Bachelor Project

Bachelor Information Engineering at the University of Konstanz
Department of Computer and Information Science

Subject of Study: Information Engineering
Specialization: Computer Science
Topic: Applied Informatics

by Mario Schreiner
Matriculation number: 01/733138
<mario.schreiner at uni-konstanz.de>
Abstract

In a time of information overflow, ways of finding, evaluating and understanding information as quickly as possible are more important than ever. Especially knowledge- and library-workers often seek information about a topic without looking for that one distinct piece of information that ends their search. Therefore, they often dig through a lot of more and less relevant information before considering their information need satisfied. Aiding them in this process is what we tried in this bachelor project. In the environment of the ‘Blended Library’, a modern library room at the University of Konstanz where traditional library work and modern technologies meet, we developed an information retrieval system that tries to achieve two main objectives: 1) Aid library workers in their way of search by not forcing them to keyword-search by metadata, but rather explore topics of interest in a modern way and 2) ease the so-called process of ‘sensemaking’ for those library workers, which is a process that is part of every search where the searcher tries to build a mental model of the information that encodes as much information as possible with as little effort as possible. For that purpose, we built a multitouch-based system that allows for a different approach to search and a different approach to ‘digging through results’.

In this text, we give an overview over the theoretical concepts behind the system, the motivation to build it and the important parts of the implementation. We then take a look at the future and how it can be extended and evaluated.
Einleitung


In diesem Text werden wir zuerst eine Übersicht über die theoretischen Konzepte und die Motivation hinter dem System geben. Danach gehen wir auf wichtige Teile der Implementierung ein und enden mit einem Blick in die Zukunft indem wir uns ansehen, wie das System in Zukunft erweitert und evaluiert werden könnte.
CONTENTS

Contents

1 Motivation ........................................ 2

2 Environment ....................................... 3
  2.1 Facet Search++ ................................... 3
  2.2 Samsung SUR40 with Microsoft PixelSense ............... 3
  2.3 WPF / C# ........................................ 4

3 Implementation .................................... 5
  3.1 Theoretical Concepts ................................. 5
    3.1.1 Multitouch interaction .......................... 5
    3.1.2 2D landscape .................................. 5
    3.1.3 Semantic Zoom .................................. 5
    3.1.4 Search ......................................... 5
  3.2 Architecture ...................................... 7
    3.2.1 Data Sources ................................... 7
      3.2.1.1 BaseX ...................................... 7
      3.2.1.2 LDA ........................................ 8
    3.2.2 SingleExecutionMethodWorker ......................... 9
    3.2.3 GlobalSearchHelper ............................... 10
    3.2.4 Main View .................................... 10
  3.3 The Timeline Visualization .......................... 12
    3.3.1 Topic Cloud .................................... 14
    3.3.2 Timeline ....................................... 17
    3.3.3 Author Details .................................. 19
    3.3.4 Token Distribution Circle ......................... 20
  3.4 Personal landscape .................................. 23
  3.5 Challenges & Lessons Learned ........................ 25

4 Future Work ....................................... 27
  4.1 Implementation .................................... 27
    4.1.1 Timeline visualization .......................... 27
    4.1.2 Personal landscape .............................. 27
    4.1.3 Backtracking .................................... 27
    4.1.4 Cooperative work ................................ 27
  4.2 User Study ........................................ 27

5 Conclusion ......................................... 29

Appendix A Example XML-Structure of a document ........... 31
1 Motivation

We live in a time where the internet provides everyone of us with an almost unlimited amount of information. Searching the internet for information is part of the day-to-day activities of most knowledge seekers. These days, the main challenge often is not where to get the information, but how to dig through the huge pile of available information and find out the most relevant, most useful.

An important part of the process of finding relevant information is the so-called ‘Sensemaking’ as described by Pirolli and Card[7, p. 2]. Sensemaking is the process of organizing information in a way that helps to create new knowledge while taking less time than simply reading through everything available.

This project wants to create a sensemaking-related information retrieval system in a multitouch environment. The goal is to make it possible to search through and discover documents without specifically knowing what to look for. Traditional systems often work with a "search by keyword, then filter" approach. This might be suited in cases where the desired result is known beforehand, but if only a rough idea of the desired outcome exists it can be difficult to find relevant documents with this approach. In this project, we try to find documents by giving the user the opportunity to search through a data space by topic and relationship rather than by pure metadata filtering. This is supposed to help the user in finding relevant documents without specifically knowing what to look for.

To do that, we determine topic probabilities for each document using the LDA algorithm as described in Section 3.2.1.2 on page 8. This allows us to group documents by topic and it allows the user to search documents by the topic he is interested in or the general direction he is looking for, rather than by more abstract metadata like author or keywords. Additionally, each topics’ most important authors are calculated and visualised and each authors’ documents can be viewed. Related documents can be seen once the user is down to individual author granularity. This way a user can browse the document space in a serendipity-like approach, going by ‘what seems to fit’ or get going from something he already knows to be relevant, like a known author or a relevant keyword of the topic he wants to explore. For making browsing the document space an easy and pleasant experience, the system is built around the ZOIL framework[9]. This allows the user to use known, understandable interaction concepts like panning, zooming, semantic zoom and more. Finally, the project linked to another project called Facet Search++. Facet Search++ allows the user to still use the search-and-filter-approach if needed, but it allows for easy building of complex boolean queries and it keeps the user in the multitouch context. We will nonetheless explore a possibility to use the system without Facet Search++, for example in a mobile environment where Facet Search++ might not be available.
2 Environment

2.1 Facet Search++

The application created in this bachelor project is tied - an extension if you will - to an already existing application that has been developed in the Blended Library at the University of Konstanz[1]. The Facet Search++ application can be thought of as modern way to filter search results, allowing the user to narrow down the document space before browsing it with our information retrieval application. Facet Search++ allows the user to search a document database via complex boolean queries. The queries can be created easily without a deeper understanding of boolean rules by using the Facet Streams concept[2]. Additionally, up to three tokens can be placed anywhere inside the stream. The search query that the stream represents at the location of the token is then sent to the underlying database. Our information retrieval application picks up those queries and displays the search results represented by the tokens.

![Facet Search++ Concept](image)

**Figure 1:** The Facet Search++ concept, allowing users to build boolean queries using tokens connected via streams. At the top is a token inside a stream.

2.2 Samsung SUR40 with Microsoft PixelSense

The application will be developed for the Samsung SUR40 with Microsoft PixelSense[4]. This decision was made for multiple reasons. For one, Facet Search++ was already built for the same platform and, while using the two applications, the user should be able to use both applications in a single interaction context to avoid confusion. By using the SUR40 the user always interacts with the same multitouch gestures and interaction concepts. Secondly, the Samsung SUR40 with Microsoft PixelSense is one of the few pieces of hardware that allows to use a modern, proven UI and tested multitouch interaction in a table-sized format. Alternatively, only a tablet-sized hardware would have been possible, but a bigger screen seemed more suitable for displaying large amounts of data, especially in a more-or-less stationary library context.

