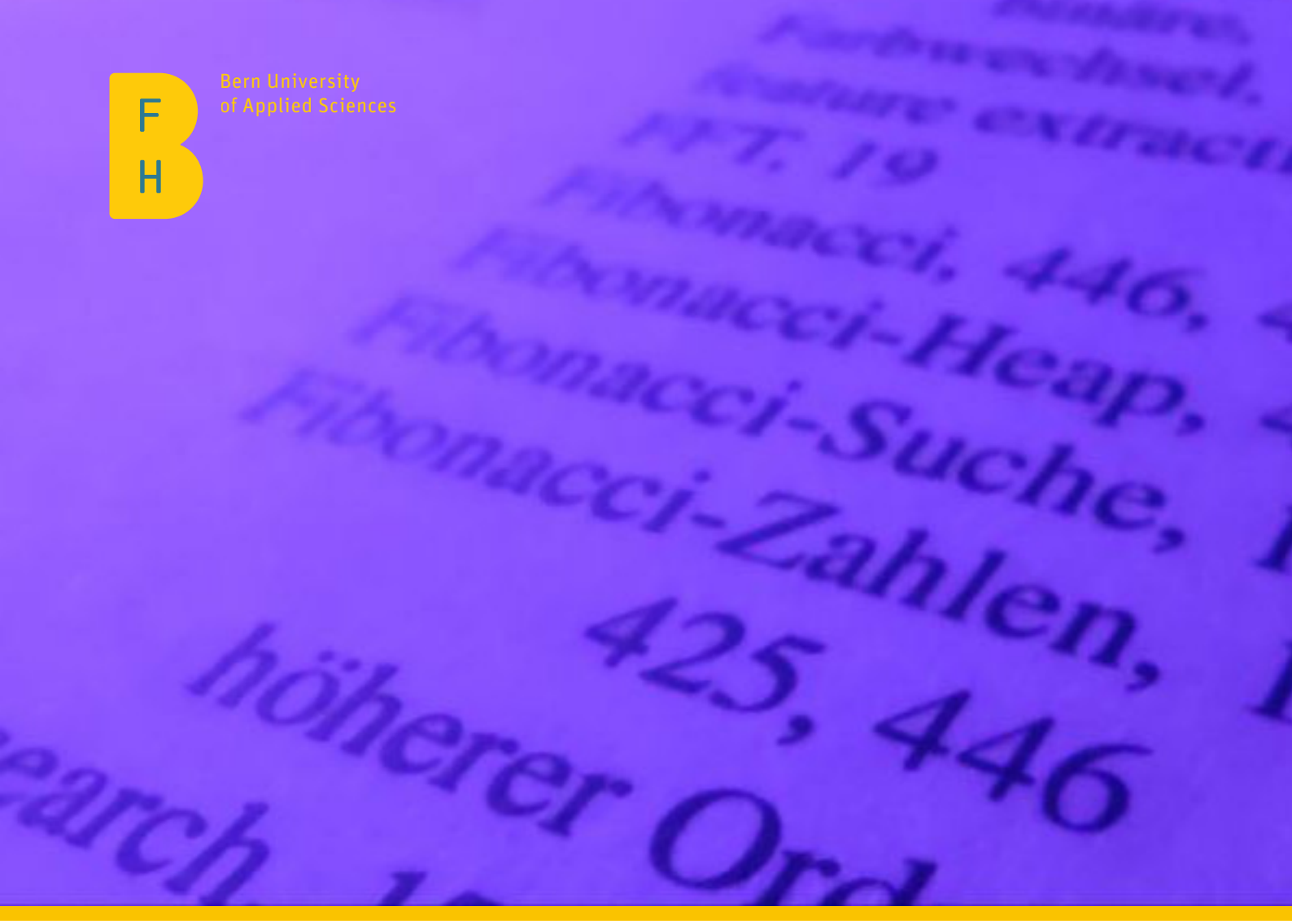


Bern University
of Applied Sciences

LaTeX-Indexer

Automatically Generating Indexes for Latex Documents

Course of study	Bachelor of Science in Computer Science
Author	David Degenhardt and Frederik Leyvraz
Advisor	Dr. Simon Kramer

Version 1.0 of June 13, 2025

Technik und Informatik

Abstract

The LaTeX-Indexer is a free, FLOSS tool that automates index creation, reducing the time it takes to create an index. An index helps the reader find significant mentions of topics. To this day however, indices are written by hand with barely any automation outside of mechanical simplifications concerning the layout. Addressing the issue, this project demonstrates a 80% reduction in indexing time is possible, by suggesting terms, allowing to track variants and automating the tagging. Future work could explore graphical interfaces, broader format support, and AI-driven term extraction.

Contents

List of Figures

1. Introduction

1.1. Initial Situation

An index in the back of a book is much more than just an alphabetic list of words and their appearance in the book. It organizes topics and allows the reader to find *useful* mentions of the concept instead of simply listing all occurrences.

Indexes were historically painstakingly compiled by hand, requiring indexers to read the text carefully and catalog the meaningful terms. With the introduction of typesetting systems like LaTeX, the process remains a manual task: The authors must identify and tag the relevant terms themselves.

1.1.1. Manual Indexing Workflow

Indexing a document follows the steps outlined in figure 1.1.

First the user has to set up their indexing tools. Second, they have to read the document and decide which words to include. The words need to be marked individually at every occurrence. Last, the document has to be rendered with the index included.

1.1.2. LaTeX

LaTeX is a typesetting system commonly used to produce academic documents. In contrast to other systems, users do not directly edit the layout and text graphically, but write plain text with markup commands, e. g. `\section{Title}` to describe a title. Later, the user calls a LaTeX compiler, who then layouts the document and outputs a PDF. LaTeX' popularity stems from its beautiful layouts and from a large ecosystem of packages and programs allowing to extend the functionality of LaTeX to e. g. include automaticly generated bibliographies and indexes.

1.1.3. MakeIndex and Latexmk

MakeIndex is a separate program that works before and after the LaTeX compiler, taking specially prepared LaTeX sources and generating a sorted typeset index.

