UPPSALA UNIVERSITET

UPTEC X 20020

Examensarbete 30 hp
Juni 2020

Enterprise Search for Pharmacometric Documents

A Feature and Performance Evaluation

Selma Edenståhl

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Enterprise Search for Pharmacometric Documents: A Feature and Performance Evaluation

Selma Edenståhl

Information retrieval within a company can be referred to as enterprise search. With the use of enterprise search, employees can find the information they need in company internal data. If a business can take advantage of the knowledge within the organization, it can save time and effort, and be a source for innovation and development within the company.

In this project, two open source search engines, Recoll and Apache Solr, are selected, set up, and evaluated based on requirements and needs at the pharmacometric consulting company Pharmetheus AB. A requirement analysis is performed to collect system requirements at the company. Through a literature survey, two candidate search engines are selected. Lastly, a Proof of Concept is performed to demonstrate the feasibility of the search engines at the company. The search tools are evaluated on criteria including indexing performance, search functionality and configurability.

This thesis presents assessment questions to be used when evaluating a search tool. It is shown that the indexing time for both Recoll and Apache Solr appears to scale linearly for less than one hundred thousand pdf documents. The benefit of an index is demonstrated when search times for both search engines greatly outperforms the Linux command-line tools grep and find. It is also explained how the strict folder structure and naming conventions at the company can be used in Recoll to only index specific documents and sub-parts of a file share. Furthermore, I demonstrate that the Recoll web GUI can be modified to include functionality for filtering on document type.

The results show that Recoll meets most of the company's system requirements and for that reason it could serve as an enterprise search engine at the company. However, the search engine lacks support for authentication, something that has to be further investigated and implemented before the system can be put into production.

Handledare: Kajsa Harling
Ämnesgranskare: Ben Blamey
Examinator: Pascal Milesi
ISSN: 1401-2138, UPTEC X 20020

Populärvetenskaplig sammanfattning

Varje dag söker miljontals människor efter information. Ett vanligt sätt att söka efter information på internet är genom att använda ett sökverktyg, som exempelvis Google. Inom några hundradels sekunder kan man få hundratusentals träffar på sina sökord och ofta hittar man också precis det man söker. Men, ofta i arbetslivet, finns informationen man söker inte tillgänglig på internet. Det kan till exempel handla om information som finns internt på ett företag, där man inom företaget vill kunna söka bland verksamhetens interna och sekretessbelagda dokument, produkter eller email. För att kunna söka efter information i dessa måste företaget implementera ett eget sökverktyg, alltså ett eget lokalt ”Google”.

Ett sökverktyg kan liknas vid en bok med ett register. I många böcker, speciellt de som behandlar vetenskap, kan det i slutet finnas ett register där bokens nyckelord finns listade. Genom slå upp ett ord i registret, kan man hitta exakt vilka sidor i boken som behandlar det ordet. Man brukar säga att dessa nyckelord ligger i bokens index. På samma sätt går det med hjälp av sökverktyg att automatiskt extrahera ord från, till exempel, flera dokument och sedan lägga dem i ett index. Det brukar kallas att sökverktyget *indexerar* dokumenten. När man sedan söker efter ett specifikt ord i indexet så vet sökverktyget precis vilka dokument som innehåller det ordet och också vart i dokumenten ordet förekommer.

Sökverktyg som gör detta brukar kallas för sökmotorer. En sökmotor kan alltså indexera datakällor såsom till exempel webbsidor, olika slags dokument eller filer och göra det möjligt att söka i dessa med hjälp av ett eller flera sökord.

Det finns många sökmotorer tillgängliga på internet som är fria för vem som helst att installera och använda för att söka i sin interna data. Men, för att veta vilken sökmotor som passar bäst för en specifik situation eller på ett visst företag, behöver en utvärdering göras utifrån de krav man som användare har på sökverktyget. Om ett företag till exempel vill kunna söka i deras interna PDF-dokument, behöver sökmotorn kunna indexera just filtypen PDF. En sökmotor som endast kan indexera Word-dokument kommer alltså inte att passa på det företaget.

Om ett företag får möjlighet att söka bland sin interna företagsdata, kan de utnyttja den kunskap och den erfarenhet som finns inom organisationen. De anställda får på så sätt möjlighet att hitta den information de behöver för att utföra sitt arbete.

I detta projekt har jag hittat och utvärderat en sökmotor åt företaget Pharmetheus AB. I projektet samlar jag in företagets krav på sökmotorn; bland annat vad de vill kunna söka efter och vilken slags data som företaget använder. Genom en litteraturstudie hittar jag sedan två sökmotorer som svarar mot företagets krav. Jag installerar och testar sedan dessa sökverktyg för att se hur väl sökmotorerna passar in i företagets befintliga system och om den faktiskt uppfyller allt det som företaget önskar.

Resultatet blir en rekommendation till företaget där jag ser den undersökta sökmotorn som en möjlig kandidat eftersom den uppfyller många av företagets krav. Samtidigt tydliggör jag att sökmotorn saknar vissa funktioner, såsom till exempel säkerhet i form av inloggningsmöjligheter.

Table of contents

Abbreviations

API	application programming interface
ES	enterprise search
PD	pharmacodynamics
PK	pharmacokinetics
PKPD	pharmacokinetic-pharmacodynamic
PMX	Pharmetheus
POC	Proof of Concept
GUI	graphical user interface

1 Introduction

In clinical pharmacology and research, it is crucial to understand the relationship between dose, concentration, and response of a drug. Since incorrect dosing of a treatment can have devastating consequences, it is important to conclude how much and how often a treatment should be given to a patient (Standing 2017). The field of pharmacometrics uses quantitative approaches, through mathematical and statistical models, to investigate dose-concentration-response relationships. In pharmacometrics, pharmacokinetic (PK) models are used to explain the relationship between dose and concentration, and pharmacodynamic (PD) models are used to explain the relationship between concentration and response. These can be combined into pharmacokinetic/pharmacodynamic models (PKPD). PK, PD, and PKPD models can be used in all stages of drug development (Standing 2017).

Pharmetheus is a consulting company that offers pharmacometric services to support drug development decisions. As a consulting company Pharmetheus works with multiple clients and projects within all stages of drug development. Information about these different projects is stored in various files and documents on a shared file server.

Today, employees at Pharmetheus have to rely on their memory in order to find previous projects and then manually navigate through the file server to find the information they need in project documents. The knowledge management could be facilitated with a search engine where the employees would be able to search the content of these files in an efficient and easy way. This would help the employees to quickly find the information they need, when they need it. It could help them answer questions like ‘how did we handle this case before?’ or ‘how did we simulate this specific model in a previous project?’.

One way to create a searchable system would be to use a document management system (Rosa *et al.* 2019, Uzialko 2019) and manually tag files with different labels such as keywords, client names, or substance names etc. At Pharmetheus it would require too much manual work to transfer documents to a document management system, since the set of files at the company is already large. Instead a search tool could be implemented in such a way that it can automatically extract file content for identification of a document and index all files in the existing file system.

1.1 Project main parts

This project is a first step towards an implemented enterprise search system at the company. Figure 1 shows the four main parts of the project. First, a requirement analysis was performed to get a better understanding of the needs at the company. Through a literature study, two search engine candidates were chosen based on the identified requirements. These two search tools were then evaluated through a Proof of Concept (POC) (Musienko 2019) where a suggested production set up was implemented and tested.

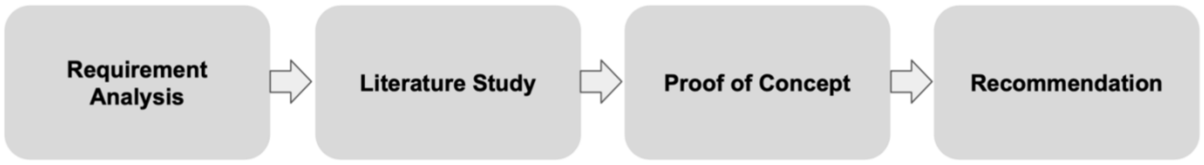


Figure 1. The project main parts.

1.2 Project aim and delimitations

The goal of this project was to:

- Find and evaluate a search tool to see how well it met the requirements at Pharmetheus

This project aimed to evaluate existing search tools on the market, not to build an own search solution from scratch. The goal was neither to investigate/evaluate underlying system architecture or design of search engines. The focus was rather on evaluation based on the functional requirements collected from the company.

1.3 Enterprise search

A term used to describe information retrieval within a company is *enterprise search*. The aim of enterprise search is to, in a business setting, provide an effective and quick way for users to find the information required to carry out their everyday tasks (Zilio *et al.* 2015). The demand for enterprise search has increased as it has been shown that managing the knowledge within an organization can be a source for innovation and organizational success. However, previous reports show that user satisfaction with enterprise search is not very high (Stocker *et al.* 2015), where users expect enterprise search to keep the same high level as a Google search. As businesses are unique and the kind information different companies hold can vary a lot, enterprise search has its unique challenges for handling content, indexing and relevance ranking (Dmitriev *et al.* 2010). An example is when compared to web search where more popular documents tends to be ranked higher than less popular ones (Zilio *et al.* 2015). This is almost never the ranking wanted within a company. In a business environment, the desire may rather be to find the most relevant documents to a given search query.

Even though the difficulties of enterprise search seems to be many and the user satisfaction low, there are relatively few research papers on the topic of implementing and evaluating enterprise search solutions (Dmitriev *et al.* 2010) (Stocker *et al.* 2015). The lack of previous studies could be due to the difficulties in building a general evaluation framework since companies can be dealing with content of very different formats. Assuming a lot of this research is performed within companies, they may not feel the urge of sharing the information with the outside world. (Zilio *et al.* 2015)

There are a lot of enterprise search tools available on the market. But in order to determine which one of these that would be most suitable for a specific company with specific needs, an evaluation has to be done.

