Towards a better Bachelor Thesis

Pietje Bell,1234567

Venlo, 2024-06-03

Information Page

Fontys Hogeschool Techniek en Logistiek
Postbus 141, 5900 AC Venlo

Towards a better Bachelor Thesis
Examples of proper LaTeX usage for a Thesis
With general tips on file formats for diagrams and tables

Name of student:	**Pietje Bell**
Student number:	**1234567**
Course:	**Informatics - Business Informatics/Software Engineering**
Period:	**Februari-2024 - June-2024**
Company name:	**Company Name**
Address:	**Default Street 123**
Postcode, City:	**1234 AB, Amsterdam**
Country:	**The Netherlands**
Company coach:	**Coach Name**
Email:	**coach.email@example.com**
University coach:	**Johnny Makes You Feel Good**
Email:	**tutor.email@fontys.nl**
Examinator:	**Dr V. Strict**
External domain expert:	**A. Einstein**
Non-disclosure agreement:	**No**
word count:	**5365 words.**

Preface

Thanks to all the students that showed me there is always room for improvement.

In this 'thesis' I wrote down some of my annoyances in reading students work.

In several spots in this thesis I write 'I' or 'we', which is bad style in a normal thesis. However, because I consider me, myself and my fellow teachers main stakeholders in all of your writing, I allow myself to do that here. *Do not do that at home* (for your own thesis that is).

Pieter van den Hombergh,
Venlo 2024-04-26

Summary

I will not bore you with a summary. That would spoil the fun.

Summaries are overrated.

TLDR;

If that is not short enough:

For the wordies If you prefer word or some other “word processor”, read the improvement suggestions in section 4.2 on page 8 and section 4.2.1 on page 9.

For all Things to avoid when including code see chapter 5 on page 11 and forget the rest. That to the wordies too.

Statement of Authenticity

I, the undersigned, hereby certify that I have compiled and written the attached document / piece of work and the underlying work without assistance from anyone except the specifically assigned academic supervisors and examiners. This work is solely my own, and I am solely responsible for the content, organization, and making of this document / piece of work.

I hereby acknowledge that I have read the instructions for preparation and submission of documents / pieces of work provided by my course / my academic institution, and I understand that this document / piece of work will not be accepted for evaluation or for the award of academic credits if it is determined that it has not been prepared in compliance with those instructions and this statement of authenticity.

I further certify that I did not commit plagiarism, did neither take over nor paraphrase (digital or printed, translated or original) material (e.g. ideas, data, pieces of text, figures, diagrams, tables, recordings, videos, code, ...) produced by others without correct and complete citation and correct and complete reference of the source(s). I understand that this document / piece of work and the underlying work will not be accepted for evaluation or for the award of academic credits if it is determined that it embodies plagiarism.

Name:	**Pietje Bell**
Student Number:	**1234567**
Place/Date:	**Venlo, 2024-06-03**

Signature:

If you add a file called `authenticity_signature.png` to root folder of your build, that will be included as signature here, instead of this text block.

Contents

List of Figures

List of Tables

Glossary

Notation	Description	Page List
bounding box	A virtual box that extends from the minimum to the maximum coordinates of an image.	8
DRY	Don't Repeat Yourselves	1
MineCraft	A Game known for its blocky images.	8, 9
persona	A fictional person for a specific role	1

Acronyms

Notation	Description	Page List
JPEG	Joint Photographic Experts Group	7
PGF	which stands for portable graphics format	14
PNG	Protable Network Graphics	7
SDK	Software Development Kit	22
TLDR;	Too Long, Did not Read	2
UML	Unified Modeling Language	8

Introducing oneself properly is always hard.
Anonymous

1 Introduction

Writing documentation is often considered a chore. But actually reading student reports is even worse. Certainly if the student is to wordy, sloppy, repeats every other section and so on.

The Casus Belli, in this case, is that I have been an examiner of a lot, not to say most students in the informatics courses at Fontys Hogeschool Venlo. There I have to read 22 reports, from 14 students that I coach and 8 others where I am the examiner. That incentivised me to write down some advice. Here you have it.

1.1 Do not bore us to death!

The worst thing that can happen to me, the poor person that has to read your whole report from front page until the last page before the appendix, is that I get the impression that some one is trying to hum me asleep. I have no clue what or who caused this, but many students have the habit to explain in the first paragraph of the chapter what they are going to tell in that chapter. Why? Why Waste my time and your time? Do you think that improves your grade? Because I have to read the whole damn thing anyway, why take the excitement of reading away. Would you read a book or watch a film if the first paragraph of each chapters gives everything away?

1.2 Things to avoid

To keep the reader awake, and more importantly: interested, and appreciative of your work you should:

Stay DRY DO NOT repeat stuff.DRY

Good titles Think of good chapter and section titles. We read and use the table of contents. If the chapter and section titles are good, they help explain the structure of the report, without any extra boring help.

Good titles

Be brief You do not get paid per written word, nor are we paid per word **read**.

brief

Use a storyline Both in your report and in your presentation. If you invent a *persona* anyway, use him or her as the protagonist to tell the store, and use him/her to explain stuff. The story need not necessarily be true to the actual chronology of the time spent on your bachelor project, but should be a logical story, in which you take the reader along to explain your reasoning, the decisions you made and why, etc.

persona

Your protagonist may not be useful in all the parts, but might be very handy to explain the problem, and assignment and how he or she can use the fruits of your labor.

Start with the **company** and every detail of that company that is relevant to the project. Then continue with the **context** of the problem, exactly as much as you need to explain the next logical thing: the **problem** that you have been asked to solve. Followed by the **assignment**.

If in any paragraph that you wrote you or a reviewer notices that you have to explain things that could have been explained in an earlier part, do not repeat any of said part, not even by saying

'as you can/have read in ..', but instead revise the earlier part so the context, the problem, the assignment etc. can be understood in one flow.

As an example, If you need to explain what a typical customer will do with the product, then we expect that you have explained the product and customer in the company description or context.

You should assume that the reader is a single-pass compiler (like LaTeX a C- or Pascal compiler). What has not been defined before cannot be used.

Forward references (like: "as you will see in ...") are **not allowed**. It is a waste of words, in particular when the structure, and thus the table of contents is any good.

In this way, you will keep the reader in his flow because he does not have to page forward or backward, and if the story is short enough, the reader will be able to pull through without being bored to death.

Also: By NOT repeating stuff, you have no chance of inconsistencies, because every paragraph is the only source of its truth.

1.3 TLDR;

TLDR; (Too Long, Did not Read)

In the remainder of this document, you will see some tips and tricks to use when you write your report in LaTeX, but the above and some things in the use of graphics also apply when you use **Word** or some other text processing application. The quality of your report should not suffer from your choice of tools. So read those chapters as well. In particular when it is about adding pictures, listings, and tables.

1.4 Use a better technology

One of the standards for documentation in open source and hence in Linux land is LaTeX, a text processing package. LaTeX is available for free and available with all Linux distributions and can be installed on Windows and Max OSX just as easily.

TeX is the machinery of LaTeX and was defined in the TeX book [TeXbook, 1984] and implemented by Prof. Donald Knuth. LaTeX is a (nowadays HUGE) set of macros built on top of that. LaTeX in its initial form is described by Leslie Lamport in [Lamport, 1986]. If you like your book thick, try the LaTeX companion [Companion, 2004].