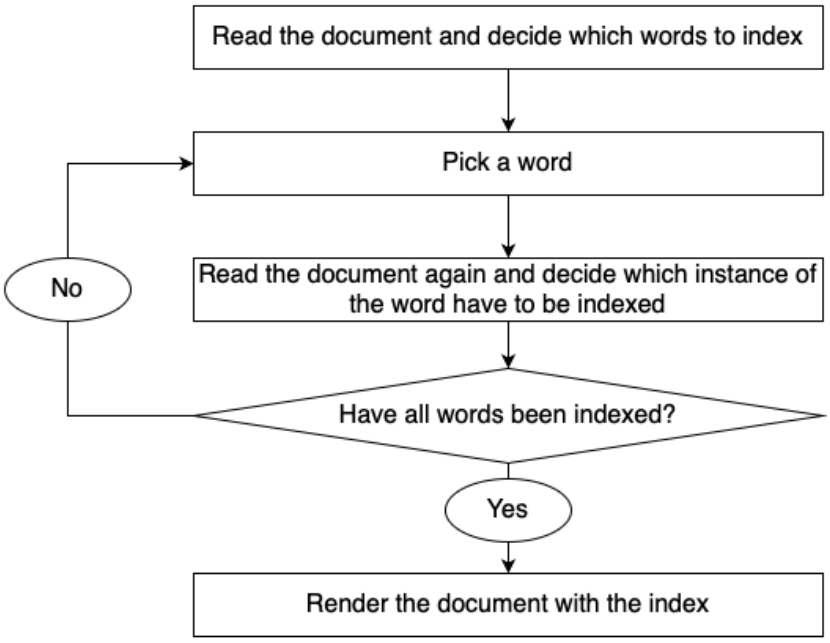


Figure 1.1.: A typical workflow when creating indexes

Even though it automates the mechanical side of index writing, it does not automate finding and tagging terms. Furthermore, when manually using MakeIndex the user has to recompile the document multiple times.

Latexmk is a tool that automatically recompiles the document with defined parameters. It detects the need for multiple compilation runs.

1.1.4. Previous Work

The paper introducing the MakeIndex package by Chen and Harrison [1] discusses indexing and its automation and explains in depth the necessary steps and considerations. The paper not only describes the MakeIndex package as it is known and used today, but also proposes an automated system to add index entries to documents. The authors describe an extension to Emacs, the GNU text editor, for ”placing index commands in the document”. Even after extensive searching, this extension could not be found on the internet; current distributions of MakeIndex do not include it.

Other attempts have been made to automate the addition of indexes to LaTeX documents [2] [3], however they do not offer the ability to add words interactively [4], are stuck in early development [5] or do not parse LaTeX at all [6].

There exists a large body of literature on keyword extraction outside of the specific application to LaTeX document processing [7] [8].

1.2. Project Goal

In this project, we develop a program, the LaTeX-Indexer, that, by using different approaches to identify relevant terms and automating the tagging process, saves authors a lot of time while making knowledge more accessible to readers.

2. Specification

2.1. Priorities

2.1.1. Product Goal

The primary goal of the proposed LaTeX-Indexer is to save time for authors when they generate larger indexes for their books. Specifically, the LaTeX-Indexer aims to save the time needed to tag relevant terms and to identify relevant terms. LaTeX sources distributed over multiple source files must be supported.

2.2. System Delimitation

2.2.1. System Environment (Statics)

The main concern with the system environment is that is very diverse and has grown over extended periods of time. LaTeX as a language does not exist as a singular specification, instead multiple implementations, additions and dialects exist. LaTeX is a full programming language, therefore rendering LaTeX means executing it. LaTeX (and especially with efforts surrounding LaTeX 3) tries to make it possible to structure documents semantically, people and machines alike have documented their struggle to correctly parse a LaTeX document non-visually.

These factors must be considered during implementation. Different existing libraries and applications exist to compile LaTeX.

2.2.2. Process Environment (Dynamics)

The top level diagram, the C4-system-context-diagram explains how the user relates to the main components. All components run locally on the user's machine.

2.3. Requirements

2.3.1. Functional Requirements (Added Value)

The main aim of the program is to save time. Derived from the given project description, the following requirements were defined:

- Extract the words from a latex source directory that are part of the text content.
- Generate a frequency distribution of the extracted words, and displaying the distribution by means of the LaTeX-library PGFplots.
- Let the user define variations for a word. Occurrences of variations in the text will be treated the same as the original word.
- Let the user define words to be sub-variants of another word, such that these words are indexed as sub-variants.
- Let the user chose which ones of the words are to be added to the index.
- Let the user chose whether all occurrences of a word are to be indexed, or whether they want to go through all occurrences manually and decide if it should be included.
- LaTeX-compiling the resulting index-command augmented LaTeX-source file so as to obtain the desired indexed LaTeX-document by means of the LaTeX-helper software MakeIndex.

2.3.2. Non-Functional Requirements

As part of the LaTeX ecosystem, the project will be published under a permissive license on CTAN and must be compatible to common LaTeX tools and packages.

2.3.3. Non-Goals

For the minimal viable product, this project does explicitly not include:

- Designing and implementing a Graphical User interface,
- Broad integration in existing tooling outside of simple build scripts,
- Support for other formats like Markup or other indexing tools.

A full reproduction of the original project description can be found in in appendix A.