1.4 Pharmetheus way of storing information

Pharmetheus (PMX) keeps information about their many client projects on a shared file server that has a strict folder structure and naming conventions. Every client has its own client folder consisting of one or several projects. Each project has a unique project code that is made up of three components:

- Client acronym: *2-4 letter abbreviation*
- Name of product investigated: *DrugName*
- PMX serial number: *PMX-number*

One example of a project code for the first project with the client 'Mostly Old Drugs' investigating the drug 'DrugName' would be: '**MOLD-DrugName-PMX-1**'. If later another project investigating the same drug and including the same company would be performed at the company, the name of that project would be '**MOLD-DrugName-PMX-2**'.

1.4.1 Folder structure

Each project follows the same strict folder structure. The main folders in each project are:

- Analysis
- FromClient
- ProjectManagment
- ToClient

These consist of several sub-folders which can contain the project plan, analysis plan, report etc.

1.4.2 Document naming

At Pharmetheus, abbreviations are used to name specific document types. For example, the acronym ANA is used for analysis plans and REP is used for analysis reports. These abbreviations are used first in the document names.

Documents at Pharmetheus consist of different versions depending on how many times they have been revised. The first final version of a document gets the number 1. If the document is revised after the first final version, the second final document gets number 2 etc. The drafts of these document versions are tracked with a letter a-z. The first version draft will have the letter a, the second draft will have the letter b etc. The version number will consist of the version of the document (1-*) with or without a draft letter (a-z), depending on if the version is finalized or not.

An example of an analysis plan during its first version and second draft is: '**ANA-1b-MOLD-DrugName-PMX-1**'. When the first version is finalized the document name will instead be '**ANA-1-MOLD-DrugName-PMX-1'.** A table of example names of different document types can be seen below.

Table 1. Document naming conventions.

Type of document	**Example file name**
Analysis plan	ANA-No-MOLD-DrugName-PMX-No.pdf
Data management report	DMR-No-MOLD-DrugName-PMX-No.pdf
Project update	PU-No-MOLD-DrugName-PMX-No.pdf
Report	REP-No-MOLD-DrugName-PMX-No.pdf

The company also stores files containing model code or code for visualizing analysis results, these resides most commonly under the folder 'Analysis' and have file endings:

- .mod
- .R
- .Rnw

2 Requirement analysis

In this section, the method, result and discussion of the requirement analysis is stated.

2.1 Method

In order to identify and understand the needs for the search engine, a requirement analysis (Demirel & Das 2018, Gunawardhana 2019) was performed in the beginning of the project. The requirement analysis was divided into two main parts: user requirements, and core technology requirements. Hereafter, the user and core technology requirements will together be referred to as the system requirements.

2.1.1 Collection of user requirements

A survey was conducted to collect example search queries and desired search results from the end users. It was designed together with the supervisor and sent as a Google Form to all employees at the company. The employees were asked to only write one example search query per submission but were able to submit the survey form as many times as they liked. The answer form was designed so that it would not be too time consuming to read or answer. It did only include three questions:

- If you could ‘Google’ the project folders, what would you search for in the search field?
- What would you hope to find when you run the above search?
- Any additional explanations, comments or questions?

The survey was held open for about two weeks in which the employees had the possibility to answer as many times as they wanted. After that, example search queries had to be sent by email and was then added to the result file. The collected example search queries and desired search results were then divided into categories based on the identified search requirements.

2.1.2 Collection of core technology requirements

The core technology requirements were discussed and set up together with the supervisor. They were grouped in priority order. Some features were regarded as required for a functional system. Some functionality was desired but not essential. Lastly some implementations were regarded as nice to have but did not have to be prioritized.

2.2 Result

The requirement analysis result was divided into two main parts: user requirements and core technology requirements.

2.2.1 User requirements

In total, 65 example search queries from 22 unique users were collected. The example search queries were divided into three different categories depending on the type of query identified. The examples given are only fictional.

- **Single word search**
  The example query only contained a single keyword such as a specific variable, method, disease or client. Example: ‘DiseaseA’.

- **Boolean search**
  The query contained a combination of words that were bound together by boolean operators such as OR, AND or NOT. Example: ‘DiseaseA’ AND ‘MethodA’ OR ‘MethodB’.

- **Semantic search**
  The query contained words/phrases that had to be interpreted together in order to give contextual meaning to the words. Example: ‘Latest report from ClientB where MethodA was used’.

The result of the search query categorization can be seen in Figure 2.

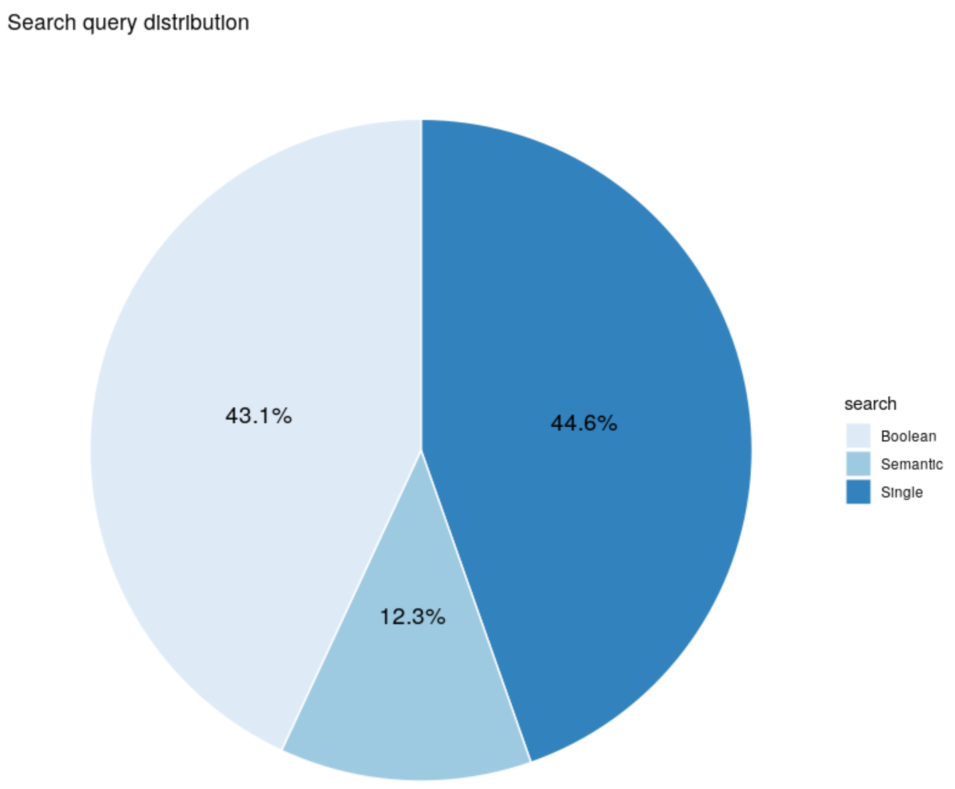


Figure 2. Distribution of example search queries collected.

About 45% of the collected example search queries contained single words. Most of these were a particular method, disease, variable or model. A lot were disease, method or model abbreviations. For example, a method called ‘very fast method’ would be written VFM. Some words were single variable names used in model codes.

The fraction of example searches categorized as Boolean queries were about 43%. Some of these were longer method, client, or disease names containing several words. In a Boolean search these would be referred to as exact phrases. To use the previous example again, ‘very fast method’ would be an example of an exact phrase where the user would want to search for the words in that exact order. Other queries were a particular method in combination with a specific model. For example, ‘MethodA’ AND ‘ModelC’. Other queries contained the OR operator and could be a search for two exact phrases using different words for a specific phrase such as ‘Selection of dose’ OR ‘Dose finding’.

Only 12% of the collected sample search queries were categorized as semantic searches. These types of queries could contain a specific version of a document in combination with a specific client, method or drug. One example could be ‘Latest analysis report from ClientY investigating DrugZ’.

The desired search results from these example search queries were also divided into different categories. These were chosen depending on the identified search functionalities.

- **All projects**
  Of all the collected answers, 18 expressed that the desired search result was a list of all projects relevant to the search query.
  Example: ‘A list of all projects were ModelA was applied.’

- **Filter**
  46 of the answers desired a search result filtered in different ways:

  - **Type of document**
    38 collected answers stated that the desired search result was hits in specific types of documents. These were: analysis plans, analysis reports, data management plans, project proposals or project updates. They also wanted to be able to search in combinations of these, at the same time.
    Example 1: ‘All analysis plans and analysis reports where ModelA was used’
    Example 2: ‘Data management reports where VariableY was included as a variable’

    Other answers stated that they wanted to filter to only search for files with a specific file ending: .mod, .Rnw, .R. These file endings represent code files.
    Example 1: ‘Search .Rnw or .R files for this word. This will allow the user to find example code quickly’
    Example 2: ‘Search all control streams (.mod) for this word’

  - **State of document**
    4 answers expressed the need for only searching for documents with a specific status, such as latest, final or signed version.

  - **Part of document**
    5 answers expressed that it was desired to be able to search in specific sections/parts of documents.
    Example 1: ‘Search among objectives or aims in proposal, analysis plan or analysis report’
    Example 2: ‘Search results section in analysis reports for these words’

- **Preview result**
  Another desired featured was a preview of the result. The result should display the part of the text where the search results are positive, with the query words highlighted.

- **Clickable result**
  Some answers desired a fast access link to the documents or the code file on the result page.

