**Gottfried Wilhelm**
**Leibniz Universität Hannover**
**Fakultät für Elektrotechnik und Informatik**
**Institut für Praktische Informatik**
**Fachgebiet Software Engineering**

# Halbautomatische Generierung eines Glossars während der Dokumentation von Anforderungen

## Masterarbeit

im Studiengang Informatik

von

**Sebastian Meyer**

**Prüfer: Prof. Dr. Kurt Schneider**
**Zweitprüfer: Prof. Dr. Nicola Henze**
**Betreuer: M. Sc. Eric Knauss**

**Hannover,** ***7. November 2007***

# Danksagungen

*Ich möchte mich bei Herrn M.Sc. Eric Knauss für die vorbildhafte Betreuung der Masterarbeit und alle interessanten und fruchtbaren Gespräche während der Entstehung bedanken.*

*Ich bedanke mich bei Prof. Dr. Kurt Schneider für die hilfreiche Begleitung während meines Masterstudiums.*

*Ich bedanke mich bei meiner Familie für die Unterstützung während des Studiums und den Rückhalt in schwierigen Zeiten.*

*Ich bedanke mich bei allen denen, die bis spät in die Nacht meine Masterarbeit Korrektur gelesen haben. Verbliebene Fehler sind allein mein Verdienst.*

*Mein Dank gilt allen denen Kommilitonen, die mich auf meinem Weg zum Master begleitet haben. Danke Arne, Benjamin, Dennis, Hannes, Leif, Moritz, und Stefan.*

*Mein Dank gilt Christina Bertz für die Unterstützung in schweren Zeiten und dafür, dass sie mir immer den Rücken freihält.*

*Ich bedanke mich bei Mareike Heups, die einen langen und manchmal sehr steinigen Weg mit mir gegangen ist.*

*Schließlich, aber nicht zuletzt, möchte ich mich bei Herrn Torben Flenner bedanken, der mich trotz Danks in der Bachelorarbeit immer noch erträgt und immer für mich da ist, wenn ich einen echten Freund brauche.*

## Zusammenfassung

Ein kritischer Fehler bei Software Projekten sind missverstandene Begriffe. Um Begriffe eindeutig und für alle nachschlagbar zu dokumentieren bietet sich ein Glossar an.

Diese Arbeit untersucht ob und in wie weit es möglichist, mit Hilfe von einfachen Methoden Informationen aus natürlich-sprachlichen Texten zu extrahieren die bei der Erstellung eines Glossar helfen. Einfache Methoden meint hierbei vorallem den Verzicht auf große, schwer zu erzeugende Wissensbasen.

Diese Methoden sollen außerdem die Erstellung und Verwaltung eines Glossar erlauben und sich als Bibliothek in beliebige andere Projekte einbinden lassen.

An einer prototypischen Implementation dieser Bibliothek am Beispiel der Use Case Design Environment soll getestet werden, wie Feedback und die Unterstützung des Byproduct-Konzepts dem Benutzer helfen können das Glossar während der Anforderungsdokumentation zu erstellen.



## Abstract

Misunderstood terms are a crucial mistake in software projects. Glossaries are appropriate tools to provide unique definitions to all stakeholders.

This thesis examines if and to what extend the extraction of information from natural-language texts using simple methods is feasible in supporting the creation of a glossary. Simple methods are those not relying on large, hard to create knowledge bases.

These methods should further allow the creation and administration of a glossary. Also, integration with other projects as a library should be possible.

Using a prototypical implementation of said library, this thesis reviews its integration into the Use Case Design Environment to find out to what extent feedback and support for the byproduct concept help the user in the creation of the glossary during the phase of requirements documentation.

# Inhaltsverzeichnis

# 1. Einführung

## 1.1. Motivation

Bei einem Softwareprojekt entfällt ein großer Teil der Kosten auf die Fehlerbehebung. Fehler, welche erst spät entdeckt werden sind aufwendiger und damit kostenintensiver zu beheben.

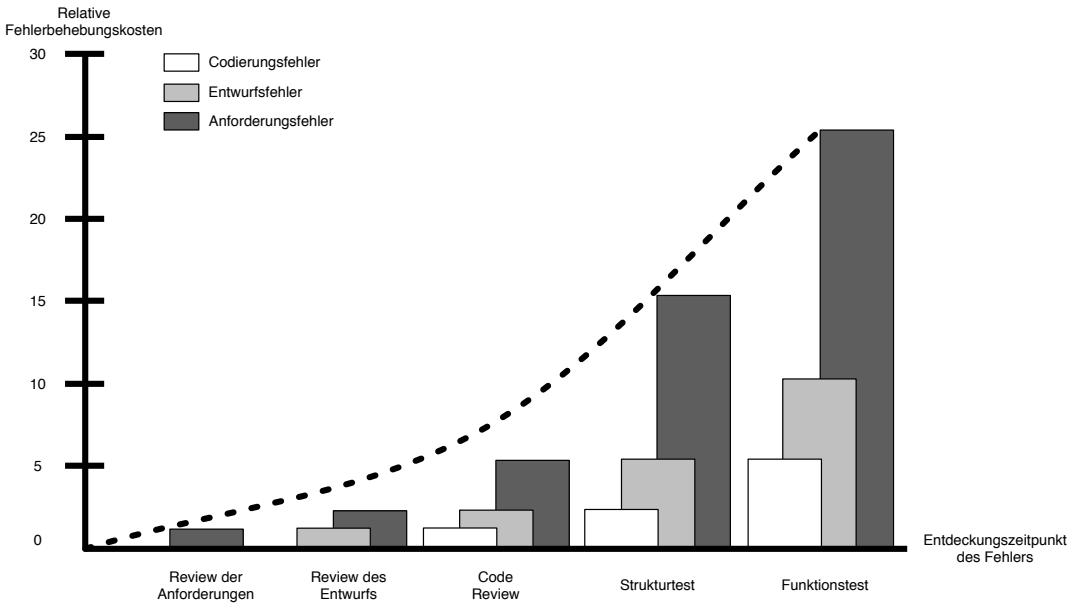


Abbildung 1.1.: Relative Fehlerbehebungskosten nach [KF91]

Wie Abbildung 1.1 zeigt, steigen die Kosten für die Behebung von Fehlern die aus falschen Anforderungen entstanden sind exponentiell an.

Kostenoptimal wäre es also alle Fehler in den Anforderungen spätestens während eines Reviews des Entwurf zu finden, besser noch während des Anforderungsreviews.

Eine häufig auftretende Quelle für Fehler in Anforderungen sind missverstanden Begriffe. Beim Einstieg in eine fachfremde Domäne ist eine gründliche Einarbeitung unumgänglich. Jedoch ist es fast unmöglich sich innerhalb kurzer Zeit so intensiv in eine fremde Domäne einzuarbeiten wie dies Menschen getan haben die schon jahrelang in ihr arbeiten.

Um nun beiden Parteien einen einheitlichen Sprachgebrauch zur Verfügung zu stellen

hat sich die Einführung eines Glossars bewährt. In diesem werden domänenspezifische Fachbegriffe aufgenommen und ihre Bedeutung erklärt um so eine eindeutige Referenz für einen späteren Bezug zu erhalten.

## 1.2. Problemstellung

Ziel dieser Arbeit ist es eine Möglichkeit zu schaffen aus natürlich-sprachlichen Texten mit einfachen Mitteln Vorschläge für die Aufnahme in ein Glossar zu extrahieren. Außerdem soll ein Glossar angelegt und bearbeitet werden können. Bei diesen Schritten der Glossarerstellung und dessen Pflege soll der Benutzer vom zu entwickelnden Programm unterstützt werden.

Um diese Funktionen zu testen und in eine reale Umgebung zu stellen, wird das entwickelte Konzept in die am Fachgebiet Software Engineering der Leibniz Universität Hannover[1] entwickelte *Use Case Design Environment* (UCDE) integriert.

Da ein Glossar abhängig von der Menge der Anforderungen und der damit einhergehenden häufigeren Verwendung uneindeutiger Begriffe schnell wächst, wäre es äußerst uneffektiv das Glossar erst nach der Erhebung der Anforderungen zu erstellen. Dem am Fachgebiet geprägten *Byproduct-Konzept* folgend wird das fertige Plugin dem Benutzer die Möglichkeit geben ein Glossar während der Anforderungserhebung assistiert zu erzeugen, ohne dabei die momentan durchgeführte Aufgabe länger als nötig zu unterbrechen.

Eine weiteres wichtiges Konzept bei der Integration in die UCDE ist die Unterstützung des Benutzers durch Mechanismen zur Rückmeldung von im Zusammenhang mit dem Glossar wichtigen Informationen.

## 1.3. Einordnung in das Software Engineering

Die Erstellung und Bearbeitung eines Glossars wird primär in der Phase der Anforderungserhebung stattfinden. Abbildung 1.2 zeigt das Referenzmodell für Anforderung und Entwurf wie in [Sch07] vorgestellt.

Die Erstellung eines Glossar berührt hierbei sämtliche Bereiche der Systemanalyse.

So führt eine Identifikation der Stakeholder dazu, zu erkennen welche Domänen für das Projekt relevant sind.

Sodann können die für diese Domäne wichtigen Begriffe identifiziert werden und zwischen den beteiligten Personen so abgestimmt werden, dass eine belastbare Definition entsteht.

Das Glossar dient gleichzeitig zur Dokumentation der domänen-spezifischen Begriffe eines Projekts und lässt sich als Referenz bei Unklarheiten sowie bei der Verifikation

[1] http://www.se.uni-hannover.de

bzw. Validierung heranziehen.

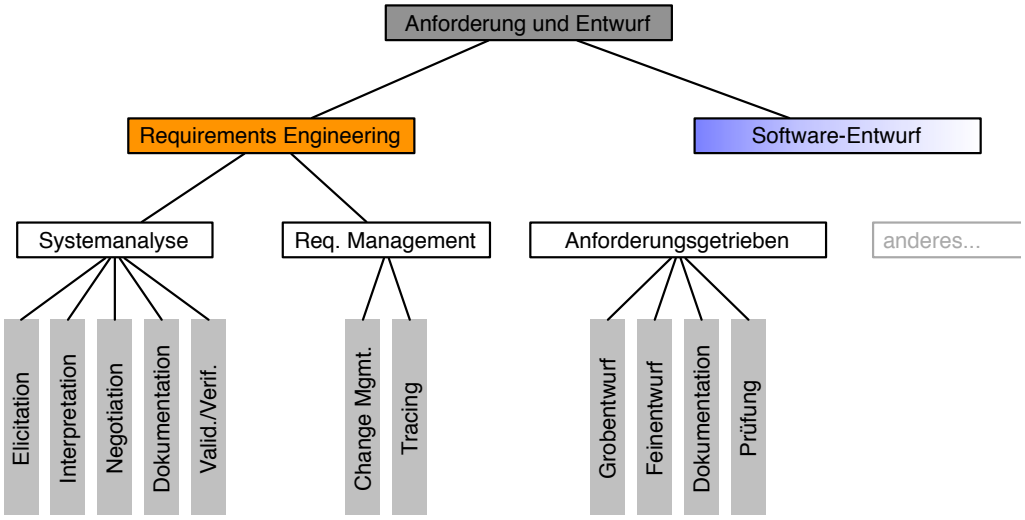


Abbildung 1.2.: Referenzmodell Anforderung und Entwurf nach [Sch07]

Die in dieser Arbeit entwickelten Technologien dienen primär den Bereichen *Interpretation* und *Dokumentation*. Zwar muss auch die *Elicitation* durchgeführt werden um überhaupt festzustellen welche Domänen - und damit einhergehend welche Begriffe - zu berücksichtigen sind. Hierbei wird der Benutzer jedoch nicht aktiv unterstützt.

## 1.4. Struktur der Arbeit

In Kapitel 2 werden kurz die Anforderungen an die zu entwickelnden Programmteile beleuchtet.

Kapitel 3 erklärt die in dieser Arbeit verwendeten Begriffe. Außerdem wird auf die verwendeten Technologien eingegangen.

Im folgenden Kapitel 4 werden die Methoden dargestellt, die für die Erstellung und Verwaltung eines Glossars benötigt werden. Ebenso werden Methoden für die Informationsextraktion aus natürlich-sprachlichem Text vorgestellt.

Kapitel 5 beschreibt ein Anwendungsbeispiel für die Einbindung des in Kapitel 4 entwickelten Glossarkerns. Hier wird eine beispielhafte Implementierung in die Use Case Design Environment vorgestellt.

Kapitel 6 stellt Qualitätsmodelle für alle beteiligten Komponenten auf und stellt fest wodurch diese beeinflusst werden.

In Kapitel 7 werden die erarbeiteten Ergebnisse bewertet und gegen die Literatur abgegrenzt. Schließlich wird ein Ausblick über zukünftige Entwicklungen gegeben.

# 2. Anforderungen

## 2.1. Technische Randbedingungen

Das fertige Produkt setzt sich aus einem Plugin für die Use Case Design Environment (UCDE) und einer Bibliothek zur Erstellung und Verwaltung eines Glossars sowie zur Extraktion von Glossar-relevanten Informationen aus natürlich-sprachlichem Text zusammen.

Da das Plugin in der Umgebung des Hosts läuft sind die Schnittstellen um auf die Daten des Anforderungsprojektes zuzugreifen sowie die eigenen darzustellen bereits vorgegeben. Ebenso ist die Programmiersprache auf Java in der Version 1.6 festgelegt.

Für die Bibliothek wird ebenfalls Java als Programmiersprache benutzt um eine einfache Benutzung durch das Plugin zu ermöglichen.

## 2.2. Benutzerkreis

Ausgehend von der der bisherigen Anwendung der UCDE lassen sich folgende Benutzerrollen ermitteln welche die UCDE mit dem zu entwickelnden Plugin benutzen.

- Anforderungs-Analyst

  Der Anforderungs-Analyst benutzt die UCDE primär um Anforderungen zu erfassen. Wenn es ihm auf einfach Weise möglich ist möchte er hierbei auch gleich ein Glossar anlegen, jedoch ist es für ihn kein Hindernis, wenn dies nicht möglich ist.

- Nachschlagender

  Der Nachschlagende benutzt die UCDE um durch vorhandene Anforderungen mit Glossar zu blättern. Damit er sofort erkennt dass ein Begriff im Glossar enthalten ist und somit evtl. eine andere als die ihm bekannte Bedeutung besitzt, müssen im Glossar eingetragene Wörter deutlich erkennbar markiert werden.

- Glossar-Autor

  Der Glossar-Autor benutzt die UCDE primär um ein Glossar anzulegen und dieses zu pflegen. Für den Fall dass er bestehende Anforderungen mit einem Glossar ergänzt möchte er - vom Plugin unterstützt - vorhandene Begriffe hinzufügen. Außerdem benötigt er Methoden zur Messung der Qualität des erstellten Glossars.

Für die Rollen *Anforderungs-Analyst* und *Nachschlagender* gilt, dass sie schon zum ursprünglichen Benutzerkreis der UCDE gehören. Es ist also wichtig für diese beiden Rollen die Einbettung des Plugins dergestalt zu realisieren, dass ihr bisheriger Workflow davon nicht gestört bzw. unterbrochen wird.

Dies bedeutet, dass die Integration der neuen Werkzeug so geschehen muss, dass sie vom Benutzer zum einen leicht zu ignorieren sind. Zum anderen soll auch die Benutzung der neuen Funktionen wie z.B. das Hinzufügen eines Glossareintrags so implementiert werden, dass die eigentliche Aufgabe des Benutzers nicht länger als nötig unterbrochen wird.

Die Rolle *Glossar-Autor* wird implizit durch die Aufgabe des Plugins neu geschaffen. Die primäre Aufgabe dieser Benutzerrolle ist es ein Glossar zu bearbeiten. Es kann also davon ausgegangen werden, dass Aktionen wie das Hinzufügen eines neuen Begriffs zum Glossar die primären Aktionen des Benutzers sind. Für diese Benutzerrolle muss nicht darauf geachtet werden, dass diese Aktionen nebenläufig geschehen.

## 2.3. Eigenschaften der Implementierung

### Trennung von Glossarkern und Plugin

Die zu erstellende Implementierung soll in zwei Projekte aufgeteilt werden: Einen Glossarkern und ein Plugin für die Use Case Design Environment.

Der Glossarkern enthält Methoden zur Erstellung und Verwaltung eines Glossars. Ebenso werden in diesem Methoden zur Extraktion von glossarbezogenen Informationen aus natürlich-sprachlichem Text integriert.

Das Plugin benutzt den Glossarkern um eine beispielhafte Integration des Glossarkonzeptes in die Use Case Design Environment zu demonstrieren. Hierbei soll Wert auf die Unterstützung der beiden wesentlichen Konzepte der UCDE gelegt werden: *Feedback* und das *Byproduct-Konzept*. Zusätzlich können für die Benutzerrolle des *Glossar-Autors* auch Funktionen integriert werden, die nicht dem Byproduct-Konzept entsprechen.

### Einfaches Hinzufügen

Das Hinzufügen von neuen Glossareinträgen darf keine längere Eingewöhnungsphase benötigen. Dies würde dazu führen, dass nur wenige Benutzer die primär mit der Erfassung von Anforderungen beschäftigt sind die neuen Funktionen zur Glossarerstellung benutzen. Hierdurch ergibt sich, dass mindestens das Hinzufügen über das bereits vorhandene Kontextmenü des Projektbaums möglich sein muss um schon erfahrene Benutzer nicht zu verwirren.

Weiterhin werden zusätzliche Möglichkeiten für das Hinzufügen von Glossareinträge geschaffen. Hierbei ist die Beachtung des Byproduct-Konzepts ausschlaggebend. Dies

führt dazu, dass die neuen Möglichkeiten kontextsensitiv auf den Benutzer und den aktuell bearbeiteten Teil der Anforderungen eingehen und ihn nicht in seiner aktuellen Tätigkeit unterbrechen sollen.

### Unterstützung durch Vorschläge

Die Beachtung der Rolle des *Anforderungsanalysten* führt dazu, dass für diese eine Möglichkeit zur Verfügung gestellt werden muss, mit der Begriffe dem Glossar hinzugefügt werden können ohne sich hierzu tiefer als nötig mit dem Anforderungsprojekt zu beschäftigen.

Um dies zu ermöglichen soll eine Liste mit Vorschlägen generiert werden. In dieser werden dem Benutzer die möglicherweise für sein Glossar relevante Vorschläge präsentiert.

Wichtig ist hierbei, dass die Vorschläge den Benutzer zwar unterstützen ihn jedoch nicht bevormunden. Deshalb muss es möglich sein die Vorschläge zu ignorieren ohne dadurch Nachteile in der Bedienung und Funktionalität des Plugins zu bekommen. Ebenso soll es möglich sein die Vorschläge so anzupassen, dass bestimmte Arten von Vorschlägen wie z.B. Bindewörter von vornherein aussortiert werden.

### Bewertung der Qualität

Für Benutzer mit der Rolle *Glossar-Autor* ist es nötig einen Überblick über die Qualität eines Glossars zu erhalten. Dies schließt auch die Qualität der einzelnen Glossareinträge mit ein.

Um Qualitätskennzahlen für Glossareinträge und damit auch für das gesamte Glossar zu erhalten, werden für die Einträge Metriken definiert. Diese tragen zur Bildung einer Qualitätskennzahl bei. Die Ergebnisse dieser Metriken sollen dem Benutzer in geeigneter Weise zur Verfügung gestellt werden.

### Markieren von Einträgen

Wie oben beschrieben ist ein großes Problem bei der Erhebung von Anforderungen das Falschverstehen von domänenspezifischen Begriffen. Zwar hilft ein Glossar dabei eine eindeutige Definition dieser Begriffe zu dokumentieren, jedoch setzt dies voraus, dass der Leser eines Anforderungsdokuments auch im Glossar nachschlägt.

Um die Wahrscheinlichkeit zu verringern, dass ein Benutzer im Glossar aufgenommene Begriffe nicht nachschlägt, sollen diese Begriffe in den bereits bestehenden Editoren für die Dokumente des Anforderungsprojekts erkennbar markiert werden.

Dies führt dazu, dass es nicht nötig ist für jeden Begriff im Text zu recherchieren ob er im Glossar aufgenommen wurde. Gerade bei Anforderungsprojekten mit einer großen

Anzahl von Glossareinträgen wird diese Methode die Nützlichkeit des Glossars verbessern.

### Integration in die Erfahrungsrückmeldung

Ein wesentlicher Bestandteil der Use Case Design Environment ist die heuristische Rückmeldung von Erfahrungen an den Benutzer. Diese wird durch ein in [Kit07] entwickeltes Plugin implementiert. Die Erfahrungen welche das Plugin dem Benutzer anbietet lassen sich per JavaScript um eigene Heuristiken erweitern

Das in dieser Arbeit entwickelte Plugin soll sich in diese Erfahrungsrückmeldung integrieren. Um dies zu erreichen sollen die erstellten Metriken dem Erfahrungs-Plugin zugänglich gemacht und JavaScript-Fragmente für die Auswertung der Metrikkennzahlen erstellt werden.