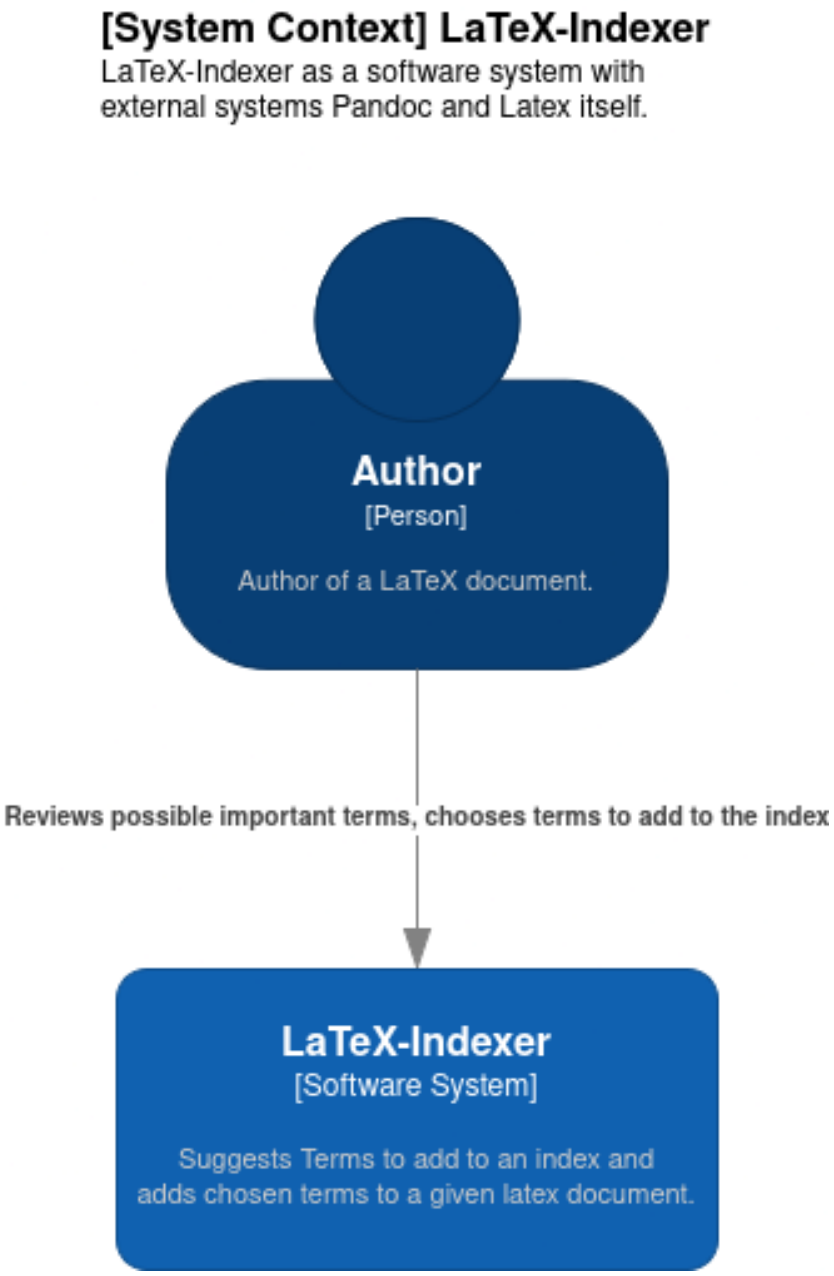


Figure 2.1.: The system context diagram corresponds to a Use-Case diagram

3. Implementation

3.1. Architecture

3.1.1. Goals

The architecture of the indexer is described in the following section. The diagrams are modeled using the C4 modeling method. C4 models are divided on four levels: Context, Containers, Components, and Code. Containers are defined as separately deployable units, the do not have to be containers in the usual sense.

The focus of architectural decisions was to keep the indexer modular, extendable, maintainable. The architecture supports incremental development steps.

External components must be exchangeable.

3.1.2. Solution Architecture

Advantages

The current architecture is modular and separates user interaction, external components and internal processing well. External components are exchangeable.

Performance, Limited Scalability

Currently, the indexer processes documents sequentially, this may lead to longer processing times when using very large documents.

Calling external programs may introduce some latency.

Components

The indexer consists of the UI, a parser and plotter component.

UI

The UI class provides an interactive command-line user interface for user in- and output. It has a simple structure with a while-loop that runs until the user quits or interrupts the program, a scanner which reads user inputs, a little bit of logic

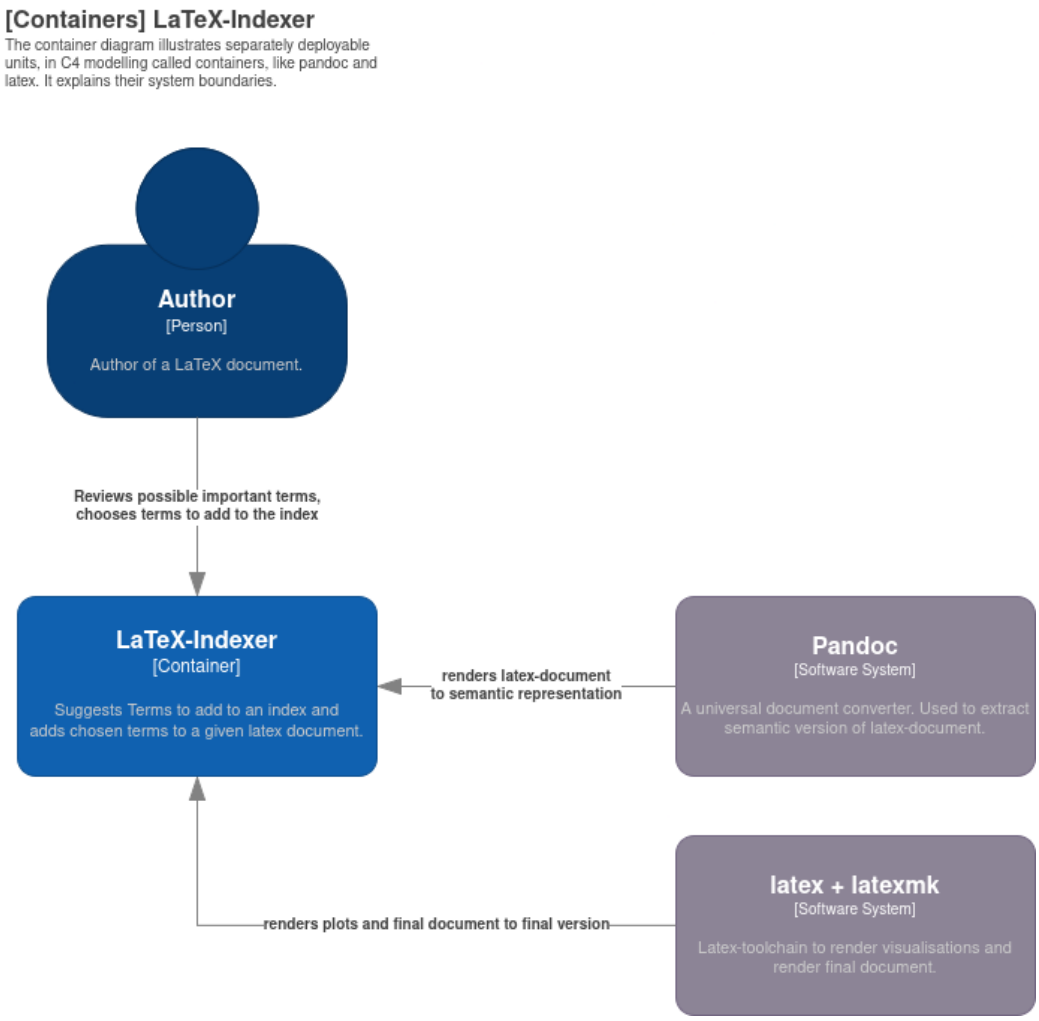


Figure 3.1.: The container diagram, each so-called container is separately deployable

which parses user commands and packages them into the custom Argument inner class such that they can easily be used by the other components, and finally a switch statement which calls the user-requested command and passes it an array containing the arguments passed with the command.

Parser

As LaTeX is a programming language, it is not possible to parse it without compiling and running the code. Extensive use of packages, most of them developed and maintained by authors, publishers and institutions just for their own needs make it difficult to preempt all possible issues.

Currently, there are two parsers available, one is disabled in the code: Pandoc is

