# Kriterienkatalog für IR- und DB-Werkzeuge: OpenText

Inhalt:



---

## 1. Kommentare zum Kriterienkatalog

**Fett** gedruckte Kriterien sind vorläufige KO-Kriterien.

- Benutzungsfreundlichkeit
  - Update der DB mit Dokumenten
    *HF: Noch keine Erfahrungen.*

  - Retrieval
    *HF: Mehre Möglichkeiten*
    - *Grafisches Frontent, für MS Windows & Motif verfügbar*
    - *'einfaches' Abfragetool (pat) mit eigener Abfragesprache*

- Suchfunktionalität
  - Suchlogik
    - **Boolesch**
      *HF: Boolsches Suchen mit Differenz, Vereinigung- und Schnittmenge.*

    - **Ranking**
      - **Methode:** probabilistisch, Vektorraum, Fuzzy, ...
      - Rankingalgorithmus offengelegt

*HF: Nicht vorhanden.*

- Relevance Feedback
  *HF: ?*

- **Unterstützung verschiedener Datentypen**
  - **Volltext**

    *HF: Dokumentstruktur kann - sofern vorhanden - berücksichtigt werden, z. B. um Überschrift(en), Titel, etc. zu durchsuchen. Auch Indexierung von SGML-Dokumentne möglich.*

  - **Textfelder**
    *stemming, wildcards, phrase search, proximity, Ähnlichkeit*
    *HF: Suchbefehle fby (followed by), near existieren, phrase search (suche nach mehreren aufeinanderfolgenden Wörtern) ist möglich. Proximity kann gesetzt werden, bei 'fby' und 'near' per default aktiv.*

  - **numerische Felder, Datumsfelder**
    *(<, >, =, vage Vergleiche)*
    *HF: Nicht möglich.*

- **Suchfelder**
  - **Dokumente können in frei festlegbare Suchfelder unterteilt werden**
    *HF: Ja, Struktur (auch hierarchisch) frei definierbar, z. B. durch Textverarbeitung (verschiedenste Formate) festgelegt oder via SGML.*

  - Suchfelder können mit unterschiedlicher Methodik invertiert werden
    *HF: ja, vgl. 'Combining Sets of Results' in "Pat Tutorial", S. 91ff. An Methoden existieren Schnittmenge, Vereinigungsmenge & Differenz.*

- Multifile Searching
  *HF: Multifile per default, auch Multi-DB-Search via PEM (leider nicht sehr gut dokumentiert).*

- Thesaurus- / Wörterbuchunterstützung
  *HF: Ja, aber noch keine Erfahrungen.*

- Suchergebnisse
  - Qualität der Ergebnisse
    *(angelehnt an Precision, Recall etc.; genaues Vorgehen muß noch festgelegt werden)*
    *HF: Bei Suche mit 'pat50' werden alle Hits mit etwas (einigen 10 Zeichen) Kontext aufgelistet. Vorangestellt wird der Byte-Offset im Gesamttext (bei mehreren Dokumenten nicht sehr Sinnvoll), der aber auch - bei strukturiertem Text - durch ein beliebiges Feld ersetzt werden kann und so z. B. den Dateinamen angehen kann.*

  - Transparenz des Ranking für den Benutzer
    *HF: what ranking.*

- Ausgabeformat und Darstellung der Ergebnisse
  - statistische Angaben über Ergebnismenge
    *HF: Es wird lediglich die Gesamtzahl der Hits angegeben.*
  - Highlighting bzw. Tagging für Highlighting
    *HF: Tagging möglich, auch speichern und weiterarbeiten mit getaggten Ergebnissen.*
  - **Auswahl von Displayfeldern**
    *HF: ja, je nach Dokumentstruktur. Im interaktiven Betrieb mit 'pat50' kann neben dem Feld, in dem der gesuchte Text gefunden wurde noch ein Feld angegeben werden, das zusätzlich angezeigt werden soll.*
  - Sortiermöglichkeiten
    *HF: Entweder der Indexierunge gemäß (d.h. eher zufällig), alphabetisch, oder gemäß dem Vorkommen im Text. Details siehe "Pat Reference Manual", Seite 67.*
  - Clustering
    *HF: ?*
- Dublettenbehandlung
  *HF: Keine, doppelte Suchergebnisse werden doppelt aufgelistet.*

