

- PIPETEX -

Requirements and Specification

Author

Max Weise

Contact Email: maxfencing@web.de

July 19, 2022

Contents

1	Introduction	2
1.1	Motivation	2
1.2	Goals	4
2	Requirements Engineering	5
2.1	Use Cases	5
2.2	Functional Requirements	6
2.2.1	Must Have Requirements	7
2.2.2	Should Have Requirements	8
2.2.3	Nice To Have Requirements	8
	Glossary	9
	List of Tables	9

1 Introduction

Hi, welcome to this requirements document. It is a collection of key-characteristics and features this project should include. The whole purpose of this project is to learn and improve my coding and software development skills. But let's not get ahead of ourselves.

In section 1.1 I will briefly talk about why this application is useful to me and might potentially be useful to you and section 1.2 will set the goals which determine the success of this project.

So without further ado, let's jump into it.

1.1 Motivation

The motivation for this project is twofold:

1. Keep expanding my knowledge in the software development field
2. Improving my workflow with L^AT_EX-documents

Let's talk about them separately for a bit.

Gaining Knowledge

First and foremost, I wanted to create a more or less complete application from start to „finish“, formulating requirements, designing software on the drawing board and packaging everything up to a nice installable package for me (and maybe others) to use.

I had the chance to participate in one or two development projects as part of my studies and I definitely learned much from them. The downside was, that none of them was a *complete* development process. Complete in the sense of including all vital steps. In the case of a project which included a huge codebase we were included in the SCRUM process and we would have regular meetings where we presented our work and which tickets were next to be worked on. The development part was very spot-on, but the planning and designing part was pretty much nonexistent.

Other projects were simple projects to be submitted at the end of the semester. While the expectations of my professors were higher each year and we had to be more specific about what we wanted to do, the project never actually came to a packaging and deployment phase.

So in essence, I have kind of experienced a whole development process, but split over multiple projects and semesters. So this project should combine the experience from all my previous projects and let me face each challenge on my own and without the pressure of good grades lurking somewhere.

Improving Workflow

L^AT_EX is a great tool to create PDF documents which look professional and uniform. I use it pretty much for every mildly important documents and even this requirements specification file is written and compiled using latex.

While it is a great tool, there are some inconveniences I'd like to get rid of. One of those inconveniences is the so called „draft mode“. This is mainly used to minimize compilation time by only showing black frames instead of fully rendered images. While it is nice in the early stages of the document because it speeds up the workflow, it becomes tedious when the final document has to be created over and over again, but the draft option can't be removed fully, so I have to manually remove and re-add it every time I want to switch modes. Sometimes I have to check if the full image is big enough so I can't use draft mode for this. And in some cases (like this document) I keep it in draft mode all the time and only switch to final mode when I need to deploy the document.

And don't mention bibliographies or glossaries. Each of them requires a separate

engine to be run on the document, sometimes multiple times and in a very specific order, otherwise the references won't work or won't even be visible. And don't let me mention the various auxiliary files generated by all the engines and features \LaTeX provides.

Wouldn't it be neat if only I can run a single command and all those tasks will be done for me? Of course I could configure my texteditor to do all those things for me using commands and keyboard-shortcuts and whatnot, but even those I'd have to input manually and I'd have to wait until the first step is finished, then input the second instruction, wait for that to finish executing and so on.

1.2 Goals

Now, let's define the goals for the project. Some of the „goals“ I listed above are really more *intentions*, as they aren't formulated sharp enough to be objectively measured or it's just their nature to be vague and abstract.

Keep in mind the bullet points below are not actually the requirements for the project, these will be listed further down (however, requirements can serve as goals in my opinion).

I envision a system which will be fed some basic information, like which tasks to run and the file which will be compiled and then do them automatically, one after the other and without me having to babysit the computer during the process and telling it when to do what. It might take a bit longer, but even if the time is long enough for me to grab a coffee, the system will be worth it for my purposes, as I work very frequently in \LaTeX .

Following is a list of bulletpoints which I want to accomplish with this application.

- Building a package in Python which builds \LaTeX documents and cleans up the working environment. This includes:
 - Running a \LaTeX -engine which compiles to PDF-documents
 - Running a bibliography-engine like biber or bibtex.
 - Running a glossary engine like makeglossaries
- Make the package installable
- Make the script be callable from everywhere in the directory structure. This means that the script should be run as a function no matter where it is called and / or installed.

2 Requirements Engineering

The following will contain a list of concrete requirements for the project. The list is not final for the moment as future updates and development processes may impact the requirements of the project.