# 3. Grundlagen

## 3.1. Glossar

**Definition 3.1.1 (nach [Dud06])**
*Ein Glossar ist ein selbstständiges oder als Anhang eines bestimmten Textes erscheinendes Wortverzeichnis mit Erklärungen dieser Wörter.*

Damit ein Glossar als effektives Nachschlagewerk für eine bestimmte Aufgabe dienen kann, ist es nötig gesuchte Wörter schnell zu finden. Für die Erstellung und Verwaltung eines Glossars ist es weiterhin nötig die gewünschten Begriffe derart einzutragen, dass nicht mehrere Einträge im Glossar enthalten sind, die lediglich unterschiedliche Formen des gleichen Worts sind.

Um sowohl dem Ersteller eines Glossars, als auch denjenigen Benutzern gerecht zu werden, welche zwar mit natürlicher Sprache jedoch nicht mit der Benutzung der Glossarerweiterung vertraut sind, wird ein Standard für die in ein Glossar aufzunehmenden Wörter benötigt.

Im vorliegenden Fall wird davon ausgegangen, dass die Texte des Anforderungsprojekts überwiegend in deutscher Sprache verfasst sind. Aus diesem Grund halten sich die erstellten Glossare an die Konventionen der Dudenredaktion, welche einen de facto Standard für Nachschlagewörter der deutschen Sprache darstellen. Dies bedeutet, dass ein Wort immer in seiner Grundform in das Glossar aufgenommen wird.

## 3.2. Glossareintrag

Jeder Eintrag in einem Glossar hat zwei wesentliche Merkmale.

Der *Name* des Glossareintrags ist die Bezeichnung unter der der Eintrag im Glossar verzeichnet ist. Dieser ist im vorliegenden Fall immer die Grundform des beschriebenen Begriffs.

Die *Beschreibung* eines Glossareintrags ist die Definition des durch seinen Namen identifizierte Begriffs. Optimalerweise sind Beschreibungen so gestaltet, dass sie selbstständig aussagekräftig sind, also keine anderen erklärungsbedürftigen Begriffe verwenden.

## 3.3. Wörterbücher

Um den Begriff des *Wörterbuchs* von dem des Glossars abzugrenzen gilt für ein Wörterbuch im Kontext dieser Arbeit die folgende Definition:

**Definition 3.3.1**
*Ein Wörterbuch ist ein Verzeichnis von Wörtern und aller von diesen abgeleiteten Formen einer Sprache ohne Erklärung dieser.*

Für Wörterbücher gibt es zwei mögliche Formen die in der Praxis benutzt und z.B. in [Lei07] beschrieben werden.

1. Vollformenlexikon

   Ein Vollformenlexikon beinhaltet nicht nur die Grundformen sondern ebenso alle davon existierenden Ableitungen. So ist es möglich durch einfachen Textvergleich mit den Einträgen im Wörterbuch zu erkennen ob ein Begriff verzeichnet ist oder nicht.

2. Grundformenlexikon

   In einem Grundformenlexikon stehen lediglich die Grundformen aller verzeichneten Wörtern mit ihren zugehörigen Flexionsklassen. Jede Wortform die nachgeschlagen werden soll muss zunächst mittels geeigneter morphologischer Regeln auf eine Grundform reduziert werden. Diese kann dann wie im Falle des Vollformenlexikons mittels Textvergleich überprüft werden. Die morphologischen Regeln ergeben sich hierbei durch die entsprechenden Flexionsklasse.

Beim direkten Vergleich dieser beiden Arten von Wörterbüchern ergibt sich, dass ein Vollformenlexikon effizienter beim Nachschlagen ist, da keine Verarbeitung des nachzuschlagenden Begriffs mehr stattfinden muss. Jedoch ist es hinsichtlich Speicherbedarf dem Grundformenlexikon deutlich unterlegen. Außerdem enthält ein Vollformenlexikon normalerweise keine morphologischen Regeln um die die enthaltenen Formen in ihre entsprechende Grundform zu überführen und umgekehrt.

So wird eine Vollformenlexikon vor allem dann in Frage kommen, wenn schnelle Verarbeitung gefragt oder der Wortschatz klein genug ist um ohne Probleme ein Vollformenlexikon im Speicher unterzubringen. Ein Grundformenlexikon ist die bessere Wahl, wenn Wert auf die morphologischen Regeln gelegt wird.

## 3.4. Grundformen

Ein Wort kann in unterschiedlichen Formen auftreten. So kann ein Nomen z.B. in seinen deklinierten Formen im Text auftreten. In diesem Fall ist es für einen menschlichen Leser einfach zu erkennen, dass diese unterschiedliche Deklinationen des gleichen Worts sind. Für ein Computerprogramm jedoch ist dies nicht einfach möglich. Um

Vergleichbarkeit herzustellen muss jede Form auf ihre eindeutige *Grundform* zurückgeführt werden.

**Definition 3.4.1**
*Die Grundform eines Wortes wird in der Linguistik üblicherweise wie folgt definiert:*

- *Die Grundform eines Nomens ist der Nominativ Singular*
- *Für Verben ist die Grundform der Infinitiv Präsens Indikativ*
- *Für alle anderen Arten von Wörtern ist die Grundform das Wort selbst*

## 3.5. Methoden zur Grundformbildung

Da Anforderungen in einem Projekt überlicherweise in natürlich-sprachlichem Text vorliegen, ist es nötig ein Verfahren zu benutzen um für eine Wortform die zugehörige Grundform zu bestimmen. Dies ist zum einen nötig um ein Wort dem Glossar hinzuzufügen, da die in Abschnitt 3.1 vorgestellte Definition eines Glossars dies fordert. Zum anderen um zu erkennen ob eine Wortform (vertreten durch ihre Grundform) im Glossar enthalten ist.

Im folgenden werden drei Methoden vorgestellt mit denen es möglich ist aus einer gegebenen Wortform eine Grundform abzuleiten. Wesentliches Kriterium für diese Methoden ist es, dass eine Wortform immer auf die gleiche Grundform zurückgeführt wird.

Gesucht ist also eine Funktion

$$f : \mathrm{w} \mapsto \mathrm{g}$$

mit $w, g \in \mathrm{String} \backslash \{\varepsilon\}$ und $w$ = beliebige Wortform, $g$ = Grundform von $w$

### Heuristiken

Daniel C. Dennet beschreibt in [Den96] Heuristiken als gewagte Methoden, welche - im Gegensatz zu Algorithmen - nicht garantieren Ergebnisse bzw. genau die gewünschten Ergebnisse zu erbringen. Ihre Aufgabe ist es hauptsächlich einen großen, komplexen oder auf sonstige Art unhandlichen Suchraum derart zu beschneiden, dass er beherrschbar wird.

Eine Heuristik für die Grundformenbildung könnte zum Beispiel sein:

„Entferne alle s am Ende eines Wortes“

Hier ist zu erkennen, dass Heuristiken tatsächlich nicht immer das gewünschte Ergebnis liefern.

Es gilt $f(\text{Vogels}) = \text{Vogel}$, $f(\text{Wortes}) = \text{Worte}$, $f(\text{Hans}) = \text{Han}$.

Diese Ergebniss entsprechen nicht den in Definition 3.4.1 festgelegten Anforderungen an eine Grundform bzw. sind linguistisch nicht korrekt.

Ein weiteres Problem ist, dass die Ergebnisse von Heuristiken nicht unbedingt vorhersagbar sind. Beispiele hierfür findet man in der Virenscanner-Technologie. Hier wird mit Heuristiken nach noch unbekannten Viren gesucht. Hierbei treten jedoch gelegentlich sog. *false positives* auf. Dies bezeichnet einen Zustand in dem die Heuristik fälschlicherweise signalisiert, dass das gesuchte Ergebnis gefunden sei.

Angewandt auf die Grundformenbildung ist ein false positive also eine Rückgabe des Algorithmus, welche nicht Definition 3.4.1 genügt.

Das Problem der geringen Vorhersagbarkeit und die damit verbundene Tatsache, dass Heuristiken nicht unbedingt definitionsgemäße bzw. linguistisch korrekte Grundformen als Ergebnis liefern führt dazu, dass Heuristiken als Technik der Grundformgewinnung für ein natürlichsprachliches Glossar nicht benutzbar sind.

### Stemming

*Stemming* ist ein Verfahren aus dem Information-Retrieval um verschiedene Formen auf einen gemeinsamen Stamm zurückzuführen. Ein Stamm ist hierbei nicht unbedingt identisch mit dem linguitisch korrekten Wortstamm. Ein häufig verwendetes Verfahren hierfür ist das nach Martin Porter benannte Porter-Stemming, welches in [RRP80] für die englische Sprache vorgestellt wurde.

Ein Porter-Stemmer führt *fishing*,*fished* und *fish* auf den Stamm *fish* zurück. Diese als overstemming bezeichneten Fehler führen dazu, dass beim stemming möglicherweise die unterschiedliche Semantik zweier Begriffe verloren geht.

Für Anwendungszwecke wie dem Information-Retrieval ist diese Einschränkung vollkommen ausreichend da es hier nur darauf ankommt große Datenmengen schnell zu durchsuchen. In diesem Fall kann vor dem eigentlichen Stemming noch eine Vorverarbeitungsphase durchgeführt werden, in der Wörter mit unterschiedlicher Bedeutung von einander getrennt werden. Hierdurch lässt sich das Problem umgehen, dass Wörter mit unterschiedlicher Bedeutung auf den gleichen Wortstamm zurückgeführt werden.

Ähnlich wie bei Heuristiken kann hier wieder festgestellt werden, dass der Algorithmus Ergebnisse liefert, welche nicht Definition 3.4.1 genügen. Außerdem führt das Problem des overstemmings dazu, dass evtl. Begriffe unterschiedlicher Bedeutung auf eine gemeinsamen Glossareintrag reduziert werden. Dies ist für das Ziel eines Glossar, eine eindeutige Erklärung zu einem Begriff zu liefern ein großes Hindernis.

Diese Beobachtungen zeigen, dass sich stemming-Verfahren für ein natürlich-sprachliches Wörterbuch nicht einsetzen lassen.

## Lemmatisierung

Bei der Lemmatisierung wird eine flektierte bzw. variierte Wortform auf ihr Lemma zurückgeführt, indem alle Flektionsendungen sowie alle Umlaute und graphemische Veränderungen entfernt werden.

**Definition 3.5.1 (nach [Bus02])**
*Ein Lemma ist ein Eintrag bzw. ein einzelnes Stichwort in einem Lexikon.*

Um eine Wortform auf ihr Lemma zurückzuführen ist es nötig ein Lexikon zu benutzen in dem die infrage kommenden Lemmata verzeichnet sind.

Die Qualität eines Lemmatisierer hängt ganz entscheidend von der Qualität der verwendeten Wörterbücher ab. Je umfassender das Wörterbuch ist, desto höher die Qualität der Ausgabe. Der optimale Zustand ist erreicht, wenn das Wörterbuch zu jeder Wortform die tatsächliche Grundform zurückliefern kann. Da Sprache sich jedoch verändert und Anforderungen zusätzlich Wörter verwenden, die in der Grundsprache nicht vorkommen (bspw. englische Fachbegriffe) ist ein solcher Zustand nicht zu erreichen. Möchte man sich jedoch diesem Zustand möglichst stark annähern, so wird man das Wörterbuch möglichst groß machen um eine hohe Abdeckung der Sprache zu erreichen.

Im Falle der Lemmatisierung wird ein Grundformenlexikon verwendet, da ein vollständiges Wörterbuch einer Sprache so umfangreich wäre, dass der Vorteil des schnelleren Nachschlagens durch den erheblichen Speicherbedarf zunichte gemacht werden würde. Hinzu kommt, dass im Falle nicht englischer Anforderungen zusätzlich zu dem englischen Wörterbuch noch mindestens ein weiteres in der Sprache der Anforderungen gebraucht wird, wodurch sich der Speicherbedarf im Schnitt verdoppelt.

Unabhängig von der Frage welches Wörterbuch verwendet wird ist es nötig morphologische Regeln zu erstellen um eine Wortform in ein Lemma zurückzuführen bzw. von einem Lemma zu einer Wortform zu kommen.

Ein Grundformenlexikon hat gegenüber einem Vollformenlexikon den Vorteil, dass die morphologischen Regeln zur Überführung einer Grundform in eine andere Wortform bereits vorhanden sind. Damit ist es auch mögliche aufgrund dieser Regeln eine Wortform auf ihre entsprechende Grundform zu reduzieren.

Da die Lemmatisierung die Reduktion auf die Grundform mit Hilfe von Wörterbüchern vornimmt, kann anders als bei der Verwendung von Heuristiken oder des stemmings garantiert werden, dass jedes Lemma linguistisch korrekt ist.

Es gilt

$$f(w) = \begin{cases} g \text{ wenn } g \text{ die Grundform zu } w \text{ ist} \\ w \text{ sonst} \end{cases}$$

Die Lemmatisierung ist damit ein geeigneter Algorithmus um Grundformen für eine natürlichsprachliches Wörterbuch zu generieren.

## 3.6. Probleme bei der Grundformenbildung

Aufgrund der spezifischen Beschaffenheit von Anforderungen ergeben sich bei der Bestimmung von Grundformen mehrere Probleme, von denen im Folgenden die schwerwiegendsten beschrieben werden.

### Mehrsprachigkeit

Sofern Anforderungen nicht in Englisch geschrieben sind, ergibt sich das Problem, dass manche verwendeten Wörter keine Entsprechung in der jeweiligen Anforderungssprache haben. Dies ist insbesondere bei Fachbegriffen in der Informatik der Fall. Dadurch reichert sich der Text mit englischen Wörtern an.

Für die Suche nach Grundformen ist diese Mehrsprachigkeit von Wörtern ein großes Problem. Denn Heuristiken und Stemmer sind nur für eine bestimmte Sprache entwickelt, so dass für jedes Wort zuerst geprüft werden muss welcher Sprache es entstammt.

Um dies festzustellen wird zunächst mit Hilfe von Wörterbüchern überprüft zu welcher Sprache ein Begriff gehört. Danach wird für diesn die Implementierung der jeweiligen Sprache aufgerufen. Der Vorteil eines möglicherweise leichtgewichtigen Verfahrens wie des stemming verliert durch Benutzung von Wörterbüchern eben diesen Vorteil gegenüber eines auf die Benutzung von Wörterbüchern ausgelegten und damit schwergewichtigeren Verfahrens.

Offensichtlich ergibt sich dieses Problem für die Lemmatisierung nicht in gleichem Ausmaße, da hier aufgrund ihrer Arbeitsweise bereits Wörterbücher benutzt werden. Um die Lemmatisierung auf die Anforderung der Mehrsprachigkeit umzustellen, wird der Algorithmus dergestalt verändert, dass mehrere Wörterbücher als Eingabe benutzt werden können.

### Abhängigkeit von Wörterbüchern

Der Einsatz von Wörterbüchern zur Umgehung des Problems der Mehrsprachigkeit führt dazu dass nun nicht mehr der Algorithmus allein für die Qualität der Grundformreduktion entscheidend ist sondern ebenso die Qualität der Wörterbücher.

Für die Lemmatisierung wird die Qualität im schlechtesten Fall gleich bleiben, da für deren Reduktion bereits die Wörterbücher ausschlaggebend sind. Im mittleren Fall wird durch die Erweiterung des Wortschatzes eine Verbesserung der Qualität stattfinden. Diese lässt sich dadurch erklären, dass die Überprüfung der Sprache nur dann ein positives Ergebnis liefert, wenn das Wort im entsprechenden Wörterbuch enthalten ist. Ist dies der Fall so wird auch die Lemmatisierung ein Ergebnis liefern, da sie mit dem gleichen Wörterbuch arbeitet.

Für die anderen Verfahren, die normalerweise nicht mit Wörterbüchern arbeiten kann

im ungünstigsten Fall eine Verschlechterung der Ergebnisse auftreten wenn die Zuordnung einer Wortform zu einer Sprache fehlerhafte Ergebnisse liefert. Dies führt dazu dass die falsche Sprachvariante eines Algorithmus benutzt wird und damit die Ergebnisse der Grundformenbildung verfälscht.

### Einträge mit mehreren Wörtern

Unabhängig vom verwendeten Algorithmus kann es für jede Domäne spezifische Fachbegriffe geben, welche sich aus zwei oder mehreren Wörtern zusammen setzen. Ein Beispiel hierfür ist *Use Case*.

Bei der Grundformenbildung stellt sich nun das Problem zu definieren, wie die Grundform solch eines Mehrwortbegriffs zu bestimmen ist.

In obigen Beispiel sind sowohl *Use* als auch *Case* korrekte Wörter der englischen Sprache. So ist es im Rahmen sprachlicher Korrektheit erlaubt sowohl *Use* als auch *Case* zu deklinieren. Jedoch nicht im Falle eines feststehenden Fachbegriffes, da z.B. *Uses Cases* keine Wortform von *Use Case* ist.

Mit Hilfe der vorgestellen Algorithmen ist die Grundformbildung zu solch einem Spezialfall nicht möglich, da sie dafür ausgelegt sind, jeweils genau ein Wort als Eingabe zu akzeptieren und zu bearbeiten.

Für die Lemmatisierung kann dieses Problem durch das Hinzuziehen eines Fachwörterbuchs abgeschwächt werden. In diesem Fachwörterbuch wären dann auch Begriffe mit mehreren Wörten sowie die entsprechenden Grundformen enthalten.

Da jedoch Fachwörterbücher in den meisten Fällen nicht zur Verfügung stehen, wird in dieser Arbeit ein anderer Ansatz gewählt um Mehrwortbegriffe zu behandeln.

Zwar besteht ein Mehrwortbegriff aus einzelnen Wörtern, jedoch bilden diese sprachlich einen feststehenden Ausdruck. Somit ist ein Mehrwortbegriff wie ein einziges Wort aufzufassen. Dies bedeutet für die Grundformbildung, das lediglich das letzte Teilwort verändert wird und alle Teilworte davor gleich bleiben.

**Definition 3.6.1**
*Für einen Begriff* $w = w_1 w_2 \ldots w_n$*, welcher aus den einzelnen Teilwörtern* $w_1 \ldots w_n$ *zusammengesetzt ist, gilt*

$$\mathit{Lemma}(w) = w_1 w_2 \ldots \mathit{Lemma}(w_n)$$

## 3.7. Die Use Case Design Environment (UCDE)

Abbildung 3.1 zeigt die Oberfläche der UCDE zu Beginn der Arbeit. Wie zu sehen ist, wird der Großteil der GUI bereits belegt, so dass diese Bereiche nicht genutzt werden kann ohne das gewohnte Erscheinungsbild zu verändern. Dies ist aber auf Rücksicht auf schon im Umgang mit dem Programm erfahrene Benutzer nicht ratsam.

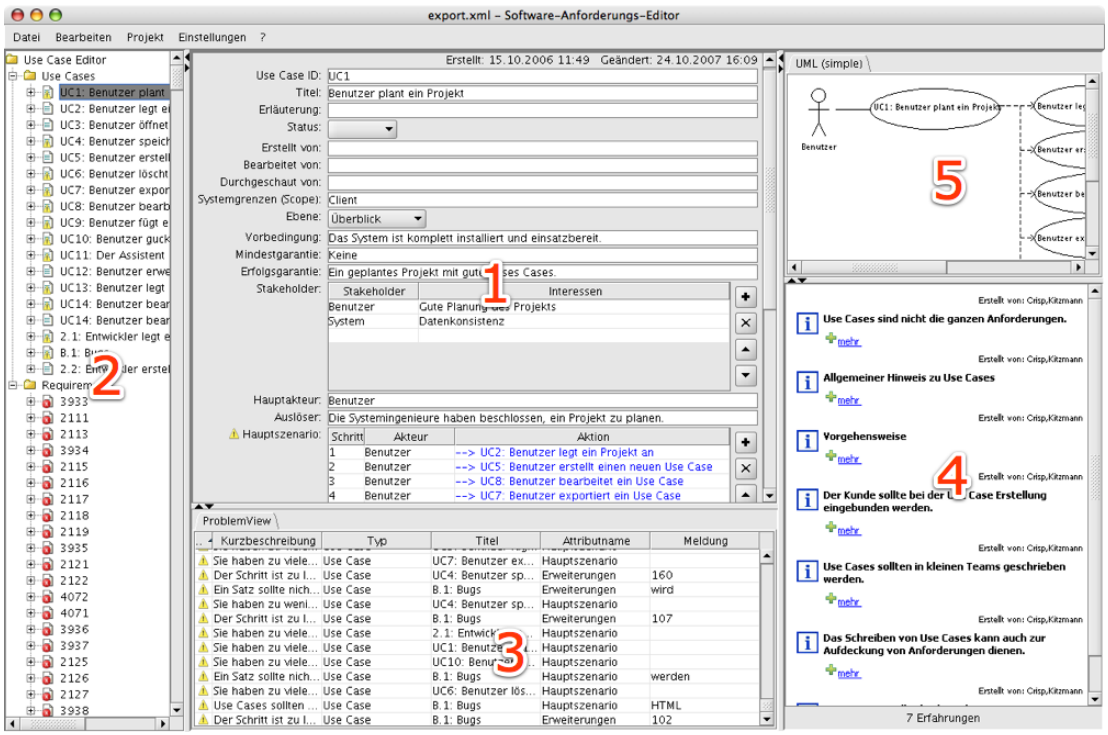


Abbildung 3.1.: Ansicht des Hauptprogramms

Ein integraler Bestandteil der UCDE sind die Erfahrungen und das damit verbundene Feedback an den Benutzer. Diese Komponente wurde von Ingo Kitzmann wie in [Kit07] beschrieben neu programmiert. Zwar sind die Komponenten für die Erfahrungen in ein Plugin ausgegliedert, jedoch kann aufgrund ihrer herausragenden Bedeutung für die Use Case Design Environment davon ausgegangen werden, dass sie aktiviert sind. Dies führt bei der Betrachtung der Layoutstruktur dazu, dass die vom Erfahrungs-Plugin benutzen Flächen mitbeachtet werden.