Let it be noted that the application was developed in a way that allows porting to other touch-compatible devices running the same Framework, for example Surface tablet devices[5].
2.3 WPF / C#

With the decision to develop for the SUR40 it was clear that the application would be developed in C# / WPF, since this is the language provided by Microsoft to develop for this platform. It comes with a multitouch-ready API that allows developers to take full advantage of the multitouch table. Since the SUR40 runs on a Windows core, using WPF makes it easy to port the application to other platforms, such as tablets or smartphones, which gives additional flexibility. Although the application might be used differently and some additional requirements or restrictions apply in other hardware environments, easy portability was still desired. For example, the application could be used in a mobile context, which makes it possible to search for documents and books while walking through the library. This, for example, could allow the user to find relevant real-world books, be guided to them or mix them with digital media easily.

WPF is not only a relatively new language, especially when it comes to multitouch support, but also introduces certain concepts that are very different from established languages like Java or C++. Noteworthy here is XAML, an XML-based language that is used to build the UI of an application, and the interaction between C# and XAML, which heavily depends on magic code running in the background that links both languages. These differences between “ordinary” object-oriented programming languages and the WPF framework made a long training period necessary before first usable results could be developed. Still, using WPF made the most sense when thinking about what is best for the enduser, so this training period was necessary to get a coherent, easy-to-use system.
3 Implementation

3.1 Theoretical Concepts

There are a number of theoretical interaction concepts that this system does make use of. The concepts and why they were used are shortly explained in the following section.

3.1.1 Multitouch interaction

Since, as described in Section 2 on page 3, the system is built for the Samsung SUR40 with Microsoft PixelSense\[4\], it naturally incorporates multitouch interaction. This means the system was built to specifically make use of tried-and-true multitouch gestures like pinching with two fingers, panning with one finger, double tapping and more.

3.1.2 2D landscape

The document space is represented by a two-dimensional landscape where objects are placed on, can be moved, have a mass and a spatial relationship to each other. Objects can be moved next to each other, on top of each other, grouped by spatial arrangement, discarded by moving them ‘out of the way’ and so on. This is to give the user the feel that these documents behave similar to real-world objects, for example a pile of paper. This is supposed to make object interaction as intuitive and the learning phase as short as possible when using this application, since the user can apply real-world knowledge to the virtual objects. ecl

3.1.3 Semantic Zoom

This system uses the ZOIL Framework\[9\] to make use of, amongst others, the landscape and semantic zooming capabilities it offers. ZOIL places views on a nearly infinite landscape and allows to zoom and pan that landscape freely. Semantic zooming gives content the possibility to change dependent on the space it occupies on the screen. So a view can show its content differently when it is fullscreen than when it is zoomed out to be only a few pixels in size. For example, a group of documents could be shown as only a number when they have little space, revealing how many documents are in that group. When the group is zoomed into, having more space available, it could change into a view where the authors in that group are displayed. Zoomed even further, single documents begin to appear.

3.1.4 Search

Since the application was built with portability, for example to mobile devices, in mind, a feature to replace Facet Search++ as a pre-filter step was needed. To do so, an application-internal search feature was developed. When this feature is active, the user is greeted with a very simple, screen-filling search box:
3 IMPLEMENTATION

Figure 2: The first screen of the application, featuring the search function as the only element on the screen.

Additionally, a search sidebar was developed that is available globally in the application via gestures. The sidebar features a search field and history of past searches:

Figure 3: The search sidebar, globally available via gestures, allows the user to enter search terms, look up his search history and add terms via ‘tap to search’.

Cross-metadata searching  We did say before that having a keyword-search has certain drawbacks, especially if the user is not exactly sure of what to look for. Nonetheless, in this scenario the user is greeted with a term-based search. Unfortunately, because of technical limitations it is simply not possible to let the user explore the entire data space - a pre-filtering was still needed. While this was a technical issue we had to deal with, we still wanted to offer the user a way of search that aids knowledge workers best. In practice, this means that every search in our application is automatically a cross-metadata search. That means that every available metadatum is searched for every single one of the entered search terms. If the user knows an author related to his research, he can enter the author name. If the user is looking for documents in a certain time period, a year can be typed in. The system offers support for different formats of time periods, so users can type in freely without worrying about formatting the query properly. For example ‘1999’, ‘1999 2000 2001’ or ‘1999-2008’ will all give the expected results. Obviously, document titles and abstracts can be searched through as well. And so on. These different types of searches can also
be combined seamlessly by typing in multiple terms. Additionally, not only document metadata (see Appendix A on page 31 for a complete list) is searched, but also the LDA-generated topics. We will go into the details of what the LDA does in Section 3.2.1.2, for now let it be said that the LDA calculates topics for every document in the search results. In practice, this means the user is able to enter a word related to the topic he is interested in and find relevant results. This gives the user an additional abstraction layer to search through, one that we think is much closer to the users’ mental model when they start searching.

**Tap to Search**  Because the internal search feature was developed with mobile devices in mind, an additional feature was added to ease mobile use of the search feature. The search sidebar features a button that activates the so-called ‘tap to search’ feature. While this button is held, another finger can tap almost any text in the application and it will be appended as a search term to the current search. For example, the user could tap the name of an author or a word describing a topic or a year. In a mobile context, typing is often difficult, especially when the user is moving. ‘Tap to search’ tries to make typing easier by allowing the user to search for anything of interest on the screen without ever having to type something in.

### 3.2 Architecture

This section will explain the technical architecture that was used in this application. While the general architecture of the system stayed the same throughout the course of this project, a lot of details changed heavily. At first, focus was on a design that should be as flexible as possible. This was simply due to requirements at that time, because the system was planned to incorporate a multitude of different visualizations and views. It became clear quickly that this magnitude of flexibility came with a price: A lot of different visualizations were possible, but each of them took a long time to implement. Since time was one of the most limiting factors and it became clearer that doing complex visualizations was not the main focus of the project, the requirements changed and with it a new kind of architecture was necessary that allowed for more rapid development. The new approach was less flexible and produced more repeated code, but allowed developing visualizations much more easily. In the following sections, some important parts of the final application architecture will be highlighted and described in more detail.

#### 3.2.1 Data Sources

**3.2.1.1 BaseX**  A BaseX database with 7305 documents was used as the data source for the application. Each document contained a multitude of metadata including the title, authors, number of pages, related documents and more. A detailed example of the XML tree of a single document can be found in Appendix A on page 31.

The data arrives in the application via BaseX database events. When a token is placed or removed from Facet Search++, a **ShowResults** or **RemoveResults** event is sent that includes the query at the tokens’ position and the ID of the token. From that query the documents to add or remove can be retrieved from the database.