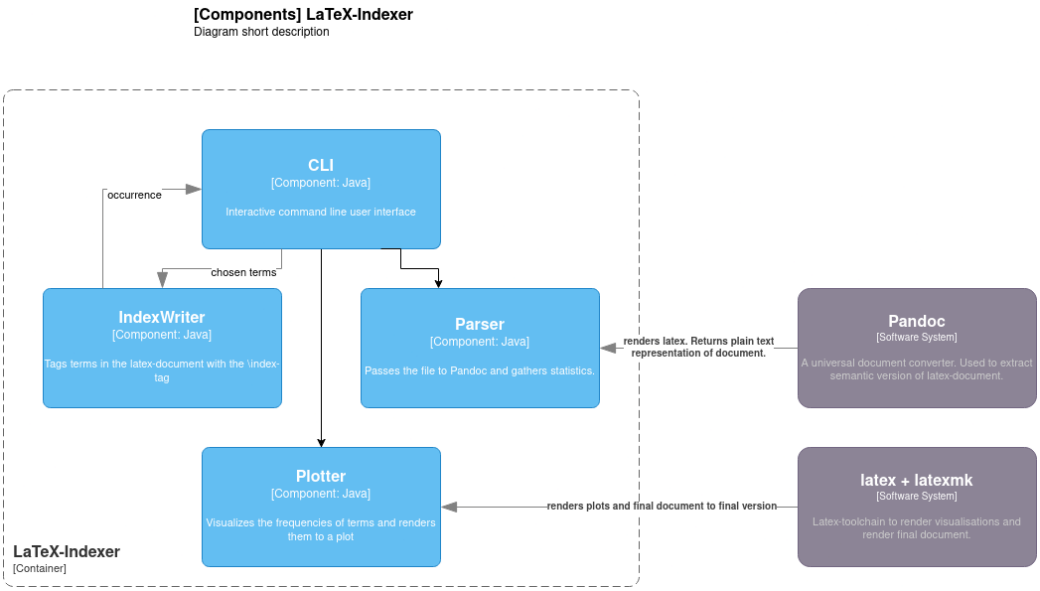


Figure 3.2.: The component diagram

active and support for detex was implemented. Pandoc is a document converter, turning virtually any document into another by first converting the inputs to an intermediate representation. Pandoc is widely used to render LaTeX documents to other formats than PDF and therefore supports most common LaTeX packages reasonably.

Detex (OpenDetex) is a small command line program shipped with most LaTeX distributions. One very useful feature of it is, that it indicates the exact source location of a word. Unfortunately, the errors it made during parsing prevented us from using it as input.

IndexWriter

The writer searches for all occurrences of a word and replaces them with the respective index keywords. The writer tries to use detex to find the location of words, however detex miscounts some lines. This is corrected in a workaround.

Plotter

The plotter visualizes term frequencies and renders them into plots. Upon creation, the plotter receives a list containing all the words the parser gathered from the document. When it is being called to generate a plot, it receives the arguments passed to the program by the user, creates a copy of the list of words and filters it according to the user's preferences. It then creates a .tex file with the

user-defined or generic name and writes latex code for generating a plot with the latex library PGFplots to it, with a plot point for each word in the list.

Similarly, the plotter is also responsible for listing the user-requested words to the command line. The same method is used for filtering the list of words in both cases, but rather than generating latex code, the plotter adds the words to a formatted string and returns that string to the UI, which can then print it to the command line.

Testing

The goal of testing software is to guarantee the correctness, performance and maintainability of software products. To attain these goals we developed unit tests and did manual end-to-end tests.

3.2. Processes, Project Management

3.2.1. Methodological Considerations

To gauge which approach is suited, we considered cultural factors, organizational aspects, volatility and risk.

As the project's objectives are variable, the product is suitable for incremental delivery. We consider the project risks to be manageable even given our very limited experience (creating indexes by hand is painful in the figurative sense only). Our team is very small and closely connected through BFH. All involved persons are interested in exploring different solutions, open to changes and motivated to use agile methods. We therefore found Scrum to fit the needs of the project, noting however that we would have to adapt Scrum to our needs, as the team is very small.

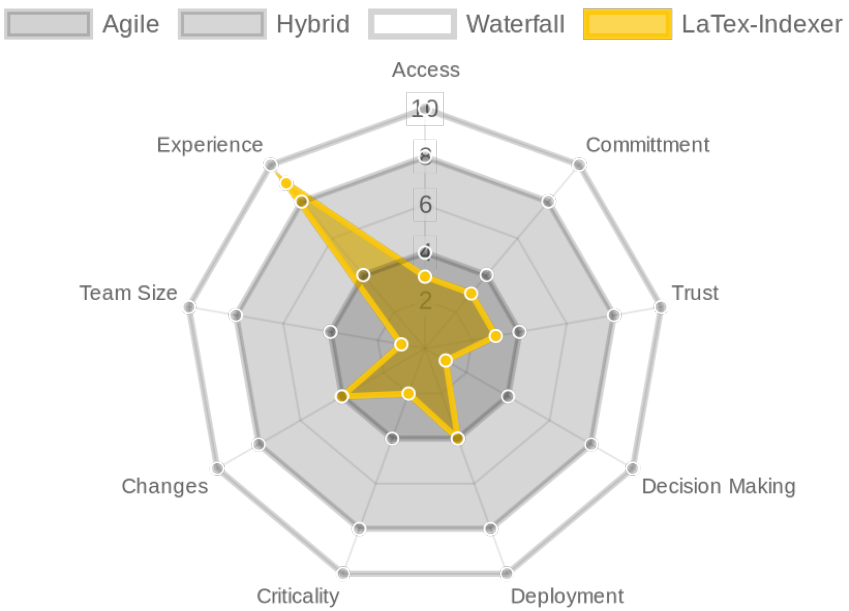


Figure 3.3.: Assessment of PM-Approach Questionnaire

3.2.2. Stakeholders and associated Scrum-Roles

Client

Our client, Simon Kramer, is a researcher, lecturer and in the context of this project a target author. He will be using the product and therefore is the main stakeholder. For this reason he also was the main contact person whenever questions about potential features arose, that were deemed to important to be called by the product owner on his own.

Product Owner

David Degenhardt is responsible for keeping the Product Backlog maintained and ensuring that the clients needs are adequately represented. He ensures the backlog is clear, visible, and prioritized based on business value and customer needs. He assesses the different product backlog items and decides on their usefulness and importance to evaluate which features are to be included in the next sprint, in an attempt to maximize the business value of the product.

Scrum-Master

Frederik Leyvraz is “accountable for establishing Scrum as defined [...]” [9]. He makes sure the Scrum Team has a common understanding of Scrum practices

and helps find techniques and methods for teammates to effectively carry out their work.

Developers

As the team consists of only three people, the product owner and Scrum master also serve as developers, developing implementation solutions for the required features and implementing the indexer as well as contributing to Scrum practices, mainly sprint planning, daily scrum and sprint review.

3.2.3. Scrum Adaptations

Sprint Duration, Velocity

As this project has a fixed deadline, we opted for the shortest (meaningful) sprint duration of two weeks, giving us six sprints in total, where each sprint is long enough to achieve meaningful improvements in the project but short enough to not lose oversight over the items that have to be completed. To be able to estimate the workload and our velocity, we used hours as a basis and fixed twenty hours per sprint as our velocity, as this seemed like a realistic amount of work we could take on per week for this project. Items were weighted using the Fibonacci estimation method since we deemed it to be relatively intuitive and practical for the velocity we set for our selves.

Daily Scrum

The low velocity allowed us to extend the daily Scrum meetings intervals. Instead of meeting every day, we met whenever we were both at school, usually leaving us with 3 meetings per week on Tuesday, Thursday and Friday.

Definition of Ready

The definition of ready (DoR) is a quality gate that all items of the product backlog must fulfill in order to be “ready”. “Ready” means that the items then may be selected for the sprint backlog.