The web is also a very good source of LaTeX documentation. A good starting point is `http://en.wikibooks.org/wiki/LaTeX`, useful for beginners and pros alike. The help pages on overleaf are also quite good.

This is a simple multi-part document. Its purpose is to show how easy it is to create a multi-part document, one that, for instance, can be worked on simultaneously by several authors. Note that most of the settings for this document are set in the file `configuration/thesis_config.tex`. Look in that file too.

You are kindly advised to keep your lab logs in simple text files. These can be turned into latex files easily, which can be used to produce a nice-looking report.

1.5 Some hints to start with

Sometimes things do not work out the way you think. LaTeX interprets some character codes in its way. Things like dollar signs or even underscore are special. LaTeX sources are littered with accolades or *curly braces* if that's the way you call them. They are special too. So here is some advice:

funny file names

Do not use *funny file names*. That is: stick to ASCII filenames without spaces or even underscores. These will lead you only into trouble. If you want to keep things portable, don't use camel case (like in JavaClassNames) either, because some OS-es do not distinguish between upper and lower case. You may of course break this rule if the files are program things like Java source files.

1.5.1 Hints for informatics (use version control)

In software projects, versioning is important. LaTeX and GIT work nicely together here.

To keep these version codes up to date, first check if your LaTeX files compile, then add and commit them and do your final LaTeX run.

2 | Motivation

To write a simple document, like a letter, a word processing package is just fine. To automate the creation of a document this is a bit harder. Using a word processor to automatically create a consistent document from unrelated sources is difficult at best, if doable at all.

However, as long as the only thing that those unrelated sources should produce are simple ASCII documents, things get much more doable. It is like comparing HTML to Microsoft Word documents. The power you have in defining the layout of a document in HTML (combined with css) with a "simple" ASCII document is almost as powerful as what you can do with Word. But now try writing the same thing with a (self-written) program. You know it is easy in HTML (certainly if you ever did some programming with e.g. PHP), but producing a Word document with a program is hard work[1].

If the document should include complex things like mathematical formulas that are laid out properly, it becomes difficult in HTML too.

Enter LaTeX.

2.1 Multi-person literal work

One of the much-overlooked advantages of LaTeX(and of any multi-file source code application like Java projects) is that the fact that you can split up the document into multiple but still coherent parts. This fact allows you to work on one final document with multiple persons as team members. This is exactly what the informatics teachers do when writing a report for external evaluation.

This makes LaTeX almost ideal for project work in which several authors have to contribute to the final product and where source files are shared using a repository. Even more so in cases where you want to include (part of) program source code into the document to explain or show certain implementation aspects. Such source code is not copied and pasted but instead read on the fly from the original file(s) during text processing. This allows you to always have the most up-to-date version.

As this sample is a multipart document, it can be used as a reference or a start for your document.

2.2 Good documentation and online support

Most documentation can be found online. Overleaf has very good documentation. Overleaf has as its main product a build service for LaTeX documents and many students are using and have used it successfully.

The original sources of the LaTeX documentation are Lesly Lamports LaTeX Book Lamport [Lamport, 1986], and expanded on that is the LaTeX Companion [Companion, 2004].

[1] It becomes easier in a modern version of word processing packages, they tend to use XML as their internal document format

3 | Mathematics to show off

Here, you see how a mathematical equation can be generated in line, for instance, $f(x) = \frac{1}{1+25x^2}$. The `$`-symbols enclose the formula. As a so-called displayed formula, it would look like

$$f(x) = \frac{1}{1+25x^2}.$$

It is customary that mathematical functions are *not* set in math italics, so LaTeX has the basic ones pre-defined; you should use the commands `\cos`, `\exp`, etc. to get $f_1(x) = \cos x$, $f_2(x) = -e^{-xt}\sin^2 x$, etc.

Here, I use some of my commands defined above: I like $\varepsilon = \varepsilon$ better than the default ϵ. A partial derivative (with 2 arguments) would be obtained as follows. If $f(x,y) = x^2y^3$, then

$$\frac{\partial f}{\partial x} = 2xy^3, \quad \frac{\partial f}{\partial y} = 3x^2y^2.$$

3.1 Sums and Integrals

When you say "capital sigma," you probably did not really mean Σ, but rather a summation symbol. You would get that as in

$$\sum_{i=0}^{\infty} r^i = \frac{1}{1-r} \quad \text{for all } |r| < 1.$$

Finally, we have

$$\int_0^1 \sin(2\pi x)\,dx = 0$$

and

$$\iint f(x)g(y)\,dx\,dy = \int f(x)\,dx \int g(y)dy.$$

Here, `\,` gives a small space, while `\!` forces things closer together; you have to work on the proper spacing for integrals, as LaTeX does not understand, what is going on.

3.2 Matrices in LaTeX

A matrix $A \in \mathrm{R}^{m \times n}$ could be defined by

$$A = \begin{pmatrix} 11 & 12 & 13 & \cdots & 1n \\ 21 & 22 & 23 & \cdots & 2n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m1 & m2 & m3 & \cdots & mn \end{pmatrix}$$

Here, the word `dots` in the commands stands for an ellipsis (i.e., three dots) placed horizontally in the center (`\cdots`), vertically (`\vdots`), or diagonally (`\ddots`); what is not mentioned is `\ldots` for horizontal dots at the baseline. Use the baseline or central version as appropriate, for instance

$$\begin{aligned} a_1, a_2, \ldots, a_n \quad &\text{and not} \quad a_1, a_2, \cdots, a_n, \\ a_1 + a_2 + \cdots + a_n \quad &\text{and not} \quad a_1 + a_2 + \ldots + a_n, \end{aligned}$$

Some more comments on the matrix are needed, I suppose: The `\left(` and `\right)` create the variable-sized parentheses around the actual array of terms. You can also use `\left[` and `\right]`, or `\left\{` and `\right\}` in other situations. The actual array arrangement is accomplished by the `array` environment. The argument `ccccc` indicates that there are five columns and you want the entries centered ("c"), other options are left ("l") and right ("r"). Notice how `&` separate columns and `\\` the rows.

Here is another matrix example. A matrix multiply used with 3D graphics:

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & X \\ 0 & 1 & 0 & Y \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4 | Graphics as easy as pie

LaTeX, combined with pdf in `pdflatex`, supports the following graphic file types: pdf, png, jpeg or jpg and gif in that order of preference. Using the vector format pdf gives the added benefit that the graphic file can be scaled up and down without loss of quality. If you want to include bitmaps, try to get them in *PNG (Protable Network Graphics)* format which is open and patent-free. It has the advantage over jpeg or jpg that is is loss-less, so you do not see any artifacts if you blow them up in your inclusion. Converting back from jpg to png is useless because the damage is already done in the jpeg format. *JPEG (Joint Photographic Experts Group)* is excellent for photographs. That is also what its name says. For all the bitmap formats: try to get them at the intended size with a resolution of 300 dpi for printouts. 75 dpi is acceptable for screen reading.