One of the first important steps in the creation of the application was to store the currently displayed documents in an efficient and convenient way. It was also important that the connection between a document and the token(s) representing that document was kept, and that storing redundant data is reduced to a minimum. To solve this problem, the first step was to introduce the *MultiValueDictionary*. The *MultiValueDictionary* is a subclass of an ordinary dictionary that can efficiently store multiple values for a single key. That makes this dictionary behave differently from a normal dictionary in a number of ways:

- When adding a key-value-pair with a key that is already in the dictionary, the new value is added to the list of values already stored for that key. No new entry in the dictionary is created.
3 IMPLEMENTATION

- When retrieving the values for a key, a list of values is returned.
- When removing a key from the dictionary, all values are removed and the entry is deleted from the dictionary.
- It is possible to remove only one of multiple values for a key. The key and all other values will remain.
- If the last value of a key is removed, the entire entry is removed from the dictionary. Empty value lists are not possible.

Additionally, it was made sure the `MultiValueDictionary` is thread-safe, making it easy to be used in a multi-threaded environment. The `MultiValueDictionary` class can then be used to store a list of documents with the token(s) containing them. To do so, a `MultiValueDictionary<Document, Token>` is instantiated. The result is a dictionary with documents as keys and the list of tokens that contain the document as the values.

Finally, it is sometimes necessary to store a so-called ‘Document Group’. This is basically the same dictionary we just described, but named. For example, a list of documents belonging to a certain author will be saved as a named `MultiValueDictionary` with the name of the author as the group’s name. Since this is a very commonly used structure a distinct class was created for this purpose called `TokenizedDocumentGroup`, which simply combines together a `MultiValueDictionary` and a name:

\[\text{Figure 4: UML Diagram of the TokenizedDocumentGroup and its’ dependencies}\]

3.2.1.2 LDA  We used MALLET’s implementation of the LDA algorithm\[^3\] to help us determine the topic affiliation of the documents. For that, the algorithm takes an abstract of each document. Full texts would have been possible, but were not available for all documents. Two things are then calculated from the abstracts:

- Topics: It calculates a fixed number of topics and each topic consists of a fixed number of ranked words. So a topic could be made out of the words ‘interaction, user, interface, usability, human, psychology’ in that order, which would mean that ‘interaction’ is the most important word, then ‘user’ and so on. The topic is not named but only defined by the words it contains - but a human is still able to conclude that the above topic is, for example, probably related around human-computer interaction.

- Document-topic relationships: The algorithm calculates the affiliation probability of each document to each topic, whereas all probabilities for a single document add up to exactly 1.0. In other words, imagine a pie chart being built for each single document, whereas the individual slices of the chart represent topics. If the words of a topic occur often in a document, the topics’ slice for that document gets bigger and the affiliation is considered higher. But this also means that other topics’ slices will get smaller, since every document must have exactly one full pie.
The algorithm returns the results in textual form and a way was needed to easily access them in the application. One such possibility would be to run the algorithm for the entire data space and store the results in the BaseX database as metadata for each document. This seemed unfavourable, though, because the database is an interchangeable module of the application. The BaseX database could be exchanged any time for, say, an external online document catalogue like Microsoft’s Academic database[6]. Besides the fact that performing the LDA for the entire Microsoft document database is impossible and that those databases are subject to frequent changes, storing the results in the external database would not be possible in this case. For this reason, a solution was needed where the LDA algorithm could be run on-demand only for the data that was actually displayed and with the possibility to locally store and work with the results, regardless of the data backend used. For that reason, the algorithm is run locally, on-demand and the textual output is then parsed into objects. The algorithm outputs its results in the following format:

```
Document {ID} {PROB 1} {PROB 2} {PROB 3} ... {PROB n}
Document {ID} {PROB 1} {PROB 2} {PROB 3} ... {PROB n}
... 
Document {ID} {PROB 1} {PROB 2} {PROB 3} ... {PROB n}
TOPIC: {WORD 1}, {WORD 2}, {WORD 3} ... {WORD m}
TOPIC: {WORD 1}, {WORD 2}, {WORD 3} ... {WORD m}
... 
TOPIC: {WORD 1}, {WORD 2}, {WORD 3} ... {WORD m}
```

n topics are determined and each topic is defined by m words. n and m are free parameters, so reasonable values must be chosen - n = m = 100 should be suitable for most cases. When the database grows larger, calculations to determine both values might be needed, but not so for the database used here. For each document, n probabilities are returned, one for each topic. These determine how much the document relates to each topic and sum up to 1. It is easy to see that working with this kind of structure in code is not very desired and that the LDA results need to be processed further to make them easily accessible during runtime. For this reason, a parser was written that processes the text file into objects:

- **Topic** objects, where each represents a single topic and contains a sorted list of the m words that define that topic and

- **DocumentTopics** objects, where each represents a document and contains the document ID together with a Topic-Probability dictionary. By using **Topic** objects in this dictionary, it is very easy to get from a document to it’s most important topic and from there to the words defining that topic.

With this kind of storage, the LDA results can be easily accessed and worked with during runtime. It is easily possible to find all topics for a certain document and vice versa, the most important topics for a document, the topics with the most documents and so on.

### 3.2.2 SingleExecutionMethodWorker

![Figure 5: UML diagram of the SingleExecutionMethodWorker class](image-url)
Another important prerequisite for a smoothly running application was the \textit{SingleExecutionMethodWorker}. There are several methods throughout the application, including the repaint of the document space, that may only run once at any given time, even in a multithreaded environment. Usually, this problem is solved by using one of the programming languages’ dedicated mechanisms for thread-safe coding, like locks or mutexes. In our case, the requirements were more precise. For example, calling the repaint method while another instance of the repaint method is already running invalidates the currently running instance - because it will be painted over anyways.

To solve this, the \textit{SingleExecutionMethodWorker} was introduced, which wraps around the multithreaded \textit{BackgroundWorker} class of WPF. The method that shall be executed is passed to the Worker as a delegate. Instead of calling the method directly, classes now call the \texttt{run()} method of the Worker. The Worker will then try to stop any currently running instance of the method. Only after the current execution was cancelled or finished, the Worker will call the method again with the most recent data. Any additional calls made in the meantime are ignored, only the latest call is then executed. This not only ensures that only one instance of the method can be run at any given time, but also that no unnecessary work is done.

### 3.2.3 GlobalSearchHelper

![Figure 6: UML diagram of the GlobalSearchHelper class](image)

For situations where the Facet Search++ application is not available, a sidebar was introduced that allows the user to search either by typing in terms or by using ‘tap to search’ as described in Section 3.1.4 on page 7. All components accessing this global search, including the search bar view itself, use the \texttt{GlobalSearchHelper} class to do so. This class is a singleton class, so all components access the same instance of the \texttt{GlobalSearchHelper}. The Helper stores the current search string, past searches and determines if tap to search is active or not. The \texttt{search()} method builds a query that searches for every term of the search string amongst all available metadata plus topics as described in Section 3.1.4 on page 6. If something changes in the \texttt{GlobalSearchHelper}, it is communicated to the outside via an Observer pattern. This way, any arbitrary class in the application can simply register to be notified of any changes in this class. The \texttt{GlobalSearchHelper} is a very simple, easy way for all component of the application to work on the same search data, be able to access any search-related data and be notified of changes to them. For example:

- The main view listens for search queries built by the \texttt{GlobalSearchHelper}, executes those queries and displays the results. This behaviour replaces receiving database events from Facet Search++ as described in Section 3.2.1 on page 7.