Considering we did not have a lot of experience in Scrum, we chose to start with all INVEST criteria (Independent, Negotiable, Valuable, Estimable, Small, Testable). As we are a very small team, we added Understandable as criterion to ensure that the items are documented in a way more than two people understand.

Definition of Done

The definition of done (DoD) is the second quality gate between the state doing and done. The following criteria were added to our user story template, but we extended and elaborated on the criteria when needed during backlog refinement and sprint plannings.

- (Functional requirements met)
- Documented
- Tested
- Accepted by the product owner

As of the time of writing, we have not yet implemented an automated deployment pipeline and therefore do not include any deployment related criteria in the DoD.

3.3. Results: Scrum-Artifacts

3.3.1. Product Vision

An index in the back of a book is much more than just an alphabetic list of words and their appearance in the book. It organizes topics and allows the reader to find useful mentions of the concept instead of merely listing all occurrences.

Historically, indexes were painstakingly compiled by hand, requiring indexers to read the text carefully and catalog the meaningful terms. With the introduction of typesetting systems like LaTeX, even when using helpers like MakeIndex, the process remains a manual task: Authors must identify and tag relevant terms themselves.

In this project, we develop a program, the LATEX-Indexer, that, by using different approaches to identify relevant terms and automating the tagging process, saves authors a lot of time while making knowledge more accessible to readers.

The primary goal of the proposed LaTeX-Indexer is to save time for authors when they generate larger indexes for their books. Specifically, the LaTeX-Indexer aims to save the time needed to tag relevant terms and to identify relevant terms. Tex sources distributed over multiple source files must be supported.

As the project moves along, multiple approaches to keyword extraction are implemented, with first priority given to a frequency-based approach. Other approaches include using large language models (LLMs commonly referred to as AI) or other context-based methods.

3.3.2. Timeline

We defined one milestone: The minimum viable product is reached when all requirements as given in the original project description are fulfilled. ie.

- Extracting the words in the .tex files of a LaTeX source directory.
- Generating a frequency distribution of the extracted words, and displaying the distribution by means of the LaTeX-library PGFplots.
- Letting the user choose which of the extracted words are to be included in the index.
- Inserting the string "\index{chosenOne}" into the LaTeX-source file right after each occurrence of each user-chosen word.
- LaTeX-compiling the resulting index-command augmented LaTeX-source file.

Date	Description
2025-02-17	Start of Semester
2025-03-10	Start of Sprint 1
2025-05-23	Milestone: Minimum Viable Product
2025-06-13	Project Due Date

3.3.3. Product Backlog

The product backlog is maintained on GitLab in the form of issues. Four different status differentiate unrefined, refined, currently being implemented, and done backlog items. Three priority levels help organize the backlog. Figures 3.4 and 3.5 give an overview of the product backlog at the time of writing at the end of the second sprint.

3.3.4. Sprint 1

Sprint Goal

The goal of the first sprint was: Build the solid foundation for the project to grow.

Sprint Backlog

The following user stories were added to the sprint backlog.

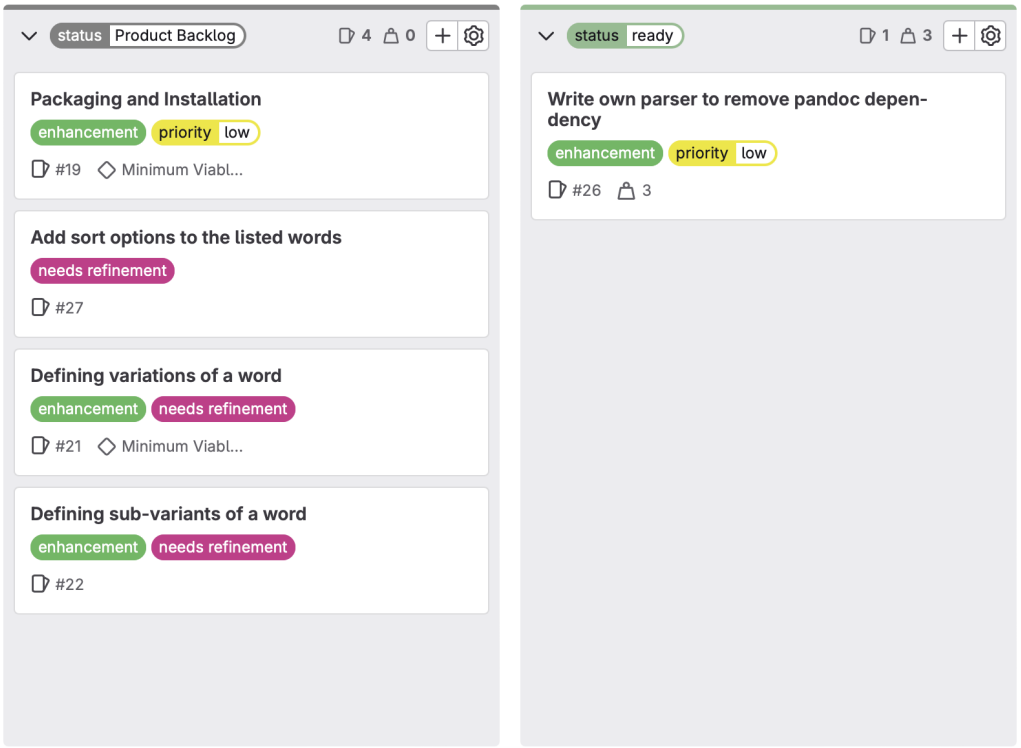


Figure 3.4.: Unrefined and refined backlog items.

10 - Define Architecture and technology
As a developer I want to understand the architecture and be able to record decisions concerning it.
Priority: high Weight: 13 Status: done

13 - Extracting words using pandoc
As a user I want to be able to extract all the words in the .tex files of a LaTeX source directory in order to list and work with them.
Priority: high Weight: 2 Status: done

17 - Setup Java project and build system
As a developer I want to build the program quickly and automatically so that builds are the same across all deployments.
Priority: high Weight: 3 Status: done