- Multimediale und Hypertext-Dokumente
  - **strukturierte Dokumente (einfache Hypertexte)**
    *HF: ja*
  - Links in / zwischen Dokumenten
    *HF: Nur existierende Hypertext-links; muß vom Frontend ausgewertet und als Link interpretiert werden.*
  - Maps für Hyperlinks
    *HF: ?*
  - Annotationen
    *HF: Pro zu indexierende Datei kann in einer eigenen, außerhalb der Datei gelagerten zweiten Datei ein Kommentarfeld (OTTEXT?) eingebracht werden, das dann beim indexieren mit in den Index aufgenommen wird.*
  - **Multimediadokumente**
    *HF: Nope, text only.*

- Performance / Technische Eigenschaften
  - Aufwand
    - Invertierung
      *HF: Was invertieren? Anfrageergebnisse über boolesche Operationen (Differenz) invertierbar.*
    - Update
      *HF: Inkrementelle Updated möglich, ohne die gesamte Datenbank neu zu indexieren.*

- Update im laufenden Betrieb oder nur offline
  *HF: Nur offline, siehe "Database Administration Guide", S. 220.*

- Speicherbedarf pro Dokument
  *HF: Noch keine Erfahrungen.*

- Systemgrenzen
  - maximale Anzahl der Dokumente
    *HF: Viele.*

  - **maximale Dokumentgröße (min. 2MB)**
    *HF: Noch keine Erfahrungen. Was ist ein Dokument - eine Datei, oder darf ein Dokument aus mehreren Dateien bestehen? In letzterem Fall sind 2MB kein Problem.*

  - maximale Anzahl der möglichen gleichzeitigen Nutzer
    *HF: Noch keine Erfahrungen, wohl aber nur durch Systemgrenzen eingeschränkt.*

- Suchzeit
  - in Abhängigkeit von Anzahl der Nutzer
    *HF: Noch keine Erfahrungen.*

  - in Abhängigkeit von Anzahl der Dokumente
    *HF: Nein. Antwortszeit ist bei OpenText nicht vom zu durchsuchenden Dokumkentenvolumen abhängig.*

- Formate
  - **Importformate (welche?) / Importfilter bzw.:**
    *HF: Siehe Handbuch "Database Administraion Reference Guide", Seite 13ff. Mitgeliefert Werden Filter für WordPerfect und Microsoft Word (vgl. DB Admin Guide, S. 8). Allerdings ist die Einbindung/Verwendung der Filter an keiner Stelle dokumentiert, was das Verständnis des Filter-Schemas sehr einschränkt.*

  - **Schnittstelle zur Integration eigener Importfilter**
    *HF: Wahrscheinlich ohne Ende, mehrphasiger Import-Ablauf ist beschrieben. Außerdem eigenes API.*

  - Speicherformate
    *HF: Proprietär, nicht offengelegte Binärdateien.*

  - **Display im Originalformat**
    *HF: Ja, falls (OpenText eigener) Viewer Format kennt.*

- Accounting, Access Control
  - Verschiedene Rechte für verschiedene Nutzer
    *HF: Accounting EXTREM schlecht beschrieben (???WO???), anscheinend nur "wer hat was gefragt" bzw. "Wer hat welches Ergebnis bekommen". Keine Vergabe von Benutzerrechten auf DB-Ebene (bzw. nur auf Unix-Ebene, für Dateien).*

- Implementierung
  - Aufwand für Implementierung einer lauffähigen Anwendung
    *HF: nicht allzu hoch, sofern die Anforderungen der Anwendung im Rahmen der Möglichkeiten von OpenText bleiben.*

  - Dokumentation
    *HF: Abfragesprache ganz gut; Zugriff von mehreren Hosts schlecht; Accounting noch schlechter; Aufbauen eigener Datenbanken im Detail sehr genau, nur der Gesamtzusammenhang kommt zu kurz. Eine "Jumpstart"-Section wäre nicht schlecht.*

