein müssen, aber trotzdem sollen die Suchergebnisse vollkommen präzise sein.

### 4.1.6. Lexical Search

TODO: Ergänzen

### 4.1.7. Semantic Search

Eine Semantische Suche arbeitet nicht anhand von Keywords, sondern anhand von Bedeutungen von Wörtern. Sie ist technisch so umgesetzt, dass durch sie die semantische Ähnlichkeit von Wörtern abgebildet wird. So wird durch sie beispielsweise die Ähnlichkeit von den Wörtern *rollout deployment"* abgebildet, und dass diese Wörter oft im gleichen Kontext verwendet werden. Um zu verstehen, welche Wörter kontextuell zusammengehören, werden die Wörter in einem n-dimensionalen Raum positioniert. Wörter, die sich sehr ähnlich sind, also im gleichen Kontext verwendet werden, haben in diesem n-dimensionalen Raum eine geringe Distanz. Wörter, die sich eher unähnlich sind, wie *"rollout" und "API"*, haben eine größere Distanz. Der Vorteil einer semantischen Suche ist, dass der genaue Begriff, welcher gesucht wird nicht bekannt ist. Wenn sich der Nutzer also über ein Thema informieren möchte, mit welchem er nicht gut vertraut ist, dann kann die semantische Suche hilfreich sein. Denn der Nutzer kann nun einen Begriff eingeben, der zu dem Thema passt, und den er kennt. Er findet anschließend Dokumente, welche vielleicht nicht genau diesen Begriff beinhalten, aber welche thematisch dennoch ähnlich sind. Genau dieser Vorteil soll bei der Implementierung später genutzt werden.

Um eine semantische Suche zu implementieren, werden die Technologien von Transformern und Vektordatenbanken verwendet. Ein Transformer erhält als Input eine große Menge an Text und mappt die einzelnen Wörter auf einen Vektor einer beliebigen Länge. Der Vektor, der am Ende herauskommt, beschreibt die Position des Wortes in dem n-dimensionalen Raum. Der Vektor beschreibt gewissermaßen, wie stark ein Wort in eine abstrakte Kategorie einzuordnen ist. Jeder Wert im Vektor entspricht einer Kategorie. Mithilfe der Vektoren können verschiedene Wörter hinsichtlich ihrer Ähnlichkeit analysiert werden. Ähnliche Wörter habe eine große räumliche Nähe, während zwei Wörter, die in vollkommen unterschiedlichen Kontexten verwende werden eine sehr große Distanz im Raum besitzen. Nehmen wir für ein Beispiel einen dreidimensionalen Raum an. Die X-Achse ist beschriftet mit dem Wort „Tier“, die Y-Achse ist beschriftet mit dem Wort „Computer“ und die Z-Achse ist beschriftet mit dem Wort „Mensch“. Nun geben wir einem Transformer das Wort „Katze“, und der Transformer berechnet einen dreidimensionalen Vektor, welcher das Wort „Katze“ im Raum positioniert. Weil eine Katze ein Tier ist, ist der X-Wert des Vektors eins. Der Wert eins bedeutet, dass das Wort vollständig zu dieser Kategorie gehört. Da eine Katze überhaupt nichts mit einem Computer zu tun hat, ist der Y-Wert des Vektors 0.

Nun ist eine Katze zwar kein Mensch, aber eine Katze ist ein Haustier von Menschen. Es ist denkbar, dass die Wörter Katze und Mensch oft im gleichen Kontext verwendet werden, sodass der Wert bei 0,3 liegen könnte. Damit der Transformer einen Vektor berechnen kann, braucht er eine Menge Daten. Diese Daten erhält er aus vielen Texten. Werden zwei Wörter oft im gleichen Text genannt oder kommen zwei Wörter in vielen Texten sehr nahe beieinander vor, dann geht der Transformer

davon aus, dass die beiden Wörter ähnlich sind, und berechnet ähnliche Vektoren. Zuvor müssen die Texte allerdings bereitgestellt werden. Dazu kann beispielsweise das Internet gecrawlt werden. Die Ergebnisse des Transformers werden in einer Vektordatenbank gespeichert. Eine Vektordatenbank ist eine Datenbank, welche Vector Embeddings, also ein Objekt als Key und dessen Vektor als Value speichert. Bei dem Objekt kann es sich um Wörter handeln, dann wird auch von Word Embeddings gesprochen. Es können aber auch Daten andere Daten, wie Bilder, Videos oder Audio gespeichert werden. Der Zweck von Vektordatenbanken ist es, Daten nicht einfach linear zu speichern, sondern in einem Raum. Die Distanz zwischen zwei Einträgen in diesem Raum beschreibt dessen Ähnlichkeit. Genau diese Informationen machen sich semantische Suchen zu Nutze.