2.2.2 Core technology requirements

The core technology requirements were identified as either required, desired or nice-to-have.

- **Required functionality**
    - Compatible with Linux distributions
    - Capable of indexing files with file endings: .pdf, .docx, .R, .mod, .Rnw.
    - Low required maintenance
    - Based on programming languages Python or Java
    - Existing web graphical user interface (GUI)

- **Desired functionality**
    - Easy division and implementation of multiple indexes
    - Built in or easy development of user authentication
    - Available API for future improvement of GUI functionality

- **Nice-to-have**
    - Administration GUI

2.3 Discussion

The analysis of user requirements showed that the majority of users desired a keyword and Boolean search on file content. A great part of users also wanted the possibility to filter search result, mainly on document/file type. Lastly, they would also benefit from having the results presented together with a preview, with hits highlighted where the search results are positive. A fast access link to the files would also be preferable. Since the demand for semantic search and searching in a specific part of a document was not high among the users, these requirements where considered not to be of high priority. Also because the implementation of these functionalities was considered to be more complicated.

The most common situations for when the employees would use this system appears to be when:

- Trying to find wordings others have used to describe something (a method, a model etc.) in a document.
- Looking for example code for how to implement a specific model or method.

- Finding variables in code files.

Since users had a limited amount of time to participate in this project, there was no time for meetings to further discuss user stories. A more extensive analysis through user feedback and input would probably have given a more detailed picture of the user requirements.

The collected core technology requirements were not many, but contained concrete system demands. However, several required some kind of own interpretation of the requirement, such as ‘low required maintenance’. The result of this requirement analysis served as a base for the literature study and POC.

3 Literature study

In this section, the method, result and discussion of the literature survey is stated.

3.1 Method

The literature study aimed to investigate candidate search engines available on the market, that fulfilled the system requirements. The goal was to find two search tool candidates for further evaluation. In the beginning of the study, evaluation questions to use both in the literature study and later in the POC were found. The questions were mainly constructed from information found in web articles on the topic of enterprise search evaluation (Buxton 2019, Hilger 2018, Laroui 2019, Singh Dang 2018). They were divided into four categories: Core technology and fit, Administration and maintenance, Indexing, and Search. These can be seen below.

Core technology and fit

- Is the framework compatible with existing systems?
- Does the base programming technology and language match competence at the company?
- Is the framework open source/commercial? Is it free?
- What was the solution created for? Where is it headed in the future?
- What in the solution can be customized? Is it easy/difficult to configure/tune?
- Is there any API available?
- What does the code look like?
    - Explore the risks regarding the technical implementation.
    - When was it last modified?

Administration and maintenance

- How complex/time consuming is the installation?

    - Subjective IT-admin experience.
- What are the skill sets needed to configure/customize the solution?
- What does the support community look like? Large/small? Online or at vendor? Is it active?

Indexing

- How much storage space does the indexing take up?
- How well does it scale?
- What is the speed of indexing and indexing latency?

Search

- Features and user experience
    - Does the solution offer a GUI for searching? Can it be customized?
    - Does the solution offer the desired features for searching with regard to user requirements? E.g. keyword, Boolean, semantic, word highlighting, spelling control, context etc.
    - How well does search results match the desired search results?
        - Evaluate using the collected example search queries and user stories.
- Relevancy
    - Is it possible to boost favored content?
    - How easily can the ranking be tuned? How flexible is it?
    - How is search scoring handled?

These questions were used as a baseline in the evaluation together with the system requirements.

In next step of the literature study, the availability and fit of available search solutions were investigated. The research was restricted to only include open source search tools. The literature study was mainly based on information found in web articles and search tool documentation. As soon as a search solution was found not to meet one of the system requirements, the evaluation was terminated. Out of the search solutions investigated, only the two most promising ones were selected to move on to the next step of the evaluation; POC.

3.2 Result

In total, 15 candidate search tools where investigated. 8 of these were found to be outdated or inactive. Others did not meet the system requirements. Here, the 4 most promising search solutions are stated. A summary of the results can be seen in Table 2. For a complete list of all search tools investigated, see Appendix 1.

3.2.1 Apache Lucene
Apache Lucene is an open source search library (Apache Software Foundation 2020) developed by Apache Software Foundation. It is written in Java and the main functionality is indexing and full-text search through Apache Lucene API (Luburić & Ivanović 2016). Multiple search engines are built on the Lucene library and the library has been likened to the engine of a car, where the engine is just a part of the entire search solution. Hence, it is not an out-of-box search solution since it cannot be used as-is (Tan 2020). Apache Lucene is a code library, not a search server. Coding has to be done using Lucene API in order to use Lucene (Smiley *et al.* 2015). Therefore, Lucene should be installed and used by developers/programmers. The library is written in Java but can be called from in a lot of other languages such as Python, .NET and C. Apache Lucene can be used to build applications for searching in data sources like databases, email clients and web sites etc. It is licensed under the Apache License, version 2.0. The solution offers detailed documentation and has a large support community. It offers many features like relevance ranking, different query types and hit highlighting (Apache Software Foundation 2020).

3.2.2 Apache Solr
Apache Solr is an open source, full-text enterprise search engine. It is built on Apache Lucene, and uses the Lucene library functionalities for indexing and search (Apache Software Foundation 2020). Here, Lucene is the engine and Solr is the rest of the car - a complete search solution. Solr uses HTTP for communication between the user and the server, this is done via multiple formats such as JSON, CSV and XML. Solr offers storage, indexing and search of data and also includes features such as an administration GUI, sample web GUI, full-text search, hit highlighting and data analysis (Smiley *et al.* 2015). Binary files such as pdf can be extracted with Solr Cell using Apache Tika. Solr is a widely used search solution and users include Netflix and Instagram etc (Apache Software Foundation 2020). It also has extensive documentation and a large support community.

3.2.3 Elasticsearch
Elasticsearch is another open source search engine. This search solution is also built on the Lucene library and written in Java. Like Apache Solr, it also has a REST-like HTTP API and stores data in JSON format (Elastic 2020a). The solution is mainly free under the Apache License but advanced features like machine learning applications have to be paid for (Elastic 2020a). Anyone can offer a contribution to the development of Elasticsearch but submissions always have to be accepted by employees at Elastic (Turnbull 2019). Elasticsearch is similar to Apache Solr as they are both built on the Lucene library, though Elasticsearch seems to be more focused on analysis and visualization of data while Apache Solr main focus appears to be text search. Elasticsearch offers users to start small and then scale the search solution, through what is called *shards,* as the need grows (Elastic 2020b).

3.2.4 Recoll
Recoll is a full-text, open source, desktop search engine available for Linux, Windows and Mac OS X (Dockes 2020). The search solution is built on the Xapian search engine library

and offers probabilistic relevance ranking, stemming, phrase and wildcard search through Boolean queries (Xapian 2020a). It can index most document formats like pdf, docx and html. Recoll offers a web GUI with preview and document download, built on the Python API (Dockes 2020). The community is small but the Recoll documentation is extensive and contains code examples.

3.2.5 Summary

Summary of investigated search engines can be seen in Table 2. The requirements matching the system requirements are colored in green, the ones not matching are colored in red.

Table 2. Investigated open source and free search engines.

	Apache Lucene	**Apache Solr**	**Elasticsearch**	**Recoll**
Written in	Java (implementation available in a lot of languages)	Java (implementation available in a lot of languages)	Java (implementation available in a lot of languages)	C++ and Python
Based on	Lucene library	Lucene library	Lucene library	Xapian library
Supported platforms	Linux/Unix/OSX and Windows	Linux/Unix/OSX and Windows	Linux/Unix/OSX and Windows	Linux/Unix/OSX and Windows
Available API	Java API for search and indexing	Java API REST-like HTTP API	Java API REST-like HTTP API	Python API
Support community	Large	Large	Large	Small
GUI	No	Yes, administration GUI and sample web GUI	N/A	Yes, qt GUI and sample web GUI
Provides required search features	Yes	Yes	Yes	Yes
Relevance ranking (The search tool has a way of estimating the relevance of each document for a specific query)	Yes	Yes	Yes	Yes
Out-of-box-solution (The search tool can be used as-is and does	No	Yes	Yes	Yes

not require programming skills)	🟧	🟩	🟩	🟩
Last updated source code	March 2020 (at time of evaluation)	March 2020 (at time of evaluation)	March 2020 (at time of evaluation)	March 2020 (at time of evaluation)

3.3 Discussion

The literature study was mostly seen as a way to get a better overview of existing search solutions. It was concluded that a lot of the evaluation questions were difficult to answer without actually installing/testing the search tools and going into deeper detail of search engine documentation.

The enterprise search market seems to mostly be dominated by search tools built on the Apache Lucene library. The majority of search engines investigated uses Apache Lucene as the underlying search engine. The two most widely used ones being Apache Solr and Elasticsearch. These are also used by many global companies. The two search engines seem very similar when it comes to how information is handled and retrieved. However, Elasticsearch seems to offer more in terms of data analysis tools available and Apache Solr appears to focus more on text mining functionality. Both Elasticsearch and Apache Solr offers advanced configuration/customization and they seem to be applicable for almost any data and search situation. The complexity of these search tools could be a challenge when it comes to implementation.

Recoll was a search tool recommended by the company to look more into. It is mainly a desktop search engine and there are limitations in how it can be configured/customized to user needs. However, it seems to meet most system requirements and its simplicity could be an advantage when it comes to implementation and maintenance.

In this study no commercial enterprise search engines were studied. This was mostly due to the fact that it was a complicated task to find relevant information from the commercial enterprise search web sites. Also, there were no companies offering a search tool test time that was enough to perform a POC.