- Schnittstellen
  - **Z39.50**
    *HF: Was ist das?*

  - (erweitertes) SQL
    *HF: nö*

  - WWW
  - **komfortable Programmierschnittstellen (C, C++, Perl, ...)**
    *HF: API mit C-Schnittstelle vorhanden. ???Details???*

  - andere DBMS (Oracle, O2,...)
    *HF: Noch keine Erfahrungen. ???*

- Plattformen
  - **UNIX**
    *HF: Client und Server*

  - Windows NT
    *HF: Client und Server*

  - Ms Windows 3.1
    *HF: Client und Server(?)*

- Kosten
  - **Version zur Evaluierung im Projektrahmen bezahlbar**
  - Demoversion
  - Hochschulrabatt
  - Lizenzen
  - Wartungsgebühren
  - Vertragsbedingungen für (kommerzielles) Online-Angebot der DB

  *HF: Keine Ahnung.*

- Firmenprofil
  - Marktpräsenz der Firma
  - **Marktpräsenz des Produkts**
  - Installationen bei Referenzkunden

- Qualität der Wartung
- **Support**

*HF: Noch keine Erfahrungen.*

---

# Comments

## 2. Open Text Datenbank Struktur

(vgl. DB Admin Guide, S. 18f)

- Data Dictionary file (`.dd`)
  - Text Group: `<Text>`, `</Text>`,
    - Text files
      - source files of various formats
      - ...
    - FileMap
      - FileMap files (`.fmp`, `.xmp`, `.lmp`, built by `mfsbld50`)
      - ...
  - Main Index Group: `<Indices>`, `</Indices>`,
    - Main index (`.idx`, built by `patbld50`)
    - FastFind files
      - FastFind indices (`.ffc`, `.ffi`, `.ffw`, built by `patffi50` and `patffw50`)
      - ...
  - Region Indices Group
    - Region indices (`.rng`), built by:
      - `multirng50` (mainregion sub-index)
      - `patrng50` or custon builder (supplementary region indices)
      - `sgmlrng50` (regions based on the inherent structure of the SGML data)
    - FastRegion indices (`.fri`, built by `patfr50`)

## 3. Zusammenhang zw. Programm und Suffix

Die Spalte "Described in" bezieht sich auf das "Database Administration Guide".

| Suffix | Program | Described in |
|---|---|---|
| `.idx` | `patbld50` | Chapter 12 |
| `.ffc` | `?` | Chapter 12 |
| `.ffi` | `patffi50` | Chapter 12 |
| `.ffw` | `patffw50` | Chapter 12 |
| `.rng` | `multirng50` | Section 12.3.3 |
| `.rng` | `patrng50` | Section 12.3.4 |
| `.rng` | `sgmlrng50` | Section 13.3 |
| `.fri` | `patfr50` | Section 12.3.5 |
| `.fmp` | `mfsbld50` | Section 12.3.1 |
| `.xmp` | `mfsbld50` | Section 12.3.1 |
| `.lmp` | `mfsbld50` | Section 12.3.1 |
| `.tag` | Editor | Section 4.1.1 |
| `.pem` | Editor | Section 15.1 |
| `.cfg` | ??? | ??? |

# 4. Indexieren eines einfachen Textes

Es soll eine einfache Textdatei `test.txt` indexiert werden.
Vorgehen:

1. Data-Dictionary mit `dbbuild50` erzeugen:

```
sunendres6% ddgen50 test
sunendres6% dir
total 342
-rw-r--r--   1 feyrer   users        147 May  8 00:03 test.cfg
-rw-r--r--   1 feyrer   users       3801 May  8 00:03 test.dd
-rw-r--r--   1 feyrer   users         65 May  8 00:03 test.tag
```