- When a label displaying an author is tapped, the label will check if ‘tap to search’ is active using the getter of \texttt{tapToSearchActive}. If so, it will add itself as a search term using \texttt{addSearchTerm()}.  

### 3.2.4 Main View

Displaying the main document landscape is the core of the application and proved difficult to implement. It underwent multiple iterations with implementation of different features that became
more or less important during the course of the project. The final architecture of the main application part is as follows:

![UML diagram of the central parts of the system. The central component is the Main-View, which derives from ZOIL’s ZComponent and displays one of multiple possible visualizations in which the current data is visualized.](image)

**Figure 7**: UML diagram of the central parts of the system. The central component is the *Main-View*, which derives from ZOIL’s *ZComponent* and displays one of multiple possible visualizations in which the current data is visualized.
Please note that this is a highly simplified UML diagram, designed to make the reader understand the basic principles of the main view module. It has several important parts:

- **The MainView**, which is basically a wrapper for the actual Visualization. The MainView receives database events and makes sure the displayed documents are correct and the Visualization is repainted when needed. It also handles certain (mostly global) events and interactions, such as the search sidebar mentioned in Section 3.2.3 on page 10.

- **The BaseVisualization**, an abstract superclass for all visualizations. It allows the actual document visualization to be interchanged without changing any of the other components of the application. Right now, there is only one main visualization (TimelineVisualization), but this very flexible system allows for changing the entire visualization based on arbitrary circumstances, such as user interaction, number of documents, kind of data to display and so on. The visualizations could also be changed back and forth ‘on the fly’ during runtime without any problems. Visualizations could even be pre-loaded in the background and then displayed when ready. In earlier versions of the application, this was made even more flexible and reusable by further splitting the visualizations themselves into multiple modules and reusing the individual parts. Since implementing visualizations this way was very inconvenient and slow, though, the system was changed for the slightly less clean, but much more efficient solution used right now.

- **The NoResultsVisualization** and **EmptyVisualizations** are basically visualizations displaying no data, that can be shown by the main view whenever seen fit. Currently, the EmptyVisualization is shown when the application first loads and no data is yet displayed, and the NoResultsVisualization is shown when a user-initiated search returns empty. Both views display a string, telling the user that there is no data.

- **The TimelineVisualization, TimelineTopic and TimelineDocumentGroupView** form the main visualization of the application. TimelineVisualization splits the documents into authors, and assigns one or more topics to each author. It then creates a TimelineTopic for each topic and passes the calculated authors to that topic. The TimelineTopic offers two semantic zoom levels: Zoomed out, it displays a word cloud with the words of the topic it represents. Zoomed in, it displays a timeline-based view with author names. Each author is a TimelineDocumentGroupView, again with multiple semantic zoom levels: The authors’ name on the outermost, and a list of author documents when zoomed in. The detailed process will be described in Section 3.3.

### 3.3 The Timeline Visualization

This section will describe the current main visualization, called the *Timeline Visualization*. We will go into detail about the visualization’s goals and possible alternatives or additions to the current state of the visualization.

The visualization had to fulfill certain requirements, with which in mind it was created:

- **Exploration.** The user should not be forced to search list upon list of text, going through tables of metadata until a suitable result is found. Instead, a natural flow should guide the user through the data space. The user should be able to explore the space without feeling like exploring ‘a database’.

- **Relations.** It should be possible to find relations between parts of the data space and finding work related to documents that are already known to be relevant.

- **Topics.** The LDA algorithm described in Section 3.2.1 on page 7 allows us to show the user the topics a document belongs to. Searching by topic seems to be a method supporting the natural workflow of a knowledge worker who knows the general topic he wants to know
about but not exactly what to look for. Therefore, making use of topics seems important to help the user narrow down the document space and find relevant results.

- **Understanding.** The presented visualization should be easy to understand. It should be clear where in the dataspace the user is, how he got there and how to get back.

- **Serendipity.** Related to exploration, the user should be able to find documents related to what he is looking for without specifically searching for them.

- **Responsive.** The system should be quick, responsive and follow UI guidelines to make the user feel ‘home’.

During the course of this project, the visualization and with it the requirements we just named, changed a lot. The current timeline visualization tries to satisfy the final requirements as good as possible. Still, the system was built in mind with the possibility to expand and add visualizations. The timeline visualization is a ZOIL-based landscape with different semantic zoom levels and certain interaction possibilities you would expect from this kind of landscape, like pinching, panning, moving object and so on. It follows Ben Shneidermanns’ ‘Overview first, zoom and filter, then details on demand’-pattern\[8\] by allowing the user to get from a general, topic-based overview down to individual documents and information about them. On the outermost level, the visualization displays the most important topics of the current search results as it can be seen in Figure 8.

![Figure 8: Clouds for four different topics. Word size indicates the importance of a word for its topic.](image)

If zoomed in far enough - either by pinching with two fingers or tapping on a topic - the topic clouds transform into a more detailed information view about authors that wrote documents related to that topic. A timeline is shown and the most important authors of that topic are displayed on that timeline, showing from when to when they were active in that topic as shown in Figure 9 on the next page.
Figure 9: TimelineView showing the most important authors for the ‘information, visual, user, interaction, ...’ topic seen in Figure 8 on the previous page. The closer to the timeline an author is, the more likely it is to be important. Width of the author determines from when to when the author wrote documents in that topic - for example it can be seen that ‘Frank Müller’ started writing in this topic in 2000 and wrote the last document in 2006.

Each author is again a zoomable instance, zooming further in reveals information about the authors, going down to document level. Documents of the author, related documents and related authors are displayed. Additionally, documents can be explored by viewing their abstract and related documents. An example for the detailed author view can be seen in Figure 10.

Figure 10: The author detail view reached by tapping ‘Limbach, Tobias’ in the timeline in Figure 9, displaying documents of an author, related authors and related documents. Individual documents can be explored further by tapping them.

This text will now go into more detail about how the individual zoom levels are created, calculated, which components are responsible for drawing and what difficulties arose.

3.3.1 Topic Cloud

The first zoom level is the topic overview. Here the user gets an overview over the general areas of expertise the search result covers. It is possible for the user to move the clouds, which, for example, allows him to engage in a first, broad sensemaking step by arranging topics that have
something in common together. Two classes are taking part in creating the cloud: First, the `TimelineVisualization`, which is the parent view of all topics and is responsible for grouping documents into topic stacks. Secondly, the `TimelineTopic`, each representing and displaying a single topic.