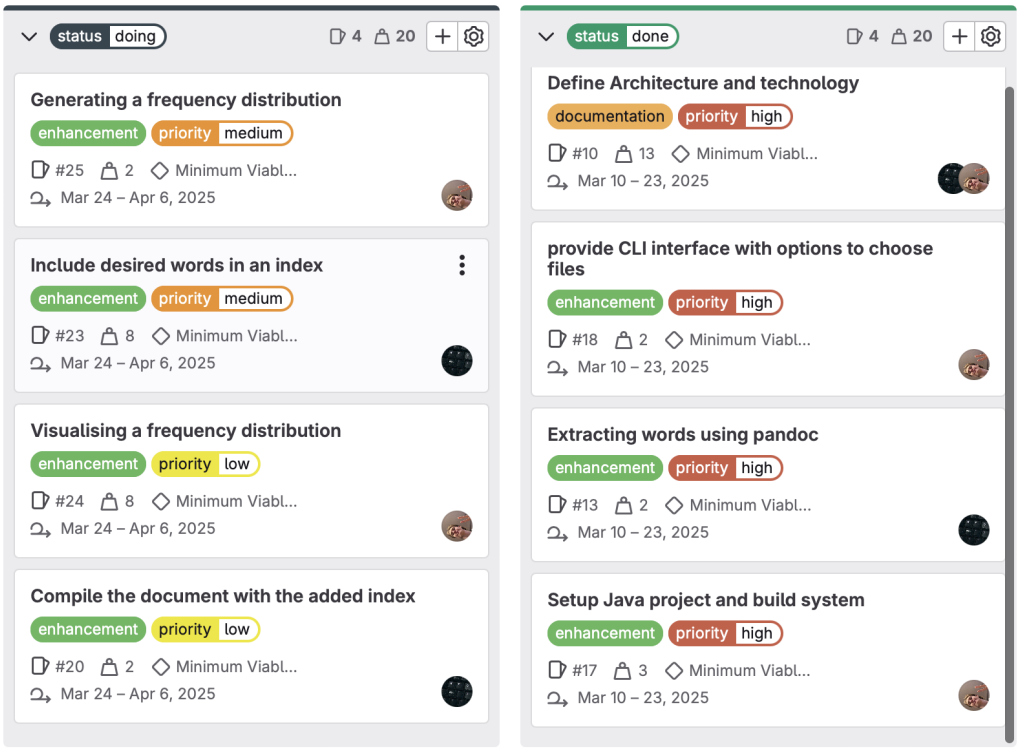


Figure 3.5.: Backlog items which are done or currently worked on.

18 - provide CLI interface with options to choose files
As a user I want to be able to LaTeX source directory to the program so that it knows what to work with.
Priority: high Weight: 2 Status: done

3.3.5. Sprint 2

Sprint Goal

The goal of the second sprint was: Calculate the frequency distribution of words in a given document and visualize it.

Sprint Backlog

The following user stories were added to the sprint backlog.

23 - Include desired words in an index
As a user I want to define a (list of) word(s) to be included in a specified index.
Priority: medium Weight: 8 Status: done

25 - Generating a frequency distribution
As a user I want to generate a frequency distribution of the words in the .tex files of a LaTeX source directory so that I can choose which words to include in an index.
Priority: medium Weight: 2 Status: done

20 - Compile the document with the added index
As a user I want to see the updated index after adding words so that I always know what my index currently looks like.
Priority: low Weight: 2 Status: done

24 - Visualising a frequency distribution
As a user I want to generate a frequency distribution of the words in the .tex files of a LaTeX source directory so that I can choose which words to include in an index.
Priority: low Weight: 8 Status: done

3.3.6. Sprint 3

Sprint Goal

The goal of the third sprint was to add the first non-MVP features, as well as to prepare the existing features for deployment, by fixing some minor bugs we found and preparing the packaging and installation.

Sprint Backlog

The following user stories were added to the sprint backlog.

19 - Packaging and Installation
As a User I want to install the indexer with a package manager, so that I can manage my tools efficiently and without manual intervention.
Priority: medium Weight: 3 Status: done

26 - Write own parser to remove pandoc dependency
As a user I want the program to be self-contained so that I do not have to install dependencies.
Priority: low Weight: 3 Status: done

27 - Add sort options to the listed words
As a user I want to list or filter the parsed words alphabetically or by frequency.
Priority: low Weight: 5 Status: done

Additionally, one backlog item concerning a bug in the code for the generation of the latex code for the word frequency plot was added in this sprint.

34 – Fix frequency plot latex code to show words vertically
Priority: medium Weight: 1 Status: done

3.3.7. Sprint 4

Sprint Goal

The goal of the fourth sprint was to add some more non-MVP features and, as we found a lot of new bugs beforehand, to fix as many bugs as possible to ensure our product is ready for deployment.

Sprint Backlog

The following user stories were added to the sprint backlog.

21 - Defining variations of a word
As a user I want to define a word to be a variation of another word so that variations are handled the same way.
Priority: medium Weight: 8 Status: done

22 - Defining sub-variants of a word
As a user I want to be able to define sub variants to a word so that so that I can distinguish more specific references from general ones.
Priority: low Weight: 5 Status: done

3.3.8. Sprint 5

Sprint Goal

The goal of the fifth sprint was to focus on preparing this scrum report, the final presentation and the live demo for it. As we had implemented all the features we wanted, and had fixed most of the bugs we had found by then, we decided that we could afford to concentrate on the presentation and report during this sprint.

Sprint Backlog

As we were more focused on preparing the final presentation, writing the Scrum report and generally finding and fixing bugs in our implementation, we didn't add any specific backlog items to the sprint backlog during this sprint ????

3.3.9. Sprint 6

Sprint Goal

The goal for the sixth sprint is to fix the final minor bugs present in the code, and to deploy the project to CTAN.

Sprint Backlog

40 - Deploy to CTAN
As a user I want download the LaTeX-Indexer from CTAN.
Priority: high Weight: 5 Status: doing

3.3.10. Project Setup Review

We found our assessment of the initial situation accurate.

We assessed the task to be to develop a "helper for a helper" program, the analysis helped us understand, that the product's purpose, use case, and motivation need to be clearly explained before discussing the technical subject matter itself.

Stakeholders were identified early on and remained unchanged throughout the project. Potential stakeholders such as future open-source contributors or platform integrators were deemed out of scope. No community-building efforts were initiated. This assessment is the one most likely to change as the project moves to its maintenance and publication phase.

There was a high degree of autonomy concerning the scope and methods of this project.

As the project team consisted of only two persons, conflicts of interest between assigned roles could have better been anticipated and accounted for.

3.3.11. Scrum review

Product Backlog

The product backlog refinement turned out to be a very important and useful activity, and was therefore prioritized. Team members shared many assumptions and perspectives, making it difficult to recognize shared blind spots.

The user story format was useful initially, though confusion persisted around how to handle non-functional requirements.

Definition of Done (DoD)

The DoD was comprehensive, but no formal control mechanisms were in place.

For future projects, integrating automated checks (e.g., code coverage, compilation) could improve consistency and accountability.

Definition of Ready

INVEST criteria were applied informally. A shared understanding among team members sufficed for this project. Relaxed handling may be acceptable in future small-team projects as well.

Sprint Backlog

The Sprint Board was effective for daily tracking. Scope issues occasionally arose; it was sometimes difficult to distinguish between simple tasks and full user stories (see also Product Backlog).

Product Increment

Due to the short sprint lengths, increments offered limited value and rendered the presentability criterion of the product increment a challenge.

Release Plan

A small release plan was created, defining dates for the MVP and final due date. No updates to the plan were needed.

Impediments

Impediments were not formally tracked, but were addressed through bilateral, direct communication.

Sprints

A sprint diary was kept in handwritten form, which proved useful during reviews. Sprint planning was central to aligning team priorities and defining scope. With only two team members, a shared understanding was quickly reached. However, this also made it more difficult to identify blind spots. The team made conscious efforts to be aware of and address this.