2. Im Data-Dictionary `test.dd` die zu indexierende Datei in <MfsDir>**,** </MfsDir> eintragen und das VERzeichnis, in dem sich die Datei befindet, in <MfsFile>**,** </MfsFile>:

```
<DB>
  <Text>
    <MfsFiles>
      <FileMap>test</FileMap>
      <FilterChain>
        <SearchView>wfw|meta</SearchView>
        <DisplayView>iflat|meta</DisplayView>
        <RawView>iflat|meta</RawView>
        <DisplayFmt>ASCII</DisplayFmt>
        <DefaultDataTag></DefaultDataTag>
        <FileGroup>
          <MfsDir>.</MfsDir>
          <MfsFile>*.txt</MfsFile>
```

```
          <MfsExpand>file</MfsExpand>
        </FileGroup>
      </FilterChain>
    </MfsFiles>
  </Text>
  ...
```

3. Indexlauf mit "`dbbuild50 -v -D test.dd`" starten.

4. Test-Abfrage mit "`pat50 test.dd`" starten:

```
sunendres6% pat50 test.dd

       Open Text Database System, Release 5.0
    Copyright 1987-1995 by Open Text Corporation

>> kernel
  1: 89 matches

>> pr sample
    33286, ..say) possible kernel    bugs or installation problems, but admin..
    51676, .. a BSD        kernel, configuring a System V kernel.            ..
    87525, .. copy of your kernel    image named '/netbsd'. You may also want..
    15846, ..n NetBSD. The kernel itself, that is the core part of    NetBSD,..
    57015, ..tituting your kernel name for BABYLON, obviously ;-). The progra..
    57753, ..r the    full kernel recompile. When the make finishes, you can ..
    53727, ..11:           kernel source for all architectures: alpha, amiga,..
    75386, ..uld patch the kernel to boot into a Retina screen of resolution ..
    19980, ..t part of the kernel, which    knows how to access the particula..
    57348, ..ompiling your kernel             You are now about ready to comp..

>> quit
```

5. Da das Auflisten der einzelnen Matches ziemlich zäh geht kann man dies durch ein optimieren der Filter-Chain optimieren. Dazu sind im Data Dictionary die folgenden Werte zu ändern (vorausgesetzt, die Eingabedatei ist eine reine ASCII-Datei!):

```
...
<SearchView>meta</SearchView>
<DisplayView>meta</DisplayView>
<RawView>meta</RawView>
...
```

   Die Filter sind also jeweils auf den `meta` Filter zu reduzieren. Anschließend ist der Index neu zu generieren.

# 5. Indexieren von HTML-Texten

Um einen Satz von HTML-Seiten zu indexieren ist wie folgt vorzugehen:

1. Data-Dictionary erzeugen:

```
sunendres6% ddgen50 bla
sunendres6% ls -l
-rw-r--r--   1 feyrer   users        147 May 14 20:59 bla.cfg
-rw-r--r--   1 feyrer   users       3801 May 14 20:59 bla.dd
```

```
-rw-r--r--   1 feyrer   users         65 May 14 20:59 bla.tag
```

2. Im Data-Dictionary die `MfsDir`- und `MfsFile`-Sections wie oben beschrieben anpassen.

3. Tags in `bla.tag` festlegen:

```
sunendres6% cat bla.tag
<region>
  <name>TITLE</name>
  <tag>TITLE</tag>
  <desc></desc>
</region>
<region>
  <name>H1</name>
  <tag>H1</tag>
  <desc></desc>
</region>
<region>
  <name>H2</name>
  <tag>H2</tag>
  <desc></desc>
</region>
<region>
  <name>H3</name>
  <tag>H3</tag>
  <desc></desc>
</region>
<region>
  <name>H4</name>
  <tag>H4</tag>
  <desc></desc>
</region>
<region>
  <name>H5</name>
  <tag>H5</tag>
  <desc></desc>
</region>
<region>
  <name>H6</name>
  <tag>H6</tag>
  <desc></desc>
</region>
```

4. Region-config-File `bla.cfg` anpassen (`DisplayFmt` -> `ASCII`, 2. `Regions`-Section entfernen):

```
sunendres6% cat bla.cfg
<Regions>
  <DisplayFmt>ASCII</DisplayFmt>
  <TagFile>bla.tag</TagFile>
</Regions>
```