## 4.2. NLP-Algorithmen

TODO: Ergänzen

## Suchmaschinen-Architektur

Die Architektur von Suchmaschinen besteht aus drei Komponenten. Aus dem Crawling, der Indexierung und der Suchfunktion selbst. Das Crawling ist zuständig für das Finden von Websites. Die Indizierung ist zuständig für das optimale Speichern der Informationen der Websites, und die Suche ist zuständig für das Verstehen der Nutzeranfrage und die Abfrage der relevantesten Informationen aus dem Index, sowie dessen Verarbeitung und Darstellung. Im Folgenden wird der Begriff Dokument verwendet, um die Dateien zu beschreiben, welche durch einen Crawler gesucht und durch den Index verarbeitet werden. Unter Dokumenten können hierbei auch eine Website verstanden, welche durch einen Webcrawler durchsucht werden.

## 4.3. Crawling

Bei der Implementierung eines Systems für eine Suchfunktion benötigt das System zunächst einen Datensatz von Dokumenten, welche überhaupt grundsätzlich über die Suchfunktion gefunden werden können. Dieser wird mithilfe eines Crawlers aufgebaut. Ein Crawler ist ein Algorithmus, welcher ein Dokument als Startpunkt bekommt, und anhand dessen neue Dokumente findet. Der Algorithmus analysiert dazu das Dokument auf Links, welchen der Algorithmus anschließend folgt. Die neuen Dokumente werden durch den Index verarbeitet und wiederum auf neue Links analysiert. Dieses Verfahren kann beliebig lange und beliebig rekursiv durchlaufen werden, um den Index zu erweitern. Neben dem Crawling können Indizes befüllt werden, indem eine Liste von Dokumenten übergeben werden, welche dem Index hinzugefügt werden sollen.

## 4.4. Indizierung

Bei der Indexierung werden Wörter und Tokens mit Dokumenten assoziiert. Dazu werden Wörter aus gecrawlten Dokumenten extrahiert und in die Datenbank geschrieben. Den Dokumenten werden IDs zugeordnet und diese IDs werden den Wörtern zugeordnet. Dem Index werden nun weitere Informationen hinzugefügt, wie die Wortfrequenz. Die Wortfrequenz gibt an, wie oft ein Wort in einem Dokument vorkommt. Es wird auch gespeichert, an welchen Stelle des Dokuments das Wort vorkommt, und auch in wie vielen Dokumenten ein Wort vorkommt. Diese Art der Indizierung wird auch als invertierter Index bezeichnet, weil den Wörtern bzw. Tokens die Dokumente zugeordnet werden, und nicht umgekehrt. Wenn der Nutzer nun ein Keyword in die Suche eingibt, dann sind diese Keywords oft bereits im Index vorhanden, sodass die Dokumente, welche diese Keywords beinhalten einfach dem Index entnommen werden können. Aufgabe der Suche ist es anschließend die Ergebnisse aufzubereiten.

### 4.4.1. Volltext-Indizierung

Bei der Indizierung der Wörter besteht die Problematik, dass gleiche Wörter in unterschiedlichen Formen existieren können. So stammen *Heizung* und *heizen* beide von dem gleichen Wortstamm *heiz* ab. Um bei der Indizierung Speicherplatz zu sparen, können Wörter auf diesen Wortstamm reduziert werden, damit sie als ein einziges Wort betrachtet werden können. Die Bildung des Wortstamms wird auch als Stemming bezeichnet. Beim Stemming kann es jedoch zu Overstemming und Understemming kommen. Overstemming bedeutet, dass zwei Wörter, die eigentlich nichts miteinander zu tun haben, also nicht semantisch gleich sind, den gleichen Wortstamm besitzen und als ein Wort betrachtet werden. Ein Beispiel hierfür sind die Wörter *Wand* und *wandere*, wie in *ich wandere*. Beide besitzen den Wortstamm *wand* und werden entsprechend als ein Wort betrachtet. Understemming bedeutet, dass zwei Wörter, die eigentlich etwas miteinander zu tun haben, also semantisch gleich sind, nicht den gleichen Wortstamm besitzen und dadurch als zwei verschiedene Wörter betrachtet werden. Ein Beispiel hierfür sind die Wörter *absorbieren* und *Absorption*, welche die Wortstämme *absorb* und *absorp* besitzen. Es gibt Techniken zur Vermeidung solcher Probleme, wie der Einsatz vollständiger morphologischer Analysekomponenten. Hierauf soll aber nicht weiter eingegangen werden.