Velocity

Time estimates are known to be a difficult problem, despite the fact that the estimates we took and the velocity we chose are quite conservative, we found them to be useful in assigning work and planning ahead. We were able to complete all planned work in time, taking a break over easter.

The two-week sprints allow for very simple scheduling. The short time between planning, review/retrospective and some backlog refinement led to very short daily Scrums, which led to high overhead.

The Fibonacci estimation method helped us break ties when estimating weights, the granularity (fixed time amounts) was not a limitation.

Daily Scrum

Daily standups were difficult to formalize and needed continuous effort to keep up. Greater discipline may be helpful in future iterations. This was discussed and corrections were implemented as a result.

Burndown Charts

GitLab uses the time of closing an issue as the time mark when to reduce weights in burndown charts. We did not close issues consistently (rather, we assigned the status “done”), leading to misleading (or rather flat) charts. Iterations can only be used in GitLab Groups, which we had not set up initially.

Sprint Reviews and Retrospectives

Reviews were generally short and focused on backlog refinement. Client reviews were conducted and provided broad feedback and guidance.

Retrospectives

Retrospectives were challenging to formalize. Defining the scope and actionable outcomes of retrospectives was difficult — the team aimed to avoid spending too much time on rituals at the expense of actual work.

The dual role of Scrum Master and Developer caused conflicts of interest, especially when the Scrum Master underperformed in their developer duties.

These issues were mitigated through honest communication and corrective measures were taken.

Communication with Client

The communication with our client put emphasis on the need to reduce bureaucracy. He gave us a lot of freedom to interpret the specifications given in the introductory document.

Tooling

GitLab was a useful platform but better suited for larger teams.

Ganttlab was not used due to narrow mapping/binding between issues on GitLab metadata and the corresponding chart.

Burn-down charts were only available retroactively, due to incomplete initial GitLab setup.

Settings and templates developed during the project could benefit future teams. CI/CD was out of scope, so no experience was gained in this area — though it may be valuable to include in future projects.

4. Deployment and Integration

4.1. Licensing

The LaTeX-Indexer is released under the GPL-3 or later.

4.2. Deployment

4.2.1. Building from Source

To build the executable JAR run the following command in the root directory of the Maven project: ‘mvn clean package‘. This will create a JAR file in the target directory.

4.3. User Manual

4.3.1. Prerequisites

Before the indexer is able to run, make sure you have the following programs installed:

- An up-to-date LaTeXinstallation
- Pandoc
- Java Version 21 or higher

4.3.2. Installation

macOS

```
 brew install pandoc
curl -O https://mirrors.ctan.org/.../indexer.zip
unzip indexer.zip
```

4.3.3. Usage

```
java -jar indexer.jar /path/to/your/file
```

The LaTeX Indexer supports the following commands, entered at the prompt:

- **h, help**: Displays a list of all available commands with brief descriptions.
- **p, parse**: Re-parses the `.tex` document to update the list of words. This is automatically performed at startup but can be rerun to refresh the word list.
- **l, list**: Lists parsed words with optional parameters:
  `-n <number>` (number of words to display, default 20),
  `-c <a|f>` (sort alphabetically or by frequency, default frequency),
  `-p <prefix>` (filter words by prefix), and
  `-r <true|false>` (reverse order, default false).
  `-h` for detailed help.
- **g, generate**: Creates a `.tex` file with a frequency plot using PGFplots, rendered with PDFLaTeX. Supports the same parameters as `list`, plus `-f <filename>` for a custom plot file name. Use `-h` for details.
- **s, subvariant**: Defines words as subvariants of a specified word, indexing them under the main word. Enter the command as `s <word1> <word2> ...`, then provide subvariant words when prompted. Use `-h` for help.
- **v, variation**: Defines words as variations of a specified word, indexing their occurrences under the main word. Enter as `v <word1> <word2> ...`, then provide variation words. Use `-h` for help.
- **a, add**: Adds specified words to the index automatically. Enter as `a <word1> <word2> ...`. The tool checks if the words exist in the document before adding them. Use `-h` for help.
- **i, interactive**: Interactively adds a single word to the index, prompting the user to confirm each occurrence. Enter as `i <word>`. For each occurrence, the tool displays the line and surrounding context, allowing the user to choose `[Y]es`, `[N]o`, or `[A]bort`. Use `-h` for help.
- **q, quit**: Exits the program.

4.3.4. Hint

While it is helpful to index an entire book at a time, the authors found it useful to consider single chapter files individually.

5. Results and Discussion

5.1. Results

5.1.1. Estimated Productivity Gain

In order to estimate the productivity benefit of the LaTeX-Indexer, we first try to estimate the time and cost it takes to generate an index manually. Then we measure the time it takes to add an index using the indexer.

Time Expenditure of Manual Indexing

An average reader reads around 250 words per minute [10] and an average, A4-sized page typeset with the report package in latex contains about 500 words. Further we assume the amount of graphical elements and so on to be negligible.

Beginning by replicating the results from [10], we found our reading pace a bit slower but comparable to the times given. Then we stopped the time it took to read a page think about what terms and candidates to include in the index and then adding the terms in the source. This coefficient α was estimated to be 1.5.

Overhead, like the time to include packages, render the document etc. were not considered, as they are the same with and without the indexer.

Mathematically, we modeled the estimated time as follows:

$$t_{think} + t_{index} = \alpha \times t_{read}$$

For a total of

$$\begin{aligned} t_{total} &= t_{read} + t_{think} + t_{index} \\ t_{total} &= 2.5 \times \frac{500 \text{ words}}{250 \text{ wpm}} = 5 \text{ minutes} \end{aligned}$$

The resulting 5 minutes of total time per page to index a page manually are in line with our (limited) experience. We used [11] as a reference book. To estimate

the total time it would take to index it we rounded the book's number of pages to 800. Using the estimation method developed above, we calculate that it takes 67 hours of work to index the entire book.

Time Expenditure of automatic Indexing

Using the LaTeX-Indexer, we indexed a different sample of 10 pages of the same book. First we listed the 50 most frequent words, generated a plot and identified a term to add. With the interactive mode, we reviewed all 18 occurrences of the term and added the two relevant occurrences of that term, skipping the others. By repeating this process (listing, then interactively or automatically adding words to the index, we were able to index the pages in 6:45 minutes.

To correct for the fact that the timed tester developed the program and was familiar with its use, the time is multiplied by a novelty-factor of 1.5 to obtain the speed of a realistic indexing process, which rounds to 1 minute per page.

Time Saved

The indexer reduces time spend on indexing tasks from 5 minutes per page to 1 minute per page; or for an entire book from 67 hours to 13 hours. This corresponds to a reduction of 80% of manual and repetitive tasks.

Cost Economy

Using the gross average salary at BFH (45 CHF per hour)[1]. the cost to index a book manually amounts to 3000 Swiss Francs. By using the indexer, more than 2300 Swiss Francs can be saved.