5. Indexieren:

```
sunendres6% dbbuild50 -c bla.cfg -u bla.tag -D bla.dd
```

6. Probe-Abfrage:

```
sunendres6% pat50 bla.dd
```

```
        Open Text Database System, Release 5.0
     Copyright 1987-1995 by Open Text Corporation

>> region TITLE including "FAQ"
  1: 2 matches

>> pr
      291, ..ril 1996 -->  <TITLE>Amiga-NetBSD-FAQ</TITLE> <H1>NetBSD/amiga F..
   105297, ..May 1994 -->  <TITLE>Networking-FAQ</TITLE> <H1>Networking-FAQ</..
```

Sind in den zu indexierenden HTML-Texten die Tags `HTML`, `HEAD` und `BODY` enthalten, so kann dies zu Schwierigkeiten beim indexieren führen. Diese Tags sind vorher zu entfernen (oder ins tag-File aufzunehmen?).

# 6. Kommentare zur TIME-Suite

- Die Stopwort-Liste in TIME.STP umfaßt ca. 340 Wörter, OpenText kann jedoch maximal 31 Stopwörter verwalten (siehe Database Admin Guide, Seite 64 Mitte).
- Auch mit Hilfe des Hinweises, daß die Textnummern in TIME.REL und TIME.QUE die *laufenden* Nummern innerhalb TIME.ALL sind, sind keine auch nur annähernd brauchbaren Suchergebnisse zu erzielen. Auch Suche mittels ’grep’ brachte keine brauchbaren Ergebnisse, was auf die Qualität der TIME-Suite schließen läßt.
- Um die TIME-Suite mit allen interessanten Feldern zu indexieren wurde folgende Tag-Hierarchie implementiert:

  ```
  <TIME>
          <TNUM>...</TNUM>
          <LNUM>...</LNUM>
          <TEXT>...</TEXT>
  </TIME>
  ```

  Die einzelnen Tags haben dabei die folgenden Bedeutungen:

  `TIME`:
  : Tag das alles einschließt. Ist nötig, um sich die Textnummer ausgeben lassen zu können, in dem ein Begriff gefunden wurde (`{SortOrder OccurHead LNUM TIME}` in pat50);

  `TNUM`:
  : Enthält die Textnummer, wie sie im TIME.ALL angegeben ist.

  `LNUM`:
  : Enthält die laufende Textnummer, von 1 ab aufsteigend numeriert.

  `TEXT`:
  : Enthält den eigentlichen Text aus TIME.ALL.
- Mit dem genannten Tag-Schema ergeben sich folgende Dateigrößen:
  - TIME.ALL: 1.549.700 Bytes
  - *.txt (mit Tag-Struktur, aber ohne ’*TEXT’-Header aus TIME.ALL: 1.539.179 Bytes
  - time2.* (Indexdateien, Data Dictionary, ...): 2.021.185 Bytes (131%)
- Das indexieren der in 423 Dokumenten enthaltenen 1,5MB Daten (*.txt) benötigte auf einer SPARCstation 5 (32MB RAM) 2 Minuten 20 Sekunden (bei Verwendung von ’-m 16m’ nur 1 Minute 20 Sekunden).

# 7. Kommentare zur MeDoc-Suite

- Auch hier eigenes Tag-Format, um CR-Klassifikation und Text in einer Datein unterbringen zu können:

```
<MEDOC>
        <ID>...</ID>
        <CR>...</CR>
        <TEXT>...</TEXT>
</MEDOC>
```

  Die einzelnen Tags haben dabei die folgenden Bedeutungen:

  `MEDOC`:
  : Allumfassendes Tag

  `ID`:
  : Identifikation, mit deren Hilfe der Name der CR-Klassifikations-Datei und der Name der eigentlichen Text-Datei bestimmt werden kann.

  `CR`:
  : CR-Klassifikation wie in der CR-Datei angegeben, mehrere Klassifikationswerte sind durch ", " getrennt.