The `TimelineVisualization` takes all documents in the search result and creates multiple stacks out of them, one for each topic that is to be displayed. It is clear that there are a number of factors deciding which documents are assigned to which topic and which topics should even show up on the screen. Developing the algorithm proved tricky, and an abstract-as-possible pseudo-code will be given and explained in the following:

```
// Input: We start with groups, one group represents all documents of one author
set groups to documents grouped by author

// Calculate topic probabilities for each group by simply averaging all documents in the group
for each group:
  set topicProbs to new array
  for each document in group:
    for each topic of document:
      add topic probability to topicProbs[topic]
  endfor
endfor

// get each topic probability with a simple rule of three
// value: topicProbs[topic], minimum: 0, maximum: number of documents in group
for each key in topicProbs:
  set finalProbs[key] to topicProbs[key]/number of documents in group
endfor

save finalProbs in group

// Assign each group to one or more topics
// A group must be affiliated at least MIN_AFFILIATION percent with a topic to get assigned
// A group can be in a maximum of MAX_TOPICS topics
set topicsWithGroups to new MultiValueDictionary
for each group:
  set groupTopics to new array
  for each topic of the group: // calculated above
    if topicProb >= MIN_AFFILIATION:
      add topic to groupTopics
  endif
endfor

sort groupTopics by probability
set relevantTopics to the first MAX_TOPICS topics of sorted groupTopics
for each topic in relevantTopics:
  add group to topicsWithGroups[topic]
endfor

// A topic must have at least MIN_GROUPS_PER_TOPIC groups to be considered worth showing
create a TimelineTopic for each topic with more than MIN_GROUPS_PER_TOPIC groups
```

It should have become clear by now that there are a number of factors to consider. For example, it is important we don’t hide important information from the user but at the same time don’t clutter the screen with noise that doesn’t add a lot of information. Topics should be correct and useful and groups inside that topic should contain enough documents that actually belong to the topic in order for the user to make sense to explore.

In order to achieve that, we perform multiple steps. First, documents are grouped by authors. If a document has multiple authors it is assigned to all of them. Since we want the user to see correct, useful results, we need to make sure an author wrote a certain amount of documents in a topic for him to show up. Each author’s topic affiliations are calculated by using the average topic affiliations of all documents of that author.

In the next step we determine the topics each author is most affiliated with. It is important
to note that, to prevent noise, a few requirements must be fulfilled by each author-topic relationship. For example the author must be affiliated enough with that topic (expressed by the MIN\_AFFILIATION threshold), and an author can only belong to a certain amount of topics (expressed by the MAX\_TOPICS threshold). In the end, each author belongs to something between zero to MAX\_TOPICS topics, which could be called his ‘areas of expertise’.

After authors are assigned to topics, the algorithm determines the most important topics - which are the ones containing the most search results/the most authors. As another noise prevention method, a topic must have a certain amount of data to be displayed - it must have MIN\_GROUPS\_PER\_TOPIC groups/authors in it. In a visualization sense, topics not fulfilling that requirement could be considered outliers of the search results, so we basically detect and remove outliers from the result set. Each topic that fulfils the requirements is then displayed on the screen by creating a TimelineTopic object for it.

The TimelineTopic class is then responsible for displaying the actual topic cloud. This is done by a most simple algorithm:

```plaintext
set i to 0
for each word of topic
  set fontSize to MAX\_FONT\_SIZE - i
  if fontSize < MIN\_FONT\_SIZE then
    set fontSize to MIN\_FONT\_SIZE
  endif
  display word with fontSize i
  increment i
endfor
```

Since words are sorted by importance for each topic, the first word is displayed biggest, and font size is then decreased linearly until a certain minimum threshold is reached. Combined with correct text wrapping, this will create a decent word cloud representing the topic, with the most important words being easily spottable:

![Figure 11: A single topic cloud as the outcome of the above algorithm. More important words are displayed bigger and word size decreases linearly until reaching a set minimum.](image)

Of course, the constant parameters determine how good or bad those algorithms perform. In the final application, the following values were used:

- \( MIN\_AFFILIATION = 0.33 \)
- \( MAX\_TOPICS = \infty \)
- \( MIN\_GROUPS\_PER\_TOPIC = 5 \)
- \( MAX\_FONT\_SIZE = 35 \)
- \( MIN\_FONT\_SIZE = 10 \)

The MAX\_TOPICS constant is set to ‘no limit’ because in our case the MIN\_AFFILIATION threshold was enough to eliminate unimportant groups and reduce noise. The final result are topics containing a number of relevant results, with authors assigned to them that actually did
contribute to that topic. When the user then decides to explore a topic, relevant results are displayed as we will see in the following.

### 3.3.2 Timeline

The second zoom level is the timeline and is presented to the user when he decides to explore one or more topic clouds in more detail. It gives the user an overview over the most influential authors of a topic and the period of time they worked in that field. The timelines, like the clouds, can be moved freely in space and therefore be arranged spatially and even compared to each other if necessary. The main component responsible for displaying the timeline is the `TimelineTopic` class. Each author is represented by a `TimelineDocumentGroupView` object. When displaying the timeline of a topic there are, again, certain requirements we wanted to be fulfilled:

- The timeline should be comparable to timelines of other topics
- Authors should be placed correctly in regard to the time axis, reaching from the year of their first publication to the year of their last publication in that topic
- Authors should not overlap
- The y-axis should have a meaning: We decided to rate the importance of authors for the topic and place the more important authors further at the top. This has the additional benefit that we can limit the number of authors to show and still be sure that the important authors are displayed

The last requirement made it necessary to create an importance rating for authors, but actually we already implicitly did this in the previous step when creating the topic clouds. In Section 3.3.1 on page 15 we calculated the affiliation of each author with a topic. We can use that affiliation as the importance rating. The higher the affiliation, the more important the author. The authors need then to be placed on the screen, abiding the given requirements. To do this, the following algorithm was developed:

```java
// Input: globalEarliest and globalLatest are passed to TimelineTopic from the TimelineVisualization
set globalEarliest to the earliest year any author in the search result wrote something
set globalLatest to the latest year any author in the search result wrote something
set sortedGroups to groups sorted by affiliation to current topic
for each group in sortedGroups:
    set earliest to earliest year the group author wrote something
    set latest to latest year the group author wrote something
    set y to 0
    // This author will be drawn from earliest to latest on the timeline
    // Determine the highest y position where other authors have been drawn already in that time period
    // To prevent overlapping, we need to draw higher than that y position
    for each year from earliest to latest:
        if highestY[year] exists and highestY[year] > y then
            y = highestY[year]
        endif
    endfor
    set x to (earliest - globalEarliest) * YEAR_WIDTH
    draw group at x, y
    add GROUP_HEIGHT to y
    for each year from earliest to latest:
        highestY[year] = y
    endfor
endfor
```

To prevent overlap the algorithm remembers the highest y position already drawn at each year. If an author will be drawn at that year, a higher y position will be used to prevent overlapping.
Of course, this algorithm does have certain drawbacks. Space can be wasted: Assume five authors are drawn that only wrote publications in 1999, occupying “slots” (lines) 1-5 for that year. A sixth author wrote something in 1998, 1999 and 2000 and will be drawn at the 6th line. Then an author wrote only publications in the year 2000 and later. Although the slots 1-5 are unused for those years, the algorithm will detect an author at slot 6 for the year 2000 and therefore draw at slot 7. Another obvious flaw is that the y position of authors is not directly tied to their influence: Authors with higher importance are much more likely to be at the top of the visualization, but not guaranteed to.