The section 2.1 will contain a list of use cases which are the base for all requirements. The requirements themselves will be listed in section 2.2.

2.1 Use Cases

The following contains a list of use cases for the system.

All use cases result in the construction of a PDF-Document which contains full images and is free from length marks.

Removing the draft-option

The user calls the pipetex-application by entering the CLI-command for the application and supplying the name of the main \LaTeX file. This is done in the directory containing the \LaTeX file.

The application will make a copy of the file, remove draft option from the class definition and compile the \LaTeX file two times. The compiled pdf file will be moved to a separate directory and all generated auxiliary files will be removed from the directory.

Creating a bibliography

The user calls the pipetex-application by entering the CLI-command for the application and supplying the name of the main \LaTeX file and a flag which specifies the creation of a bibliography file. This is done in the directory containing the \LaTeX file.

The application will make a copy of the file, remove draft option from the class definition and compile the \LaTeX file. The application will call a bibliography-engine on the \LaTeX file and then compile the file again. The compiled pdf file will be moved to a separate directory and all generated auxiliary files will be removed from the directory.

Creating a glossary

The user calls the pipetex-application by entering the CLI-command for the application and supplying the name of the main \LaTeX file and a flag which specifies the creation of a glossary file. This is done in the directory containing the \LaTeX file.

The application will make a copy of the file, remove draft option from the class definition and compile the L^AT_EX file. The application will call a glossary-engine on the L^AT_EX file and then compile the file again. The compiled pdf file will be moved to a separate directory and all generated auxiliary files will be removed from the directory.

2.2 Functional Requirements

Now following is a compilation of all requirements for the application. The requirements will be grouped into three categories. The categories are ordered by importance:

- Must Have Requirement
- Should Have Requirement
- Nice to Have Requirement

The requirements have an ID, a name and a short description.

2.2.1 Must Have Requirements

The following contains all Must Have Requirement.

ID	Name	Description
MHR1	L ^A T _E X-Compilation	The application must be able to compile L ^A T _E X Documents using a T _E X-Engine.
MHR2	Draft-Mode-Removal	The application must be able to find and remove the draft option in the class definition
MHR3	Creation of bibliography	The application must be able to run a script to create a bibliography on the document
MHR4	Creation of glossary	The application must be able to run a script to create a glossary on the document
MHR5	Working-dir-agnosticism	The script must be callable from everywhere in the file system without the need of a copy in any specific location
MHR7	CLI-Compatibility	The application must be callable from a CLI
MHR8	Lossless-Execution	The application must not make changes to the main T _E X file. Instead, it should operate on a copy of the file to ensure nothing is lost.
MHR9	Working-Dir-Cleanup	The application should remove all generated auxiliary files once the compilation process is over.

Table 1: Functional Must Have Requirements

2.2.2 Should Have Requirements

The following contains all Should Have Requirement.

ID	Name	Description
SHR1	Meaningful-Exit-on-Error	The application should only quit if the L ^A T _E X-compilation fails. All other engines should log the error and continue with the execution
SHR2	Logging	The application should output log messages to the CLI.
SHR3	Changing-Output-Name	The application should provide the option to rename the PDF document which will be generated.

Table 2: Functional Should Have Requirements

2.2.3 Nice To Have Requirements

The following contains all Nice to Have Requirement.

ID	Name	Description
NtHR1	Colored-Logs	The logs printed to the CLI should be colored.
NtHR2	Log-File	The application should provide the option to redirect the logs to a logfile.
NtHR3	Verbose-Mode	The application should provide the option to print the output of the T _E X / Bib / Gloss engine to the CLI.
NtHR4	Changing-Logging-Level	The application should provide an option to increase or decrease the logging level.
NtHR5	Changing-Engine	The application should provide a way to change specific compilation engines if the user is not satisfied with the standard.

Table 3: Functional Nice to Have Requirements

Glossary

bibliography-engine The programm / script used to compile L^AT_EX documents to create a bibliography.. 5

glossary-engine The programm / script used to compile L^AT_EX documents to create a glossary.. 6

Must Have Requirement Thees requirements are part of the minimum usable subset of requirements to deliver a final application.. 6, 7

Nice to Have Requirement These requirements are desirable, but not important. If they are not present in the current version, the goal should be to implement them in the next. No patching needed.. 6, 8

Should Have Requirement Thees requirements are important, but not vital. If they are not fulfilled for a specific version, it is not vital, but needs to be patched.. 6, 8

List of Tables

1	Functional Must Have Requirements	7
2	Functional Should Have Requirements	8
3	Functional Nice to Have Requirements	8