  `TEXT`:
  : Der eigentliche Text aus der Text-Datei. Sämtliche "<" Sind dabei durch "<" ersetzt, sämtliche ">" durch ">" und sämtliche "&" durch "&". Wird dies nicht gemacht, so werden in den Texten auftauchende "<CR>"-Zeichenfolgen falsch zugeordnet.
- Indexieren der 8,5MB mit '-m 16m' 5 Minuten, 30 Sekunden; mit '-m 1m' 15:38 Minuten.
- Unerklärliche Probleme ohne '-m 16m'-Switch
- Dateigrößen:
  - Urspruengliche Textsammlung: 1.556 Bytes CR-Klassifikation und 8.768.664 Bytes Texte.
  - Texte inkl. Tags: 8.796.578 Bytes
  - Größe des erzeugten Index': 7.462.540 Bytes (85%)

# 8. Kommentare zum HTML3-Buch (Tolksdorf)

- Dateigrößen:
  - Zu indexierende HTML-Texte: 705.933 Byes
  - Erzeugte Indexdateien: 763.863 Bytes (108%)
- Indexierungszeit: 1:01 Minute.

# 9. Kommentare zum Stoecker-Buch

- Dateigrößen:
  - Zu indexierende HTML-Texte: 8.414.117 Bytes
  - Erzeugte Indexdateien: 8.118.499 Bytes (96%)
- Indexierungszeit: 16 Minuten.

# 10. Antwortsverhalten

## 10.1 Lokal

Ist die OpenText-Datenbank auf dem selben Rechner installiert, auf dem die Abfragen gemacht werden, so ist keine nennenswerte Verzögerung bei der Beantwortung der gestellten Anfragen zu sehen, alle Anfragen wurden in 1-2 Sekunden beantwortet. Die Antwortszeit war dabei unabhängig von der Komplexität der gestellten Anfrage und dem zu durchsuchenden Datenvolumen.

## 10.2 Über's Internet

An dieser Stelle soll nur auf die OpenText-eigenen Abfrage-Werkzeuge mittels pat50 (bzw. der darauf basierenden Oberfläche) eingegangen werden. Der Zugriff via WWW soll an anderer Stelle evaluiert werden.

Die OpenText-eigenen Abfrage-Werkzeuge benutzen zum Zugriff auf entfernte Daten das rlogin-Protokoll, und arbeiten dann mit den selben Werkzeugen und Methoden wie beim lokalen Zugriff. Ist also der rlogin-Zugriff "schnell genug", so wird sich der Zugriff über's Netz nicht merkenswert langsamer gestalten aus ein lokaler Zugriff.

*Ein Wort zur Sicherheit:* Ist der rlogin Erlaubt, so kann von der Kennung auf dem Rechner, auf dem die Anfrage an die OT-Datenbank gestellt werden soll, auch der Zugriff auf den Account und damit den gesamten Rechner erfolgen, auf dem der OT-Datenbank-Server läuft. Da dies ein nicht unerhebliches Sicherheitsrisiko darstellen kann ist beim Aufsetzen der abfragenden Dienste darauf zu achten, dass diese nach Möglichkeit nicht an interaktive Kennungen gebunden sind.

# 11. Berechnen der Recall- und Precision-Werte

### 11.1 Mit eigenem Ranking

Die folgenden Schritte werden durchgeführt:

1. Anfrage an OpenText generieren.
   Dies geschieht mit Hilfe des perl-Scripts '*mkquery2*'. Dabei wird aus der Datei *TIME.QUE* die Anfragen ausgelesen, und OpenText-gerecht aufbereitet. Stopwörter, die in *TIME.STP* vorkommen werden vorher ausgesondert. Der Output des Scripts kann als Eingabe für pat dienen, in Kommentar ist außerdem eine Liste der erwarteten Resultate aufgelistet.

   ```
   ./mkquery2 >mkquery2.med
   ```

2. Anfragen an OpenText stellen und Ausgabe zwischenspeichern:
   `pat50 time2.dd mkquery2.out`

   Dieser Punkt kostet in der gesamten Auswertung mit Abstand die meiste Zeit (ca. 15 Minuten)!