Die Hauptansicht der UCDE ist in 5 Bereiche geteilt:

- Der mit 1 markierte Bereich ist der *Bearbeitungsbereich* der UCDE. Hier werden die einzelnen Dokumente des Anforderungsprojekts bearbeitet. Dazu können für jeden Dokumententyp unterschiedliche Editoren zur Verfügung gestellt werden.
- Der *Projektbaum* (2) zeigt die einzelnen Dokumente des Projekts an. Diese sind nach Dokumenttyp geordnet. Durch einen Klick auf ein Dokument wird es mit dem entsprechenden Editor im Bearbeitunsbereich geöffnet.
- Der *Problemview* (3) ist dazu gedacht alle im Projekt auftretenden Probleme übersichtlich in einer Liste darzustellen. Dieser Bereich wird vom Erfahrungs-Plugin benutzt.
- Der mit 4 gekennzeichnete Bereich ist das *Criticview*. Hier werden die Erfahrun-

gen, Hinweise und Warnungen des Erfahrungs-Plugins angezeigt und es kann mit ihnen interagiert werden.

- Der *Simulationsbereich* (5) dient dazu die Ausgaben von Plugins anzuzeigen, die verschiedenste Informationen über das Projekt sammeln und auswerten. Ein Beispiel hierfür ist das in [Cri06] entwickelte UML-Plugin.

# 4. Methoden zur Glossarerstellung

Um die Generierung eines Glossar nicht auf die Use Case Design Environment festzulegen wurde bei der Entwicklung auf eine Trennung zwischen Grundoperationen und Methoden zur Einbindung in ein Programm (im speziellen der UCDE) Wert gelegt.

Dieses Kapitel beschreibt diejenigen Funktionen, deren Aufgabe die Generierung und Verwaltung eines Glossars sowie die Bildung von Grundformen und die Generierung einer Liste mit Vorschlägen einschließlich deren Filterung.

## 4.1. Lemmatisierung

Bei der Anforderungserhebung entstehen natürlich-sprachliche Texte. Ebenso sollen die Einträge im Glossar den Kriterien von Definition 3.4.1 gerecht werden.

Um zu einer Menge von Wörtern, welche aus einem natürlich-sprachlichen Text extrahiert wurden die jeweilige Grundform zu bilden, wurden in Abschnitt 3.5 verschiedene Methoden zur Grundformbildung vorgestellt.

Für die vorliegende Arbeit wird zur Grundformenbildung ein Lemmatisierer benutzt. Die genauen Gründe für diese Entscheidung werden detailiert in Abschnitt 6.2 beschrieben.

Eine Vorraussetzung für einen qualitativ hochwertigen Lemmatisierer ist das verwendete Wörterbuch. Dieses muss möglichst umfangreich sein um eine hohe Abdeckung der verwendeten Sprache zu gewährleisten.

Da das Erstellen eines solchen Wörterbuches allerdings außerhalb der Reichweite dieser Arbeit liegt muss auf ein fertiges Wörterbuch zurückgegriffen werden. Das OpenOffice-Projekt[1] bietet als Teil des Lingucomponent-Projekts[2] Wörterbücher für die Rechtschreibprüfung innerhalb ihrer Office-Suite an. Diese stehen unter der (L)GPL-Lizenz und eignen sich daher auch lizenzrechtlich für die Benutzung in dieser Arbeit.

Das Format diese Wörterbücher entspricht dem von MySpell welches in [Wik07] genauer beschrieben wird. Dieses zeichnet sich dadurch aus, dass es eine Datei mit Grundformen benutzt. Diese sind mit ihren jeweiligen Flexionsklassen annotiert. Eine zweite Datei, die sog. Affixdatei, enthält die zu den Flexionsklassen gehörenden morphologischen Regeln um aus einer Grundform die entsprechende Wortform herzuleiten. Es handelt sich also um ein Grundformenlexikon.

[1] `http://www.openoffice.org`
[2] `http://lingucomponent.openoffice.org/`

## Rechtschreibprüfung

Abbildung 4.1 zeigt den prinzipiellen Ablauf einer Rechtschreibprüfung.

Wie zu sehen ist prüft der Algorithmus zur Rechtschreibprüfung zunächst einmal ob das übergeben Wort selbst eine Grundform ist. Ist dies der Fall wird *TRUE* zurückgegeben.

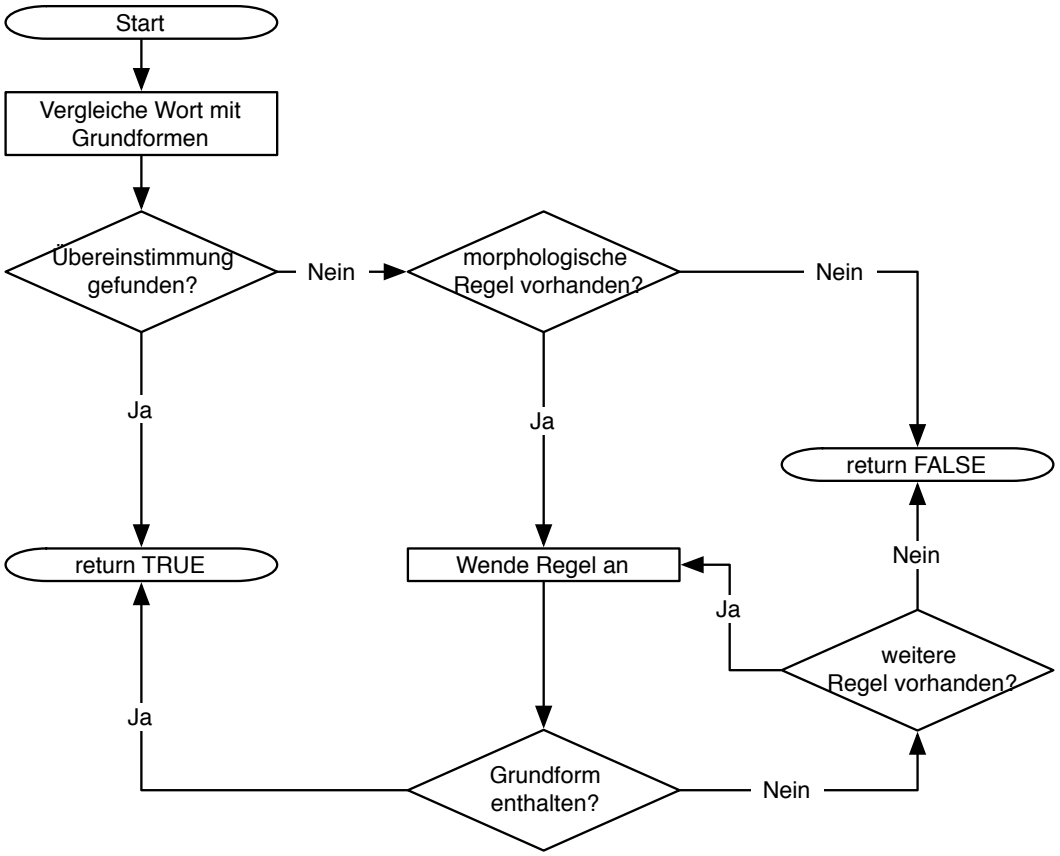


Abbildung 4.1.: Funktionsweise einer Rechtschreibprüfung

Andernfalls wird angenommen, dass das übergebene Wort eine Variation einer bekannten Grundform ist. Nun wird überprüft ob es eine oder mehrere gültige morphologische Regeln gibt, welche zu den Flexionsklassen des Worts gehören. Gibt es keine solche Regel, so gilt das Wort als unbekannt und somit im Rahmen der Rechtschreibprüfung als falsch geschrieben. Es wird *FALSE* zurückgegeben.

Falls es jedoch eine oder mehrere Regeln gibt wird für jede überprüft, ob das Wort durch deren Anwendung in eine gültige Grundform überführt werden kann. Ist dies der Fall, so wird *TRUE* zurückgegeben. Ansonsten wiederum *FALSE*.

Eine Rechtschreibprüfung prüft also ob eine Wortform mit Hilfe von morphologischen

Regeln auf eine Grundform in der betrachteten Sprache zurückzuführen ist. Ist dies der Fall und die Grundform im verwendeten Wörterbuch verzeichnet, so gehört das Wort zu Sprache.

Fast dem gleichen Prinzip folgt eine Lemmatisierung. Hierbei ist es jedoch nicht interessant ob ein Wort zur Sprache gehört sondern wie seine Grundform lautet.

Für den Fall, dass das übergebene Wort $w$ selbst eine Grundform ist, wird dieses wieder zurück gegeben. Es gilt also $f(w) = w$. Wenn das übergebene Wort selbst keine Grundform ist, so muss die zugehörige Grundform $g$ bestimmt werden. In diesem Fall gilt $f(w) = g$.

Falls $w$ ein domänen-spezifisches Wort ist, welches nicht in den benutzten Wörterbüchern enthalten ist, so kann keine Grundform gebildet werden. Gleichzeitig kann auch nicht festgestellt werden, ob $w$ selbst eine Grundform ist, da hierfür keine Referenz vorhanden ist. Für diesen Fall gilt folgende Definition:

**Definition 4.1.1**
*Für ein Wort $w$ welches im Wörterbuch einer Lemmatisierung nicht vorhanden ist wird $w$ selbst als Lemma gewählt. Es gilt $f(w) = w$.*

Um eine Rechtschreibprüfung zur Lemmatisierung zu benutzen ist es nötig dass in jedem Schritt nicht nur überprüft wird ob die variierte Wortform eine Grundform ist sondern sie muss auch dem aufrufenden Programmteil bekannt sein.

Wenn nun eine Übereinstimmung festgestellt wird, so wird diese veränderte Eingabe zurückgegeben. Hier gilt $f(w) = g$.

Da für eine Lemmatisierung garantiert wird, dass jede Rückgabe linguistisch korrekt ist, wird für den Fall dass die Rechtschreibprüfung *FALSE* liefert die Eingabe selbst zurückgeliefert, so das gilt $f(w) = w$.

Da das MySpell-Format öffentlich zugänglich ist gibt es schon fertige Implementierungen welche sich zu einem Lemmatisierer erweitern lassen. Für diese Arbeit wurde auf JMySpell[3] zurückgegriffen, da diese Implementierung unter der LGPL veröffentlich wurde und vollständig in Java geschrieben ist.

Zudem bietet sie den Vorteil dass, wie oben beschrieben, in jedem Kombinationsschritt auf die veränderte Eingabe zurückgegriffen werden kann, so dass diese Implementierung leicht zu einem Lemmatisierer zu erweitern ist.

## 4.2. Glossar und Glossareinträge

Ein Glossar ist wie in Definition 3.1.1 beschrieben eine Sammlung von einem oder mehreren Wörtern mitsamt der Erklärung.

[3] `http://jmyspell.javahispano.net/en/index.html`

Das Glossar wird durch die Klasse `Glossary`, deren Klassendiagramm in Abbildung 4.2 zu sehen ist, repräsentiert.

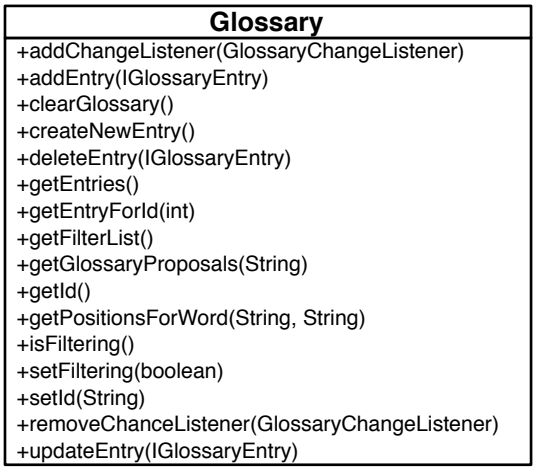


Abbildung 4.2.: Klassendiagramm der Klasse Glossary

Diese Klasse enthält zum einen Funktionen zur Verwaltung von Glossareinträgen. Zum anderen sind hier auch alle Funktionen untergebracht die benötigt werden um aus einem Text mögliche Vorschläge für die Aufnahme in ein Glossar zu extrahieren. Diese werden im Detail im Abschitt 5.1.1 beschrieben.

Um ein Glossar eindeutig identifizierbar zu machen, hat jede Instanz der Glossarklasse `Glossary` eine eindeutige Id. Diese wird durch die Methoden `randomUUID` der Standardklasse `java.util.UUID` generiert und ist somit zu einer sehr hohen Wahrscheinlichkeit einzigartig.

Die Einträge in einem `Glossary` werden durch vom in Abbildung 4.3 gezeigten Interface `IGlossaryEntry` abgeleiteten Klassen repräsentiert.

Zum einen den Namen des Glossareintrags. Dieser ist die Bezeichnung unter der der Eintrag im Glossar nachgeschlagen werden kann. Im Falle eines Glossars nach der Definition dieser Arbeit ist dies die Grundform eines Wortes.

Das zweite notwendige Attribut ist die Beschreibung. Hauptaufgabe eines Glossars ist es eine Erklärung zu einem verzeichneten Wort zu liefern. Um dieser Aufgabe gerecht zu werden enthält ein `IGlossaryItem` zusätzlich zu seinem Namen ein Feld für seine Beschreibung.

Als drittes hat jedes `IGlossaryItem` ein Feld für eine Id. Dies hat den Hintergrund dass sich während der Bearbeitung eines Glossars aus verschiedenen Gründen der Name eines Glossareintrags ändern kann. Ebenso kann sich die Beschreibung eines Glossareintrags ändern. Aus diesem Grund eignen sich beide Attribute weder allein noch in Kombination zur eindeutigen Identifizierung eines Eintrags. Hier wird ähnlich wie bei Datenbanken ein künstliches Attribut eingeführt um die eindeutige Identifikation

eines Glossareintrags zu gewährleisten.

Damit diese eindeutige Identifikation gewährleistet ist darf es nicht passieren dass eine Id mehrfach vorkommt. Die Klasse `Glossary` stellt hierfür die Methode `createNewEntry` zur Verfügung. Für alle durch diese Methode erstellten Glossareinträge wird garantiert, dass sie unterschiedliche Ids innerhalb des entsprechenden Glossars haben, da diese zentral durch die Klasse `Glossary` verwaltet werden.

Schließlich enthält ein `IGlossaryEntry` noch ein Attribut für die Zuordnung zu einem Glossar. Dieses speichert die einzigartige Id einer Glossar-Instanz.

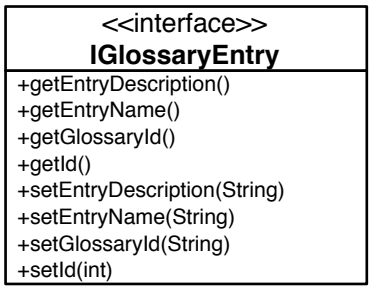


Abbildung 4.3.: Klassendiagramm des Interface IGlossaryEntry

## 4.3. Extraktion von Vorschlägen

Um aus einem Text die möglichen Vorschläge zu extrahieren wird wie folgt vorgegangen:

Zuerst werden bereits in das Glossar aufgenommene Einträge aus dem Text entfernt um so zu verhindern dass diese oder - bei Glossareinträgen mit mehreren Wörtern - Teilwörter von diesen wieder vorgeschlagen werden.

Wenn Informationen über schon einmal einem Glossar hinzugefügte Begriffe vorliegen, werden diese in die Rückgabeliste eingefügt und ebenfalls aus dem Text gelöscht. Zusätzliche erhalten sie eine hohe Priorität. Wie in Abschnitt 5.1.1 beschrieben sollen bereits einmal aufgenommene Begriffe zuerst in der Vorschlagliste auftauchen, was durch die Zuordnung einer hohen Priorität gewährleistet wird.

Nachdem nun alle bereits im Glossar aufgenommenen oder schon einmal hinzugefügten Begriffe aus dem Text gelöscht worden sind, bleiben nur noch die möglichen Vorschläge der normalen Priorität übrig.

Zunächst werden alle Sätze in einzelne Worte zerlegt. Die erhaltenen Worte werden nun mittels Lemmatisierung auf ihre Grundformen zurückgeführt. Schließlich wird für jede Grundform geprüft ob sie schon einmal vorgekommen ist. Ist dies der Fall, so wird der Liste der Vorkommen die gefundene Position hinzugefügt. Ansonsten wird die

Grundform mit dieser Position neu aufgenommen.

Die Rückgabe dieser Funktion ist eine Liste aller Grundformen zu den gefundenen Begriffen. Um Entwicklern, welche diese Klasse benutzen, immer wiederkehrende Arbeiten wie z.B. das Suchen nach einem Eintrag abzunehmen wird bewusst auf die Benutzung der Standardklasse `List` verzichtet.

Stattdessen wurde die Klasse `ProposalList` die in Abbildung 4.4 zu sehen ist eingeführt. Diese nimmt eine Menge von Vorschlägen, die durch die Klasse `Proposal` repräsentiert werden, auf.

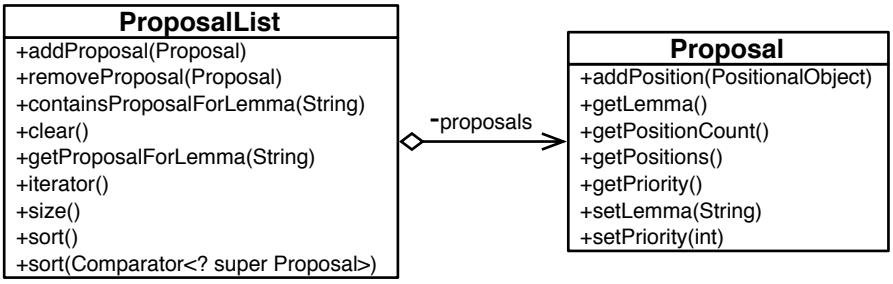


Abbildung 4.4.: Klassendiagramm der Klassen ProposalList und Proposal

Ein `Proposal` speichert zu jedem Vorschlag die Grundform (Lemma), die Priorität und eine Liste alle gefundenen Positionen innerhalb eines Textes. Über die Länge dieser Liste lässt sich auch ermitteln wie oft ein Begriff vorkommt.

Aus der Priorität und der Vorkommenhäufigkeit lässt sich die Position eines Vorschlags in der Gesamtliste bestimmen. Nach Abschnitt 5.1.1 gilt, dass Vorschläge zuerst nach Priorität und danach nach Häufigkeit geordnet werden.

Da jeder Vorschlag diese Informationen speichert kann eine ProposalList die Sortierung selbst vornehmen. Die Ordnungskriterien werden mittels einer von `Comparator` abgeleiteten Klasse bestimmt.

Für die beschriebenen Ordnungkriterien ist die Klasse `ProposalComparator` (Abbildung 4.5) bereits implementiert. Jedoch ist es im Falle von selbst definierten Ordnungskriterien auch möglich eine eigene Implementierung zu benutzen, welche diese beachtet.

## 4.4. Architektur der Filterung

Das Ziel der Filterung ist es die Menge an Informationen so zu begrenzen dass sie dem Benutzer helfen ein Glossar zu erstellen anstatt ihn zu überfordern.

Dies wird zum Teil dadurch erreicht, dass Bindewörter ausgefiltert werden. Hierfür wird dem Plugin ein Filter beigelegt, der eine Menge von häufig verwendeten Bindewörtern

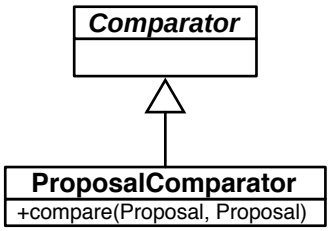


Abbildung 4.5.: Klassendiagramm der Klasse ProposalComparator

enthält. Dieser kann dann vom Benutzer in der Filterverwaltung aktiviert werden um so die Bindewörter auszufiltern. Da dieser Filter keinen Anspruch auf Vollständigkeit erheben kann, ist es evtl. nötig ihn bei Bedarf zu erweitern. Dies kann genauso geschehen wie bei jedem andern Filter auch. Da Bindewörter jedoch nicht die einzigen Wörter sind, welche der Benutzer nicht vorgeschlagen bekommen möchte, ist es nötig mehrere Filter gleichzeitig zu berücksichtigen.

Zu diesem Zweck werden alle aktivierten Filter in einer Liste (`FilterList`) zusammengefasst. Jeder Filter besteht wiederum aus mehreren `FilterItem` welche die Einträge enthalten auf die der Filter wirken soll. Abbildung 4.6 zeigt das Beziehungs- und Klassendiagramm für die an der Filterung beteiligten Klassen.