Nonetheless, the algorithm performs quite well in practice and is quick and suitable for the task at hand. The final result will be a timeline where authors are placed correctly on the time axis and authors at the top are more likely to be relevant to the topic than authors at the bottom, as seen in Figure 12.

![Figure 12](image.png)

**Figure 12:** The timeline view for the topic shown in Figure 11 on page 16, displaying authors on a time axis. The further on top an author, the more likely he is more important for a topic.

A problem remaining was that the current view basically only tells the user when an author first wrote something and when he or she last wrote something. It is unknown to the user if the author was very active in that field and if the author even wrote anything in the years between. To solve this problem, a linechart representing the amount of documents written per year was added to the timeline visualization for each author. The linecharts are normalized globally. This has the distinct advantage that the charts become much more comparable, because every linechart can be compared to every other linechart, even those from other topics. The final timeline visualization can be seen in Figure 13 on the next page.
Figure 13: The same timeline view as in Figure 12 on the previous page, but featuring additional linecharts for each author, visualizing the amount of documents wrote by the author each year.

3.3.3 Author Details

The third zoom level is the author detail level. After exploring a topic, the user now has the chance to zoom into an author, revealing the work of that author. This includes documents in that topic, all documents of the author, documents related to the author’s documents and popular co-authors, as shown in Figure 14.

Figure 14: The author detail view after zooming into ‘Limbach, Tobias’ in the timeline in Figure 13, revealing the work of that author to the user.

Tapping a document title reveals information about the document as seen in Figure 15 on the next page. This transition does not, like the other transitions so far, use the semantic zoom paradigm. This decision was made for multiple reasons: First of all, we wanted to limit the depth of semantic zoom operations, because going in ‘too deep’ was found to be confusing. Secondly, we wanted to stay in the context of the author detail view, showing name and token distribution of the author. If a document is tapped, the view simply switches to the details of the document, and a ‘back’-button allows the user to go back. The document details include the title, authors, abstracts and related documents to help the user find other relevant documents.
Figure 15: The details of a single document showing the authors, title, abstract and related documents.

In the end, the user went from a very general topic overview down to document content level fluidly, without switching contexts and the zoomable landscape allows the user to go back and forth between different detail levels at any time.

### 3.3.4 Token Distribution Circle

In the author detail view in Figure 14 on the preceding page we saw the Token Distribution Circle in the top left. This visualization was developed specifically for this system and makes it possible to assess the token-membership for sets of documents. In this case, the documents of an author are assessed. The visualization allows the user to determine the amount of documents in different results sets and the amount of overlap between those result sets quickly and without taking much space. It was originally developed when the focus of the application was a lot more on comparing the different tokens placed on the Facet Search++ streams. Back then, it was used a lot more amongst different visualizations of the system. Now, it is just a small part of one of the zoom levels, but can still be very useful. For example, if different users place different tokens on Facet Search++, this visualization can help assessing where the ideas and mental models of users overlap and where they differ.

The token distribution circle tries to solve the problem of displaying differences and similarities of up to three result sets. Additionally, the visualization was supposed to fulfil the following requirements:

- It should be easy to understand
- It must be comparable. It is more important that multiple instances of this visualization can be compared to each other easily than to get exact numbers out of it.
- It needs to be space-saving since it will be embedded into larger visualizations. The target size it should still work well on was 50 by 50 pixels.
- It should work for one, two or three result sets
- The intersection of any two result sets must be visible for any combination of two sets
- The intersection of all three result sets must be visible

This proved more difficult than originally thought and required to develop a very individual visualization. It became clear quickly that a circle-like shape seemed the most fitting for this task:
When creating an ordinary pie chart with one segment for each token, there naturally exists a border for any combination of two tokens, so it seemed this could be used to display those overlaps.

After some testing, the final result was the Token Distribution Circle, which is, very basically described, a modified pie chart. First, each segment represents one result set - to get back to our application, each segment represents the result set of the token of that color. The bigger that segment, the more documents are in that result set. So a first glance at the circle in Figure 16 reveals that the green token represents only a few documents while the other two tokens represent the majority of the documents. The overlap between two result sets is represented by how much a segment ‘bleeds’ into another one. The border between green and red is ‘hard’ or ‘sharp’, there seems to be no overlap. On the other hand, the border between yellow and green is not that clean, yellow ‘bleeds’ into green, they have a lot of documents in common. The border between yellow and red bleeds very little, they have only a few documents in common.

Lastly, the intersection of all three result sets is represented by the the purple circle in the center. The bigger that circle, the more documents are in all three result sets. If the circle is not visible, there is no intersection. If the entire visualization is purple all documents are in all three token result sets and therefore no individual token colors are shown. That also means that the purple circle can exceed the ‘empty’ (black) space in the middle.

How is this circle created? First of all, additional information needs to be calculated. Internally, the following geometrics are needed and calculated with standard geometric formulas:

There are a total of four circles: The purple all-tokens circle. The black, empty inner circle. The normal circle with radius \( r \) that makes up the token segments. And finally an outer (orange) circle, that is used during the drawing process but clipped away afterwards. Since we clip the visualization to the form of the normal circle, it will have the size of the shown (turquoise) rectangle, which is \( 2r \times 2r \). The drawing will be performed by the following algorithm, which will be explained in detail further below:
The basic principle of the algorithm is easy: One segment for each token is drawn and the size of the segment is determined by the number of documents in that token's result set. So, if the documents of a token make up 50% of the entire documents, the segment will make up 180° of the circle. Similar, the size of the purple circle is made up by the fraction of documents all tokens have in common. If every single document is in all three result sets the radius of the purple circle will be \( r = 1.0 = r \), so it will fill the entire visualization.

Next we want to achieve the ‘bleeding’ effect as a visual cue of the overlap of two tokens. To do so, we ‘walk’ further on the outer and normal circle than on the inner circle - exactly the fraction of overlapping documents between the two adjacent tokens. So if the current token has 50% of the documents to itself, and shares 10% with the next token, we will walk 180° on the inner, and \( 360\times(0.5+0.1) = 216° \) on the outer and normal circle. This achieves a segment end that will ‘reach’ or ‘bleed’ over into the next segment, representing the amount of intersection between the two segments. The following image shows the circle ends calculated for the yellow segment:
3 IMPLEMENTATION

Figure 18: Calculated circle ends of the inner, normal and (invisible) outer circle for the yellow segment.

which leads to the following area of intersection between the two segments:

Figure 19: The area of intersection between the yellow and green segment.