Zur Implementierung eines Volltext-Index werden Dokumente und Wörter in einer m x n Matrix angeordnet, wobei m die Anzahl der Dokumente ist und n die Anzahl der Wörter. Die Werte in dieser Matrix werden anhand einer ausgewählten Metrik bestimmt. Eine oft verwendete Metrik ist das Verhältnis der Wortfrequenz mit der invertierten Dokumentfrequenz. Die Wortfrequenz gibt dabei an, wie häufig das in dem Dokument vorkommt. Die Dokumentfrequenz gibt an, in wie vielen Dokumente ein Wort vorkommt. Das Verhältnis gibt damit an, wie charakteristisch das Wort für ein bestimmtes Dokument ist. Wenn das Wort charakteristisch

für ein Dokument ist, dann kommt es in diesem Dokument häufig vor, aber in anderen Dokumenten nur selten. Wenn das Wort nicht charakteristisch für das Dokument ist, dann kommt es in anderen Dokumenten genauso häufig oder häufiger vor als in diesem Dokument. Die Metrik wird auch als tf-idf-Wert bezeichnet.

TODO Rechnung besser verstehen

Wir können uns nun einen m-dimensionalen Raum vorstellen, in dem jedes Dokument eine Dimension darstellt. Jedes Dokument liegt räumlich auf der Achse der eigenen Dimension. Wenn der Nutzer nun Die Matrix lässt sich konzeptionell so verstehen, dass sie aus n Merkmalsvektoren besteht. Verwenden wir die eben beschriebene Metrik, dann gibt jeder Merkmalsvektor eine Gewichtung an, wie charakteristisch ein Wort für die verschiedenen m Dokumente ist. Nun kann der Nutzer Keywords in die Suche eingeben.

### 4.4.2. Vektorindizes

Neben einer Volltext-Indizierung können Dokumente in Form von Vektoren indiziert werden. Dazu werden Dokumente zunächst, wie auch bei der Volltext-Indizierung gecrawlt. Anschließend durchlaufen die Inhalte der Dokumente ein Preprocessing. Dieses kann je nach Implementierung variieren. Das Kapitel *Einspielen der Daten in Weaviate* beschreibt, wie in der hier aufgeführten Implementierung das Preprocessing durchgeführt wird. Das Preprocessing hat den Zweck die Daten an das Schema der Datenbank anzupassen und die Qualität der Daten zu erhöhen. Außerdem sorgt es für eine kürzere Indizierungszeit.

Nach dem Preprocessing werden durch einen Transformer für die Inhalte der Dokumente Vektoren berechnet. Ein Transformer wird mithilfe von Trainingsdaten darauf trainiert, Vektoren für Wörter zu generieren. Das trainierte Modell wird nach dem Preprocessing durchlaufen. Zuletzt werden die Daten in einer Vektordatenbank gespeichert. Eine Vektordatenbank speichert Daten, wie eine dokumentenbasierte Datenbank. Dort werden nun sowohl die rohen Daten als auch die Vektoren gespeichert, welche von dem Transformer berechnet wurden. Die Vektoren haben den Vorteil, dass die Daten in der Datenbank nicht linear gespeichert sind. Sie sind in einem n-dimensionalen Raum gespeichert, mit dessen Hilfe die semantische Nähe zwischen Dokumenten ausgedrückt werden kann. Das funktioniert auf die gleiche Weise, wie bereits in dem Kapitel *Semantic Search* beschrieben.