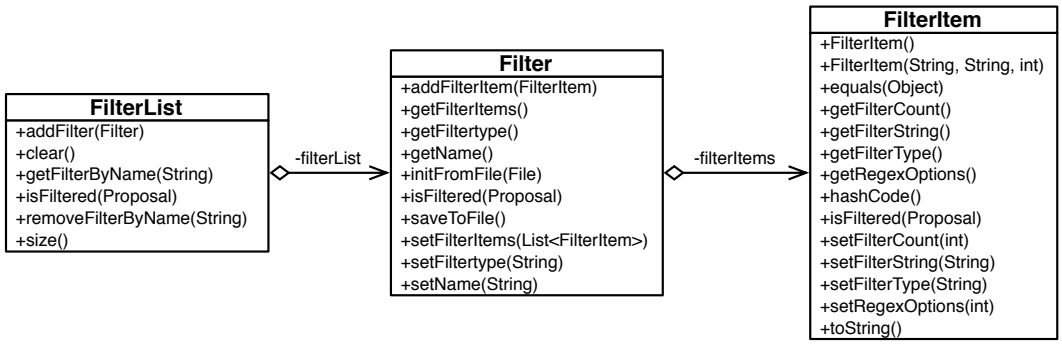


Abbildung 4.6.: Beziehungs- und Klassendiagramm der Filterkette

Ein Filter besteht wie beschrieben aus mehreren `FilterItem`. Ein FilterItem besteht aus einem regulären Ausdruck und Bedingungen welche eine Einschränkung bezüglich der Häufigkeit treffen.

Durch die Einschränkung der Häufigkeit lässt sich ein Eintrag filtern wenn er maximal bzw. mindestens $n$-mal vorkommt.

Die Zusammenfassung mehrerer FilterItems wird durch die Klasse `Filter` geregelt. Hierbei kann für jeden Filter die Art angegeben werden wie eine Übereinstimmung

zwischen zu filternden Wort und Filtereintrag interpretiert wird.

Es gibt zwei Arten von Filtern:

- Eine negative Filterart, die alle Wörter welche mit einem oder mehreren Einträgen übereinstimmt aus der Menge der möglichen Vorschläge herausfiltert
- Eine positive Filterart, die alle Wörter ausfiltert welche *nicht* mit einem oder mehreren Einträgen aus der Menge der möglichen Vorschläge übereinstimmt

Eine `FilterList` schließlich umfasst mehrere einzelne `Filter`. Der Glossarkern enthält ledigliche eine einzige FilterList in die alle aktivierten Filter aufgenommen werden. Für jeden möglichen Vorschlag wird nun für alle Filter der Liste überprüft ob der übergebene Vorschlag in die Rückgabe aufgenommen werden soll oder nicht. Um zu ermöglichen dass Programme, die den Glossarkern benutzen die Filterung selbst durchzuführen (z.B. um Ergebnisse zwischenzuspeichern) kann die automatische Filterung im Glossarkern ausgeschaltet werden.

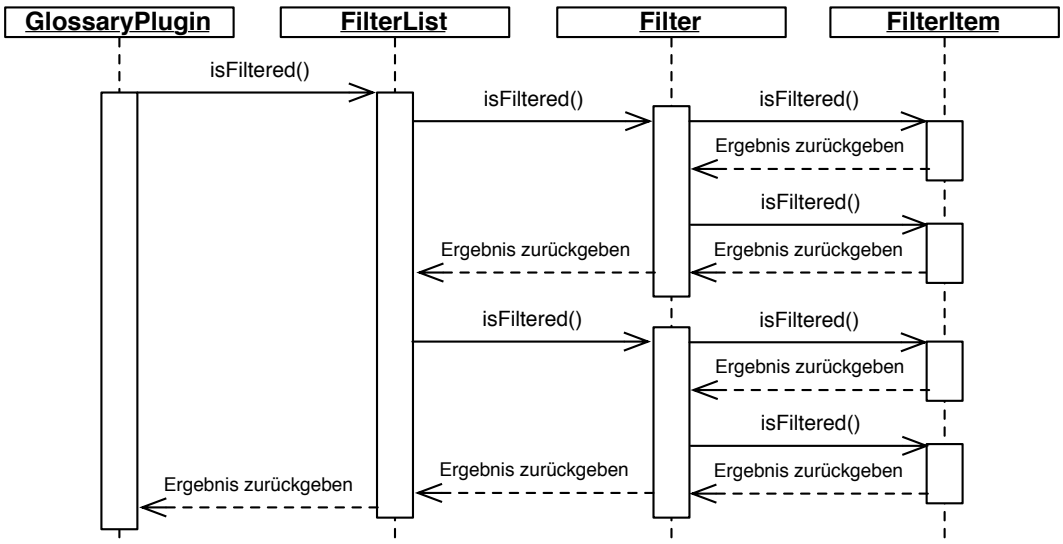


Abbildung 4.7.: Sequenzdiagramm der Filterkette

Abbildung 4.7 zeigt das Sequenzdiagramm für die Filterung. Hier wird noch einmal die Verkettung der einzelnen Komponenten deutlich. Die tatsächliche Filterung findet erst in der Klasse `FilterItem` statt. Die Methoden `isFiltered` in den Klassen `FilterList` und `Filter` iterieren jeweils über die Liste der untergeordneten Klassen und rufen dann für jeden Eintrag in der Liste wiederum `isFiltered` auf.

## 4.5. Nutzung des Erfahrungskreislaufs

In der Konzeption der Use Case Design Environment spielt die Nutzung von Erfahrungen eine bedeutenden Rolle. Diese Wiederverwendung bereits gemachter Erfahrungen unterstützt den Benutzer bei der täglichen Arbeit. Die Erfahrungsbasis kann durch den Benutzer selbst erweitert werden um so seine eigenen Erfahrungen mit andern Benutzern zu teilen.

Diese Nutzung von Erfahrungen zur Verbesserung der Wirksamkeit einer Komponente kann auch auf die Arbeit mit Glossaren übertragen werden. Insbesondere profitiert die Extraktion von Informationen aus dem Anforderungsprojekt von einmal gemachten und für gut befundenen Erfahrungen.

Abbildung 4.8 zeigt den Erfahrungskreislauf wie in [Sch02] beschrieben. Im folgenden soll betrachtet werden wie dieser Erfahrungskreislauf genutzt werden kann um die Qualität der Filterung und die der Vorschläge bzw. der Vorschlagsliste zu verbessern.

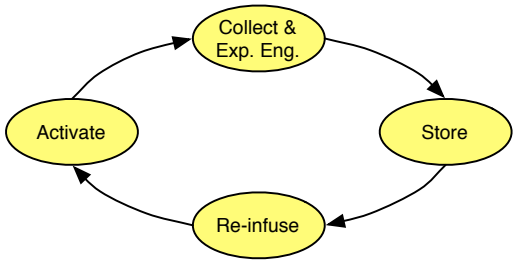


Abbildung 4.8.: Erfahrungskreislauf nach [Sch02]

Wie in Abschnitt 6.5 beschrieben, kann die Qualität von Filtern dadurch verbessert werden, dass mehrere Projekte der gleichen Domäne untersucht werden. Vorraussetzung hierfür ist, dass die Projekte abgeschlossen sind, sich also weder Glossar noch Anforderungen ändern.

Der Ausgangszustand ist ein Filter, der mindestens ein FilterItem enthält. Dies ist der *activate*-Schritt im Erfahrungskreislauf.

Der Schritt des *collecting and experience engineering* gestaltet sich wie folgt: Für jeden Eintrag in diesem Filter wird zuerst geprüft ob der vom Filtereintrag abgedeckte Begriff in der Liste der Vorschläge zu einem Projekt vorkommt. Ist dies nicht der Fall, wird der Filtereintrag nicht weiter berücksichtigt. Kommt der abgedeckte Begriff in der Liste der Vorschläge vor, so wird überprüft ob dieser Begriff in das Glossar aufgenommen wurde. Der Zähler für das Auftreten dieses Begriffs wird um 1 erhöht. Ist der Begriff im Glossar aufgenommen, wird auch der Zähler für die Aufnahme in ein Glossar für diesen Begriff um 1 erhöht. Nachdem alle Projekte durchsucht wurden kann aus diesen beiden Werten eine neue Kennzahl für die Qualität des Filters berechnet werden. Hierbei ist zu beachten, dass diese je nach Art des Filtern unterschiedlich ermittelt wird:

- Für negative Filter ist es schlecht wenn die von ihnen abgedeckten Begriffe im Glossar aufgenommen werden, da sie diese aus der Vorschlagsliste entfernen. Wird ein Begriff aufgenommen obwohl er durch einen Filter aus der Vorschlagsliste entfernt wurde, so arbeitet der Filter in diesem Fall nicht zufriedenstellend.

  Die Qualitätskennzahl wird errechnet als $1 - \frac{\text{Anzahl der Aufnahmen}}{\text{Anzahl der Vorkommen}}$

- Bei positiven Filtern ist es gut, wenn der Begriff aufgenommen wurde. Hier wird die Qualitätskennzahl als $\frac{\text{Anzahl der Aufnahmen}}{\text{Anzahl der Vorkommen}}$ berechnet

Mit Hilfe der neuen Qualitätskennzahl für jedes FilterItem lässt sich der Filter nun optimieren in dem z.B. schlechte Einträge entfernt werden. Dieser verbesserte Filter wird im Schritt des *Re-Infuse* wieder zur Verfügung gestellt um von den Verbesserungen zu profitieren.

Wird der Filter danach verändert oder stehen neue Projekte zur Verfügung kann dieser Kreislauf wiederholt werden.

Ebenso wie die Filter lassen sich Erfahrungen auch für Vorschläge benutzen.

Hierbei startet der erste *activate*-Schritt mit einer leeren Erfahrungsbasis. Für jedes abgeschlossene Projekt werden nun die Vorschläge in die Erfahrungsbasis aufgenommen. Hierbei bekommt jeder Eintrag in der Erfahrungsbasis noch eine Kennzahl für die Anzahl der Aufnahmen und eine Kennzahl für das Vorkommen in unterschiedlichen Projekten.

Wird ein bereits in der Erfahrungsbasis vorkommender Eintrag in einem Projekt nicht in das Glossar übernommen, so erhöht sich lediglich der Zähler, der die Vorkommenshäufigkeit notiert, nicht jedoch der, welcher die Anzahl der Aufnahmen speichert. So lässt sich eine Qualitätskennzahl für einen Vorschlag in der Erfahrungsbasis als $\frac{\text{Anzahl der Aufnahmen}}{\text{Anzahl der Vorkommen}}$ berechnen.

Diese Kennzahl lässt sich zur Laufzeit berechnen. Um schlechte Vorschläge auszusortieren muss ein zusätzliches Ordnungskriterium eingeführt werden, welche die Vorschläge innerhalb der Prioritätsklassen zuerst nach Qualität und danach nach Vorkommenshäufigkeit sortiert. Wird nun zusätzlich die Filterung so erweitert, dass sie ebenfalls die Qualitätskennzahl von Vorschlägen berücksichtigt, so führt dieses Verfahren dazu, dass Vorschläge die häufig vorkommen aber nicht oder nur selten in ein Glossar aufgenommen werden in der Vorschlagsliste nach unten rücken und beim Erreichen einer durch den Filter festgelegten Grenze ganz entfernt werden.

Allerdings ist hierbei zu beachten, dass Vorschläge die einmal in einem Dokument vorkommen aber nicht in ein Glossar aufgenommen wurden bereits eine Qualitätskennzahl von 0 erhalten. Für die Erweiterung der Filterung muss also berücksichtigt werden, dass eine stabile Qualitätskennzahl erst erreicht werden kann, wenn der Vorschlag in einer gewissen Mindestanzahl von Projekten vorkommt.

Der Schritt des *re-infuse* besteht für die Vorschläge darin, die Erfahrungsbasis nach jeder Änderung allen Benutzern zur Verfügung zu stellen.

## 4.6. Automatische Aufnahme von Einträgen

Mit den entwickelten Methoden zur Extraktion von Informationen aus Anforderungen und der Verwaltung von Glossaren kann eine eingeschränkte automatische Aufnahme von Einträgen in ein Glossar realisiert werden.

Vorraussetzung für diese Art der automatischen Aufnahme ist ein Anforderungsdokument, welches ausreichend formalisiert ist. Ein Beispiel für einen Text dieser Art ist [HP02], welches am Fraunhofer Institut für experimentelles Software Engineering entwickelt wurde. Dieses Dokument beschreibt die Anforderungsspezifikation für ein Türsteuergerät.

Bemerkenswert ist hierbei, dass alle im Dokument verwendeten Signale nach dem gleichen Schema benannt sind. Ein Signalname besteht nur aus Großbuchstaben und einem Unterstrich.

Diese Begriffe lassen sich durch die vorgestellten Methoden aus dem Text extrahieren und in einer Vorschlagsliste zusammenfassen. Aus dieser Liste lassen sich mit Hilfe regulärer Ausdrücke genau die Begriffe auswählen, die dem gewünschten Muster entsprechen. Diese lassen sich danach automatisch in das Glossar übernehmen. Abbildung 4.9 zeigt einen prototypischen Dialog für dieses Verfahren.

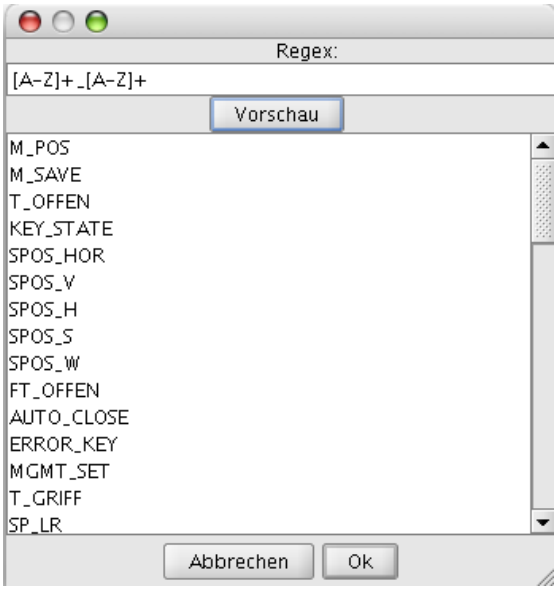


Abbildung 4.9.: Prototypischer Dialog für das automatische Hinzufügen

Dieses Verfahren stößt an seine Grenzen, wenn die gewünschten Einträge nicht nach einem bestimmten Schema benannt sind. Außerdem ist es hiermit nicht möglich zusätzlich zu den Begriffen auch deren Beschreibung in das Glossar aufzunehmen.

# 5. Feedback und Byproduct

In diesem Abschnitt wird auf die speziellen Implementierungen eingegangen die nötig waren um den Glossar-Kern in die UCDE zu integrieren.

Die Funktionen welche die Erstellung bzw. Benutzung eines Glossars im Byproduct-Ansatz und ein Feedback an den Benutzer ermöglichen sind spezifisch auf das jeweilige Umfeld abzustimmen. Im Gegensatz zu den Methoden des Glossarkern können sie deshalb nicht als direkt wieder zu verwendendes Paket zur Verfügung gestellt werden. Jedoch können die hier vorgestellten Methoden in andere Projekte übertragen werden.

## 5.1. Feedback

### 5.1.1. Generierung der Vorschlagsliste

Die Vorschlagsliste dient dazu dem Benutzer Wörter zu präsentieren die möglicherweise in sein Glossar aufgenommen werden sollten.

Der Ablauf um aus den Anforderungsdokumenten eines Projekts eine Vorschlagsliste zu generieren ist in Abbildung 5.1 dargestellt.

Im folgenden sollen die nötigen Schritte näher betrachtet werden.

#### Ermittlung der Vorschläge

Um eine Liste mit Vorschlägen generieren zu können müssen zunächst einmal alle Anforderungen eines Projektes nach möglichen Vorschlägen durchsucht werden.

Um Vorschläge aus den Anforderungen zu generieren sind nur die tatsächlichen Daten interessant nicht ihre Position innerhalb eines bestimmten Elements.

Die Mechanismen um aus einem einfachen String eine Menge von Vorschlägen für die Aufnahme eine Glossars zu extrahieren werden in Abschnitt 4.3 genauer beschrieben.

Im Folgenden soll kurz erläutert werden, wie im Fall des Glossar-Plugins vorgegangen wird. Dieses Verfahren kann leicht auf andere Situationen übertragen werden.

Die UCDE speichert die zu einem Projekt gehörenden Anforderungen in ihren Datenobjekten. Hier ist zu beachten, dass zwar jedes Element der Anforderung von `IModelElement` abstammt, jedoch erweitert sein kann.

Da die UCDE durch Plugins zu erweitert werden kann, ist nicht auszuschließen dass

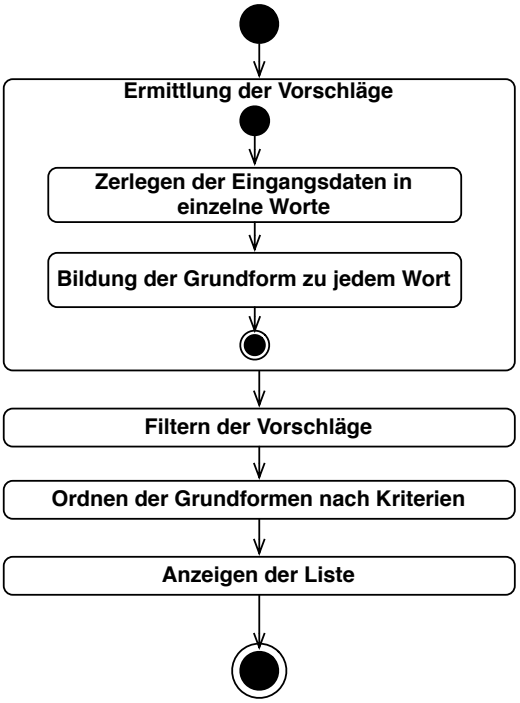


Abbildung 5.1.: Generierung einer Vorschlagsliste

in Zukunft zusätzliche Datentypen hinzugefügt werden. Ein Beispiel hierfür ist die im XP-Labor 2007 am Institut für Software Engineering entwickelte Erweiterung für Requirements.

Aus diesem Grund und um die Auswertung von der tatsächlichen Implementierung zu entkoppeln werden Adapterklassen eingesetzt, wie sie in [GHJV95] beschrieben werden. Diese implementieren das Interface `IModelElementAdapter`, dessen Klassendiagramm in Abbildung 5.2 zu sehen ist und lesen die Daten aus den konkreten Elementen aus. Eine Factory-Methode liefert für ein übergebenes `IModelElement` den passenden Adapter. Dies hat den Vorteil das für weitere spezialisierte Datenklassen weitere Adapter hinzugefügt werden können ohne am Rest des Glossar-Plugins etwas zu verändern. Sie müssen lediglich der Factory bekannt gemacht werden.

Die Arbeitsweise der Vorschlagsermittelung ist hierbei die folgende:

Der Adapter iteriert schrittweise über alle Attribute des adaptierten Elements. Attribute eines Elements können einfache Strings sein oder wiederum zusammengesetzte Elemente. Über diese zusammengesetzten Elemente wird solange weiter iteriert bis schließlich ein String-Element vorliegt.

Aus jedem String werden die möglichen Vorschläge extrahiert. Dazu werden wird die Funktion `Glossary.getGlossaryProposals(String)` aufgerufen. Die Rückgabe hiervon ist eine `ProposalList`.

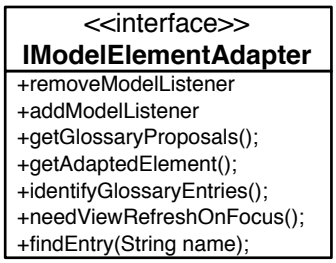


Abbildung 5.2.: Interface IModelElementAdapter

Um nun eine Liste für das gesamte Anforderungsprojekt zu generieren wird diese Vorgehensweise auf jedes Element des Projekts einmal angewandt. Danach werden die einzelnen Rückgabelisten zu einer Gesamtliste kombiniert.

### Filtern der Vorschläge

Die Liste aus dem vorherigen Schritt enthält nun die Grundformen aller auftretenden Wörter mit den jeweiligen Positionen ihres Vorkommens.

Nach der Erhebung möglicher Vorschläge lassen sich die gefundenen Grundformen in zwei Kategorien einteilen: Bindewörter und alle anderen Wörter.

Die Gruppe der Bindewörter ist für ein Glossar nicht interessant da sie nur zur Strukturierung des Textes beitragen aber nicht selbst bedeutungstragend sind.

Die restlichen Wörter haben selbst eine Bedeutung. Jedoch ist die Bedeutung für verschiedenen Dömänen evtl. unterschiedlich. Dies kann dazu führen, dass es zwischen zwei beteiligten Personen unterschiedlicher Fachrichtungen zu einer möglicherweise sogar gegensätzlichen Interpretation der Bedeutung kommen kann.

Diese Wörter sind deshalb für ein Glossar besonders interessant, da hierdurch eine einheitliche Definition der Worte sichergestellt werden kann und somit Fehler aufgrund falschen Verständnisses verhindert werden.