3.3.1 Motivation for choice of search tools

Recoll was chosen as one of the candidates to further investigate as it seemed to meet most system requirements. It could be implemented on Linux/Unix systems, used relevance ranking through Xapian search engine library, provided a web GUI for Boolean search, and could index most document types such as pdf and docx. Recoll was also the search tool recommended by the company to look further into. Even though the main purpose of Recoll seemed to be desktop search, there was no indications that the search tool would not be a good candidate for serving as a web application used by multiple users.

It seemed interesting to compare the Recoll desktop search tool to one of the more advanced and widely used search tools available. It was concluded that building an own application from scratch only using Apache Lucene would be too technical and time consuming for this project. Therefore, the choice for the second candidate search engine to be further evaluated was between Apache Solr and Elasticsearch. As Elasticsearch seemed to be moving more towards data analysis functionality and the Apache Solr documentation seemed to be more extensive and detailed, Apache Solr was chosen as the second candidate.

4 Proof of concept

In this section, the method, result and discussion of the POC is stated.

4.1 Method

A POC was performed to demonstrate and explore the feasibility of the search tools Recoll and Apache Solr. In this step, Recoll and Apache Solr were installed, configured and tested on a local machine.

4.1.1 Installation

Both tools were installed on a 18.04.2-Ubuntu x86_64, with 4 cores and 11GB of RAM, local machine. Recoll 1.26.5 with Xapian 1.4.5 was installed through a binary copy for Ubuntu distribution, from Personal Package Archives. Before the installation, external tools for pdf and msword text retrieval were installed. These were supporting packages `pdftotext` and `antiword`.

Apache Solr version 8.4.1 was installed from a .tgz file, downloaded and verified from downloads.apache.org. The installation was followed out using instructions from Apache Solr documentation (Apache Software Foundation 2020).

4.1.2 Configuration

Recoll 1.26.5 was used with default configuration parameters. Supporting packages `pdftotext` and `antiword` was used to for extracting text-content from pdf and docx. Apache Solr 8.4.1 was run in standalone mode with the example configuration `sample_techproducts_config`. Pdf and docx file contents were extracted with Solr Cell using Apache Tika.

4.1.3 Benchmarking

The performance test was only done as a benchmark and did not involve real project documents. Instead, 100000 pdfs were simulated using text from the Blog Authorship Corpus

(Schler *et al.* 2006), the Enron Dataset (Klimt & Yang 2004), and Wikipedia Links data (Google Code Archive).

For each pdf, a file from each data source was randomly drawn, resulting in a pdf containing an image, a blogpost, a wiki post and an email. The number of pages were between 3-26 and the size 50-150 kB. Since files were randomly drawn, this could result in files from the data sources being picked multiple times. But still, no document contained the exact same content. The total size of 100000 pdfs were 7GB.

The testing was run on 10, 100, 1000, 10000 and 100000 documents. All measurements were performed using a bash script in command line. Time was measured using the built-in bash command `usr/bin/time` and was done 3 times for each command. For each number of documents the following parameters were measured:

- **Indexing time**
  Recoll and Apache Solr were both configured to index 10, 100, 1000, 10000 and 100000 documents. Between each step, indexes were removed and documents were reindexed.

- **Index size**
  For each indexing step, Recoll and Apache Solr index sizes were measured.

- **Query time**
  For the query time measurement, Recoll and Apache Solr query times were compared against the combined Linux command line tools `grep` and `find`. `Grep` and `find` commands can be used to search for file content in command line. In this test, `grep` and `find` was used as a control group. See Appendix 2 for more information about `grep` and `find`. Recoll and Apache Solr were both configured to only show 10 hits in the command line prompt. Only one query containing a single keyword was run.

4.2 Result

In the POC, Apache Solr and Recoll was tested out to see how well it met system requirements and how it answered to the evaluation question. The POC was divided into performance, system implementation, configuration, web GUI and relevance ranking.

4.2.1 Performance

The indexing time between Recoll and Apache Solr was very similar. It ranged from about 1 second for 10 files to about 32 minutes for 100000 files, see Table 3 and Figure 3. The indexing time appears to increase linearly. The use of CPU during indexing differed quite a bit with Recoll using almost 100% of all 4 cores available (see Figure 4), while Apache Solr used about 50% (see Figure 5).

Table 3. Recoll and Apache Solr mean indexing times.

Number of documents	**Recoll indexing time (s)**	**Apache Solr indexing time (s)**
10	0.78	0.77
100	1.71	2.15
1000	13.81	18.88
10000	154.05	199.26
100000	1949.35	1951.34

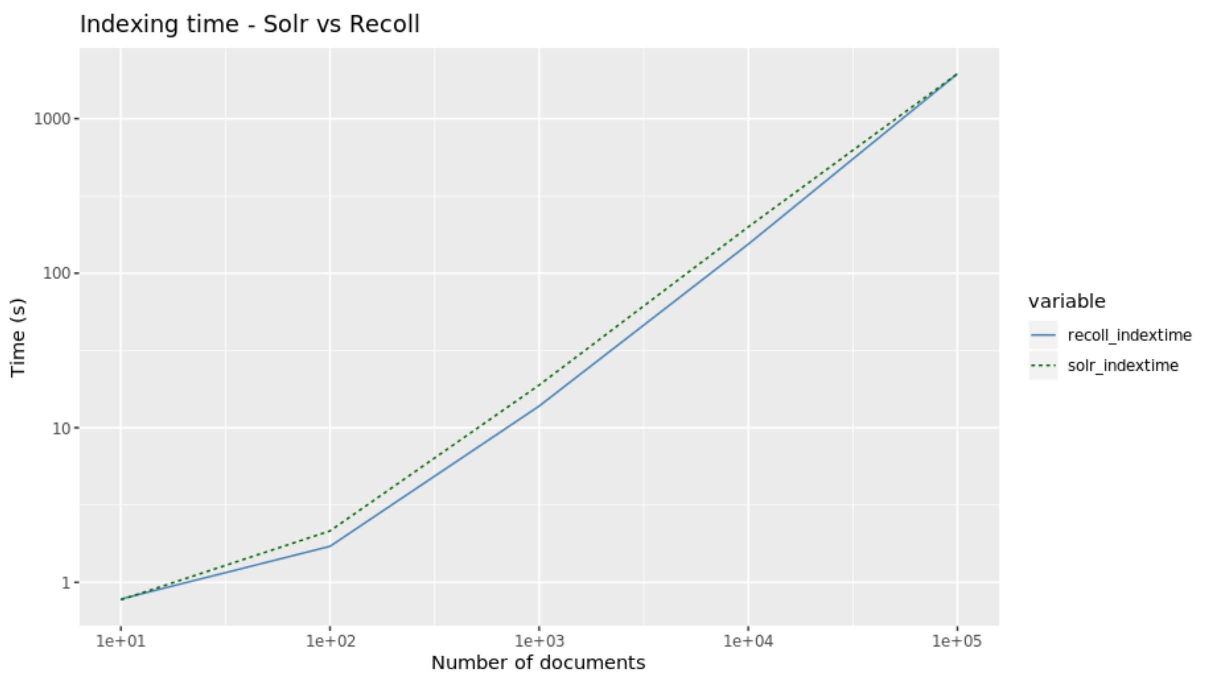


Figure 3. Indexing time Recoll vs Apache Solr.

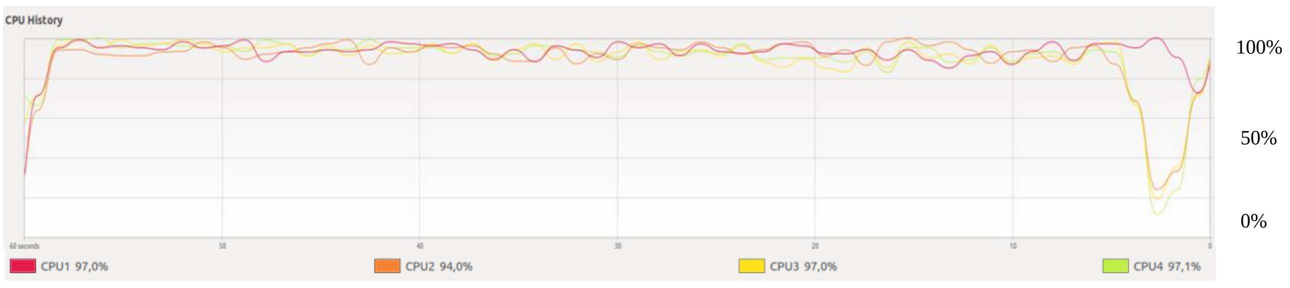


Figure 4. CPU usage Recoll when indexing 100000 documents.

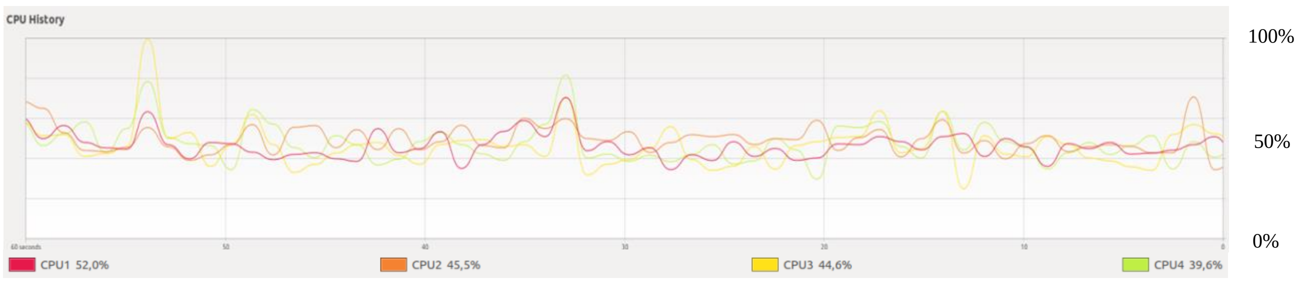


Figure 5. CPU usage Apache Solr when indexing 100000 documents.