# 5. Implementierung

In diesem Kapitel soll die Implementierung einer alternativen Suchfunktion zu der Suchfunktion von Confluence dargestellt werden. Diese Implementierung wird anschließend mithilfe des erläuterten Versuchsaufbaus mit der bestehenden Suche von Confluence verglichen. Die Implementierung verwendet eine Vektordatenbank, welche von Haus aus eine semantische Suche ermöglicht. Das Filtern nach bestimmten Properties ermöglicht die Vektordatenbank ebenfalls. Es soll die Vektordatenbank Weaviate verwendet werden. Der Vergleich soll zeigen, ob eine semantische Suchfunktion mit Filtern *besser* ist als die Confluence-Suche. Die Confluence-Suche soll hier als Benchmark für die Performance der implementierten Suche dienen. Grund dafür ist die Verwendung von Confluence in vielen Softwareprojekten (TODO Quelle).

Um die bestehende Confluence-Suche mit einer neuen Implementierung zu vergleichen, erscheint die Entwicklung eines Confluence Plugins als logische Herangehensweise. Die Implementierung eines Plugins für Confluence kann über zwei Arten erfolgen. Es können Plugins für die Confluence Cloud entwickelt werden, einer kostenfreien Version von Confluence. Und es können Plugins für die Self-Hosted Variante geschrieben werden. Über Forge, einem Entwicklungsframework von Atlassian, kann ein Plugin für Confluence Cloud entwickelt werden. Die Entwicklung eines Plugins für die Self-Hosted Variante erfolgt über die Implementierung einer Spring Boot Anwendung. Diese Anwendung wird anschließend als Jar-Datei bereitgestellt. Ein plugin-descriptor beschreibt, wie die Jar in dem Self-Hosted Confluence Space einzubinden ist.

Nun ist für einen Vergleich der Suchfunktionen nicht nur die Implementierung einer neuen Suchfunktion nötig. Es werden auch Daten benötigt, welche die Suchfunktion abfragen kann. Dazu wurde dem Autor ein Self-Hosted Confluence Space bereitgestellt, um die Suchfunktionen auf Basis der bestehenden Daten zu testen. Für die Confluence Cloud stehen dem Autor solche Daten nicht zur Verfügung. Aus organisatorischen Gründen ist die Installation eines Plugins auf dem Self-Hosted Confluence Space nicht möglich. Aus diesem Grund werden die Daten lediglich aus dem Confluence Space exportiert, und anschließend einmalig in der Weaviate Datenbank indiziert. Die genaue Vorgehensweise wird im Folgenden weiter erläutert.

## 5.1. Aufsetzen der Vektordatenbank Weaviate

Für das Aufsetzen von Weaviate werden zwei Komponenten benötigt. Zum einen die Vektordatenbank selbst. Zum anderen ein Transformer, welcher Text entgegen-

nimmt, und diese in Vektoren umwandelt, sodass diese in der Datenbank gespeichert werden können. Um die beiden Komponenten aufzusetzen hat der Autor Docker verwendet. Das entsprechende docker-compose-yml File ist im Anhang zu finden. Hier werden die beiden Komponenten definiert. Unter *t2v-transformers* wird der Transformer konfiguriert. Es wird das Image eines bereits vortrainierten Transformers verwendet. Unter *weaviate* wird die Datenbank konfiguriert. Hier wird mithilfe von den Environment-Variablen $DEFAULT\_VECTORIZER\_MODULE$, $ENABLE\_MODULES$ und $TRANSFORMERS\_INFERENCE\_API$ konfiguriert, welcher Transformer verwendet werden soll. So wird konfiguriert, dass der Transformer der anderen Komponente verwendet werden soll.

Mithilfe der Dependecy *io.weaviate:client:4.0.1* verwendet der Autor die Client API von Weaviate. So wird nun eine Verbindung zu der Vektordatenbank aufgebaut. Diese beinhaltet zu diesem Zeitpunkt noch keine Daten. Bevor die Daten in die Datenbank eingespielt werden, muss das Schema der Daten angegeben werden. Der entsprechende Code ist ebenfalls im Anhang zu finden. Er definiert eine Klasse *Document*. Diese Klasse beinhaltet die Properties *documentUrl*, *h1*, *h2* und *p*. Es werden in dieser Klasse also die URL des Dokuments, sowie die Inhalte aller h1-, h2 und p-Tags gespeichert. Außerdem wird als Vectorizer *text2vec-transformers* konfiguriert.