Ein wichtiges Konzept der Arbeit ist es den Benutzer zu unterstützen und ihn weder zu bevormunden noch mit zu vielen Informationen zu überladen.

Aus diesem Grund ist es nötig aus der Menge aller vorzuschlagenden Wörter diejenigen herauszufiltern, die eine geringe Wahrscheinlichkeit aufweisen in das Glossar aufgenommen zu werden.

Zu diesem Zweck wird auf die ermittelte Liste der möglichen Vorschläge eine Menge von Filtern angewandt um zu entscheiden ob ein möglicher Vorschlag auch tatsächlich dem Benutzer präsentiert wird.

Die genaue Architektur dieser Filterung ist in Abschnitt 4.4 beschrieben.

Aus Gründen der Performance-Erhöhung werden die Vorschläge nur dann neu extrahiert, wenn sich eine Anforderung geändert hat. Für die anderen Anforderungen werden die extrahierten Vorschläge zwischengespeichert. Da aber die Benutzung und die Manipulation der Filter unabhängig von den Anforderungen geschieht, wird auf eine Filterung durch den Glossarkern während der Extraktion verzichtet und diese erst vor der Anzeige vom Plugin selbst durchgeführt.

### Ordnungskriterien

Die nach dem vorherigen Schritt erhaltene gefilterte Liste ist noch ungeordnet, jedoch gehören alle Wörter die zu diesem Zeitpunkt noch in der Liste der Vorschläge enthalten sind potenziell zu der Gruppe von Wörtern die für die Aufnahme in ein Glossar interessant sind. Um den Benutzer weiter zu unterstützen soll ihm ein Anhaltspunkt geboten werden welche dieser Wörter eher in das Glossar aufgenommen werden sollten und welche eher vernachlässigbar sind. Hierfür müssen Kriterien festgelegt werden nach denen die Liste geordnet wird.

Kommt ein Wort seltener vor bedeutet dies auch, dass es in weniger verschiedenen Kontexten benutzt wird und somit die Wahrscheinlichkeit sinkt falsch interpretiert zu werden. Je häufiger jedoch ein Wort vorkommt,in desto mehr verschiedenen Kontexten tritt es auf. Dies sorgt dafür dass die Einordnung des Wortes in einen konkreten Kontext weniger erkenntlich wird, was wiederum die Wahrscheinlichkeit falsch interpretiert zu werden erhöht.

Weiterhin ist ein Wort welches weniger häufig im Projekt vorkommt vermutlich nicht so bedeutungstragend für das gesamte Projekt wie Wörter die immer wieder benutzt werden und somit häufiger vorkommen.

*Anmerkung:* Natürlich gehören auch Bindewörter zu denjenigen Wörtern die sehr oft im Projekt vorkommen. Jedoch wird angenommen dass diese schon durch den Schritt der Filterung entfernt worden sind und somit nur noch die tatsächlich für ein Glossar interessanten Wörter geordnet werden müssen.

Ein zusätzliches Ordnungskriterium sind Begriffe die schon einmal benutzt worden sind. Dieser Heuristik liegt die Annahme zu Grunde, dass der Benutzer immer in der gleichen Domäne arbeitet. Glossareinträge, die schon einmal in ein domänenspezifisches Glossar aufgenommen sind scheinen also für die gewählte Domäne eine besondere Bedeutung zu haben.

Wenn diese Einträge nun in einer Art Wissensdatenbank gespeichert werden, können sie bei einem neuen Anforderungsprojekt wieder in die Vorschlagsliste aufgenommen werden. Da von diesen Wörtern bekannt ist, dass sie schon einmal in einem Glossar aufgenommen worden sind, ist die Vorschlagspriorität für diese Wörter sehr hoch.

Aus den oben genannten Gründen wird für die vorliegende Arbeit ein zweischrittiges Ordnungskriterium gewählt. Zuerst werden die Vorschläge in Prioritätsklassen eingeteilt. Hierbei bilden die schon einmal benutzten Wörter eine Klasse mit hoher Priorität und die restlichen Wörter eine Klasse mit normaler Priorität. Innerhalb der Klassen

werden die möglichen Vorschläge noch einmal nach Häufigkeit sortiert so dass die Begriffe, welche öfter vorkommen eher vorgeschlagen werden.

Die Vorschlagsliste zeigt also zuerst die bereits einmal in ein Glossar aufgenommenen Einträge sortiert nach der Vorkommenshäufigkeit an. Danach werden alle anderen, nicht ausgefilterten Vorschläge, ebenfalls nach ihrer Vorkommenshäufigkeit sortiert, angezeigt.

### 5.1.2. Anzeigen der Vorschlagsliste

Eine Möglichkeit dem Benutzer ein Feedback zu geben ist die Liste mit Vorschlägen für die Aufnahme in das Glossar zu präsentieren. Hierdurch erhält der Benutzer Informationen über die in seinem Anforderungsdokument vorkommenden, für ein Glossar möglicherweise interessanten Begriffe.

Die Liste wurde durch die Filterung bereits von potenziell uninteressanten Vorschlägen bereinigt. Außerdem ist sie nach den in Abschnitt 5.1.1 definierten Ordnungskriterien sortiert. Aus diesem Grund stehen am Anfang der Liste die Wörter, die die höchste Wahrscheinlichkeit haben ins Glossar aufgenommen zu werden.

Um dem Benutzer nun die Möglichkeit zu geben mit dieser Liste produktiv zu arbeiten muss eine Möglichkeit gefunden werden diese benutzerfreundlich darzustellen.

In der vorliegenden Arbeit ist weiterhin zu beachten, dass die dargestellte Liste sich in die Grundstruktur der UCDE einfügen muss. Diese soll im oberen rechten Informationsbereich, dem *SimulationView* als neue Ansicht hinzugefügt werden, da dieser Bereich sich beliebig durch weitere Fenster erweitern lässt welche dann als Tabs dargestellt werden. Dieser Bereich ist auch der einzige, der noch nicht mit einer festen Aufgabe belegt ist.

Da diese Fläche sich vom Benutzer beliebig in der Größe verändern lässt muss das Layout variabel sein. Abhängig von der Filterung kann die Liste sehr lang werden. Deshalb muss das Layout sowohl scrollbar als auch bei großen Datenmengen noch zu bedienen sein.

Aus den oben genannten Überlegungen wird für das Layout der Vorschlagsliste ein Html-View gewählt. Dies hat den Vorteil, dass sämtliche Bedienelemente mittels Html erstellt werden können und vom View automatisch dargestellt werden. Außerdem wird für die in [Kit07] eingeführte Benutzerführung der Erfahrungen ebenfalls ein Html-view verwendet, so dass die Liste sich in das Bedienkonzept der direkt angezeigten Erweiterungen einfügt.

In Abbildung 5.3 ist die fertige Vorschlagsliste zu sehen. Wie zu sehen ist, sind die Vorschläge als Hyperlinks ausgeführt. Ein Klick auf einen Vorschlag fügt ihn in das Glossar ein.

Ein Klick auf den dahinter stehenden Link *Ignorieren* führt dazu, dass der Vorschlag in den Ignorieren-Filter aufgenommen wird und fortan nicht mehr angezeigt wird, wenn dieser Filter aktiviert ist.

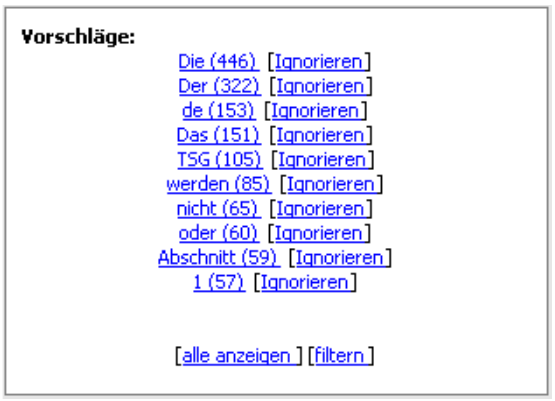


Abbildung 5.3.: Vorschlagsliste

Zusätzlich kann über den Link *filtern* zu den in Abschnitt 5.3 dargestellten Filtereinstellungen gewechselt werden.

Ein Nachteil des Html-Views ist, dass es bei größeren Dokumenten etwas dauert bis der übergebene Html-String in eine Darstellung umgewandelt worden ist. Daher werden standardmäßig nur die ersten zehn Einträge gezeigt. Falls der Benutzer alle Einträge sehen möchte kann er dies durch einen Klick auf *alle anzeigen* erreichen.

### 5.1.3. Markierung bereits aufgenommener Begriffe

Für einen Benutzer der sich nicht primär mit dem Glossar beschäftigt soll es dennoch möglich sein leicht zu erkennen, welche Begriffe bereits aufgenommen sind.

Dadurch wird dem Benutzer bereits bei der Betrachtung der Anforderungen bewusst, welche Begriffe im Glossar enthalten sind. Dies führt dazu, dass der Benutzer sofort erkennt, ob ein Begriff zu dem er möglicherweiqse eine eigene Definition im Glossar aufgenommen ist.

Um dem Benutzer ein Feedback zu geben welche Begriffe bereits im Glossar aufgenommen sind sollen diese in der Anzeige markiert werden. hierbei ist zu beachten, dass die Markierung zwar deutlich zu erkennen sein soll, den Benutzer jedoch nicht bei seiner Arbeit behindert.

Eine Möglichkeit der Kenntlichmachung wäre es alle Begriffe, die sich im Glossar befinden in Hyperlinks zu wandeln. So wäre zum einen erreicht, dass der Benutzer erkennt, dass ein Begriff im Glossar enthalten ist, zum anderen kann über das Anklicken des Hyperlinks direkt zum entsprechenden Eintrag gesprungen werden.

Der Nachteil dieser Lösung ist allerdings, dass hierfür alle bereits bestehenden Editoren so umgestaltet werden müssten, dass sie statt Textfeldern für die Eingabe Html-Komponenten benutzen. Dies bedeutet zum einen erheblichen Mehraufwand zum anderen würde es die Neuentwicklung von weiteren Editoren erheblich beschränken.

Aus diesen Gründen soll eine andere Lösung für die Kenntlichmachung von Glossareinträgen gefunden werden.

Textverarbeitungsprogramme bieten die Möglichkeit während des Schreibens die korrekte Orthographie zu überprüfen. Hierbei werden gefundene mögliche Fehler wie in Abbildung 5.4 gezeigt mit einer Wellenlinie unterstrichen. Dies gibt dem Benutzer zum einen ein Feedback, dass das markierte Wort möglicherweise nicht korrekt geschrieben ist, behindert ihn zum anderen aber nicht beim weiteren Schreiben, falls er beschließt es zu ignorieren.

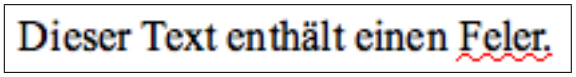


Abbildung 5.4.: Automatische Rechtschreibprüfung in Neooffice

Das gleiche Verfahren soll benutzt werden um Glossareinträge im Text zu markieren. Hierbei ist allerdings zu beachten, dass ein Benutzer zu einer hohen Wahrscheinlichkeit schon einmal eine Textverarbeitung mit aktivierter automatischer Rechtschreibprüfung benutzt hat. Aus diesem Grund darf für die Markierung nicht Rot als Farbe gewählt werden um deutlich zu machen, dass es sich hier nicht um einen Rechtschreibfehler handelt sondern um eine andere Markierung

Im Unterschied zur Markierung durch Hyperlinks ist es hier nicht möglich die markierten Begriffe auf ein Anklicken reagieren zu lassen. Stattdessen wird die Möglichkeit zum entsprechenden Glossareintrag zu gelangen über ein Kontextmenü wie in Abbildung 5.5 gezeigt realisiert.


Abbildung 5.5.: Kontextmenü für markierte Einträge

Da die bisherigen Editoren für die Texteingabe Komponenten aus dem Swing-Toolkit[1] benutzen, die in der Ableitungs-Hierachie alle von `JTextComponent` abstammen, bietet es sich an, für die Markierung eine eigene Implementierung eines Highlighters zu benutzen.

Abbildung 5.6 zeigt die Struktur eines `JTextComponents`. Eine Highlighterklasse kann zu einem `JTextComponent` hinzugefügt werden und trägt dann zur endgültigen Darstellung dieser Komponente bei. Die funktioniert auch bei Tabellen, solange der CellRenderer von `JTextComponent` abgeleitet ist.

Für die Benutzung des Highlighters sind lediglich kleine Anpassungen am UCDE und den jeweiligen Editoren nötig gewesen. Der Hauptvorteil für dieses Verfahren ist jedoch, dass die Komponenten für die bereits bestehenden Editoren nicht durch andere

[1] `http://java.sun.com/docs/books/tutorial/uiswing/index.html`

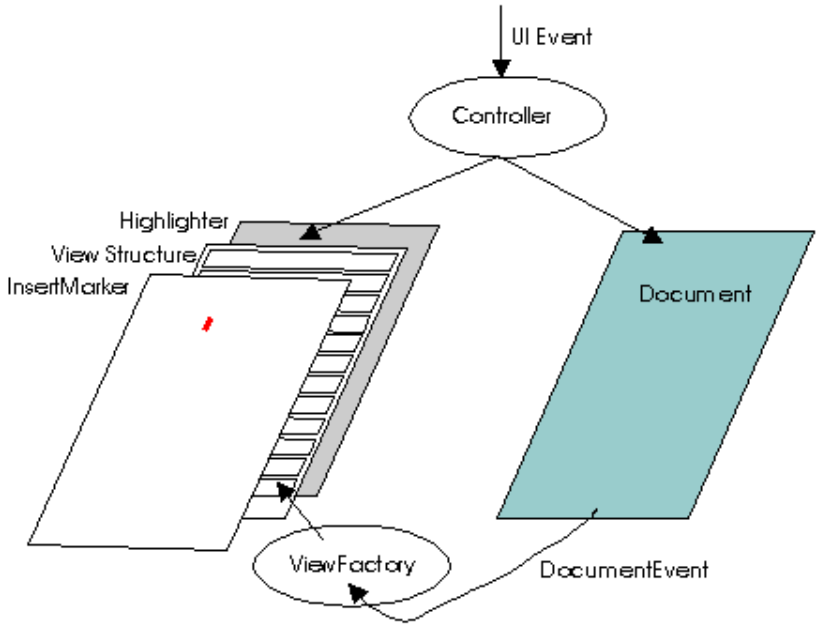


Abbildung 5.6.: Struktur eines JTextComponets

ersetzt werden mussten und dass für neue Editoren keine Einschränkungen bezüglich der zu verwendenden Komponenten entstanden sind.

### 5.1.4. Anzeige von Metriken

In [Kit07] wurde ein Plugin entwickelt mit dem Erfahrungen im Kontext der Anforderungen angezeigt werden können.

Die Heuristiken für dieser Erfahrungen lassen sich durch eigene, in JavaScript geschriebene Heuristiken ergänzen.

Dies soll benutzt werden um die Maßzahlen der Metriken für Glossareinträge oder das gesamte Glossar einzubinden und den Benutzer auf schlechte Maßzahlen hinzuweisen.

Um innerhalb der Heuristiken auf die Metriken zugreifen zu können, müssen diese der JavaScript-Engine bekannt gemacht werden. Hierzu bieten sich zwei gegensätzliche Methoden an:

Zum einen kann der JavaScript-Engine ein Metrik-Objekt hinzugefügt werden, welches zu einem übergebenen Glossareintrag die entsprechene Metrik berechnet. Zum anderen kann jedes Glossarelement mit einer Methode erweitert werden die Maßzahl für eine ausgewählte Metrik zu berechnen.

Gegen die erste Möglichkeit spricht, dass hierzu Eigenschaften der JavaScript-Engine

verändert werden müssten. Da die JavaScript-Engine jedoch nicht Teil des Hauptprogramms sondern im Experience-Plugin verankert ist, würde dies eine Abhängigkeit von eben diesem Plugin bedeuten.

Für die zweite Methode muss lediglich die Klasse `GlossaryEntry` erweitert werden. Da diese Klasse vollständig innerhalb der Grenzen des Glossar-Plugins liegt, führt diese Implementierung nicht zu einer ungewünschten Abhängigkeit.

Innerhalb der Heuristik kann dann analog zur Verarbeitung von Use Cases über die Glossareinträge iteriert werden. Jedes `GlossaryEntry`-Objekt verfügt über eine Methode `getMetricFactory`. Diese liefert eine Instanz der in Abbildung 5.7 gezeigten Factory-Klasse `MetricFactory` zurück. Über diese Klasse kann entweder eine bestimmte Metrik oder eine Liste aller verfügbaren Metriken abgefragt werden.

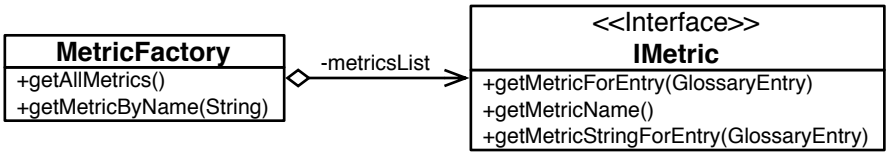


Abbildung 5.7.: Klassendiagramm der Klasse MetricFactory

Jede über die `MetricFactory` erhaltene Metrik implementiert das ebenfalls in Abbildung 5.7 dargestellte Interface `IMetric`. Dieses bietet Methoden um sowohl die Kennzahl für die Metrik als auch ein Html-String mit einem Text für die Metrik abzufragen.

Nach diesem Schema kann nun für jede Metrik eine Heuristik in die Erfahrungsbasis integriert werden. Dabei ist es möglich die Heuristik so zu konstruieren, dass sie nur aktiv wird, wenn die Maßzahl in einem bestimmten Bereich liegt.

In Listing 5.1 ist die Heuristik für eine Häufigkeitsmetrik dargestellt. Diese berechnet als Kennzahl das Vorkommen des Glossareintrags im gesamten Anforderungsdokument. Falls kein Eintrag gefunden wurde, wird der entsprechende Glossareintrag im Projektbaum mittels des Experience-Plugins markiert. Ebenso wird ein Eintrag in der Problemliste und im Erfahrungsview erstellt.

```
for(var i = 0; i &<; project.getAllElements().length ; i++){
  var element = project.getAllElements()[i];
  if(element.getClass().getName().equals("de.unihannover.se.uce.
     glossary.model.GlossaryEntry"))
  {
    var metricFactory = element.getMetricFactory();
    var countMetric = metricFactory.getMetricByName("UsesMetric");

    if (countMetric.getMetricForEntry(element)==0)
    {
      contextList.addContext("<html>"+countMetric.
         getMetricStringForEntry(element), project, element);
```

```
        }
    }
}
```

Listing 5.1: Eine Heuristik für die Häufigkeitsmetrik

Für Anwender mit der Benutzerrolle *Glossar-Autor* wurde festgelegt, dass sie einen Überblick über die Metriken bekommen sollen. Aus diesem Grund wird während im Projektbaum ein Glossareintrag ausgewählt ist eine Übersicht über die den Eintrag betreffenden Metriken anstelle der Vorschlagsliste angezeigt. Diese ist in Abbildung 5.8 zu sehen.

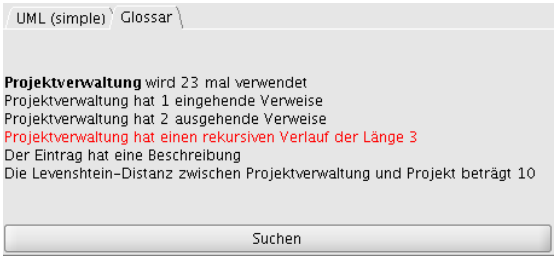


Abbildung 5.8.: Übersicht über die Metriken für einen Glossareintrag

## 5.2. Unterstützung des Byproduct-Konzepts

Neben dem beschriebenen Feedback an den Benutzer ist das zweite wesentliche Merkmal für die Integration in die UCDE die Berücksichtigung des Byproduct-Konzeptes.

Dies bedeutet, dass alle Funktionen für den Benutzer nebenbei zu erreichen sind. Es ist also nicht nötig die eigentliche bzw. angefangene Arbeit zu unterbrechen um bestimmte Funktionen auszuführen.

Das entwickelte Plugin unterstützt das Byproduct-Konzept zum einen beim Hinzufügen von Begriffen in das Glossar und dem Nachschlagen von bereits im Glossar aufgenommenen Begriffen. Zum anderen wird das Hinzufügen von Einträgen in die Liste der zu ignorierenden Vorschläge im Byproduct-Konzept umgesetzt.

### 5.2.1. Hinzufügen von Begriffen

Das Hinzufügen von neuen Begriffen zum Glossar kann ebenso im Byproduct-Konzept ermöglicht werden.

Hier ist es allerdings schwieriger zu bewerten ob das Byproduct-Konzept vollständig umgesetzt werden soll. Das Problem hierbei ist zu entscheiden ob ein neu dem Glossar hinzugefügter Begriff angezeigt werden soll oder nicht.