Basically, this is everything needed. A polygon is then created between the end of the last segment and the ends just calculated. The polygon is then filled with the segment color and clipped with the shape of the normal circle. There remains a problem where artefacts occur for segments of over $90^\circ$ in size. This is because for those segments, the created polygon lies completely within the normal circle, therefore not filling it out. This is solved by introducing the outer circle ‘checkpoints’. Every $90^\circ$, a ‘checkpoint’ is created on the outer circle that is also used as one more edge of the polygon. This way, we make sure the visualization is drawn correctly without artefacts.

3.4 Personal landscape

To support sensemaking and the process of materializing the mental model further, a personal landscape was added that allows the user to drop interesting parts of the search results on it and then arrange and annotate them.

Currently, topics and authors can be put on the personal landscape. Unfortunately, due to time and technical restrictions, documents cannot be put on the personal landscape yet. Putting objects onto the personal landscape is done with the so-called ‘personal drop zone’. When a personalizable object is dragged, the drop zone appears at the bottom of the screen. If the object is dragged onto that zone and released, the object is copied into the personal landscape. The use of the personal drop zone is shown in Figure 20 and Figure 21. Once onto the landscape, the user has the possibility to spatially arrange and resize objects to, for example, group objects that belong together and size them by importance or relevance.
Figure 20: “Deussen, Oliver” is dragged so the personal drop zone appears at the bottom of the screen. Since the dragged author is not above the zone, the zone appears inactive.

Figure 21: “Deussen, Oliver” was dragged above the personal drop zone, therefore transforming it into active state. If the author is released now, it will be added to the personal landscape.

Due to time restrictions, the personal landscape was not finished in its entirety during the course of the project. To further support the sensemaking process, an extended personal landscape is currently still in development, which features additional possibilities. Most importantly, it features the possibilities to annotate, putting a canvas on the personal landscape where the user can then draw or write freely to take down his thoughts about the landscape. Secondly, the possibility to link objects on the landscape is currently in development. With this, the user can explicitly mark connections between objects that he thinks are in some way related to each other. Combining all those features, we hope to create a personal landscape where the user can take down his view on the search results in a mind-map kind of way. We hope that in the end, users will be able to create landscapes similar to that shown in Figure 22 on the following page.
3.5 Challenges & Lessons Learned

A project of this scale naturally comes with a multitude of challenges and problems. What may have proven the hardest in this particular case was narrowing down all the possibilities and choices into something streamlined, focused. A lot of features were implemented in the course of this project that were not used in the final system because the requirements to the application changed in a matter that made them obsolete. This unnecessary work cost a lot of resources that could have been used on other parts of the application. This could probably have been prevented by further refining requirements and painting a clearer picture of what the final application should do and not do before starting implementation.

Nonetheless, there are still additional concepts that could be explored and implemented in this application, but with very limited time only so much could be achieved.

Besides some conceptional problems during the project, there obviously were a number of technical challenges as well. A long training period was necessary to get familiar with some of the very different concepts of C#, WPF and XAML. Since I had no prior experience working with those languages and they were pretty different from other ‘famous’ object-oriented languages, the time it took until the first acceptable results could be produced span multiple months of training.

And even after that initial training period, I kept learning a lot of new concepts which sometimes forced me to change existing code, architectures and concepts.

To get a little more specific, a very annoying problem in particular was with events in WPF. On the one hand there were lots of events that conflicted each other. For example, there were
times when a view was supposed to collapse when you tap it but at the same time there were buttons in it that needed to be tap-able without the view collapsing. The same view is also drag- & drop-able, so tapping it to initiate a drag should not collapse it. Additionally, parent views might have their own event handling that could cause conflicts as well. A lot of care had to be taken to make sure events and interactions don’t conflict each other or have unforeseen side effects. This was made even worse because sometimes, handling or not handling events can have very strange side effects in the WPF API. For example, it sometimes happened that receiving a TouchDown event, but not marking it as ‘handled’, caused the TouchUp event to get lost - even though those two should not be related. On another occasion, it was impossible to receive pinching and panning gestures inside a scrollable view. The event chain should actually take care of this - delivering those events to my custom view first, and only if the custom view does not handle them the events are then given to the scrollable view. WPF did not do this, detecting events inside a scrollable view often remained unstable to the very end of the project. May those be bugs, side effects or for some reason intentional behaviour, bugs and problems like that occurred much too often in the WPF API and took a long time to be found and fixed, especially for someone who was not familiar with WPF.

Successfully including ZOIL proved problematic as well. Although getting it up and running was very quick and flawless, and the advantages of ZOIL became clear quickly, there were numerous problems with the already existing views I had created. In particular, ZOIL expects all its’ components to have a fixed size, but since content was created dynamically this sometimes proved difficult to assure. This often caused content to be shown in the wrong sizes or views could not be viewn in their entirety. Even now, there are still some of those problems left. For example, the size of text in the author details is dependent on the width of the TimelineTopic view. Forcing the same text size on all author details seems simply not possible. Additionally, semantic zoom only triggers when a view reaches a certain width on the screen. Doing the same with height is not possible. This proved problematic for TimelineTopic views with a very large width - text in the author detail view becomes very small in those views.
4 Future Work

4.1 Implementation

Since the subject at hand is large and time was the most limiting factor, things remain that could still be done, improved, explored or changed.

4.1.1 Timeline visualization

Author sorting The sorting algorithm for authors could be improved. As described earlier, there are still some problems with this algorithm: More relevant authors are not guaranteed to be displayed above less relevant ones and space is not necessarily used efficiently. Large empty gaps can occur. Further improvements of this algorithm could help display more authors on the same space and make their ordering more clear. Additionally, other visual variables like color have not yet been used and could be integrated to display additional information without using up additional space.

User interaction There are still some problems with user interaction. Event handling is not always as smooth as could be and there are still some display and sizing problems with some ZOIL landscapes. With more time this could surely be fixed.

Additional visualizations A time consuming but helpful effort could be to implement additional visualizations and display them when appropriate, or let the user chose when to display which visualization. This could help the user in exploring the data space in different ways and with different goals or requirements in mind. For example, right now it is easy to get from a general topic to relevant documents of that topic. There could be an alternative visualization that displays a topic-related author network, allowing the user to get from a well-known author to others that wrote relevant material. Since this application has been built with the ability to easily add, display and switch between different visualizations, adding these requires no architectural changes.

4.1.2 Personal landscape

The personal landscape currently implemented is very rudimentary and only a ‘proof of concept’ if you will. Of course, a lot of features could still be added to the personal landscape, for example further annotations, creating of stacks, additional views on the data, sharing and exporting and so on.