## 5.2. Einspielen der Daten in Weaviate

Um nun die Daten aus Confluence in Weaviate einzuspielen, müssen zuerst die Daten aus Confluence exportiert werden. Beim Export von Confluence Seiten werden HTML-Dateien generiert. Diese müssen preprocessed werden, um dem zuvor definierten Schema zu entsprechen. Es werden zuerst mithilfe von regulären Ausdrücken alle Inhalte von h1-, h2- und p-Tags herausgefiltert. Anschließend werden Punctuations und Stopwords entfernt und die Inhalte druchlaufen einen Tokenizer und einen Stemmer. Punctuations sind Zeichen, wie die folgenden: *.,;:*. Stopwords sind Wörter, welche für einen Leser notwendig sind, aber für die Verarbeitung durch einen Algorithmus als unwichtig erachtet werden. Beispiele für Stopwords sind *aber, denn, der*. Ein Tokenizer trennt einen Text in einzelne Wörter auf. Aus einem Text, wie *das deployment erfolgt durch ein bash-skript* wird also ein Array, welches folgendermaßen aussieht:

$$[\text{"}das\text{"}, \text{"}deployment\text{"}, \text{"}erfolgt\text{"}, \text{"}durch\text{"}, \text{"}ein\text{"}, \text{"}bash\text{"}, \text{"}skript\text{"}].$$

Ein Stemmer bestimmt den Wortstamm für die einzelnen Wörter auf Basis von Grammatikregeln. Konkret verwendet die Software den Porter-Stemmer-Algorithmus. Aus dem Wort *deployment* wird dadurch beispielsweise *deploy*. Duplikate von Wörtern werden anschließend verworfen. Nachdem die Inhalte dieses Preprocessing durchlaufen haben, werden sie in die Datenbank eingespielt.

Im Kapitel *Vektorindizes* wurde bereits von Performancegründen gesprochen, aus denen das Preprocessing durchgeführt wird. Da nun die einzelnen Schritte des Preprocessings erklärt wurde, kann auch erklärt werden, warum das Preprocessing die Performance beim indizieren erhöht. Zuallererst werden viele Wörter gänzlich verworfen, weil sie Stopwords sind. Durch den Stemmer werden anschließend ähnliche Wörter zusammengruppiert. Die Wörter *Heizung* und *heizen* stammen beide vom gleichen Wortstamm *heiz*. Das bedeutet, dass aus zwei verschiedenen Wörtern, welche beide indiziert werden müssen, ein einziges Wort gemacht wird. Denn am Ende werden Duplikate, wie bereits erwähnt, verworfen. Die Anzahl der Wörter, welche indiziert werden müssen, wird dadurch reduziert, und damit auch die Last auf dem Transformer, welche die Vektoren für die Wörter berechnen muss.

## 5.3. Verwendung der Suchfunktion von Weaviate

Weaviate verwendet eine API, welche GraphQL queries entgegennimmt. Um eine Suche durchzuführen muss ein GET-Request durchgeführt werden. Nun muss angegeben werden, welche Klasse aus der Datenbank abgefragt werden soll. In diesem Fall *Document*.

### 5.3.1. Verwendung der Semantische Suche

Um die semantische Suche zu verwenden, muss die NearText Funktion verwendet werden.

### 5.3.2. Verwendung von Filtern

TODO: Ergänzen

### 5.3.3. Darstellung der Ergebnisse

TODO: Ergänzen

# 6. Vergleich der Suchfunktionen

Für den Vergleich von Suchfunktionen müssen auf Grundlage der Anwendungsfälle realistische Suchanfragen entwickelt werden. Zu diesem Zweck sei angenommen, dass ein neuer Softwareentwickler in einem bestehenden Softwareprojekt eingearbeitet wird. Es wird also der Anwendungsfall des Onboardings betrachtet.

Zuerst wird dem Softwareentwickler aufgetragen, sich das *Getting Started* im Confluence durchzulesen. Er macht also die Sucheingabe **Getting Started**, und erwartet ein Dokument mit ebendieser Überschrift. TODO: Sucheingabe weiter beschreiben

Um sich mit der Software vertraut zu machen, wird dem neuen Softwareentwickler aufgetragen, einmal die Anwendung bei sich lokal zu starten. Nachdem er sie gestartet hat, soll er sich mit der Funktionalität der Software vertraut machen. Um die Anwendung lokal zum Laufen zu bringen, sucht der Softwareentwickler nach einer **Installationsanleitung**. Er gibt also genau dies als Suchbegriff ein und erwartet als Ergebnis ein Dokument, welches beschreibt, wie die Software installiert wird. TODO: Sucheingabe weiter beschreiben

Nachdem die Software installiert ist, startet er die Anwendung. Dazu muss er sich anmelden. Er sucht also nach **Testdaten**, welche die Anmeldedaten enthalten. Nachdem er sich angemeldet hat, möchte er sich eine Übersicht über die Funktionen der Software verschaffen. Dazu sucht er nach den **Use-Cases** der Anwendung. Nachdem er die Use-Cases gefunden hat, probiert er mehrere davon aus.