PNG

JPEG

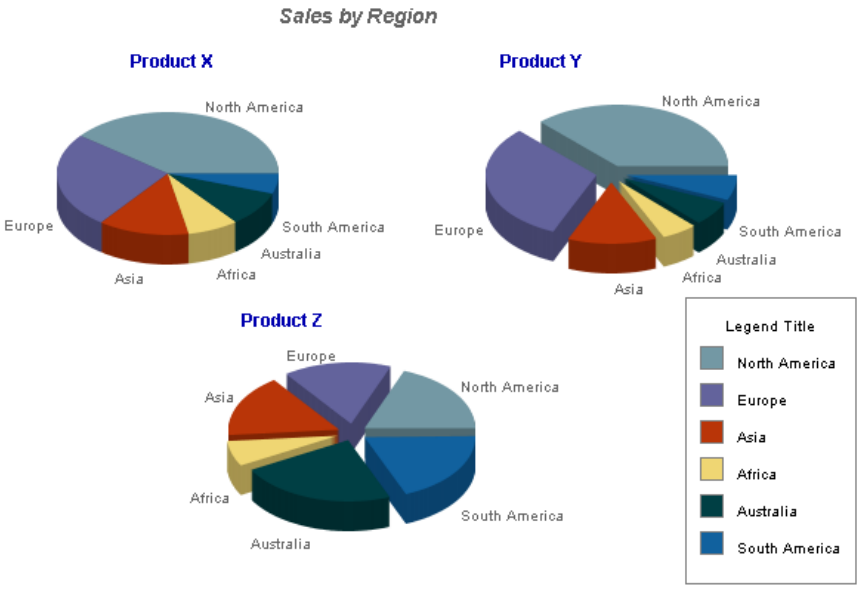


Figure 4.1: Stolen from the net. Google for a pie chart...

Many graphics packages can produce pdf files. Embedded postscript (file extension .eps) are also a good candidate, after converting them to pdf with the epstopdf tool. By the way: the native format of Adobe Illustrator (ai) is similar enough to eps, so that too can be processed with `ps2pdf`. Programs like *Visual Paradigm* can produce pdf files too. Sometimes openoffice can lend a helping hand. For instance, by cutting and pasting a Windows graphics into a single page OOdraw drawing, you can produce a very usable pdf file.

Bitmap file types like png and jpeg take up a lot of space in your final pdf document. Bitmap files take even more space if encoded into a pdf file. As much as possible, stick to pdf (if necessary derived from svg or eps files).

4.1 A png example

If latex cannot fit the diagram on this page (page 7), then you may find the diagram as figure fig. 4.1. And as you can see, you can easily reference pages and images.

4.2 PDF from a UML package

If the documentation you write is a design document of some software package, you may want to include design diagrams. No software engineering without a *UML (Unified Modeling Language)* diagrams.... You can see one generated with "dia", a vector drawing program that understands the UML in figure 4.2 on this page. This diagram is 'wrapped' in a **`wrapfigure`** environment, so the text may flow around it

UML

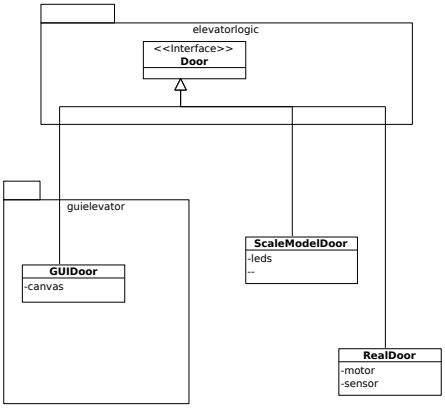


Figure 4.2: A class diagram made with Visual Paradigm

The diagram is not very sophisticated but shows an example of a vector format file included via an eps→ pdf conversion by `epstopdf`.

Open source programs like Umlet, but also commercial ones like Visual Paradigm are also able to produce vector format graphics files. And sometimes it is helpful to add a box that is a bit bigger than the picture you want to include. This ensures that the *bounding box* does not cut off any lines you want in your picture. Sometimes it is necessary to give these tools a helping hand with Inkscape, that is, do some tinkering to get all the details right. Or with **`pdfcrop`** which typically comes with your LaTeXinstallation.

bounding box

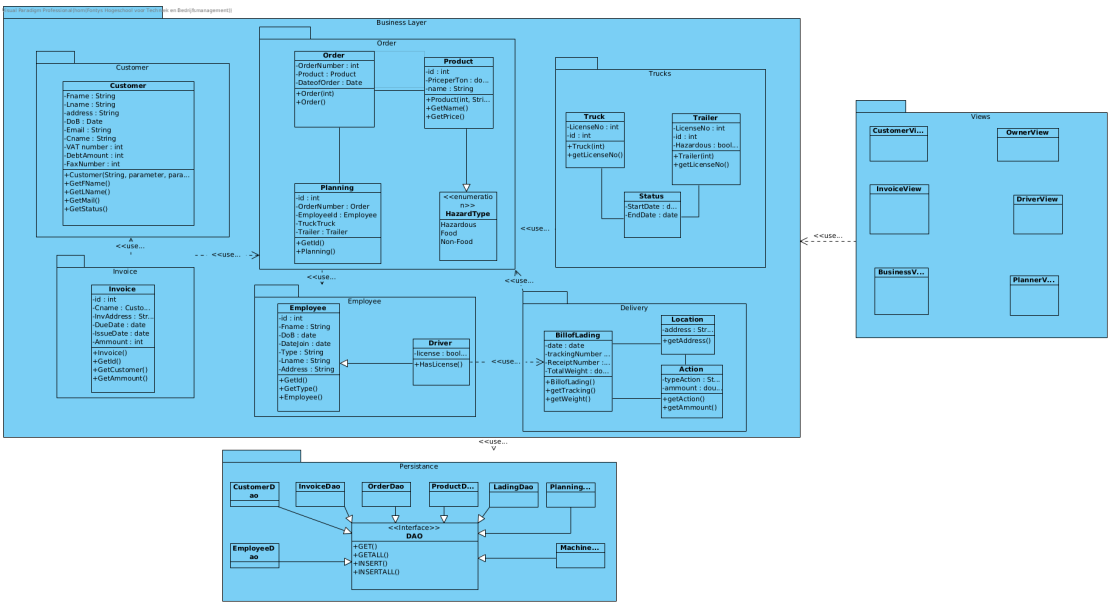


Figure 4.3: You have been warned, NO PNG

The picture above is wrong on many fronts.