A comparison of index size showed that Recoll index occupied almost the same disk space as the total size of indexed files. Apache Solr index on the other hand, took up about 20-30% of the total size, see Table 4 and Figure 6. For 100000 files of 7 GB, the Recoll index size was 6.5 GB while Apache Solr index size was 1.6 GB.

Table 4. Recoll and Apache Solr index sizes compared to the total size of indexed files.

Number of documents	**Total size of indexed documents (MB)**	**Recoll index size (MB)**	**Apache Solr index size (MB)**
10	0.74	0.776	0.264
100	6.8	4.9	1.4
1000	71	53	16
10000	707	632	147
100000	7000	6500	1600

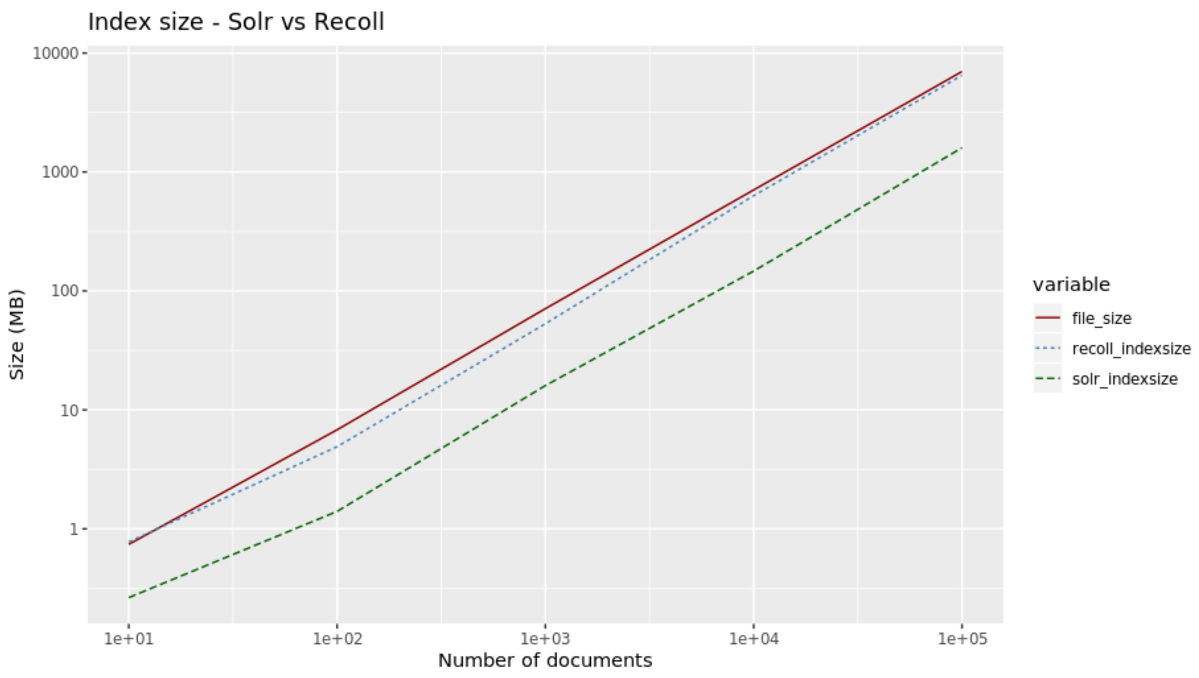


Figure 6. Index size Recoll vs Apache Solr.

The query time was also measured for 10, 100, 1000, 10000 and 100000 indexed files. This showed that both Recoll and Apache Solr could find hits among 100000 files at the same speed as 10 files, both search tools managed to do this under 0.1 seconds. The combined commands grep and find on the other hand, showed an increased search time, see Table 5 and Figure 7. A search amongst 100000 files took about 27 minutes.

Table 5. Query times for commands grep and find, Recoll and Apache Solr.

Number of documents	**grep+find query time (s)**	**Recoll query time (s)**	**Apache Solr query time (s)**
10	0.16	0.01	0.046
100	1.39	0.013	0.053
1000	14.99	0.013	0.046
10000	141.8	0.013	0.02
100000	1649	0.03	0.07

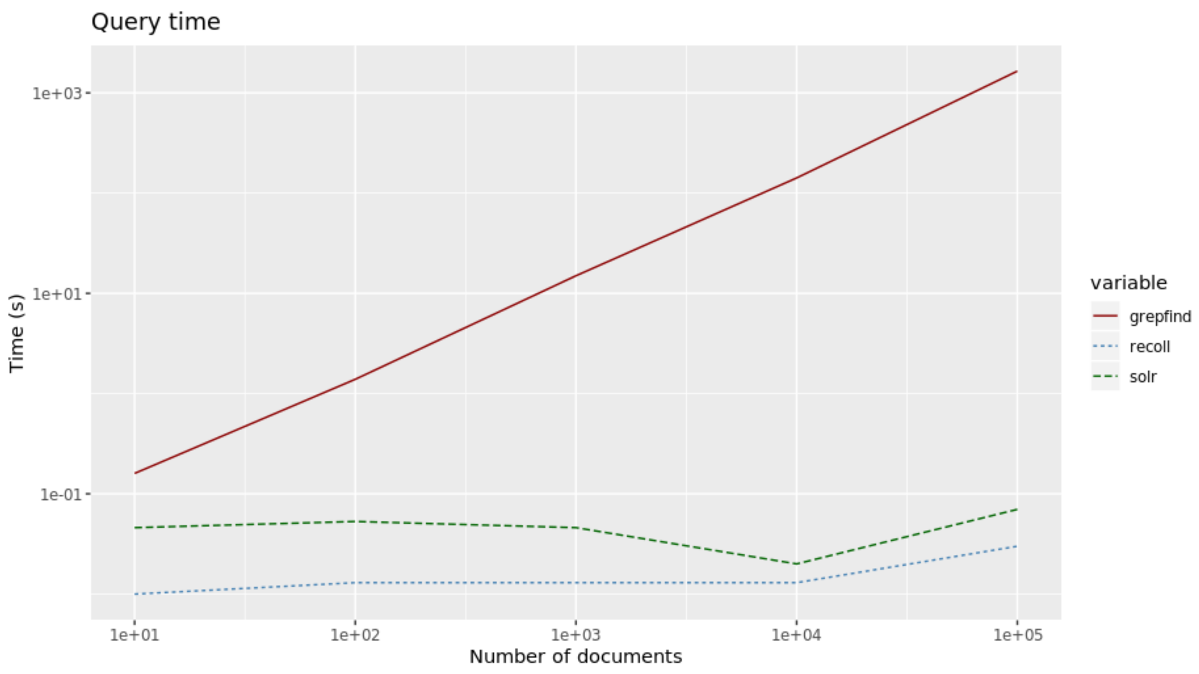


Figure 7. Query time when searching for a single keyword with grep and find, Recoll and Apache Solr.

When searching from the sample web GUI for both search tools, having 100000 documents indexed, the search response time were about 1 second for both Recoll and Apache Solr.

4.2.2 Apache Solr out of scope

After indexing company documents using Apache Solr, it was discovered that due to the lack of document metadata, Apache Solr could not automatically extract the file name from each document. The only fields that could be extracted from the documents were; the absolute path to the file, the mime type, the date of last modification, and document content. Some attempts were done on trying to extract the file name from the absolute path, but that required some configuration that was considered too time consuming. It was also discovered that implementation of desired features like hit highlighting also needed specific configurations in Apache Solr. Recoll could, by default, both identify the file name and include search features like hit highlighting. Therefore, together with the company, it was decided that the priority would be to continue the POC with only Recoll.

4.2.3 System test implementation

To demonstrate how Recoll could be implemented, a conceptual model was created. The model consisted of a file server and an application server, see Figure 8. After discussions with IT administration, the application server was set up as a virtual server 18.04.2-Ubuntu x86_64. On the application server, Recoll and an Apache2 web server was installed.

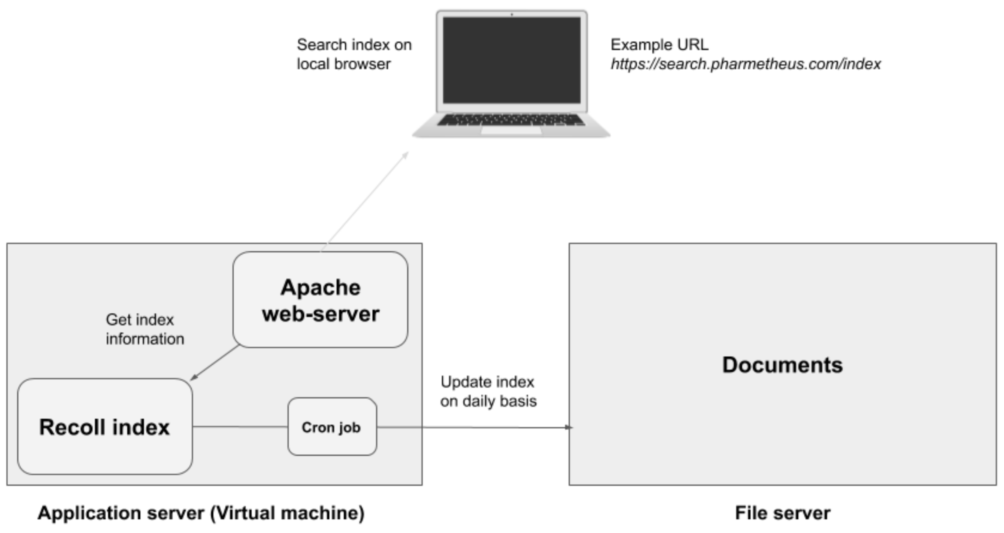

Search index on local browser
Example URL
https://search.pharmetheus.com/index
Apache web-server
Get index information
Recoll index
Cron job
Update index on daily basis
Documents
Application server (Virtual machine)
File server

Figure 8. A conceptual production setting model was set up to demonstrate the feasibility of the search tool Recoll.