These estimates are conservative, as more books at BFH are presumably written by highly paid professors, resulting in higher hourly rates.

5.2. Discussion

5.2.1. Conclusion

The project clearly shows that the problem set is relevant and that modular, extensible solutions are a valuable addition to the vast LaTeX universe.

We were able to estimate cost and time benefits using a simple mathematical model. The assumptions taken do not limit the significance and meaningfulness

[1] Excluding anciliary wage costs, calculated by taking the wage costs and dividing them by the FTE as reported in [12].

of the results: Time savings of 80% are significant, especially as the indexer reduces manual, boring tasks.

At first, the frequency heuristic looked like it would be little in identifying terms, during its use, it has proven to be quite helpful. The advantage over more involved methods is, that it is quick and transparent to the user why the words were ranked as they are.

The current implementation using the a command line interface limits its wider use for non-technical people. An Integration to text editors, as suggested by the initial MakeIndex authors, was not realized.

As a result of this project, an indexer was developed that fulfills the requirements

5.2.2. Future Work

Other Input Formats

Currently, the indexer limits itself to LaTeX files. It would be possible to leverage pandoc and use multiple input file formats as base. Also, it would be interesting to extend pandoc-based templating engines to include a markdown-equivalent to the LaTeX `\index{...}`, quite like Emacs allows users to manually index their org-mode documents.

Key Extraction, Large Language Models

Currently, the indexer supports only frequency based suggestions. As discussed in the literature (cf. section 1.1.4) it is possible to use much more involved techniques to extract relevant keywords. It still is a great challenge to make these automated decisions transparent, i.e. understandable to the user, as all automated approaches cannot truly make semantic arguments (yet?).

At the time of writing, there are different AI assistants on the market to help users navigate authoring LaTeX documents. None of the reviewed assistants specifically advertise indexing as a feature, though. Considering the fact that indexes are prone fail by small but significant syntax errors, this may be an interesting avenue to pursue.

User Interfaces

The current command line user interface may deter inexperienced users or users that do not wish to interact with the terminal. By virtue of the modularity of the current indexer, it would be possible to extend this project with a graphical user interface in Java. By separating the indexer even further and offering its services over e.g. a network, different interfaces would be possible, especially

when integrating in CI/CD pipelines, e.g. offered by BFH. This would require extensive work.

Integration in Existing Tools

Following from the last point, it would be interesting to include the indexer or similar functionality in other FLOSS-LaTeX editors or IDEs like MacTeX or Overleaf (which maintains a FLOSS edition as well).

Bibliography

[1] P. Chen and M. A. Harrison, “Index preparation and processing,” vol. 18, no. 9, pp. 897–915. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/spe.4380180907

[2] txwikinger. Is there an automatic process to create index creation? TeX - LaTeX Stack Exchange. [Online]. Available: https://tex.stackexchange.com/q/856

[3] Geremia. Answer to ”is there an automatic process to create index creation?”. [Online]. Available: https://tex.stackexchange.com/a/172906

[4] CTAN: Paket addindex. [Online]. Available: https://ctan.org/pkg/addindex

[5] Indexmeister. SourceForge. [Online]. Available: https://sourceforge.net/projects/indexmeister/

[6] “Feup-infolab/latex-index-tools,” FEUP Information Systems Laboratory. [Online]. Available: https://github.com/feup-infolab/latex-index-tools

[7] N. Firoozeh, A. Nazarenko, F. Alizon, and B. Daille, “Keyword extraction: Issues and methods,” vol. 26, no. 3, pp. 259–291. [Online]. Available: https://www.cambridge.org/core/journals/natural-language-engineering/article/keyword-extraction-issues-and-methods/84BFD5221E2CA86326E5430D03299711

[8] Z. Wu, Z. Li, P. Mitra, and C. L. Giles, “Can back-of-the-book indexes be automatically created?” in *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, ser. CIKM ’13. Association for Computing Machinery, pp. 1745–1750. [Online]. Available: https://dl.acm.org/doi/10.1145/2505515.2505627

[9] K. Schwaber and J. Sutherland. The scrum guide. [Online]. Available: https://scrumguides.org/scrum-guide.html

[10] M. Brysbaert, “How many words do we read per minute? a review and meta-analysis of reading rate,” vol. 109, p. 104047. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0749596X19300786

[11] T. Ottmann and P. Widmayer, *Algorithmen und Datenstrukturen*. Springer. [Online]. Available: http://link.springer.com/10.1007/978-3-662-55650-4

[12] Rektorat BFH, “Annual report 2024.” [Online]. Available: https://www.bfh.ch/en/about-bfh/facts-and-figures/annual-report-2024/

Index

A. Project Description

LaTeX-Indexer

Description

The **goal (what)** of this project is to deliver a FLOSS-licensed, platform-independent piece of meta-helper software (a helper software for a helper software ;-), called the *LaTeX-MakeIndex Helper,* short, *LaTeX-Indexer,* that helps automate the generation of an index for LaTeX-documents by

1. recursively extracting the words (not the LaTeX-commands) in the (.tex) files of a LaTeX-source directory;
2. generating a frequency distribution of the extracted words, and displaying the distribution by means of the LaTeX-library PGFplots;
3. letting the software user choose which ones of the extracted words - and possible variants thereof - are to be included in the desired index;
4. inserting the string "\index{chosenOne}" into the LaTeX-source file right after each occurrence of each user-chosen word "chosenOne"; and
5. LaTeX-compiling the resulting index-command augmented LaTeX-source file so as to obtain the desired indexed LaTeX-document by means of the LaTeX-helper software MakeIndex.

The **purpose (why)** of this project is to relieve the pain of LaTeX-users who want to generate an index for their documents, but who most frequently capitulate during the first couple of hours of manually inserting "\index{chosenOne}"-strings into their LaTeX-source files due to well-justified doubts about the meaning of such a transhuman if not robotic existence.

The code should be minimal, modular, and self-explaining.

The project report should be concise (maximally informative, minimally long).
It must contain this project description as a quotation.

Technologies

Java, LaTeX (https://ctan.org/pkg/makeindex, https://ctan.org/pkg/pgfplots)

Related Projects

https://ctan.org/pkg/luahttp
https://ctan.org/pkg/javascripthttp
https://ctan.org/pkg/latex-dependency-grapher
https://ctan.org/pkg/latexscreenshooter

Advisor

Dr. Simon KRAMER
(https://www.simon-kramer.ch/Simon-Kramer.vcf)

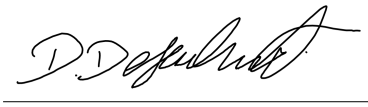
Declaration of Authorship

I hereby declare that I have written this thesis independently and have not used any sources or aids other than those acknowledged.

All statements taken from other writings, either literally or in essence, have been marked as such.

I hereby agree that the present work may be reviewed in electronic form using appropriate software.

June 13, 2025



David Degenhardt

Frederik Leyvraz