- it uses PNG as the image format, whereas a vector format is provided by Visual Paradigm. However, when you like a pixellated world such as in the game MineCraft, you will feel right at home if you zoom in a bit.
- The arrows point in all directions, which is bad style. A proper UML class diagram should be stylish to improve comprehensibility, as it is recognizable at a glance. Think of

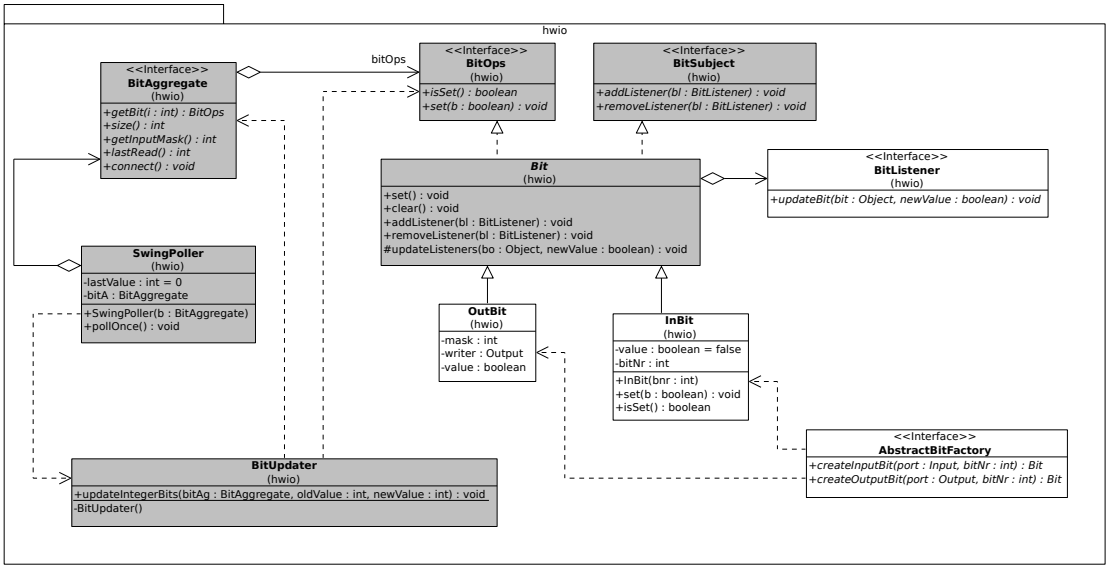


Figure 4.4: UML made with Visual Paradigm, exported to SVG and then adapted with Inkscape and in the end exported to pdf

Da Vinci's Vitruvian man.

- It has the VPP blues, that is the colors used are completely non-functional, and also impair readability because the used color lowers the contract.

Look on the next page how it is done properly.

4.2.1 Stylish UML Class diagram

Figure fig. 4.5 on the next page above has style and uses functional colors which are easily explained with a legend.

Good style in a class diagram means:

- There is always a direction. Arrows or triangles. They express dependencies.
- Inheritance in the vertical direction only.
  - Any line that leaves at the top of a class box implies that this implements (dashed line) or extends (solid line), without looking where the line ends.
  - Any triangle that enters at the bottom says that this class is extended or implemented for interfaces.
- All other relationships enter or leave at the left or right edge of the box, so you know things are used or this class is used/known.
  - If the local name is relevant, then that name is at the side of the using class, like the barcode at the PhysicalProduct which identifies a Product in the system. Internally the barcode is a simple number like an `int` or a `long`.

Other than style there are other advantages to this kind of diagram.

- You can zoom in as much as your viewer allows without getting a *MineCraft* world or worse.

MineCraft

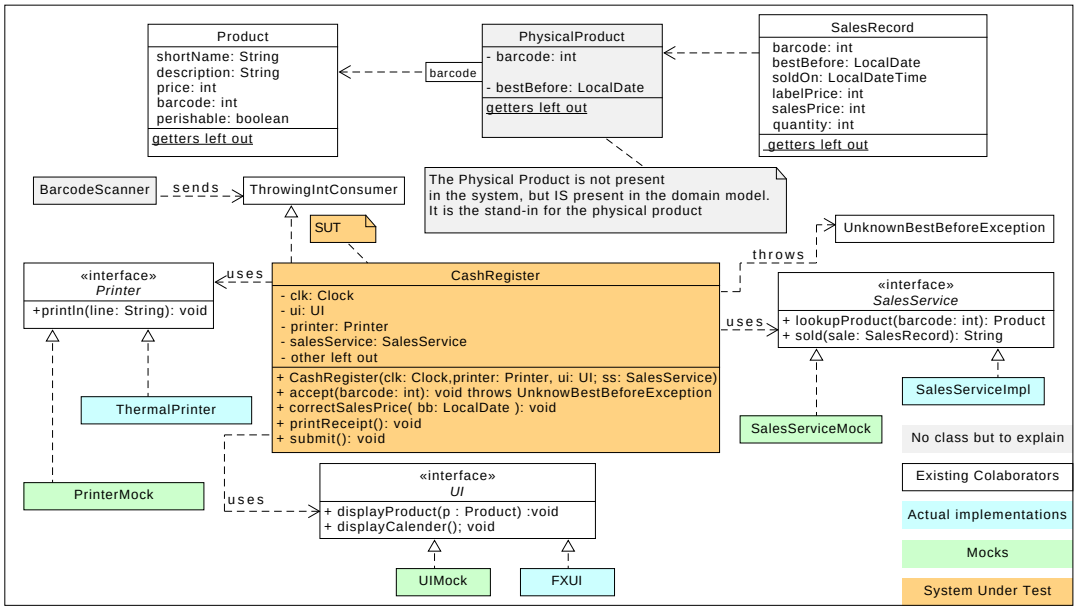


Figure 4.5: Vector format (exported as pdf), with functional color and a legend, Made with Umlet

- A reviewer (the examiner, your coach or someone working with your documents) can select the texts in the diagram and can mark it. You too could do that to embellish the diagram.
- There is no way you can do that nicely with a pixel based format like PNG or JPEG.

This last diagram has been made with UMLet.

5 Listings and Code Documentation

Just as much as you can have bad style when including diagrams, the same can happen if you want to show code snippets.

- NEVER use screenshots from your IDE and even less so, screenshots with a dark theme. Typically your report is read on a white background and the contrast with dark themes is really really bad.
- In the times you had to print your report I would ask the student who soaked the page with black ink.
- Big diagrams tend to produce big files and thus humongous PDF files.

To make this work, you use the `package` listings.

Note that the line numbers on the right hand border are the line numbers in the included sources.

5.1 Source code

The most simple case: include the whole thing with a command like
`\lstinputlisting[language=java]{code/Hi.java}`

Listing 5.1: Mandatory first program

```java
/**
 * The obiquiteous Hello World program, a Java variant.
 * @author Pieter van den Hombergh (879417)
 */
public class Hi{
    @Override
    public String toString(){
        return "Hello world";
    }

    public static void main(String[] args){
        System.out.println(new Hi());
    }
}
```

Sometimes it is useful to include just a part of a file, for instance when explaining things. Like what line 11 is all about.
`\lstinputlisting[language=java, firstline=11,lastline=11]{Hi.java}`

Listing 5.2: this is the start of a Java program.