Entscheidet man sich dafür, so wird die eigentlich durchgeführte Aufgabe, nämlich das Erfassen von Anforderungen, unterbrochen. Entscheidet man sich dagegen so kann es passieren, dass die Beschreibung des Glossareintrags möglicherweise leer bleibt weil der Benutzer vergisst, dass diese noch auszufüllen ist.

Die hier vorgestellte Implementierung setzt abhängig vom Kontext beide Methoden ein.

Beim Hinzufügen eines neuen Glossareintrags über den Projektbaum wird der neue Eintrag direkt angezeigt. Dies hat den Grund dass davon ausgegangen werden kann dass der Benutzer sich direkt mit dem neuen Eintrag beschäftigen will. Da die über den Projektbaum hinzugefügten Einträge lediglich Platzhalternamen besitzen ist es nötig diese zu ändern um den gewünschten Begriff schließlich im Glossar zu haben. Hierfür ist es nützlich direkt zum neuen Eintrag zu gelangen.

Außer dem Projektbaum gibt es noch zwei weitere Methoden einen Eintrag zum Glossar hinzuzufügen.

Zum einen kann dies über die Vorschlagsliste passieren. Hierzu ist es lediglich nötig den entsprechenden Begriff anzuklicken.

Zum zweiten werden die Eingabefelder in den bestehenden Editoren um ein Kontextmenü - wie in Abbildung 5.9 gezeigt - erweitert, mit dem es möglich ist, Einträge zum Glossar hinzuzufügen.

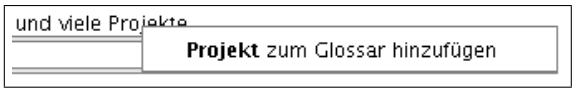


Abbildung 5.9.: Kontextmenü zum Hinzufügen neuer Einträge

Hiermit ist es dem Benutzer möglich gezielt Einträge auszuwählen, die möglicherweise nicht in der Vorschlagsliste auftauchen.

Der Unterschied dieser beiden Verfahren zum Hinzufügen über den Projektbaum ist, dass hierbei direkt der Begriff als Name des Eintrags festgelegt wird. Der Benutzer muss sich also nicht darum kümmern diesen einzutragen.

Bei diesen beiden Methoden wird der neue Eintrag nicht angezeigt, da angenommen wird, dass der Benutzer den Eintrag zwar hinzufügen will, jedoch gerade mit der Erfassung von Anforderungen beschäftigt ist. Diese beiden Methoden setzen also das Byproduct-Konzept um.

Wie oben beschrieben besteht hierbei das Problem dass hierbei Glossareinträge entstehen, die eine leere Beschreibung enthalten.

Um dies zu verhindern wird einen Heuristik in das Erfahrungs-Plugin aufgenommen, die einen Glossareintrag mit einer leeren Beschreibung markiert. Hierdurch wird dem Benutzer angezeigt, dass bei diesem Glossareintrag noch Nacharbeit nötig ist.

Die Heuristik für diese Erfahrung ist in Listing A.1 beschrieben.

### 5.2.2. Ignorieren von Einträgen

Wie oben beschrieben ist die Filterung ein wesentlicher Bestandteil der Vorschlagsliste. Hierdurch werden mögliche Vorschläge aussortiert, die vom Benutzer als nicht interessant für einen Aufnahme in das Glossar festgelegt wurde.

Um einen Filter zu erstellen muss der Benutzer über die Filterverwaltung den Dialog für neue Filter (Abbildung 5.11) aufrufen. Das Erstellen eines Filters ist ein Prozess, der nicht nebenbei erledigt werden kann. Hierzu ist es nötig die Erstellung der Anforderungen zu unterbrechen und sich exklusiv mit der Erstellung des Filters zu beschäftigen.

Dieses Verfahren ist gut geeignet wenn entweder eine Liste von Begriffen oder ein regulärer Ausdruck erfasst werden soll. Dies ist ein Vorgang für den ohnehin die Erfassung von Anforderungen unterbrochen werden muss.

Für den Fall, dass ein Benutzer einen bestimmten Begriff aus der Vorschlagsliste ignorieren möchte ist diese Verfahren allerdings nicht gut geeignet.

Um dem Benutzer die Möglichkeit zu geben einzelne Vorschläge zu ignorieren ohne einen Filter dafür konstruieren zu müssen wird vom Programm ein Filter *ignore* mitgeführt. Dieser ist standardmäßig aktiv und enthält alle Einträge welche der Benutzer ignorieren möchte. Da dieser sich ansonsten wie ein normaler Filter in das Programm einbindet, lassen sich die ignorierten Einträge mit den normalen Werkzeugen zur Filterverwaltung bearbeiten oder auch ganz wieder daraus entfernen.

Um einen Begriff zu diesem Filter hinzuzufügen genügt es in der Vorschlagsliste den Link *Ignorieren* hinter dem entsprechenden Begriff zu klicken. Eine Unterbrechung des aktuellen Vorgangs ist nicht nötig.

## 5.3. Integration der Filterung in die UCDE

Um dem Benutzer zu ermöglichen die Filterung an seine Bedürfnisse anzupassen stehen diesem zwei verschiedene Oberflächen zur Verfügung.

Für die Verwaltung bestehender Filter wird - wie auch für die Vorschlagsliste - ein Htmlview benutzt. Diese zeigt alle verfügbaren Filter an und erlaubt diese zu (de)aktivieren, zu bearbeiten oder zu löschen.

Dieses in Abbildung 5.10 dargestellte View wird anstelle der Vorschlagliste dargestellt. Dies hat den Grund, dass so für das Glossar nur ein zusätzliches View in die UCDE integriert werden muss. Außerdem ist es für die Funktionalität von sowohl Vorschlagsliste als auch Filtermanagement nicht nötig gleichzeitig beide Ansichten zu sehen. Dies hilft den ohnehin knappen Platz im UCDE sinnvoll zu nutzen.

Für die Erstellung eines neuen sowie die Bearbeitung eines bereits bestehenden Filters ist der in Abbildung 5.11 dargestellte Dialog zuständig. In ihm ist es möglich sämtliche Parameter für einen Filter und die in ihm enthaltenen FilterItems einzustellen. Um dem Benutzer ein schnelles Verstehen des Filters zu ermöglichen sind sämtliche Einträge mit Ausdruck und Einschränkung der Häufigkeit in einer Liste dargestellt.


Abbildung 5.10.: Filtermanagement

Hierbei wird die allgemein gebräuchliche Form der Darstellung **Ausdruck** $\leq$ **Anzahl** *bzw.* **Ausdruck** $\geq$ **Anzahl** benutzt.

Zusätzlich enthält der Dialog eine Liste mit allen möglichen Vorschlägen. Diese wiederum werden nicht gefiltert damit es möglich ist sämtliche im Projekt gefundenen Vorschläge direkt in einen Filter einzubinden.

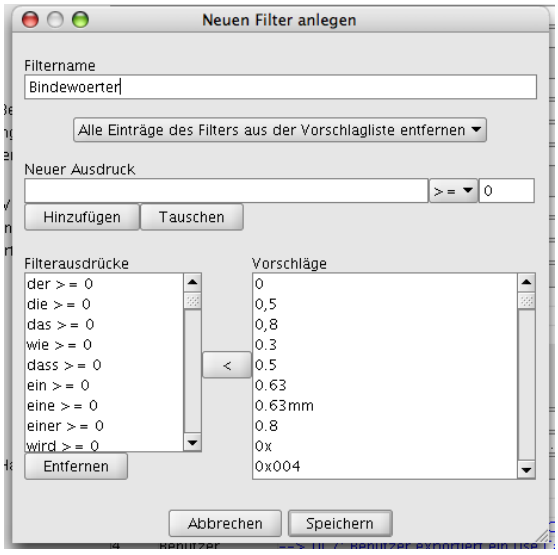


Abbildung 5.11.: Dialog für die Erstellung eines neuen Filters

## 5.4. Glossareinträge

Um die Integration von Glossareinträgen in die UCDE zu vollziehen ist es zunächst nötig deren Stellenwert innerhalb eines Anforderungsdokumentes zu klären.

Es ist das Ziel eines Glossar die erstellten Anforderungen durch die Erklärung ausgewählter Begriffe zu erläutern. Hierdurch soll verhindert werden, dass im weiteren Verlauf des Projektes Fehler aufgrund von falsch verstandenen Begriffen entstehen.

In das Glossar aufgenommene Begriffe sollen für den UCDE-Benutzer deutlich gekennzeichnet sein. So wird der Benutzer darauf hingewiesen, dass ein Begriff eine spezifische Bedeutung besitzt welche im Glossar nachgeschlagen werden kann.

Weiterhin soll dem Benutzer die Möglichkeit gegeben werden während der Erstellung von Anforderungen so einfach und schnell wie möglich Begriffe in das Glossar aufzunehmen. Hierfür ist eine Benutzerführung nötig bei der der Benutzer sich nicht erst durch verschiedene Dialoge klicken muss und so sein Hauptanliegen - nämlich das Erstellen von Anforderungen - aus den Augen verliert.

Aus diesen Gründen sollen Glossareinträge gleichwertig zu anderen Elementen eines Anforderungsprojektes wie z.B. Use Case's behandelt werden.

Dies bedeutet zum einen, dass Glossareinträge in der Projektstruktur der UCDE erscheinen. Zum anderen bedeutet dies, dass für die Bearbeitung von Glossareinträgen ein eigener Editor zur Verfügung steht, der im Hauptteil der UCDE angezeigt wird.

Für die Integration von Glossareinträgen in ein Anforderungsdokument wurde die Klasse `GlossaryEntry` erstellt. Diese implementiert drei Interfaces:

Zunächst werden `IModelElement` und `IPersistentModelClass` implementiert. Dies ist nötig um die Klasse in das vorgegebene Framework der UCDE einzubinden und um sie mittels der UCDE speichern bzw. laden zu können.

Danach wurde noch `IGlossaryEntry` implementiert um eine Kompatibilität zum Glossar-Kern zu schaffen und die Glossareinträge auch von diesem intern verwalten zu lassen.

Die Verwaltung der einzelnen Einträge durch den Glossarkern ist nicht unbedingt nötig. Jedoch ermöglicht er dass beispielsweise beim Extrahieren von Vorschlägen aus einem Text automatisch alle vorhandenen Glossareinträge entfernt werden und so nicht mehr in der Liste der Vorschläge auftauchen oder das Suchen nach Glossareinträgen innerhalb eines Texts.

Um einen Glossareintrag zu bearbeiten wird ein eigener Editor erstellt. Dieser muss den Namen des Glossareintrags und die Beschreibung anzeigen und bearbeiten können.

Für diesen Editor soll es zwei Zustände geben. Der erste Zustand zeigt den Glossareintrag an. Im zweiten Zustand kann der Eintrag bearbeitet werden. Diese Unterteilung ermöglicht ein unterschiedliches Verhalten des Editors abhängig von seiner aktuellen Aufgabe.

Der Vorteil dieser Unterteilung ist, dass für Bearbeitung und Ansicht jeweils unterschiedliche Komponenten benutzt werden können. So werden für die Bearbeitung normale Texteingabe-Komponenten benutzt, für die Ansicht jedoch ein Html-View. So wird es ermöglicht in der Beschreibung eines Glossareintrags Verlinkungen zu benutzen.

Für den Fall, dass die Beschreibung eines Glossareintrags einen Begriff verwendet,

der ebenfalls im Glossar aufgenommen ist, wird dieser Begriff in der Beschreibung als Hyperlink angezeigt. Dies gibt zum einen dem Benutzer ein Feedback dass diese Begriffe ebenfalls erklärt sind. Zum anderen ermöglicht es die Navigation innerhalb des Glossars auf einfache Weise. Abbildung 5.12 zeigt beide Ansichten.


Abbildung 5.12.: Glossareintrag im Ansichts- und im Bearbeitungsmodus

# 6. Qualitätsbestimmung

## 6.1. Qualität von Wörterbüchern

Wie beschrieben werden für die Einordnung eines Wortes in eine Sprache ein oder mehrer Wörterbücher herangezogen. Ebenso arbeiten Algorithmen wie die zur Grundformenbildung benutzte Lemmatisierung auf Grundlage von Wörterbüchern.

Die Einordnung von Wörtern in eine Sprache mit Hilfe eines Wörterbuchs erfolgt, indem für das übergebene Wort geprüft wird ob es einem Eintrag im Wörterbuch entspricht. Für die Lemmatisierung wird dies ebenfalls überprüft und zusätzlich noch dessen Grundform ermittelt.

Die Verwendung der Wörterbücher erfolgt also in beiden Fällen gleich, so dass ein Qualitätsmodell für Wörterbücher lediglich eine einzige Kenngröße beachten muss. Dies ist die Abdeckung einer Sprache durch das entsprechende Wörterbuch.

Die Qualitätskennzahl für ein Wörterbuch wird 0 sein, wenn kein Wort der Sprache enthalten ist. Die ist für leere Wörterbücher der Fall. Die Qualitätskennzahl wird 1 sein, wenn das Wörterbuch die komplette Sprache abdeckt.

Offensichtlich ist es nicht möglich eine lebende Sprache komplett durch ein Wörterbuch abdecken, da sie ständig im Wandel begriffen ist. Jedoch kann man sich einer kompletten Abdeckung beliebig gut annähern.

## 6.2. Qualität von Grundformen-Algorithmen

Durch die Entscheidung nur Grundformen für das Glossar zuzulassen ist die Qualität der Grundformen-Algorithmen von herrausragender Mitbedeutung für die Qualität des Glossars. Ein schlechter Grundformen-Algorithmus würde entweder dazu führen, dass Begriffe mit unterschiedlicher Bedeutung auf die gleiche Grundform reduziert werden oder dazu, dass bei Begriffen gleicher Semantik verschiedene Wortformen in das Glossar aufgenommen werden.

Für die Qualität von Grundformen-Algorithmen lassen sich folgende Kenngrößen feststellen:

- Die Reduzierungseffektivität
- Die Übereinstimmung der Rückgabe mit Definition 3.4.1

Die Reduzierungseffektivität beschreibt hierbei den tatsächlichen Erfolgsfaktor um eine Wortform auf ihre Grundform zurück zu führen.

Ein Algorithmus mit einer Reduzierungseffektivität $< 1$ wird etwa zwei unterschiedliche Wortformen nicht auf die gleiche Grundform zurückführen, so dass das Glossar beide Einträge enthält was jedoch nicht beabsichtigt ist. Ein Algorithmus mit einer Reduzierungseffektivität $> 2$ wird eine zu große Anzahl von Wortformen auf die gleiche Grundform zurückführen.

Die Reduzierungseffektivität ist 0, wenn kein Begriff auf eine Grundform zurückgeführt wird, die Reduzierungsfunktion also $f(w) = w$ ist. Eine Reduzierungseffektivität von 1 bedeutet, dass jeder Begriff auf seine Grundform reduziert wird. Bei einer Reduzierungseffektivität von 2 wird jeder Begriff auf die gleiche Grundform reduziert. Die Funktion ist hier $f(w) = C$.

Die Kennzahl für die Übereinstimmung der Rückgabe mit Definition 3.4.1 ist 0, wenn keine Rückgabe mit dieser Definition übereinstimmt. Sie ist 1, wenn jede Rückgabe mit der Definition übereinstimmt.

Die Kennzahl des Qualitätsmodell ist 1, wenn die Reduzierungseffektivität und die Übereinstimmung mit der Definition 3.4.1 ebenfalls 1 ist.

Die Übereinstimmung der Rückgabe mit Definition 3.4.1 geht linear in die Kennzahl des Qualitätsmodells ein, ebenso wie eine Reduzierungseffektivität $< 1$.

Für Reduzierungseffektivitäten $> 1$ wird sich die Kennzahl des Qualitätsmodells stark verschlechtern. Hierbei geht die Kennzahl als $\frac{1}{k} - 0.5$ in die Gesamtkennzahl ein.

Der Grund für die unterschiedliche Behandlung der Reduzierungseffektivität ist folgendermaßen zu erklären:

Bei einer Reduzierungseffektivität $< 1$ enstehen für die gleiche Grundform möglicherweise verschiedene Glossareinträge unterschiedliche Wortformen. Dieses Problem lässt sich jedoch dadurch beheben, dass bspw. ein Benutzerwörterbuch eingeführt wird, das diese Wortformen einer Grundform zuordnet.

Bei einer Reduzierungseffektivität $> 1$ kommt es vor, dass mehrere Wortformen auf eine gemeinsame Grundform zurückgeführt werden obwohl sie unterschiedliche Bedeutungen haben. So könnten bspw. *Fische* und *Fischern* beide auf die Grundform *Fisch* zurückgeführt werden, was eine Unterscheidung beider Formen nicht mehr ermöglicht. Da hierbei ein Informationsverlust erfolgt, dem im Nachhinein nicht mehr entgegen gewirkt werden kann, verschlechtert dies die Kennzahl des Qualitätsmodells erheblich.

### 6.2.1. Bewertung der vorgestellten Algorithmen

Im folgenden sollen die in Abschnitt 3.5 vorgestellten Algorithmen zur Grundformbildung in das beschriebene Qualitätsmodell eingeordnet werden. Hierbei wird bewusst darauf verzichtet die Qualität der Algorithmen für Mischtexte unterschiedlicher Sprachen mitzubewerten. Es gilt die Annahme, dass durch eine Vorverarbeitung jeweils die korrekte Implementierung für die Sprache des Eingangswortes gewählt wird.

### Heuristiken

Da die Ergebnisse für Heuristiken nicht unbedingt vorhersehbar sind, fällt es schwer diese mit Kennzahlen zu belegen.

Offensichtlich ist jedoch, dass die Rückgaben einer Heuristik nicht unbedingt Definition 3.4.1 entsprechen. Die Kennzahl hierfür wird also $< 1$ sein.

Die Reduzierungseffektivität ist deutlich schwerer zu bestimmen, da sie für jede Heuristik anders sein kann. Bei sorgfältiger Konstruktion der Heuristiken kann angenommen werden, dass sich die Reduzierungseffektivität der Kennzahl 1 beliebig genau annähert. Jedoch ist der Aufwand für die Konstruktion eines Satzes von Heuristiken die solche eine gute Kennzahl liefert vermutlich deutlich zu hoch. Deshalb wird angenommen, dass die mittlere Kennzahl für Heuristiken sich bei 0.5 befindet.

### Stemming

Da das Stemming ähnlich wie Heuristiken nicht zwangsläufig Wortformen nach Definition 3.4.1 liefert, ist auch hierfür die Kennzahl $< 1$.

Für den Fall dass der eingesetzte Stemmer *overstemming* produziert, also eine zu lange Zeichenkette abschneidt und damit Wörter mit unterschiedlicher Bedeutung gleichsetzt, ist die Kennzahl für die Reduzierungseffektivität $> 1$. Für Stemming-Algorithmen sind overstemming-Fehler jedoch nicht vollständig auszuschließen, so dass die Kennzahl für die Reduzierungseffektivität im Normalfall $> 1$ angenommen werden kann.

Dies bedeutet, dass Stemmer im Allgemeinen eine Qualitätkennzahl deutlich kleiner 1 haben, da die Kennzahl zur Reduzierungseffektivität für einen Wert $> 1$ wesentlich stärker ins Gewicht fällt als die Kennzahl für die Ausgaben nach Definition 3.4.1.

### Lemmatisierung

Für die Lemmatisierung ist es entscheidend ob das Wörterbuch eine Grundform zu einer ihr übergebenen Wortform enthält.

Da die Lemmatisierung im einfachsten Fall nur im Wörterbuch nachschlägt müssen dessen Einträge entsprechend der Definition 3.4.1 vorliegen. Da diese Definition sich an schon existierenden Wörterbüchern orientiert, ist anzunehmen dass die in den benutzten Wörterbüchern verzeichneten Grundformen dieser Definition entsprechen.

Im Falle eines idealen Wörterbuchs - also eines Wörterbuchs mit der Qualitätskennzahl 1 - ist die Kennzahl für die Rückgabe 1 da alle möglichen Wörter der Sprache im Wörterbuch vorliegen und sich alle Wortformen auf ihre Grundform zurückführen lassen. Im Normalfall wird die Kennzahl $< 1$ sein, da ein Wörterbuch nicht die komplette Sprache erfasst.

Die Reduzierungseffektivität ist $< 1$. Da die durch die eingesetzten Wörterbücher vorgegebenen Regeln zur Reduzierung nicht zulassen Wortformen mit unterschiedlicher

Bedeutung auf die gleiche Grundform zurückzuführen, hängt auch diese Kennzahl direkt von der Wörterbuchqualität ab. Ein Wörterbuch welches die komplette Sprache abdeckt kann auch jede Wortform zu einer Grundform zurückführen, wenn die morphologischen Regeln hierfür vorhanden sind.

Die Gesamtqualität der Lemmatisierung wird sich also linear an 1 annähern je besser das eingesetzte Wörterbuch ist.

#### Fazit

Aus den oben angegeben Qualitätskennzahlen lässt sich die Lemmatisierung als Algorithmus der Wahl ableiten.

Das Stemming disqualifiziert sich aufgrund der Probleme mit overstemming und der geringen Übereinstimmung der Rückgaben mit Definition 3.4.1.