Einer der Use-Cases ist die Versendung von Korrespondenzen am Ende eines Workflows. Um diesen Use-Case genauer nachvollziehen zu können gibt er den Suchbegriff **Korrespondenz** ein.

Im Anhang ist eine Auflistung der beispielhaften Sucheingabe zu finden.

## 6.1. Diskussion des Studienaufbaus

TODO: Ergänzen

## 6.2. Durchführung der Studie

TODO: Ergänzen

## 6.3. Diskussion der Ergebnisse

TODO: Ergänzen

# 7. Zusammenfassung und Ausblick

TODO: Ergänzen

## 7.1. Zusammenfassung

TODO: Ergänzen

## 7.2. Ausblick

TODO: Ergänzen

# 8. Literaturverzeichnis

[1] Antoniol, Canfora, Casazza und De Lucia. „Information retrieval models for recovering traceability links between code and Documentation“. In: *Proceedings International Conference on Software Maintenance ICSM-94* (2000).

[2] Sonia Haiduc, Gabriele Bavota, Andrian Marcus, Rocco Oliveto, Andrea De Lucia und Tim Menzies. „Automatic query reformulations for text retrieval in software engineering“. In: *2013 35th International Conference on Software Engineering (ICSE)* (2013).

[3] Y. Li, D. McLean, Z.A. Bandar, J.D. O'Shea und K. Crockett. „Sentence similarity based on semantic nets and corpus statistics“. In: *IEEE Transactions on Knowledge and Data Engineering* 18.8 (2006), S. 1138–1150.

[4] Pavel Sirotkin. *On Search Engine Evaluation Metrics*. 2012.

[5] Christoph Treude, Mathieu Sicard, Marc Klocke und Martin Robillard. „TaskNav: Task-based navigation of software documentation“. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering* (2015).

[6] Xin Ye, Hui Shen, Xiao Ma, Razvan Bunescu und Chang Liu. „From word embeddings to document similarities for improved information retrieval in software engineering“. In: *Proceedings of the 38th International Conference on Software Engineering* (2016).

# Anhang

# A. Anhang

## A.1. docker-compose.yml File für Weaviate

```
version: '3.4'
services:
  weaviate:
    image: semitechnologies/weaviate:1.18.3
    ports:
      - "8080:8080"
    environment:
      QUERY_DEFAULTS_LIMIT: 20
      AUTHENTICATION_ANONYMOUS_ACCESS_ENABLED: 'true'
      PERSISTENCE_DATA_PATH: "./data"
      DEFAULT_VECTORIZER_MODULE: text2vec-transformers
      ENABLE_MODULES: text2vec-transformers
      TRANSFORMERS_INFERENCE_API: http://t2v-transformers:8080
      CLUSTER_HOSTNAME: 'node1'
    volumes:
      - /var/weaviate:/var/lib/weaviate
  t2v-transformers:
    image: semitechnologies/transformers-inference:sentence-transformers-
        msmarco-distilroberta-base-v2
    environment:
      ENABLE_CUDA: 0
```

## A.2. Initialisieren des Schemas in Weaviate

```
  client.schema()
    .classCreator()
    .withClass(
      WeaviateClass.builder()
        .className(ConfluenceDataService.DOCUMENT_CLASS)
        .properties(
          buildProperties(
            mapOf(
              ConfluenceDataService.DOCUMENT_URL to WEAVIATE_TEXT_DATATYPE
              ConfluenceDataService.H1_TAG to WEAVIATE_TEXT_DATATYPE,
              ConfluenceDataService.H2_TAG to WEAVIATE_TEXT_DATATYPE,
              ConfluenceDataService.PARAGRAPH_TAG to WEAVIATE_TEXT_DATATYP
```

```
            )
          )
        )
        .vectorizer(VECTORIZER)
        .build()
).run()
```

# Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe alle Stellen, die ich aus den Quellen wörtlich oder inhaltlich entnommen habe, als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Essen, den 29. Juni 2023