4.1.3 Backtracking

It would be desirable to make it possible for the user to backtrack his work. This means that the user should be able to re-view past search results and explore them. Compare search results, see which result might have triggered new searches and so on.

4.1.4 Cooperative work

The topic of social and cooperative work was not explored during this bachelor project. This topic could open up a whole new part to the sensemaking process, giving partly or completely different requirements to the system but also allowing us to aid the user in another very important part of his daily work.

4.2 User Study

We will perform a user study and try to find out if our efforts to improve searching and sensemaking in the Blended Library succeeded. Doing a comparative study was found to be difficult, as there
are no comparative systems that can be used scientifically in such a study. Instead, we will perform an open study, giving the participants one or more open exercises and let them freely work with the system. Possible exercises could include finding information about an author - for example his field of expertise, popular co-authors, when the author was at the peak of his work and so on. Another possible exercise could be to give the participant a term and take a look at what interesting information the participant finds out - or, in other words, what mental model of the term the user creates using the system. This and other similar exercises could be used. To evaluate the study, screenshots of the final results, videos of the way the user works with the system, which path he is taking through the search results and a post-study-questionnaire are thinkable.
5 Conclusion

In conclusion, we created an information retrieval system that tries to achieve multiple things. It is made specifically for knowledge workers and takes into account that very often, those researchers don’t have a specific idea of what the outcome of their searches will look like. We tried to provide a different approach to search, not forcing the user to search by strict metadata and then digging through tables of textual results, basically picking results on the off chance. By tying the system to Facet Search++ or alternatively providing our own search helper, the user can search by traditional metadata or topic. The results are then presented in a way we believe fits the work of knowledge workers - by topic, ready to be explored in a two-dimensional space that allows the user to encode implicit knowledge in spatial relationships and size of objects. We tried to allow the user to assess the relevance of documents even for topics he has no idea about, by first grouping documents by topic and then assessing the importance of authors for those topics. The user can use multitouch technology to navigate through the document space, allowing him to get from very general topic clouds down to individual documents and document abstracts fluidly. Documents are linked to other documents by showing related authors and documents, making it possible to get from one interesting document to another. By displaying authors on a timeline the user can also assess the relevance of certain topics or authors over time. When was the topic first discovered? Did the relevance increase or decrease over time? Are there certain groups of people who were very active in a topic recently? We hope those and similar questions can be answered easily.

Obviously, with more time there is still more that could have been done. More visualizations, the ability to change them on the fly and additional ways to search the documents space could be helpful to the user. The application was built with expansion in mind, so the foundation to do those things exists.

On a personal note, I also learned a lot about what to do and what not to do in such a project. Even though the planning phase was already extensive, it became quite clear that it should have been longer. A lot of resources were wasted on features that never made it into the final application - not because they were badly done, but because they didn’t fit the final concept and requirements. A longer conceptual phase could have prevented a lot of that, working out every detail and especially the goals of the application beforehand is vital. This is definitely a lesson I learned and that I will pay much more attention to next time.
Appendices
Appendix A  Example XML-Structure of a document

The following listing contains an example document ("Medium") as it occurs in the BaseX database. The related documents were shortened. The Page-tags inside the Fulltext-tag contain the content of the document if it is available.

```xml
<Medium mv_id="1317362" bib_id="260125822">
   <Title>Related</Title>
   <Rdium>
      <Id>281148201</Id>
      <Value>0.28867513</Value>
   </Rdium>
   <Rdium>
      <Id>280675496</Id>
      <Value>0.25</Value>
   </Rdium>
   <Rdium>
      <Id>280366647</Id>
      <Value>0.2236068</Value>
   </Rdium>
   <Rdium>
      <Id>27171753</Id>
      <Value>0.20412414</Value>
   </Rdium>
   <Rdium>
      <Id>286813823</Id>
      <Value>0.18898225</Value>
   </Rdium>
   <Rdium>
      <Id>279220731</Id>
      <Value>0.25</Value>
   </Rdium>
   <Rdium>
      <Id>25985560</Id>
      <Value>0.0</Value>
   </Rdium>
   <Rdium>
      <Id>265622433</Id>
      <Value>0.26358542</Value>
   </Rdium>
   <Rdium>
      <Id>266486886</Id>
      <Value>0.2550179</Value>
   </Rdium>
   <Rdium>
      <Id>266575341</Id>
      <Value>0.2546428</Value>
   </Rdium>
   <Rdium>
      <Id>266441696</Id>
      <Value>0.2543339</Value>
   </Rdium>
   <Rdium>
      <Id>266483887</Id>
      <Value>0.25288472</Value>
   </Rdium>
   <Rdium>
      <Id>260117137</Id>
      <Value>0.24481438</Value>
   </Rdium>
   (....)
</Title>
<Authors>Related</Authors>
<Abstract>Related</Abstract>
   <Abstract>The paper presents design and results of an empirical study on organizational learning (OL). OL is conceptualized as a communication based process, changing organizationally shared reality constructions. A model is developed, which allows the description of organizations as learning systems. It serves as a frame for collecting and structuring data on reality constructions and communication relations. Gathered by interviews in two municipal administrations, data is further processed into cognitive maps (reality constructions) and networks (communication relations). Their analysis leads to</Abstract>
</Medium>
```

The paper presents design and results of an empirical study on organizational learning (OL). OL is conceptualized as a communication based process, changing organizationally shared reality constructions. A model is developed, which allows the description of organizations as learning systems. It serves as a frame for collecting and structuring data on reality constructions and communication relations. Gathered by interviews in two municipal administrations, data is further processed into cognitive maps (reality constructions) and networks (communication relations). Their analysis leads to
propositions about the nature of OL processes. It is demonstrated that changes ←
of organizationally shared reality constructions (OL) originate in self←
organizing communication networks. Furthermore, a structural and a strategic ←
organizational learning mode are distinguished. They differ with respect to the ←
information which is processed in the networks, the outcome they produce and the←
logic they follow.

A EXAMPLE XML-STRUCTURE OF A DOCUMENT

65 <Author>Klimecki, Rüdiger G.</Author>
66 <Author>Laßleben, Hermann</Author>
67 <Title>Modes of organizational learning</Title>
68 <Desc>Rüdiger G. Klimecki ; Hermann Laßleben</Desc>
69 <Town>Konstanz</Town>
70 <Publisher>Bibliothek der Universität Konstanz</Publisher>
71 <Year>1998</Year>
72 <Format>Online-Ressource</Format>
73 <Note>ResearchPaper</Note>
74 <Signature>/320</Signature>
75 <Language>Deutsch</Language>
76 <Type>Publikation</Type>
77 <Type>Online-Ressource</Type>
78 <Category>Politik/Verwaltungswissenschaft</Category>
79 <Url url="http://nbn-resolving.de/urn:nbn:de:bsz:362-opus-335"/>

80 </Fulltext>
81 </Medium>
References