Due to project confidentiality, only a limited amount of project documents were mounted onto the application server. These were only a small subset of the real projects on the file server and contained about 20 projects from 6 different clients. Recoll was configured to index a specific set of documents (see 4.2.4 Configuration) and a cron job was set up to have the Recoll index updated every day.

An Apache2 web server with the Recoll web GUI was set up so that it could retrieve information about the Recoll index and make it searchable from a local web browser, when connected to the company private VPN. The model was set up so that the users could access the search interface from their local browser using a specific URL.

4.2.4 Configuration

Recoll was configured to only index the finalized versions of analysis plans (ANA), analysis reports (REP), data management reports (DMR), project plans (PROPLAN), and project updates (PU). This was done with the parameter `onlyNames` and with the help of wildcards so that files only allowed to be indexed were files matching the pattern: DocumentType-AnyNumber-AnyText-PMX-AnyNumber.pdf:

```
onlyNames = ANA-[0-9]-*-PMX-[0-9].pdf /
REP-[0-9]-*-PMX-[0-9].pdf DMR-[0-9]-*-PMX-[0-9].pdf /
PROPLAN-[0-9]-*-PMX-[0-9].pdf PU-[0-9]-*-PMX-[0-9].pdf
```

This was done in order to not include all the draft versions of a document to the index as it would create multiple hits in essentially the same document. This resulted in an index only containing finalized versions of all ANA, REP, DMR, PROPLAN, and PU files from all mounted projects.

Recoll allows for configuration of wildcard paths where indexing should not take place and indexing crawler should never look. Since all documents relevant for indexing resides under the folder ‘ToClient’ for all projects, Recoll was configured through the parameter `skippedPaths` to skip indexing all other main directories under a project. These were the directories Analysis, FromClient, and ProjectManagement:

```
skippedPaths = /mnt/cluster/*/*/*/Analysis /
/mnt/cluster/*/*/*/FromClient /
/mnt/cluster/*/*/*/ProjectManagement
```

The stars in the absolute paths stands for matching all directories on that folder level. In this case it should match all current and archived projects, all clients and all projects under that client.

According to user requirements, users also want to be able to search in model code files. At Pharmetheus, these files have the file ending .mod. These .mod files have mime type audio/x-mod which is a type that is not recognized by Recoll. To be able to search in these files they have to be converted to a mime type that Recoll can index. This was done in two steps to first define the mime type inside Recoll:

```
.mod = audio/x-mod
```

and then assign which handler that should process this specific type:

```
[index]
audio/x-mod = internal text/plain
```

Recoll was configured to handle these mime types as plain text to be able to index and search the content of these files. The configuration for other file types like R or Rnw could be handled in the same way.

In Recoll, multiple indexes can be created and queried in parallel. This concept was tested by having Recoll indexing different sub-parts of the file share and then having them be queried together. In Recoll this is done through the use of environment variables. The main index is stored in RECOLL_CONFDIR, while extra indexes can be added in the environment variable RECOLL_EXTRA_DBS:

```
export RECOLL_CONFDIR=/home/user/Recoll/config_1
export RECOLL_EXTRA_DBS=/home/user/Recoll/config_2/
xapiandb:/home/user/Recoll/config_3/xapiandb
```

Additional configuration and index directories were created and configured to index different sub-parts of the file-share. To be queried in parallel these were then added as environment variables.

4.2.5 Relevance ranking

Recoll is based on the search engine library Xapian which uses probabilistic information retrieval to rank results based on calculated relevance. More specifically, Xapian uses the probabilistic weighting scheme called BM25 (Robertson & Zaragoza 2009) to estimate the probability of relevance of a document, given a specific search query. The model assumes that documents are either relevant or not relevant to a search query and uses several parameters to estimate the probability of a document being relevant (Xapian 2020b), some of these are:

- Within query frequency - how many times a term occurs in a query
- Within document frequency - how many times a term occurs in a document i.e. the number of times a term is extracted from a document during indexing
- Number of documents that are indexed by a term
- Total number of documents indexed
- Normalized document length - calculated by counting how many words a document contains divided by the average number of words in a document among all indexed documents

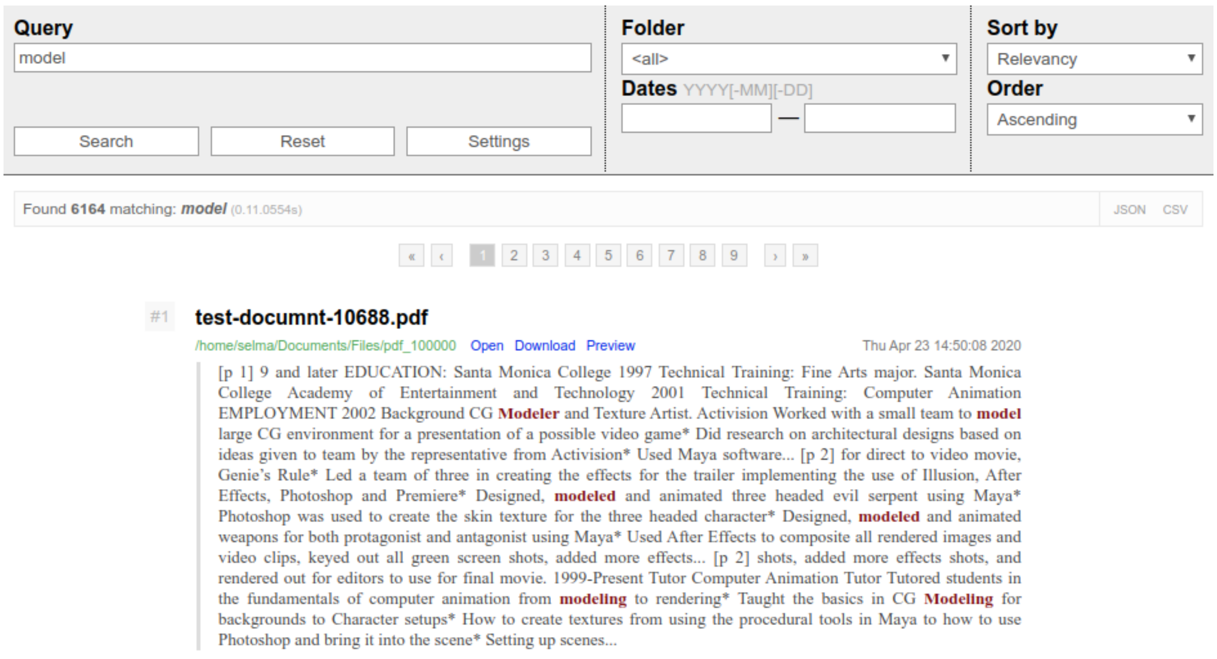
The BM25 uses these parameters to estimate how probable a document is to be relevant, for a given search query (Xapian 2020b). Recoll does not include any functionality for boosting relevance when searching in the web GUI.

4.2.6 Web GUI

The provided sample web GUI for Recoll allows the user to search in a query field (see Figure 9). Here, the user can use the Boolean query language to extract information from indexed documents.

The user is able to sort the result by relevancy or date and can also choose to have the result presented in ascending or descending order. If the user knows which directory in the file system to search in, it has the possibility to choose folder in the dropdown menu ‘Folder’. The results can also be filtered by date of modification, which the user can specify under Dates.

The search results are presented with a snapshot where the search words are highlighted. It also shows the absolute path to the file (in green) and allows the user to preview or download the file. It is also possible to see that Recoll uses stemming as searching for the word ‘model’ also gives hits on other word endings, in this case ‘modeler’, ‘modeled’, and ‘modeling’. The GUI uses paging for handling multiple hits. Clicking the arrows allows going to the next page of hits. Each page shows 25 results per default, this can be configured.