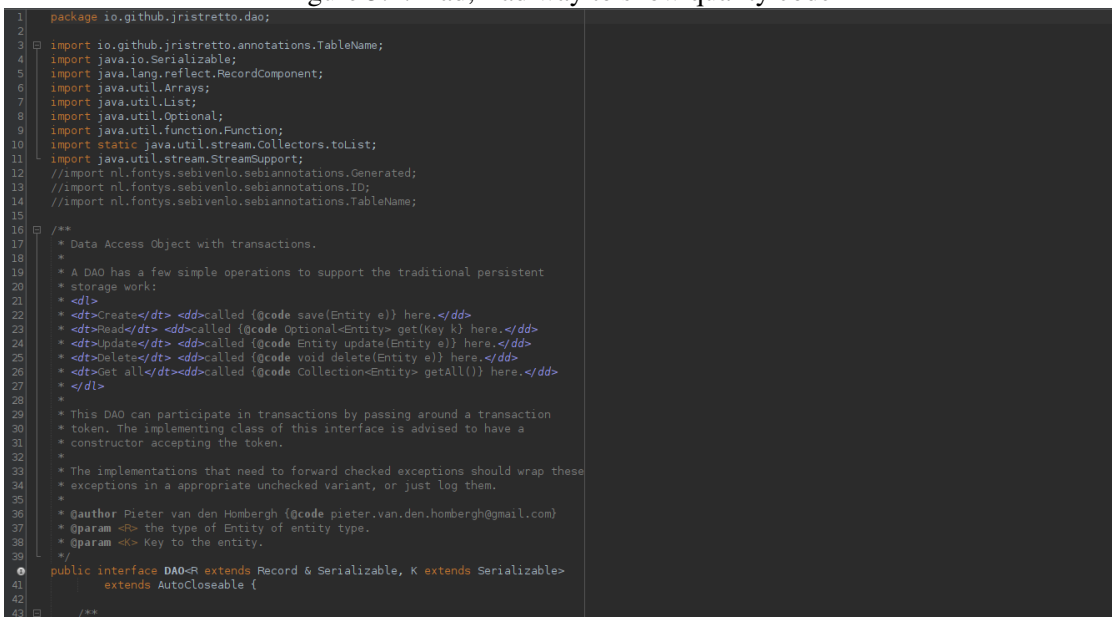
```java
    public static void main(String[] args){
```

The snippet below shows a similar code as the snippet above. Again, you can zoom in and select text.

Listing 5.3: The proper way to show your code.

```java
package io.github.jristretto.dao;
```

Figure 5.1: Bad, Bad way to show quality code



```
package io.github.jristretto.dao;

import io.github.jristretto.annotations.TableName;
import java.io.Serializable;
import java.lang.reflect.RecordComponent;
import java.util.Arrays;
import java.util.List;
import java.util.Optional;
import java.util.function.Function;
import static java.util.stream.Collectors.toList;
import java.util.stream.StreamSupport;
//import nl.fontys.sebivenlo.sebiannotations.Generated;
//import nl.fontys.sebivenlo.sebiannotations.ID;
//import nl.fontys.sebivenlo.sebiannotations.TableName;

/**
 * Data Access Object with transactions.
 *
 * A DAO has a few simple operations to support the traditional persistent
 * storage work:
 * <dl>
 * <dt>Create</dt> <dd>called {@code save(Entity e)} here.</dd>
 * <dt>Read</dt> <dd>called {@code Optional<Entity> get(Key k} here.</dd>
 * <dt>Update</dt> <dd>called {@code Entity update(Entity e)} here.</dd>
 * <dt>Delete</dt> <dd>called {@code void delete(Entity e)} here.</dd>
 * <dt>Get all</dt><dd>called {@code Collection<Entity> getAll()} here.</dd>
 * </dl>
 *
 * This DAO can participate in transactions by passing around a transaction
 * token. The implementing class of this interface is advised to have a
 * constructor accepting the token.
 *
 * The implementations that need to forward checked exceptions should wrap these
 * exceptions in a appropriate unchecked variant, or just log them.
 *
 * @author Pieter van den Hombergh {@code pieter.van.den.hombergh@gmail.com}
 * @param <R> the type of Entity of entity type.
 * @param <K> Key to the entity.
 */
public interface DAO<R extends Record & Serializable, K extends Serializable>
        extends AutoCloseable {

    /**
```

```
import io.github.jristretto.annotations.TableName;
import java.io.Serializable;
import java.lang.reflect.RecordComponent;
import java.util.Arrays;
import java.util.List;
import java.util.Optional;
import java.util.function.Function;
import static java.util.stream.Collectors.toList;
import java.util.stream.StreamSupport;

/**
 * Data Access Object with transactions.
 *
 * A DAO has a few simple operations to support the traditional persistent
 * storage work:
 * <dl>
 * <dt>Create</dt> <dd>called {@code save(Entity e)} here.</dd>
 * <dt>Read</dt> <dd>called {@code Optional<Entity> get(Key k} here.</dd>
 * <dt>Update</dt> <dd>called {@code Entity update(Entity e)} here.</dd>
 * <dt>Delete</dt> <dd>called {@code void delete(Entity e)} here.</dd>
 * <dt>Get all</dt><dd>called {@code Collection<Entity> getAll()} here.</dd>
 * </dl>
 *
 * This DAO can participate in transactions by passing around a transaction
 * token. The implementing class of this interface is advised to have a
 * constructor accepting the token.
 *
 * The implementations that need to forward checked exceptions should wrap these
 * exceptions in a appropriate unchecked variant, or just log them.
 *
 * @author Pieter van den Hombergh {@code pieter.van.den.hombergh@gmail.com}
 * @param <R> the type of Entity of entity type.
 * @param <K> Key to the entity.
 */
public interface DAO<R extends Record & Serializable, K extends Serializable>
        extends AutoCloseable {
```

5.2 Makefiles

You can also include make files. Note that makefiles have a peculiar syntax. *Spaces and tabs in Makefiles* are meaningful. That's why I made them show up in the next listing with the command

Spaces and tabs in Makefiles

```
That's why I made them show up in the next listing with the command\\
```

```
\lstinputlisting[firstline=65,lastline=67]{chapters/ch05_codelisting.tex}
\lstset{showspaces=true,showtabs=true}
```

Spaces show up as ␣, tab characters as an extended version of the same thing (␣␣␣␣).

As can be expected, spaces and tabs have no special meaning in makefile comments, the lines starting with a hash (#) sign. If you want to know more about Makefiles try google `man make` on a decently installed Linux box.

The make file for this entire document looks like this:

```
# Makefile for LaTeX.
.SUFFIXES=
.SUFFIXES=.tex .pdf

LATEX=pdflatex -recorder -output-directory=out -synctex 15 -shell-escape
BIBTEX=biber --output-directory=out --input-directory=out
DEPENDENCIES=partials/* configuration/*.tex chapters/*.tex images/*.pdf images/*.pdf images/*.jpg appendix/* ./*.png tables/*.* wordcount.txt

all: main.pdf

fast: fast.pdf

main.pdf: main.tex out $(DEPENDENCIES)
	$(LATEX) main.tex
	$(LATEX) main.tex
	$(BIBTEX) main
	makeglossaries -d out main
	$(LATEX) main.tex
	$(LATEX) main.tex
	ln -sf out/*.pdf .

fast.pdf: main.tex out $(DEPENDENCIES)
	$(LATEX) main.tex
	ln -sf out/*.pdf fast.pdf

out:
	mkdir -p out/chapters out/partials out/appendix

wordcount.txt: chapters/*.tex
	texcount -inc -sum -1 chapters/*.tex > wordcount.txt

clean:
	rm -fr out *.aux *.log *.synctex *.bbl *.lof *.toc *.out *.bcf *.blg wordcount.txt
	rm -f main.pdf
```

6 | Drawing in LaTeX

There are quite a few developers who add useful packages to the TeX world. One of them is Till Tantau of the "Technische Üniversität" in Berlin Germany. He produced the package called PGF (which stands for portable graphics format), which stands for portable graphics format.

The pgf package contains the macro tool `tikz`, that allows you to draw quite nice graphics with just a few commands. Like this small ellipse: ⬭

The pgf manual has many nice examples that may prove usefull, certainly if you want to use the package beamer[1] to create professional looking pdf presentations with LaTeX.