Die Heuristiken können zwar potenziell so angepasst werden, dass sie sich beliebig nahe einer Qualität von 1 annähern jedoch ist der Aufwand dafür sehr hoch.

Die Qualität der Lemmatisierung lässt sich direkt durch die benutzten Wörterbücher steuern. Da für nahezu alle Sprachen Wörterbücher vorliegen ist der Aufwand eine erste Lemmatisierung zu benutzen sehr gering. Außerdem bietet sich der Vorteil durch Austauch der Wörterbücher auch später noch die Qualität zu beeinflussen ohne Änderungen am Quellcode durchzuführen. Nimmt man weiterhin an, dass auch nicht-freie Wörterbücher zur Verfügung stehen, wird sich die Qualität im Vergleich zu den in dieser Arbeit benutzten, freien Wörterbüchern vermutlich nocheinmal deutlich steigern lassen.

## 6.3. Qualität der Vorschläge

Ein Vorschlag ist dann von hoher Qualität, wenn er in das Glossar aufgenommen werden kann. Ein Qualitätsmodell für Vorschläge hat als Eingangsgröße lediglich die Aufnahme in ein Glossar.

Eine Qualitätskennzahl von 1 bedeutet dabei, dass der Vorschlag aufgenommen wurde, eine Kennzahl von 0 bedeutet, dass er nicht aufgenommen wurde.

Betrachtet man nur ein einziges Anforderungsprojekt, kann die Qualitätskennzahl für einen Vorschlag lediglich den Wert 1 oder den Wert 0 annehmen, er kann entweder in ein Glossar übernommen werden oder nicht.

Aus dieser Beobachtung leitet sich der Schluss ab, dass ein Qualitätsmodell für Vorschläge nur Sinn macht, wenn mehrere Projekte untersucht werden. Hierbei ist zu beachten dass die Projekte ähnlichen Typs sein müssen um vergleichbar zu sein.

Für Projekte der gleichen Domäne kann angenommen werden, dass die Menge an Fachbegriffen immer fast identisch ist. Zwar wird sich gesamt betrachtet die Menge

etwas verändern indem neue Fachbegriffe hinzukommen und veraltete nicht mehr benutzt werden, jedoch wird der Kern immer gleich bleiben.

Untersucht man nun verschiedene Projekte der gleichen Domäne, so kann eine Qualitätskennzahl für einen Vorschlag gebildet werden, die nicht nur 0 oder 1 sondern auch jeden Wert dazwischen annehmen.

Hierzu untersucht man für jedes Projekt welche Vorschläge aus den Anforderungen ermittelt werden konnten und welche davon in ein Glossar aufgenommen wurden. Die genaue Kennzahl ergibt sich dann aus der Summe der Einzelkennzahlen geteilt durch die Summe der Vorkommen des Begriffes.

Wenn ein Begriff in jedem Projekt, in dem er vorkommt in das Glossar aufgenommen wurde, ergibt sich eine Qualitätskennzahl von 1. Falls ein Begriff in keinem Projekt in dem er vorkommt in das Glossar aufgenommen wurde so ist die Kennzahl 0. Ansonsten bewegt sie sich zwischen diesen beiden Grenzen.

## 6.4. Qualität der Vorschlagsliste

Die Vorschlagsliste ist ein wesentliches Prinzip des in dieser Arbeit entwickelten Plugins. Hierrüber wird ein direktes Feedback an den Benutzer gegeben, welche Begriffe möglicherweise für ein Glossar in Betracht kommen.

Das Ziel der Vorschlagsliste ist es eine Menge von Begriffen zu präsentieren, die in ein Glossar aufgenommen werden. Hierbei ist zu beachten, dass die endgültige Vorschlagsliste abhängig von der angewandten Filterung möglicherweise nicht alle gefundenen Begriffe enthält.

Mit der Vorschlagsliste soll erreicht werden, dass der Benutzer die Begriffe präsentiert bekommt, die er auch ins Glossar aufnehmen würde. Ein Qualitätsmodell für eine Vorschlagsliste bekommt als Eingangsgröße die Anzahl der in das Glossar aufgenommenen Vorschläge und die Anzahl aller präsentierten Vorschläge. Die Qualitätskennzahl ergibt sich dann als $\frac{\text{Aufgenommene Vorschläge}}{\text{Anzahl Vorschläge}}$.

Werden alle Vorschläge in das Glossar aufgenommen hat die Vorschlagsliste also die Qualitätskennzahl 1, wurde garkein Vorschlag aufgenommen die Kennzahl 0.

Durch die Beschränkung der Vorschlagsliste durch eine Filterung ist die Kennzahl abhängig von der Qualität der verwendeten Filter. Ein Filter, der alle Vorschläge herausfiltert, die eine Qualitätskennzahl kleiner 1 haben wird zur Verbesserung der Qualität der Vorschlagsliste beitragen. Ein Filter, der nur Vorschläge mit der Qualität 1 herausfiltert, wird somit zur Verschlechterung der Qualität der Vorschlagsliste führen.

## 6.5. Qualität eines Filters

Die Aufgabe eines Filter ist es aus der Menge aller gefundenen Vorschläge diejenigen herauszunehmen, die nicht für die Aufnahme in ein Glossar in Frage kommen.

Die Qualitätskennzahl für einen Filter ist 1, wenn er nur die Vorschläge filtert die nicht aufgenommen werden und genau diejenigen unangetastet lässt, welche aufgenommen werden. Die Qualitätskennzahl ist 0, wenn genau die Vorschläge ausgefiltert werden, die in ein Glossar aufgenommen werden und die uninteressanten Einträge übrig bleiben.

Ein Nachweis über die Qualität eines Filters lässt sich nur indirekt führen. Hierzu ist es zuerst nötig das Anforderungsprojekt inklusive Glossar vollständig zu erstellen. Ist das Projekt abgeschlossen bedeutet dies, dass alle Vorschläge aus dem Projekt bekannt sind ebenso wie die aufgenommenen Glossareinträge. Wendet man nun den Filter auf die Menge aller Vorschläge an, so erhält man die Qualitätskennzahl für den Filter.

Analog zur Qualität der Vorschläge ist die Qualitätskennzahl umso belastbarer, je mehr gleichartige Projekte untersucht werden.

## 6.6. Qualität eines Glossars

Die Qualität eines Glossar setzt sich aus der Qualität der einzelnen Einträge zusammen. Im Folgenden werden Metriken aufgestellt um die Qualität einzelner Glossareinträge zu bewerten. Für jede Metrik ist angegeben wie sie die Qualitätskennzahl des Eintrags und des gesamten Glossars beeinflusst.

Die endgültige Qualitätskennzahl setzt sich dann aus den einzelnen Metriken zusammen. Das Gewicht einer Metrik ergibt sich als $\frac{1}{\text{Anzahl Metriken}}$.

Die entwickelten Metriken werden im Rahmen dieser Arbeit in das Plugin eingebunden und lassen sich über das Erfahrungs-Plugin in die Erfahrungsbasis der UCDE integrieren.

### 6.6.1. Leere Definitionen

Die Aufgabe eines Glossar ist es durch eindeutige Definition eines Begriffs dazu beizutragen, dass keine Fehler aufgrund von falsch verstandenen Begriffen entstehen.

Begriffe, die keine oder nur eine leere Definition haben, sind zwar Bestandteil des Glossar, tragen aber nicht zur Klärung bei. Sie können im schlechtesten Fall sogar zur Verwirrung von Projektbeteiligten führen. Es kann passieren, das zwar jedem Beteiligten bekannt ist, dass ein Begriff im Glossar enthalten ist, jedoch wird aufgrund der fehlenden Definition wiederum jeder seine eigene Interpretation des Begriffs benutzen. Durch die Aufnahme in das Glossar wird eventuell sogar jeder Beteiligte angeregt sich eine eigene Bedeutung zu überlegen ohne dass diese am Glossar verifiziert werden

kann.

Eine Metrik hierfür wertet lediglich aus ob die Begriffsdefinition leer ist oder nicht und gibt entsprechend 0 oder 1 zurück.

Die entsprechende Heuristik für das Erfahrungs-Plugin ist in Listing A.1 abgedruckt.

Eine Kennzahl von 0 wird die Qualität des Eintrags und die des gesamten Glossars negativ beeinflussen, eine Kennzahl von 1 hingegen positiv.

### 6.6.2. Verwendung innerhalb des Projekts

Eine weitere Metrik, die zur Bildung einer Qualitätskennzahl für einen Glossareintrag benutzt werden kann ist die Vorkommenshäufigkeit.

Im Zusammenhang mit der Vorschlagsliste ist die Vorkommenshäufigkeit benutzt worden um die Stelle eines Begriffs in der Liste festzulegen. Dabei war die Position umso höher, desto öfter der Begriff vorkam.

Diese gleiche Heuristik kann auch für Begriffe benutzt werden, die im Glossar aufgenommen wurden. Hierbei gilt die Annahme, dass ein Begriff um so bedeutender für einen Text ist desto häufiger er vorkommt. Zwar kommen auch Bindewörter häufig im Text vor, jedoch können diese in diesem Kontext vernachlässigt werden, da diese Metrik sich lediglich auf bereits aufgenommene Begriffe bezieht. Von diesen wird angenommen, dass sie zur Bedeutung des Anforderungsprojektes beitragen.

Um zu entscheiden ob die Aufnahme eines Begriffes in das Glossar gut war soll eine Metrik prüfen wie oft der Begriff im gesamten Anforderungsprojekt vorkommt.

Für die Interpretation des Wertes gilt, dass Begriffe, welche nicht im Projekt vorkommen keine gute Wahl für die Aufnahme waren. Hier wird ein Begriff zwar definiert aber nicht verwendet. Einem Benutzer der solch ein Projekt durchschaut wird nicht erkenntlich werden wieso dieser Begriff aufgenommen wurde und welche Bedeutung er für das gesamte Projekt hat. Für alle Begriffe die mindestens einmal im Projekt vorkommen gilt, dass die Aufnahme eine gute Wahl war.

Für die Qualität des Glossareintrags ebenso wie für die Qualität des gesamten Glossars bedeutet ein Wert von 0 eine Verschiebung der Qualitätskennzahl gegen 0. Ein höherer Wert verschiebt die Qualitätskennzahl des entsprechenden Eintrag gegen 1.

Listing 5.1 zeigt eine Heuristik, die Glossareinträge markiert, welche nicht im Projekt verwendet werden.

### 6.6.3. Verweis auf den Glossareintrag

Ähnlich wie die Vorkommenshäufigkeit ist ein weiteres Indiz für die Güte eines aufgenommenen Glossareintrags die Anzahl der Verweise auf diesen Eintrag. Ein Verweis entsteht, wenn ein Glossareintrag in seiner Beschreibung einen anderne referenziert.

Ein Glossareintrag, der eine große Anzahl auf ihn zeigender Verweise hat ist im Glossar

gut aufgehoben, da er wesentlich zur Erläuterung der ihn benutzenden Einträge beiträgt.

Um zu einer Interpretation zu gelangen muss zunächst festgelegt werden wie Verweise gezählt werden. Die Beschreibung eines Glossareintrags kann mehrfach einen anderen Eintrag verwenden. Es wird festgelegt, dass lediglich das Vorkommen einer Referenz gezählt wird, nicht die Häufigkeit. Ein Eintrag, der einen anderen mehrfach verwendet wird also nur eine Erhöhung des Verweiszählers um 1 nach sich ziehen.

Dies bedeutet, dass die maximale Zahl für den Verweiszähler bei $n$ Glossareinträgen genau $n$ ist. Da ein Verweis eines Glossareintrags auf sich selbst als Kreisreferenz gelten und somit durch eine andere Metrik behandelt werden soll, wird der betreffenden Eintrag selbst nicht untersucht. Die maximale Größe für den Verweiszähler ist also $n - 1$.

Abbildung 6.1 zeigt einen Begriff mit $n - 1$ eingehenden Verweisen.

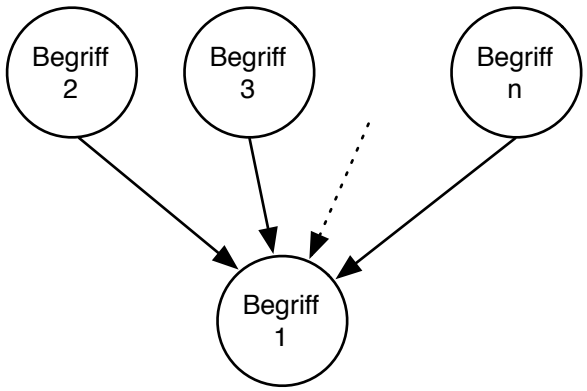


Abbildung 6.1.: Beispiele für Verweise auf einen bestimmten Begriff

Dieser Wert kann wie folgt interpretiert werden: Eine Verweiszähler größer 0 steigert die Güte der Aufnahmeentscheidung des untersuchten Begriffs.

Anders als bei der Vorkommenshäufigkeit bedeutet eine Kennzahl von 0 allerdings nicht, dass die Aufnahmeentscheidung nicht gut war. So kann es vorkommen, dass ein Glossareintrag zwar sehr häufig im Projekt vorkommt jedoch nicht bei der Beschreibung anderer Glossareinträge benutzt wird.

Für die Qualität des Glossareintrags selbst gilt, dass eine Kennzahl von $n - 1$ die Qualitätskennzahl am stärksten gegen 1 steigen lässt. Eine Kennzahl von 0 lässt die Qualitätskennzahl am stärksten gegen 0 sinken. Eine Kennzahl von $\frac{n-1}{2}$ beeinflusst die Qualitätskennzahl nicht.

Für die Qualität des gesamten Glossar gilt das Gegenteil. Hier lässt eine Kennzahl von $n - 1$ die Qualitätskennzahl gegen 0 sinken, eine Kennzahl von $0$ lässt jedoch die Qualitätskennzahl gegen 1 steigen.

Die gegensätzliche Interpretation lässt sich wie folgt erklären: Aus Sicht eines einzelnen Glossareintrags ist eine hohe Verweiszahl gut. Denn dies bedeutet, dass der Eintrag eine große Rolle bei der Definition anderer Einträge spielt. Die Aufnahme in das Glossar war also eine gute Wahl.

Aus Sicht des gesamten Glossar bedeutet eine hohe Verweiszahl hingegen, dass das die anderen Einträge des Glossars nicht selbsterklärend sind. Das Nachschlagen eines einzigen Eintrags bedeutet dann, dass der Benutzer sich weiter durch das Glossar bewegen muss. Das Glossar erfüllt so nicht die Aufgabe eine klare Definition eines Begriffes zu liefern.

Das Listing A.2 markiert einen Glossareintrag, wenn die Anzahl der eingehenden Verweise größer als $\frac{n-1}{2}$ ist.

### 6.6.4. Verweise auf andere Glossareinträge

Im vorherigen Abschnitt wurden eingehende Verweise auf einen Eintrag betrachtet. In diesem Abschnitt sollen nun ausgehende Verweise betrachtet werden.

Ein ausgehender Verweis ist die Verwendung eines anderen Begriffs in der Beschreibung des betrachteten Glossareintrags. Hierbei soll wieder gelten, dass ein mehrfach auftretender Begriff nur einfach gezählt werden kann. Lässt man weiterhin den betrachteten Glossareintrag außer Acht, so kann bei $n$ Glossareinträgen der Maximalwert für die Anzahl der ausgehenden Verweis $n - 1$ sein. Abbildung 6.2 zeigt einen Begriff mit $n - 1$ ausgehenden Verweisen.

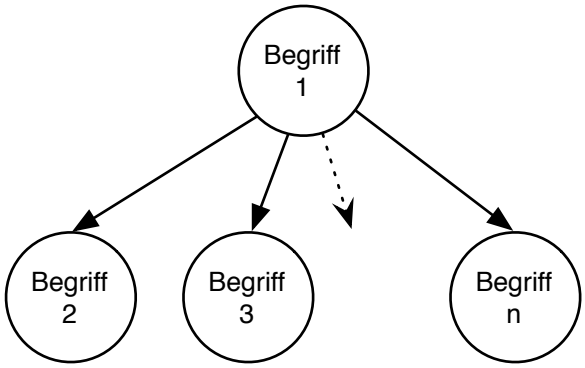


Abbildung 6.2.: Beispiele für ausgehende Verweise

Die Kennzahl dieser Metrik lässt sich wie folgt interpretieren: Ein Wert $n - 1$ bedeutet, dass der betrachtete Begriff zur Beschreibung sämtliche anderen Begriffe im Glossar benutzt. Dies widerspricht aber dem Konzept, dass ein Begriff zum Glossar hinzugefügt wird um ihn deutlich zu definieren. Die Aufnahme des Begriffes in das Glossar war keine gute Wahl. Ein Wert von 0 bedeutet, dass die Aufnahme gut war.

Für die Anteile an der Qualität des Glossareintrags und des gesamten Glossar bedeutet dies, dass für eine Kennzahl von 0 die Qualitätskennzahl in Richtung 1 beeinflusst wird. Für die Kennzahl $n - 1$ wird die Qualitätskennzahl in Richtung 0 beeinflusst.

Das Listing A.3 markiert einen Glossareintrag, wenn die Anzahl der ausgehenden Verweise größer als $\frac{n-1}{2}$ ist.

### 6.6.5. Rekursive Verweise

Im Falle der Metriken für ein- und ausgehende Verweise wurde der aktuell betrachtete Glossareintrag von der Auswertung ausgeschlossen. Beachtet man auch diese Glossareinträge selbst, so können eventuelle rekursive Verweise festgestellt werden.

Rekursive Verweise bezeichnen eine Kette von aus- und eingehenden Verweisen, die wieder beim Ausgangsbegriff ankommen. Abbildung 6.3 zeigt Beispiele hierfür. Die Länge einer rekursiven Referenz ist die Anzahl der Verweise denen gefolgt werden muss bis wieder der Startbegriff erreicht wird.

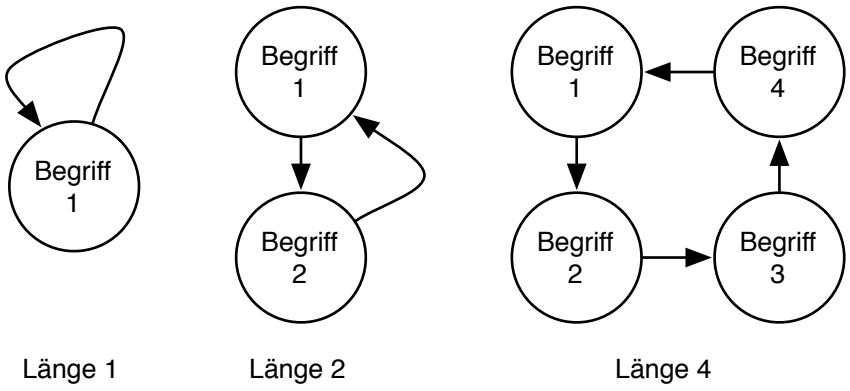


Abbildung 6.3.: Beispiele für rekursive Verweise

Ein rekursiver Verweis bedeutet aus Sicht des Benutzers, dass er in der Beschreibung eines Begriffs auf einen anderen verwiesen wird. Diese Beschreibung verweist wieder auf eine andere bis zum Schluss der Verweiskette wieder der ursprüngliche Begriff erreicht wird. Einfacher ausgedrückt bedeutet dies, dass ein Begriff mit sich selbst erklärt wird.

In Abbildung 6.3 ist zu sehen, dass rekursive Referenzen auch auftreten, wenn in der Beschreibung eines Begriffs der Begriff selbst auftritt. Solche rekursiven Verweise mit der Länge 1 sollen nicht beachtet werden, da es häufig vorkommt, dass der Begriff selbst in der Beschreibung wiederholt wird.

Für rekursive Verweise mit einer Länge größer 2 gilt, dass es hier Probleme in der Beschreibung gibt.

Werte $< 2$ verbessern die Qualitätskennzahlen von Glossareintrag und gesamten Glossar. Werte $\geq 2$ verschlechtern dieses Qualitätskennzahlen.

Listing A.4 markiert alle Glosseinträge, die rekursive Verweise mit einer Länge $> 1$ besitzen.

### 6.6.6. Ähnliche Einträge

Ähnliche Einträge sind Einträge, die sich nur geringfügig von einander unterscheiden. Um zu überprüfen wie ähnlich zwei Einträge sind wird in der Verarbeitung natürlicher Sprache überlicherweise die *Levenshtein-Distanz* benutzt.

Die Levenshtein Distanz wurde 1966 von Wladimir Levenshtein in [Lev66] beschrieben. Sie bezeichnet die minimale Distanz zweier Zeichenketten bezüglich der Operationen *löschen*, *einfügen* und *tauschen*.

Diese Methode eignet sich um beispielsweise Schreibfehler in einem Begriff zu finden. So hat ein Begriff bei dem ein Zeichen vergessen wurde eine Levenshtein-Distanz von 1 zum gemeinten Begriff. Ein Begriff bei dem zwei Buchstaben vertauscht sind hat die Levenshtein-Distanz 2.

Das Problem bei der Interpretation der Levenshtein-Distanz ist, dass kleine Distanzen nicht automatisch Fehler bedeuten. So haben beispielsweise *Sucht* und *Sicht* eine Levenshtein-Distanz von 1 und dennoch unterschiedliche Bedeutungen.