Query: model

Folder: <all>

Sort by: Relevancy

Order: Ascending

Found **6164** matching: ***model*** (0.11.0554s)

test-documnt-10688.pdf

/home/selma/Documents/Files/pdf_100000

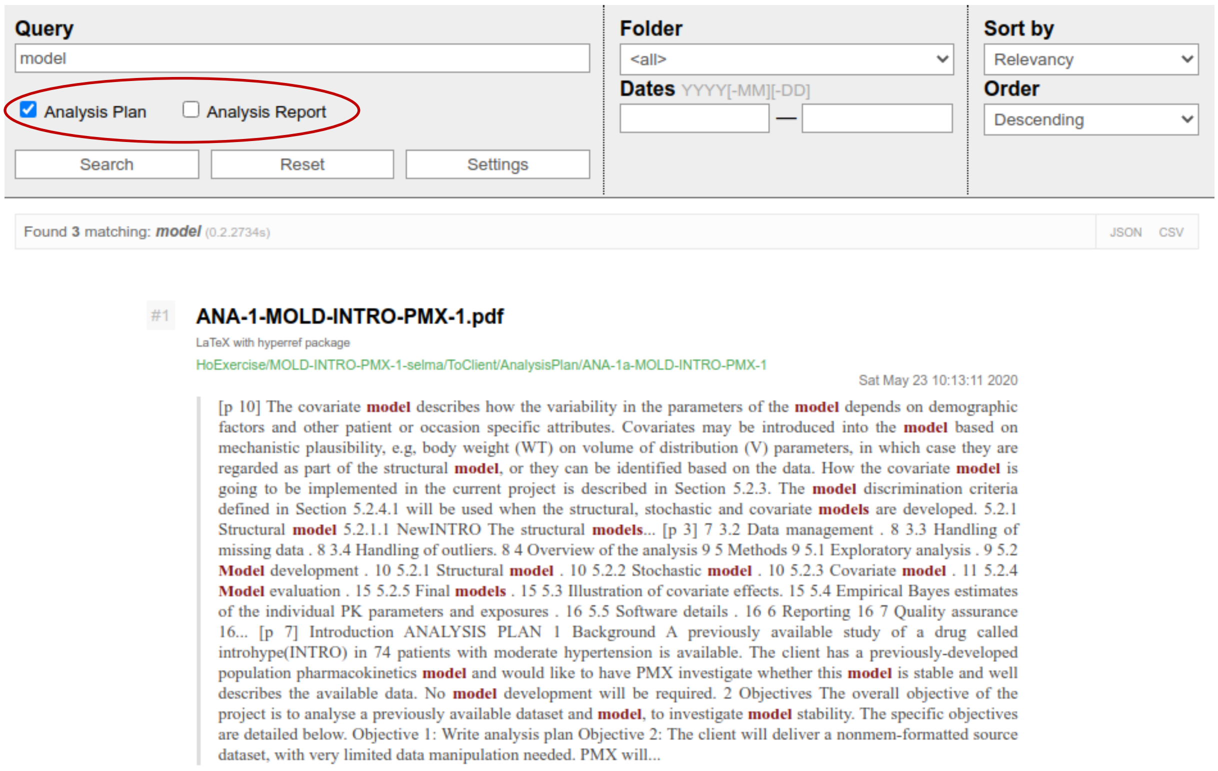
Thu Apr 23 14:50:08 2020

[p 1] 9 and later EDUCATION: Santa Monica College 1997 Technical Training: Fine Arts major. Santa Monica College Academy of Entertainment and Technology 2001 Technical Training: Computer Animation EMPLOYMENT 2002 Background CG **Modeler** and Texture Artist. Activision Worked with a small team to **model** large CG environment for a presentation of a possible video game* Did research on architectural designs based on ideas given to team by the representative from Activision* Used Maya software... [p 2] for direct to video movie, Genie’s Rule* Led a team of three in creating the effects for the trailer implementing the use of Illusion, After Effects, Photoshop and Premiere* Designed, **modeled** and animated three headed evil serpent using Maya* Photoshop was used to create the skin texture for the three headed character* Designed, **modeled** and animated weapons for both protagonist and antagonist using Maya* Used After Effects to composite all rendered images and video clips, keyed out all green screen shots, added more effects... [p 2] shots, added more effects shots, and rendered out for editors to use for final movie. 1999-Present Tutor Computer Animation Tutor Tutored students in the fundamentals of computer animation from **modeling** to rendering* Taught the basics in CG **Modeling** for backgrounds to Character setups* How to create textures from using the procedural tools in Maya to how to use Photoshop and bring it into the scene* Setting up scenes...

Figure 9. The Recoll sample web GUI.

The sample web GUI is built on the Python API. This allows for the modification of the web GUI or development of an own search interface.

To test the concept of users being able to filter the results by document type, which was one of the system requirements, the Recoll web GUI was modified to include functionality for searching among all analysis plans or analysis reports. Checkboxes were added so that the user would be able to filter the results by checking the boxes (see Figure 10).



Found **3** matching: ***model*** (0.2.2734s)

ANA-1-MOLD-INTRO-PMX-1.pdf

LaTeX with hyperref package

HoExercise/MOLD-INTRO-PMX-1-selma/ToClient/AnalysisPlan/ANA-1a-MOLD-INTRO-PMX-1

Sat May 23 10:13:11 2020

[p 10] The covariate **model** describes how the variability in the parameters of the **model** depends on demographic factors and other patient or occasion specific attributes. Covariates may be introduced into the **model** based on mechanistic plausibility, e.g, body weight (WT) on volume of distribution (V) parameters, in which case they are regarded as part of the structural **model**, or they can be identified based on the data. How the covariate **model** is going to be implemented in the current project is described in Section 5.2.3. The **model** discrimination criteria defined in Section 5.2.4.1 will be used when the structural, stochastic and covariate **models** are developed. 5.2.1 Structural **model** 5.2.1.1 NewINTRO The structural **models**... [p 3] 7 3.2 Data management . 8 3.3 Handling of missing data . 8 3.4 Handling of outliers. 8 4 Overview of the analysis 9 5 Methods 9 5.1 Exploratory analysis . 9 5.2 **Model** development . 10 5.2.1 Structural **model** . 10 5.2.2 Stochastic **model** . 10 5.2.3 Covariate **model** . 11 5.2.4 **Model** evaluation . 15 5.2.5 Final **models** . 15 5.3 Illustration of covariate effects. 15 5.4 Empirical Bayes estimates of the individual PK parameters and exposures . 16 5.5 Software details . 16 6 Reporting 16 7 Quality assurance 16... [p 7] Introduction ANALYSIS PLAN 1 Background A previously available study of a drug called introhype(INTRO) in 74 patients with moderate hypertension is available. The client has a previously-developed population pharmacokinetics **model** and would like to have PMX investigate whether this **model** is stable and well describes the available data. No **model** development will be required. 2 Objectives The overall objective of the project is to analyse a previously available dataset and **model**, to investigate **model** stability. The specific objectives are detailed below. Objective 1: Write analysis plan Objective 2: The client will deliver a nonmem-formatted source dataset, with very limited data manipulation needed. PMX will...

Figure 10. The Recoll web GUI modified to filter search among analysis plans.

4.3 Discussion

In this section, the results from the POC are discussed. The discussion is divided into performance, system test implementation, configuration, web GUI, and relevance ranking.

4.3.1 Performance

Overall, Apache Solr showed better performance by using about 70% less disk space for the index and using less CPU power while still indexing documents at the same speed as Recoll. However, both Recoll and Apache Solr appears to scale linearly when it comes to indexing time and index size. A more extensive proof of scalability could have been shown if both search tools would also have been tested on a set of 1 000 000 files. Due to a limited amount of computer disk space and time, such a test was never performed.

In real projects, Pharmetheus file sizes are in the range of 0.1-10 MB which is much larger than the files indexed in the benchmarking, these were about 50-150 KB in size. It would have been preferred to run the performance test on real project documents, which was not possible due to document confidentiality. This would have given a better estimation of the real index size. The amount of disk space available on the computer was also a restricting factor since all files used in the benchmark had to be stored on disk before they could be indexed. Even though the benchmarking was not done on real project files, it gives an indication of the amount of disk space needed when implementing the system in a production setting.

For the measurement of query time, the only query run was a single keyword. In this test, a grep and find command was used as a control group to show how long such a search would take without the use of an index. It was interesting to see that neither Recoll or Apache Solr showed an increase in query time as the set of files grew. This could be due to the fact that only the ten most relevant hits were printed in the command line prompt. If more hits would have been printed, maybe the query time would have increased. Another interesting test would have been to use more complicated queries with Boolean operators like OR and AND to see if the query time would increase.

In a real situation, users would not be searching directly from the command line but from a web GUI. That would increase the search response time as the result would have to be rendered in the web interface. One interesting test would have been to have multiple users try out the search tool from the web GUI and see what happens when Recoll has to handle multiple search requests at the same time.

4.3.2 System test implementation

For the experimental system set up, IT administration at Pharmetheus was involved in the discussion and set up of the system design. The POC demonstrated that this system design is feasible. Keeping the index and the documents on separate servers seems to work well and all components were successfully implemented. Other system design alternatives are to have the web server on a separate server or to put the index on the same server as the documents.

One component that is missing from the system design is authentication and authorization. As Pharmetheus holds a lot of classified projects and documents, a system for authentication would have to be implement before the search engine could ever be put into production. This security requirement should probably have been set as a required functionality rather than a desired feature in the requirement analysis. The Recoll database containing the index would also have to be protected in a production setting. Recoll does not provide plugins or API for security implementation, but there might be other ways to implement security features. Due to lack of time, this was not investigated during this project.

Apart from the security aspects, it would be wise to implement logging features for the cron job to make sure that the index update proceeded as expected.

4.3.3 Configuration

In the configuration testing, it was concluded that the strict file structure and naming conventions at the company can be used to index specific sub-parts of the file share and also specific document types. Recoll can be configured to full-text index the desired file types .pdf, .docx, .mod, and .R. With the use of wildcards, Recoll has been proven to be able to index specific sub-parts of the file share and also match specific file names to be indexed. It has also been shown that Recoll can make use of multiple indexes, and query them in parallel. Something that allows for indexes to be updated at different times, which could improve indexing performance. In a production setting, it would be important to handle the configuration files in the company’s existing source control system. For unexpected events, such as a system crash, it will be possible to recreate the index from the configuration files.

4.3.4 Relevance ranking

Recoll uses probabilistic relevance ranking to find the most relevant documents from a given query but does not support any functionality for tuning or boosting favored content. In more advanced search engines, such as Apache Solr, it is possible to boost favored content by adding weights to certain fields or adding relevancy to those documents where the search terms appears closer together in the document (Mihalcea 2020). For example, the document gets a higher relevance if the search query appears in the document title. In that way, the relevance ranking can be customized to a company’s needs. This may or may not be a desired functionality within Pharmetheus, but it is worth mentioning that more advanced search engines can offer such functionalities.