A simple figure from the pgf manual:

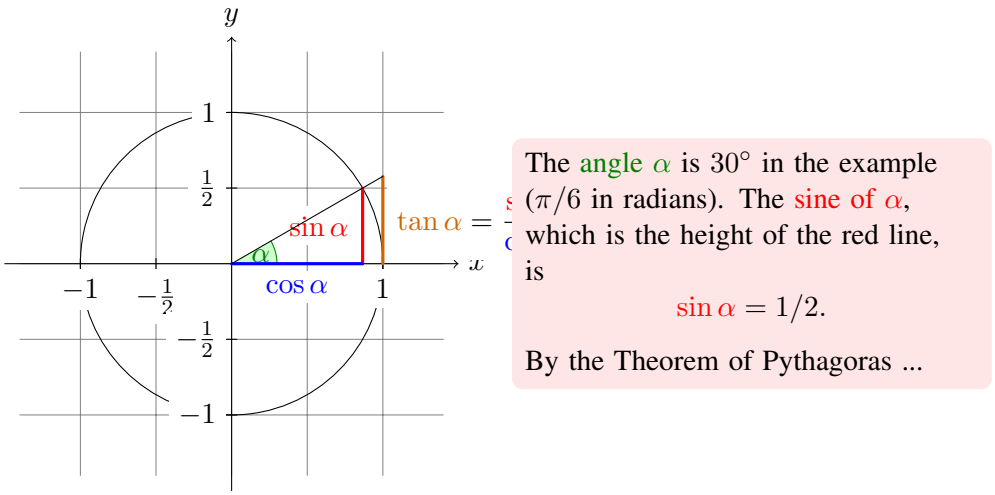


Figure 6.1: A drawing taken from the pgf manual/tutorial by Till Tantau

[1] also by Till Tantau

Never start reading a difficult book at the wrong side.
It makes you feel stupid.
Private experience

7 | Starting yourselves?

This document is indeed a bit of a showcase, but there is more to it. In essence, most documents are mainly texts. And those plain texts take mainly typing and not much more. The minimal, hello world style LaTeX file is not much longer[1] then the C classic.

Listing 7.1: Hello LaTeX world

```
\documentclass{report}
\begin{document}
\chapter{Hi}
\pagestyle{empty}
  Hello world.


 \[
  E = \int_{t0}^{te} Pdt
 \]
\end{document}
```

And the pictures, well they are made with other packages, and as long as those can produce a supported format, you can use them. TeX and LaTeX have an own drawing language, but explaining that would blow up this space. There is a lot of documentation available on TeX and LaTeX and I could recommend some good books on it. But you need not run off to the nearest book shop. Lots of documentation is on the Net, just as the TeX program suite itself.

Look for instance at http://www.ntg.nl for the Dutch TeX user group and http://www.dante.de for the German user group. They are both very alive and kicking.

A very good starting point nowadays is http://en.wikibooks.org/wiki/LaTeX/

7.1 My own definitions

Standard LaTeX output looks a bit dull but are nicely formatted nonetheless. If you want to make the most of your own style for your own or your projects documentation, put your style definitions into a separate file. That way you can keep all definitions of style and *macros* in one spot. This works especially nicely in an multi part document, so the other files (and authors) can concentrate on content.

macros

You can *define* your own macros in LaTeX. As an example a macro, `\define`, which I use to let words stand out at the outer margin. I use it at the first use of a word or concept in the text, to aid the reader in finding the definition. Of course this macro could be extended to put the word into an index. Which makes it a nice exercise.

define

[1] Counting headers too!

The macro is defined as follows:

Listing 7.2: The macro is defined as follows

```
% This macro '\define' puts the argument in em
% and in boldface in the margin.
\newcommand{\define}[1]{% 1 argument
  \mbox{}{\textit{#1}}% italics or em
  \marginpar{\raggedright% no adjust
    \bfseries\hspace{0pt}#1}% bold
} % end of macro
```

and is used as `\define{new word}`.

Anyway, in modern times you expect to find the examples somewhere convenient, so here you go. You can find this example in github at https://github.com/homberghp/bachelor_thesis_tips

Everything should be made as simple as possible, but not simpler.
Albert Einstein

8 | Citing is simple

Adding proper references and citations to you document can be a real burden. Not so in LaTeX where you can use the facilities of a kind "database" and many entries available on the web that be added to that database.

This database can be one file, but just as well a set of files, which you use to organize you bibliography. The files with these data are called .bib files.

8.1 BibLaTeX

Bibtex is the traditional way of using citations and creating a bibliography, but its role has all but been taken over by the more modern BibLaTeX. Both are equally simple to use for the most common use. For each reference that you want to use in your report you should have a definition like below. Look in the file `references.bib` for more examples.

For instance if your bib contains this entry for `Nobody`:

```
@misc{ Nobody06,
   author = "Nobody Jr",
   shorthand={Nobody},
   sortname={nobody},
   title = "My Article",
   year = "2006"
}
```

Then citing Nobody [Nobody] is easy as pie:`\parencite{Nobody06}`

With textcite it looks like Jr [Nobody], with cite like [Nobody], and with parencite [Nobody].

Then you need to add one compilation step to your normal workflow:

```
$ pdflatex myarticle
$ biber myarticle
$ pdflatex myarticle
$ pdflatex myarticle
```

You can of course easily add that to the makefile introduced in section 5.2 on page 12.

BibLaTeX is the standard you want to use if preparing an article of a journal or magazine. The journal typically also prescribes a specific bibliography style (defined in a .bst file), which is most likely already define for or by that journal.

8.1.1 BibLaTeX and Biber

A bit more modern then bibtex is biblatex, which has a much simpler definition format for bst files. There is a separate **biber** program to do the processing instead of the bibtex run. I have used biber in this version of this latex sample. [BibLaTex project, 2024].

References

Mittelbach, Frank and Michel Goossens (1993). *The LaTeXCompanion Second Edition*. Addison-Wesley.

Einstien, Albert (1905). "Zur Elektrodynamik bewegter Körper. (German)". In: *Analen der Physik* 322.10, pp. 891–921. DOI: http://dx/doi.org/10.1002/andp.19053221004.

Multimedia LLC (1999). *MS Windows NT Kernel Description*. Accessed: 2010-03-09.

Stormstout, Chen (no date). *Beer-making Blog*. Accessed: 2018-10-29. URL: www.stormstout-brewery.com/process.

Figure 8.1: No labels that are easily matched with the references in the report text.

8.2 What style to choose

Avoiding complexity is always a good idea. If you want to support the reader in the easiest and least complex way for your as a author, choose a numeric citation style, like the ieee style. But hey, the people at IEEE are a betas, the same as a software engineer.