3. Da OpenText kein Ranking besitzt (bzw. nicht lt. Handbuch funktionert?) muß ein eigens Ranking eingeführt werden. Dazu wird die Ausgabe der pat-Anfrage (mkquery2.out) eingelesen, und pro Anfrage gezählt, wieviele Hits jeder Text hatte. Anschliessend wird nach der Anzahl der Treffer

aufgelistet, wieviele Treffer welcher Text hatte.

Bewerkstelligt wird dieses Zählen (pro Anfrage ca. 1000 Treffer, ein Auswerten per Hand scheidet wohl aus :-) von dem perl-Script "*rank_results*". Die Ausgabe wird zur Weiterverarbeitung in der Datei *rank_results.out* abgespeichert:
`./rank_results | tee rank_results.out`

4. Nachdem nun eine gewertete Liste von Dokumenten vorliegt für jede Anfrage bleibt nur noch, für verschiedene Werte von Q diese mit den vorgeschriebenen Suchergebnissen aus *TIME.RES* zu vergleichen, entsprechend die Werte für A, B und C zu errechnen, und daraus wiederum Recall und Precision zu bestimmen.

   All dies wird von dem perl-Script "*calcRec+Prec*" gemacht:
   `./calcRec+Prec | tee calcRec+Prec.out`

5. Es ergeben sich folgende Werte:

| Q | Recall | Precision |
|---|---|---|
| 1 | 7.17821% | 28.91566% |
| 2 | 18.70192% | 31.32530% |
| 3 | 27.73289% | 31.32530% |
| 4 | 33.32953% | 28.01205% |
| 5 | 36.57960% | 25.54217% |
| 10 | 50.79829% | 18.79518% |
| 15 | 62.41274% | 15.58233% |
| 20 | 68.96499% | 13.01205% |

Alle hier angesprochenen Programme liegen auf der sunendres6 in
*~feyrer/work/Time-Suite/TIME.ALL.d-tagged2*

## 11.2 OpenText-Ranking

Nachdem man (Ricarda! :-) OpenText doch noch entlocken konnte, wie man Texte mit Ranking versieht hier nun die zweite Version inkl. OpenText-Ranking.

Vorgehen:

1. *mkquery3* erstellt aus den o.g. Dateien eine Liste von Anfragen der folgenden Art:

   ```
   >> {SortOrder OccurHead LNUM TIME}
   >> {RankMode NumberOccurs}
   >> region TIME RankedBy (KENNEDY + ADMINISTRATION + PRESSURE + NGO + DINH + DIEM
   ```

   Ausgaben werden in *mkquery3.queries* abgelegt:

```
./mkquery3 >mkquery3.queries
```

2. Die Anfragen in *mkquery3.queries* werden an OpenText gestellt und die Ergebnisse in *mkquery3.results* abgelegt:

   ```
   pat50 time2.dd < mkquery3.queries | sed 's/^>>.//g' >mkquery3.results
   ```

   Die Abfrage dauerte in diesem Fall ca. 15 Sekunden.

3. Die in *mkquery3.results* enthaltenen Abfrageergebnisse werden mit Hilfe des Scripts *calcRec+Prec3* ausgewertet, und die Recall- und Precision-Werte ausgelesen.

   Für die verschiedenen Werte von Q ergeben sich die folgenden Recall- und Precision-Daten:

| Q | Recall | Precision |
|---|---|---|
| 1 | 3.44996% | 14.45783% |
| 2 | 6.94111% | 13.85542% |
| 3 | 10.12425% | 14.85944% |
| 4 | 14.84931% | 15.96386% |
| 5 | 20.38327% | 17.34940% |
| 10 | 50.19588% | 18.67470% |
| 15 | 50.19588% | 12.44980% |
| 20 | 50.19588% | 9.33735% |
| 25 | 50.19588% | 7.46988% |
| 30 | 50.19588% | 6.22490% |

Alle hier angesprochenen Programme liegen auf der sunendres6 in *~feyrer/work/Time-Suite/TIME.ALL.d-tagged2*

---

*Autoren des Kriterienkatalogs: Dr. Michael Schwantner, Matthias Razum, FIZ Karlsruhe*

---

Hubert Feyrer (hubert.feyrer@rz.uni-regensburg.de)