4.3.5 Web GUI

The existing GUI fulfils the basic user requirements such as Boolean search, search result snapshots, word highlighting, document preview, and document download. By the use of the Python API it is possible to develop a customized web GUI. I have shown that it possible to modify the GUI so that it includes functionality for filtering results by document type. In a production setting, this code for the web interface should of course also be documented in the company source control system.

4.3.6 User feedback on implemented test system

At the end of the project, I held a live demonstration of the implemented search system where the users at the company had the chance to give their input. The users gave positive feedback on the search system. Most of the user requirements appears to have been captured and met. The users seemed especially happy with the possibility of full-text search of the indexed documents. Again, the users stated that filtering on document type is a relevant feature.

5 Conclusion and recommendation

This project included requirement analysis, literature study and POC to find and evaluate a candidate search engine to be implemented at the pharmacometric consulting company Pharmetheus. The purpose was to facilitate the knowledge management at the company by helping employees find information in previous projects. A requirement analysis showed that the majority of users desired a full-text file search through Boolean queries (see 2.2 User requirements). Users also wanted the possibility to filter documents by document type, such as analysis plan, report or model code. Another desired functionality was to be able to search filetypes .pdf, .docx, .R and .mod.

Through a literature study, open source search engines were investigated. The evaluation was based on the collected system requirements and assessment questions (see 3.1 Method). It was interesting to see that the field of open source enterprise search seemed to be dominated by applications build on the Apache Lucene search library. However, some search engines built on the Xapian search library were also found. It also proved difficult to answer many of the assessment questions without actually installing and testing the search engines. However, two search engines, Recoll and Apache Solr, were selected as promising candidates.

In the POC, a performance test with Recoll and Apache Solr showed no significant difference in indexing time between the two. For 100000 documents, indexing time appeared to scale linearly with both Recoll and Apache Solr, but further investigations has to be done to demonstrate scalability. When searching in command line, neither Recoll or Apache Solr show a significant increase in search time between 10 and 100000 documents and they both outperform command line tools `grep` and `find`, demonstrating why the use of an index is needed. (see 4.2.1 Performance)

Due to the lack of document metadata, Apache Solr was not able to directly extract the file name of the indexed documents (see 4.2.2 Apache Solr out of scope). As Apache Solr required more configuration than Recoll, it was concluded that it would be too time consuming to get something up and running with Apache Solr within the time frame of this project. A decision was made to only perform the system test implementation with Recoll.

The Recoll test implementation demonstrated the possibility to utilize the company’s strict folder structure and naming conventions to index specific documents in sub-parts of the file share. Another interesting discovery was how mime types for not supported file types could be configured to be handled as plain text inside Recoll. Allowing indexing and full-text search on model files (.mod) as well. (see 4.2.4 Configuration)

It was also discovered that the Recoll web GUI met user requirements by allowing for Boolean search and by how it could be modified to add filtering on document type (see 4.2.5 Web GUI).

Overall, the search engine Recoll has proved to meet system requirements when it comes to core technology, indexing, and search. However, to be able to be used in a production setting at the company, the system has to include security in terms of authentication (see 4.2.3 Recoll system test implementation). This is something that has to be further evaluated at the company.

6 Acknowledgements

I would like to express my thanks to my supervisor at Pharmetheus, Kajsa Harling, for her guidance and support throughout this project. I would also like to extend my gratitude to my subject reader, Ben Blamey, for his valuable feedback on this project and the report. Lastly, I want to thank Rikard Nordgren for his support and feedback throughout the project.

References

Apache Software Foundation. 2020. Apache Lucene - Apache Lucene Core. online 2020: https://lucene.apache.org/core/. Accessed May 16, 2020.

Apache Software Foundation. 2020. Apache Solr -. online 2020: https://lucene.apache.org/solr/. Accessed May 16, 2020.

Buxton S. 2019. The Essential Guide to Enterprise Search. online June 25, 2019: https://www.marklogic.com/blog/the-essential-guide-to-enterprise-search/. Accessed May 16, 2020.

Demirel ST, Das R. 2018. Software requirement analysis: Research challenges and technical approaches. 2018 6th International Symposium on Digital Forensic and Security (ISDFS), pp. 1–6.

Dmitriev P, Serdyukov P, Chernov S. 2010. Enterprise and desktop search. Proceedings of the 19th international conference on World wide web, pp. 1345–1346. Association for Computing Machinery, Raleigh, North Carolina, USA.

Dockes J-F. 2020. RECOLL: a personal text search system for Unix/Linux. online 2020: https://www.lesbonscomptes.com/recoll/features.html. Accessed May 17, 2020.

Elastic. 2020a. What is Elasticsearch? online 2020: https://www.elastic.co/what-is/elasticsearch. Accessed May 17, 2020.

Elastic. 2020b. Scalability and resilience: clusters, nodes, and shards | Elasticsearch Reference [7.7] | Elastic. online 2020: https://www.elastic.co/guide/en/elasticsearch/reference/current/scalability.html. Accessed May 17, 2020.

Ellingwood J. 2013. Using Grep + Regex (Regular Expressions) to Search Text in Linux. online 2013: https://www.digitalocean.com/community/tutorials/using-grep-regular-expressions-to-search-for-text-patterns-in-linux. Accessed June 4, 2020.

Google Code Archive. Google Code Archive - Long-term storage for Google Code Project Hosting. online: https://code.google.com/archive/p/wiki-links/downloads. Accessed May 18, 2020.

Gunawardhana P. 2019. Process of Requirement Analysis Link to Software Development. Journal of Software Engineering and Applications 12: 406–422.

Klimt B, Yang Y. 2004. The Enron Corpus: A New Dataset for Email Classification Research. In: Boulicaut J-F, Esposito F, Giannotti F, Pedreschi D (ed.). Machine Learning: ECML 2004, pp. 217–226. Springer, Berlin, Heidelberg.

Luburić N, Ivanović D. 2016. Comparing Apache Solr and Elasticsearch search servers. 5.

Mihalcea O. 2020. Tuning search relevance with Apache Solr. online May 8, 2020: https://medium.com/techlabs-emag/tuning-search-relevance-with-apache-solr-part-1-5d56c6b67f8f. Accessed May 24, 2020.

Musienko Y. 2019. What is Proof of Concept (POC) in Software Development - Merehead 1162. online November 25, 2019: https://merehead.com/blog/proof-concept-software-development/. Accessed May 27, 2020.

Ornbo G. 2020. Linux and Unix find command tutorial with examples. online 2020: https://shapeshed.com/unix-find/. Accessed June 4, 2020.

Robertson S, Zaragoza H. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. Foundations and Trends in Information Retrieval 3: 333–389.

Rosa ATR, Pustokhina IV, Lydia EL, Shankar K, Huda M. 2019. Concept of Electronic Document Management System (EDMS) as an Efficient Tool for Storing Document. Journal of critical reviews 6.

Schler J, Koppel M, Argamon S, Pennebaker J. 2006. Effects of Age and Gender on Blogging. 6.

Smiley D, Pugh E, Parisa K, Mitchell M. 2015. Apache Solr Enterprise Search Server - Third Edition. Packt Publishing Ltd

Standing JF. 2017. Understanding and applying pharmacometric modelling and simulation in clinical practice and research. British Journal of Clinical Pharmacology 83: 247–254.

Stocker A, Richter A, Kaiser C, Softic S. 2015. Exploring barriers of enterprise search implementation: a qualitative user study. Aslib Journal of Information Management 67: 470–491.

Tan K. 2020. Lucene vs Solr - Lucene Tutorial.com. online 2020: http://www.lucenetutorial.com/lucene-vs-solr.html. Accessed May 16, 2020.

Turnbull D. 2019. Stop Worrying about Solr vs Elasticsearch Decisions. online February 28, 2019: https://opensourceconnections.com/blog/2019/02/28/stop-worrying-solr-elasticsearch/. Accessed May 17, 2020.

Uzialko AC. 2019. Choosing a Document Management System: A Guide. online February 20, 2019: https://www.businessnewsdaily.com/8026-choosing-a-document-management-system.html. Accessed May 27, 2020.

Xapian. 2020a. The Xapian Project : Features. online 2020: https://xapian.org/features. Accessed May 17, 2020.

Xapian. 2020b. Theoretical Background. online 2020: https://xapian.org/docs/intro_ir.html. Accessed May 23, 2020.

Zilio D, Agosti M, Turato D. 2015. An Approach to the Design and Evaluation of an Enterprise Search Application. 7.

Appendix 1 – Investigated search engines

Table 1. Active open source search engines investigated.

Name	**Code last modified**
Apache Lucene	March 2020
Apache Solr	March 2020
Elasticsearch	March 2020
Fess	March 2020
Recoll	March 2020
Sphinx	N/A
Terrier	March 2020
Tracker	March 2020

Table 2. Inactive open source search engines investigated.

Name	**Code last modified**
DataparkSearch	July 2018
DocFetcher	N/A
Google Search Appliance	N/A
Ht-//Dig	June 2004
KinoSearch	April 2012
mnoGoSearch	N/A
SearchDaimon	July 2017
Swish-E	Nov 2016

Appendix 2 – Grep and find search commands

`Grep` and `find` are command-line tools for Linux and Unix-based systems. `Grep` can be used to search file content for strings matching regular expression patterns (Ellingwood 2013). The command `find` can be used to find directories and files in a file hierarchy (Ornbo 2020). If used together, grep and find can be used to find files and search file content in a Linux/Unix environment in a similar way to a search tool. `Grep` and `find` uses no index, relevance ranking or search GUI. But can be easily be implemented and used on a file share without the need for any installation or server.