It appears that Fontys Venlo promotes the so called Harvard style, but that has three issues:

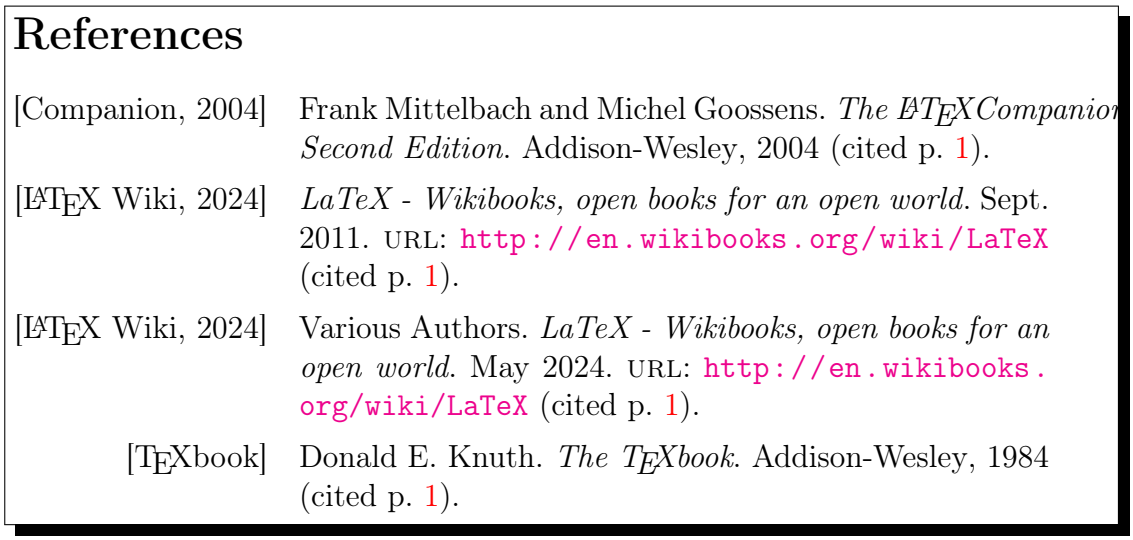
1. It is not well defined.
2. Finding a reference in the bibliography is not trivial, because the style does not use the label. in the bibliography, making it less useable or at least reader friendly.
3. It is NOT what Fontys Venlo uses internally.

The official 'harvard'-like style that is promoted by the the slides on canvas produces the bibliography in fig. 8.1, which is actually authoryear.

The style that Fontys uses internally can be emulated with the settings used in this report and with a bibliography that uses the fields `shorthand` and `sortname` like below in listing 8.1 to give the author control on both what is used in either citation and the bibliography and also the sorting applied in the bibliography. Since the shorthand defines what is shown in the citation, it also lets you control per citation what is show, title, author, or some other mnemonic name. If you want a short hand for the Patterns Book by Gamma, Helm, Johnson, and Vlissides the textcite looks like this: Gamma et al. [GoF 1994], and by following the link you can have a look in the references to see what it makes of the label and the reference.

Listing 8.1: Using `shorthand` for label and `sortname`for sorting

```
@BOOK{latexcompanion,
  author = "Frank Mittelbach and Michel Goossens",
  shorthand ={Companion, 2004},
  sortname ={companion},
```



References

[Companion, 2004] Frank Mittelbach and Michel Goossens. *The LaTeXCompanion Second Edition*. Addison-Wesley, 2004 (cited p. 1).

[LaTeX Wiki, 2024] *LaTeX - Wikibooks, open books for an open world*. Sept. 2011. URL: http://en.wikibooks.org/wiki/LaTeX (cited p. 1).

[LaTeX Wiki, 2024] Various Authors. *LaTeX - Wikibooks, open books for an open world*. May 2024. URL: http://en.wikibooks.org/wiki/LaTeX (cited p. 1).

[TeXbook] Donald E. Knuth. *The TeXbook*. Addison-Wesley, 1984 (cited p. 1).

Figure 8.2: Labels same as with the citations.

```
   title = "The {\LaTeX }Companion Second Edition",
   publisher = "Addison-Wesley",
   year = 2004
}
```

With these addition, and the biblatex configuration in listing 8.2, you get the more pleasing bibliography in fig. 8.2 which has back references as an additional bonus. These will not work in the image above, but does work with this Mittelbach and Goossens [Companion, 2004] citation.

A screenshot (still ugly, but useful for once) in fig. 8.3 on the next page shows what that looks like in the pdf-viewer **evince** on a ubuntu box.

Listing 8.2: Using `shorthand` for label and `sortname`for sorting

```
%% citation and bibliography style.
%% This style depends on 'sortname' and 'shorthand' fields in the bib file.
%% This gives you control on what is shown in the labels.
%% If you ommit the shorthand, the styles fall back to a numeric label
\usepackage[
backend=biber,
hyperref=true,
backref=true,
]{biblatex}
\DefineBibliographyStrings{english}{
  backrefpage={cited on p.},
  backrefpages={cited on pp.}
}
% end citation bibliography style.
```

As most LaTeX documentation, with a complete installation you can use the command `texdoc biblatex` to be presented with the documentaton of **biblatex** in this example in the default

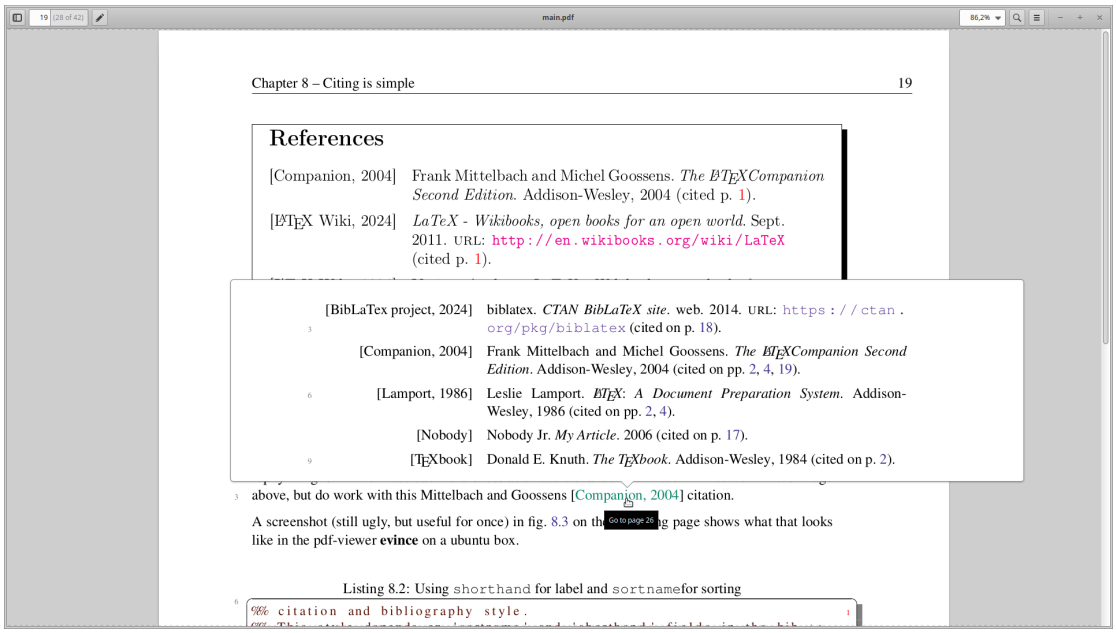


Figure 8.3: Citations called out when pointed to in evince.

pdf viewer.

9 | Tables, tables

Although LaTeX has its strengths, it is not particularly time efficient when you want to design nice tables.

A spreadsheet (excel, libreoffice calc) may serve you better than having to hand craft tables.

To show what handcrafting can mean you might want to take a look at the sources of the next table.