Für den Fall, dass in einem Anforderungsprojekt Begriffe vorkommen, die nur eine geringe Levenshtein-Distanz zu einander aufweisen, jedoch trotzdem eine unterschiedliche bedeutung haben, verkehrt sich diese Metrik von einer Hilfe in eine potenzielle Störung. Der Benutzer erhält Warnungen obwohl kein Problem vorliegt. Dies wird dazu führen, dass der Benutzer sich entschließt diese Erfahrung zu ignorieren.

Aus diesem Grund ist die Levenshtein-Metrik zwar implementiert und in Listing A.5 ihre Heuristik abgedruckt, jedoch wird sie nicht standardmäßig in die Erfahrungsbasis aufgenommen.

## 6.7. Abhängigkeit der Glossarqualität von der Vorschlagsqualität

Für eine Vorschlagsliste bedeutet eine hohe Qualität, dass der überwiegende Teil der Vorschläge auch in das Glossar aufgenommen wird.

Eine hohe Qualität eines Glossar setzt sich aus der durchschnittlichen Qualität der einzelnen Glossareinträge zusammen. Die Qualität der Glossareinträge wiederum wird aus den für die Glossareinträge aufgestellten Heuristiken gebildet.

Im folgenden soll untersucht werden ob eine Vorschlagsliste von hoher Qualität automatisch zu einem Glossar von ebenfalls hoher Qualität führt. Als Einschränkung wird

gewählt, dass das restliche Anforderungsdokument nicht mehr verändert wird. Dies verhindert z.B. dass Begriffe die über die Vorschlagsliste in das Glossar aufgenommen wurden danach aus dem Text entfernt uns so Qualitätskennzahlen verändert werden.

Lässt man zu, dass Einträge nicht nur über die Vorschlagsliste dem Glossar hinzugefügt werden können sondern auch über andere Wege die dem Benutzer erlauben die Begriffe selbst zu wählen, so führt dies dazu dass auch Einträge im Glossar vorkommen können welche nicht in der Vorschlagsliste präsentiert werden. Dies ist z.B. der Fall wenn der Benutzer sich bewusst entschließt die Vorschlagsliste zu ignorieren.

Dies bedeutet, dass es keinen offensichtlichen Zusammenhang zwischen vorgeschlagenen und tatsächlich in das Glossar aufgenommenen Begriffen gibt. Aus dieser Beobachtung ergibt sich die Folgerung, dass es ebenso keine Korrelation zwischen Qualität der Vorschlagsliste und Qualität des Glossars gibt.

Beschränkt man den Benutzer darauf Einträge zum Glossar nur über die Vorschlagsliste hinzuzufügen kann dies zwei Effekte haben.

Zum einen kann es passieren, dass der Benutzer die Vorschlagsliste nicht durch z.B. Filterung beschneidet um so die Möglichkeit zu behalten alle auftauchenden Begriffe zum Glossar hinzuzufügen. In diesem Fall wird die Qualität der Vorschlagsliste gegen 0 tendieren.

Zum anderen kann es passieren dass der Benutzer sich intensiv mit der Erstellung von Filtern beschäftigt um eine Vorschlagsliste zu erhalten welche genau seinen Ansprüchen genügt. In diesem Fall wird die Qualität der Vorschlagsliste gegen 1 tendieren.

Das Hinzufügen über die Vorschlagsliste bedeutet, dass die Namen der Glossareinträge automatisch gesetzt werden. Von den oben vorgestellten und benutzten Metriken bezieht sich lediglich die Vorkommenshäufigkeit allein auf den Namen eines Eintrags

Die Metrik der Vorkommenshäufigkeit wird immer einen Wert $> 0$ erhalten. Dieser ist dadurch bedingt, dass die Vorschlagsliste aus allen Texten des Anforderungsprojekts generiert wird. Dadurch kommen alle Einträge, die über die Vorschlagsliste hinzugefügt wurden mindestens einmal im Anforderungsprojekt vor. Die Garantie, dass diese Metrik immer $> 0$ ist, ist unabhängig von der Qualität der Vorschlagsliste.

Alle anderen Metriken beziehen sich auf die Beschreibung eines Begriffs. Der Zweck der Vorschlagsliste ist es lediglich Begriffe vorzuschlagen, die für eine Aufnahme in ein Glossar in Frage kommen. Die Beschreibung eines Begriffs muss unabhängig von ihr erarbeitet werden.

Da die Metriken welche die Beschreibung eines Glossareintrags bewerten ebenso zur Bildung der Qualitätskennzahl des Eintrags und somit zu der des gesamten Glossars beitragen, kann es passieren dass das Glossar eine schlechte Qualitätskennzahl bekommt obwohl die Vorschlagsliste aus der es entstanden ist eine gute Qualitätskennzahl hat. Ebenso kann ein schlechte Vorschlagsliste zu einem guten Glossar führen.

Wie gezeigt sind die Qualitäten von Vorschlagsliste und Glossar unabhängig von einander.

# 7. Fazit und Ausblick

## 7.1. Bewertung des Ergebnisses

Wie die vorhergehenden Kapitel zeigen ist es durchaus möglich mit einfachen Mitteln ein Glossar zu erstellen und zu verwalten und ebenso für das Glossar nützliche Informationen aus einem natürlich-sprachlichen Text zu extrahieren.

Einfache Methoden beschreiben in diesem Kontext vor allem solche, die größtenteils mit Standard-Bibliotheken auskommen. Außerdem kann eine Methode dann als einfach gelten, wenn für ihre Anwendung keine spezialisierten Daten zur Verfügung stehen müssen, wie dies z.B. bei der in [LZ06] beschriebenen Korpuslinguistik der Fall ist.

Die Extraktion der Informationen aus den natürlich-sprachlichen Texten erfolgt mit Hilfe der Java-Klasse `BreakIterator`, welche Bestandteil einer Standardumgebung ist. Außerdem werden reguläre Ausdrücke verwendet. Diese sind ebenfalls ein übliches Werkzeug in der Informatik.

Ebenso beruht die Filterung auf regulären Ausdrücken und dem simplen Zählen der Vorkommen der untersuchten Begriffe.

Diese Methoden funktionieren auch für sich allein genommen. Allerdings ermöglicht es erst das Zusammenspiel der einzelnen Methoden Informationen wie z.B. Vorschläge für die Aufnahme in ein Glossar von hoher Qualität zu erzeugen.

Bindet man diese verknüpften Methoden zusätzlich, wie in Abschnitt 4.5 vorgeschlagen, in einen Erfahrungskreislauf ein, so erhöht sich die die Qualität dieser Methoden noch einmal. In diesem Punkt eröffnet sich für den Terminus *halb-automatisch* ein neuer Blickpunkt. Hier unterstützt nicht mehr der Algorithmus den menschlichen Benutzer sondern wird selbst vom Menschen durch die Bereitstellung von dessen Erfahrungen unterstützt.

Im Gegensatz zu komplizierteren Methoden wird bei den hier vorgestellten Mechanismen auf eine Auswertung des Kontexts verzichtet. Dies bedeutet dass es z.B. nicht möglich ist Begriffe zu erkennen, die möglicherweise falsch verwendet werden. Außerdem ist eine Unterscheidung der Begriffe aufgrund ihrer Wortart nicht möglich. So werden Bindewörter genauso wie Fachbegriffe behandelt.

Diese Nachteile lassen sich allerdings zum Teil vermindern. So ist es möglich nur bestimmte Attribute eines Anforderungsdokuments auszuwerten, indem nur diese dem Glossarkern zugeführt werden. Eine echte Beachtung des Kontexts kann mit diesen Methoden allerdings nicht erreicht werden.

Ebenso ist zwar keine tatsächliche Unterscheidung von Wortklassen möglich, jedoch

lässt diese sich zum Teil durch die Verwendung spezialisierte Filter emulieren. Ein Beispiel hierfür ist der Filter zur Entfernung von Bindewörtern.

Zusammenfassend lässt sich feststellen, dass eine halb-automatische Generierung von Glossaren mit einfachen Mitteln ausreichend gut für den praktischen Einsatz funktioniert. Die Einbindung in den Erfahrungskreislauf erlaubt den Methoden darüber hinaus ihre Nützlichkeit durch Einbeziehung menschlicher Erfahrungen zu verbessern.

## 7.2. Abgrenzung zu anderen Arbeiten

### 7.2.1. Semantische Informationsextraktion in medizinischen Informationssystemen

In [HGE07] wird eine Methode vorgestellt, mit deren Hilfe Informationen aus medizinischen Informationssystemen extrahiert werden können. Diese sollen danach sowohl nutzbar als auch brauchbar gemacht werden. Das Hauptinteresse dieser Arbeit liegt jedoch auf der semantisch korrekten Erfassung der Informationen. Weiterhin ist aufgrund der Festlegung auf die Medizin zu beachten, dass ein Grundkorpus für diese Fachsprache verwendet wird, der nicht auf die formelle deutsche Sprache anzuwenden ist. Schließlich werden Verfahren benutzt welche auf eine Wissenbasis zurückgreifen. Diese ist das Ergebnis jahrelanger Erfahrung in der medizinischen Dokumentation.

Durch die Verwendung einer umfassenden Wissenbasis und der Festlegung auf den Bereich der Medizin unterscheidet sich die in [HGE07] vorgestellten Methoden deutlich von den in dieser Arbeit aufgezeigten Möglichkeiten.

### 7.2.2. Part-of-Speech Tagging

In [Zie07] werden Methoden zur maschinellen Extraktion von Informationen aus Texten dargestellt. Diese verwenden hauptsächlich sog. Part-of-Speech (POS) Tagger. Die Hauptaufgabe von POS Taggern ist es für jedes Wort in einem natürlich-sprachlichen Text die entsprechenden Wortart herauszufinden. Hierzu werden für jede Sprache Kopora benötigt, aus denen die benötigten Informationen über Auftreten und Kontext eines Worts extrahiert werden können. Außerdem werden linguistische Ontologien für diese Methoden benutzt.

Ein Problem bei, POS Tagging ist, dass viele vorhandene Algorithmen nur für die englische Sprache zur Verfügung stehen. Für diese gibt es den British National Corpus[1] (BNC) als Referenz. Als linguitische Ontologie kann WordNet[2] verwendet werden. Für die deutsche Sprache steht zum Beispiel das in [Fel97] beschriebene GermaNET als im Aufbau befindliche Alternative zur Verfügung.

Im Gegensatz zu denen in dieser Arbeit verwendeten Methoden bieten die in [Zie07]

[1] `http://www.natcorp.ox.ac.uk/`
[2] `http://wordnet.princeton.edu/`

vorgestellten Techniken die Möglichkeit die extrahierten Wörter in Wortklassen einzuteilen. Allerdings wird für die Verwendung dieser Methode ein Korpus der entsprechenden Sprache benutzt. Für viele Sprachen, wie auch das Deutsche, sind diese Korpora entweder nicht verfügbar oder noch im Aufbau.

### 7.2.3. Ontologie-Extraktion

In [Kof05] wird eine Methode zur Extraktion domänenspezifischer Ontologien unter besonderer Berücksichtigung des Requirement Engineerings vorgestellt. Diese wird in [Wen04] durch den Einbezug von Listen und Tabellen erweitert.

Diese Methode unterscheidet sich von denen in dieser Arbeit vorgestellten vor allem durch das Ziel nicht direkt Begriffe zu extrahieren sondern eine komplette Ontologie einer Domäne zu erstellen.

### 7.2.4. Erstellung von Verzeichnissen

Die meisten aktuellen Programme zur Textverarbeitung bieten Methoden an um ein Verzeichnis zu erstellen. Im Unterschied zu den Methoden dieser Arbeit wird der Benutzer jedoch nicht bei der Auswahl der enthaltenen Begriffe unterstützt. Außerdem können die erstellten Verzeichnisse später nicht qualitativ ausgewertet werden.

## 7.3. Ausblick

Ein großes Problem für die Lemmatisierung ist die Qualität der quelloffener Wörterbücher, welche für die Lemmatisierung benutzt werden. Dies kann zum einen durch die Verwendung von hinzugekauften Wörterbüchern hoher Qualität behoben werden.

Eine andere Möglichkeit stellt das in [QW99] vorgestellte Projekt *Deutscher Wortschatz* der Universität Leipzig da. Dieses bietet unter anderem die Möglichkeit über einen Webservice[3] die Grundform eines Wortes zu erfragen. Diese Projekt wird seit 1995 gepflegt, so dass es eine zumeist deutlich höhere Qualität der Lemmatisierung aufweist als die Benutzung von Wörterbüchern.

Der Einsatz dieses Webservices bringt jedoch auch einige Probleme mit sich. Zum einen sind Fachbegriffe nicht unbedingt in der Datenbasis enthalten. Zum anderen lässt sich auf den Webservice nur online zugreifen. Diese bedeutet, dass eine Lemmatisierung durch den Webservice immer eine Netzverbindung haben muss. Außerdem bringt jeder Zugriff zwangsläufig eine Verzögerung mit sich, die in Summe deutlich spürbarer sein wird als das Nachschlagen in einem, im lokalen Speicher vorgehaltenem, Wörterbuch. Schließlich ist auch zu bedenken, dass eben nur der deutsche Wortschatz von diesem Projekt abgedeckt wird und für andere Sprachen Alternativen gesucht werden müssten.

[3] `http://wortschatz.uni-leipzig.de/Webservices/`

Eine weitere Verbesserung wird wie schon beschrieben durch die Überprüfung von mehreren Projekten der gleichen Domäne erreicht. Dies ist aufgrund der nicht vorhandenen Projekte eine Aufgabe für zukünftige Untersuchungen.

Für das entwickelte Plugin ist eine mögliche Erweiterung die Berücksichtigung von Attributen der Anforderungsdokumente bei der Extraktion und Filterung der Vorschläge.

Außerdem ist zu überlegen ob für die Vorschlagsliste ein optimiertes View benutzt werden kann, welches bei großen Textmengen eine kleinere Verzögerung als das benutzte Html-View aufweist.

Eine mögliche Erweiterung des Glossarkerns wäre die Berücksichtigung von Kontexten. Dies kann beispielsweise mit Hilfe von Ontologien oder Korpuslinguistik geschehen. Die Verwendung solcher Methoden liegt jedoch ausserhalb des Ziel die Extraktion mit einfachen Mitteln durchzuführen.

# A. Erfahrungsscripts

```
for (var i = 0; i < project.getAllElements().length ; i++){
  var element = project.getAllElements()[i];
  if(element.getClass().getName().equals("de.unihannover.se.uce.
     glossary.model.GlossaryEntry"))
  {

    var metricFactory = element.getMetricFactory();
    var countMetric = metricFactory.getMetricByName("UsesMetric");

    if (element.getEntryDescription() == "")
    {
      contextList.addContext(project, element);
    }
  }
}
```

Listing A.1: Erkennen von Einträgen ohne Beschreibung

```
var n = project.getAllElements().length;
for (var i = 0; i < n ; i++){
  var element = project.getAllElements()[i];
  if(element.getClass().getName().equals("de.unihannover.se.uce.
     glossary.model.GlossaryEntry"))
  {
    var metricFactory = element.getMetricFactory();
    var metric = metricFactory.getMetricByName("InCountMetric");

    if (metric.getMetricForEntry(element) > (n-1)/2)
    {
      contextList.addContext("<html><font color=red>"+metric.
         getMetricStringForEntry(element)+"</font>", project,
         element);
    }
  }
}
```

Listing A.2: Erkennen von Einträgen mit einer hohen Eingangsverweiszahl

```
var n = project.getAllElements().length;
for (var i = 0; i < n ; i++){
  var element = project.getAllElements()[i];
  if(element.getClass().getName().equals("de.unihannover.se.uce.
     glossary.model.GlossaryEntry"))
  {
    var metricFactory = element.getMetricFactory();
    var metric = metricFactory.getMetricByName("OutCountMetric");

    if (metric.getMetricForEntry(element) > (n-1)/2)
    {
      contextList.addContext("<html><font color=red>"+metric.
         getMetricStringForEntry(element)+"</font>", project,
         element);
    }
  }
}
```

Listing A.3: Erkennen von Einträgen mit einer hohen Ausgangsverweiszahl

```
var n = project.getAllElements().length;
for (var i = 0; i < n ; i++){
  var element = project.getAllElements()[i];
  if(element.getClass().getName().equals("de.unihannover.se.uce.
     glossary.model.GlossaryEntry"))
  {
    var metricFactory = element.getMetricFactory();
    var metric = metricFactory.getMetricByName("recursiveMetric");

    if (metric.getMetricForEntry(element) > 1)
    {
      contextList.addContext("<html><font color=red>"+metric.
         getMetricStringForEntry(element)+"</font>", project,
         element);
    }
  }
}
```

Listing A.4: Erkennen von rekursiven Verweisen

```
for (var i = 0; i < project.getAllElements().length ; i++){
  var element = project.getAllElements()[i];
  if(element.getClass().getName().equals("de.unihannover.se.uce.
     glossary.model.GlossaryEntry"))
  {

    var metricFactory = element.getMetricFactory();
    var metric = metricFactory.getMetricByName("
       LevenshteinDistanceMetric");

    if (metric.getMetricForEntry(element) <= 2)
    {
      contextList.addContext(metric.getMetricStringForEntry(element)
         , project, element);
    }
  }
}
```

Listing A.5: Makierung von Einträge mit einer Levenshtein-Distanz kleiner 3

# Literaturverzeichnis


[Bus02] BUSSMANN, Hadumod: *Lexikon der Sprachwissenschaft*. Kröner Verlag, Stuttgart, 2002

[Cri06] CRISP, Christian: *Konzept und Werkzeug zur erfahrungsbasierten Erstellung von Use Cases*, Leibniz Universität Hannover, Masterarbeit, 2006

[Den96] DENNETT, Daniel C.: *Darwin's Dangerous Idea (Penguin Science)*. Penguin Books Ltd, 1996. – ISBN 014016734X

[Dud06] DUDENREDAKTION: *Duden. Deutsches Universalwörterbuch*. Bibliographisches Institut, 2006

[Fel97] FELDWEG, H.: GermaNet - ein lexikalisch-semantisches Netz für das Deutsche. In: *Lexikalische Semantik aus kognitiver Sicht - Perspektiven im Spannungsfeld linguistischer und psychologischer Modellierungen* (1997)

[GHJV95] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman, Amsterdam, 1995. – ISBN 0201633612

[HGE07] HOLZINGER, Andreas ; GEIERHOFER, Regina ; ERRATH, Maximilian: Semantische Informationsextraktion in medizinischen Informationssystemen. In: *Informatik-Spektrum* 30 (2007), S. 69–78

[HP02] HOUDEK, Frank ; PAECH, Barbara: Das Türsteuergerät - eine Beispielspezifikation / Fraunhofer Institut für Experimentelles Software Engineering. 2002. – Forschungsbericht

[KF91] KAROL FRÜHAUF, Helmut S.: *Software-Prüfung. Eine Fibel*. vdf Hochschulverlag AG an der ETH Zürich, 1991. – ISBN 3519021544

[Kit07] KITZMANN, Ingo: *Kritiksystem für ein erfahrungsbasiertes Anforderungswerkzeug*, Leibniz Universität Hannover, Bachelorarbeit, 2007

[Kof05] KOF, Leonid: *Text Analysis for Requirements Engineering*, Technische Universität München, Dissertation, 2005

[Lei07] LEISS, Hans: *Computerlinguistik*, Universität München, Vorlesungsfolien, 2007

[Lev66] LEVENSHTEIN, Vladimir I.: Binary codes capable of correcting deletions, insertions, and reversals. 1966 ( 8). – Forschungsbericht. – 707–710 S

[LZ06] LEMNITZER, Lothar ; ZINSMEISTER, Heike: *Korpuslinguistik. Eine Einführung (Narr Studienbücher)*. Narr, 2006. – ISBN 3823362100

[QW99] QUASTHOFF, Uwe ; WOLFF, Christian: Korpuslinguistik und große einsprachige Wörterbücher. In: *Linguistik online* 2 (1999)

[RRP80] Kap. 6 In: VAN RIJSBERGEN, C.J. ; ROBERTSON, S.E. ; PORTER, M.F.: *New models in probabilistic information retrieval*. London: British Library, 1980

[Sch02] SCHNEIDER, Kurt: LIDs: A Light-Weight Approach to Experience Elicitation and Reuse. In: *Lecture Notes in Computer Science* 1840/2000 (2002), S. 407–424

[Sch07] SCHNEIDER, Kurt: *Anforderung und Entwurf*, Leibniz Universität Hannover, Vorlesungsfolien, 2007

[Wen04] WENDT, Armand: *Textanalyse für Requirements Engineering - Einbeziehung von Tabellen und Listen in die Analyse*, Technische Universität München, Diplomarbeit, 2004

[Wik07] WIKIPEDIA: *MySpell — Wikipedia, The Free Encyclopedia*. 2007. – [Online; accessed 15-July-2007]

[Zie07] ZIEGLER, Cai: Wissen aus Wörtern. In: *iX, Magazin für professionelle Informationstechnik* 9 (2007)

# Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den *7. November 2007*

_____

Sebastian Meyer