Table 9.1: Research results summarized. By Torben Westphalen

Criteria	Alt 1	Alt 2	Alt 3	Alt 3
.NET 6.0 Support	✅ Yes	✅ Yes	✅ Yes ⚠ Since 2022-08-25	✅ Yes
Documentation	✅ Online Documentation ✅ Examples	✅ CHM Documentation ✅ Examples ✅ YouTube Tutorials	✅ Code Documentation ✅ Examples ✅ Tutorials	✅ GitHub Documentation
Security	✅ Signed ✅ Sign & Encrypt ✅ Certificates	✅ Signed ✅ Sign & Encrypt ✅ Certificates	✅ Signed ✅ Sign & Encrypt ✅ Certificates	✅ Signed ✅ Sign & Encrypt ✅ Certificates
License types	✅ Single Developer License ✅ Branch License (=postal address)	✅ Single Developer License ❌ Branch License	✅ Single Developer License ❌ Branch License	✅ Branch License
Support	✅ inc. 12 months Top Level Support ✅ Support and update renewal 12 months (price depending on license)	✅ Dedicated support team ⚠ Support is valid for 3 years and can be extended in 1-year steps afterward	✅ Includes first-year maintenance package ⚠ Includes only 15 support incidents	✅ No support nor guarantees

Continued on next page

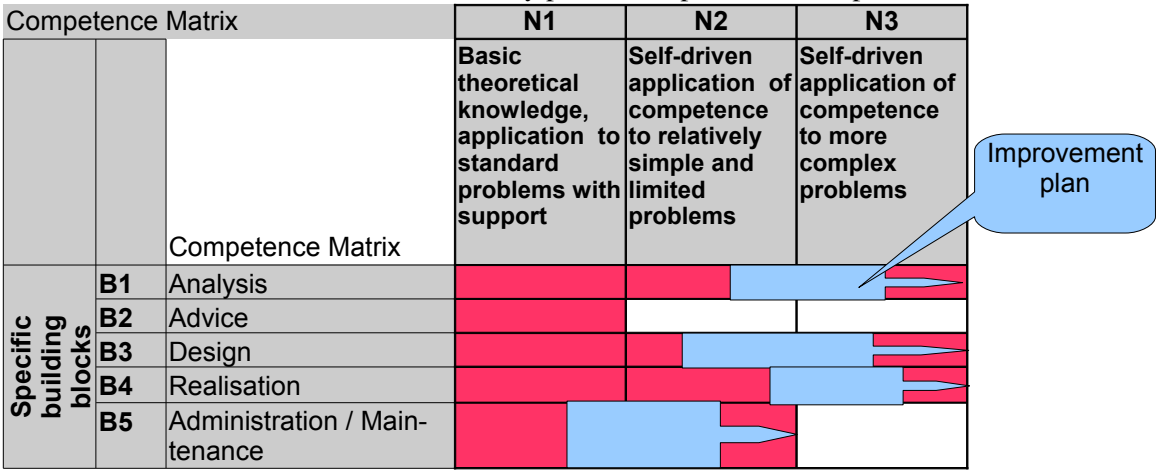
Table 9.1 Research results summarized – continued from previous page

Criteria	Alt 1	Alt 2	Alt 3	Alt 4
License	✅ Unlimited license lifetime ✅ No royalty fees	✅ All Software updates included ✅ Unlimited runtime of the SDK past of maintenance contract ✅ No royalty fees ✅ Available in source code and binary	✅ Includes one UaModeler license ✅ Includes one UaExpert license	✅ OPC Foundation license ✅ Open source
Price [1]	✅ Single developer license Client: 990€ Server: 2.175€ Combined: 2.600€ ✅ Branch license Client: 2.000€ Server: 4.350€ Combined: 5.200€	✅ Single developer license Client: 3.150€ Server: 6.150€ Combined: 9.662€	✅ Single seat developer license Combined: 4.900€ ⚠ Further support costs yearly: 1.470€	✅ Free
Costs for 5 yrs [1] [2]	✅ 8.320€ ✅ Unlimited developers	✅ 15.862€ ✅ One developer	⚠ 10.780€ ✅ One developer	✅ 0€ ✅ Unlimited developers
Other	✅ Branch license (Client & Server) already bought ⚠ Further support costs yearly 780€	✅ Integrated log system	⚠ May not be stable on .NET 6.0	✅ Limited features
Weblink	🔗 Fontys Home	🔗 This document	🔗 UA Bundle	🔗 OPC Foundation General

[1] VAT excluded
[2] with no prior license

Table 9.2: We commonly put table captions at the top

Competence Matrix			N1	N2	N3
		Competence Matrix	**Basic theoretical knowledge, application to standard problems with support**	**Self-driven application of competence to relatively simple and limited problems**	**Self-driven application of competence to more complex problems**
Specific building blocks	**B1**	Analysis			
	B2	Advice			
	B3	Design			
	B4	Realisation			
	B5	Administration / Maintenance			



At the end of the study, 3 of 5 Specific (B) competences should be on level 3.

9.1 The easy way or the high way

Luckily, spreadsheets can also create good looking tables. That is their bread and butter right? And of course, what applies to diagrams and code listings applies here too. We want to be able to zoom in to any depth of our liking without landing in the world of MineCraft, so export from the spreadsheet using svg(and convert to pdf) or pdf directly. That way you can have the reviewer zoom in and select text to make his/her remarks.

Here are some examples.

Note that the *enviroment* is table, but we include the table as if it were an image with includegraphics.

To prepare the tables, export or print *a selection* of the spreadsheet to pdf without header and footer. Then use pdfcrop, part of TeXLive, to crop the pdf to just the content. With cropping you remove the white space around the table, so it becomes its true size and appearance.

You can embellish the table in the spreadsheet as much as you like. The fruit of that work will then be visible in its appearance in your report.

All the above works for charts created with a spreadsheet too.

Table 9.3: Another table, this time made with word, but included as pdf!

Title	Projects 2			
Code	PRJ2			
Credits	10			
Academic year	2010-2011			
Education type	Theory (%)	Practical (%)	Project (%)	Self-study (%)
			100	
Description	The students apply their knowledge from the database, programming and modeling courses to develop a small web based information system. They work in groups of 4-6 students. In the first half of the project the students create analysis artifacts (user specification, use cases, domain model, prototypes) and design artifacts (class diagrams, sequence diagrams, ER-diagrams) and verification artifacts (test plans). In the second half of the project, the application is developed using Java web technology (Java Faces (JSF)), an oracle database and JDBC to access the database from the web application and tested according the test plans developed in the first part.			
Literature				
Classroom language	Dutch, German			

Note: 1 credit = 28 working hours

Table 9.4: ESD, still going strong?

Week of Semester	**1**	**2**	**3**	**4**	**5**	**6**	**7**		**8**	**9**	**10**	**11**	**12**	**13**	**14**		**Total Time**
Lecture/workshop	2	2	2	2	2	2	2		2	2	2	2	2	2	2		28
Laboratory	4	4	4	4	4	4	4		4	4	4	4	4	4	4		56
Self study	3	3	3	3	3	4	4		4	4	4	4	4	4	4		51
Exam preparation																4	4
Oral exam (distributed)																1	1
Total Time	9	9	9	9	9	10	10	0	10	10	10	10	10	10	10	5	140

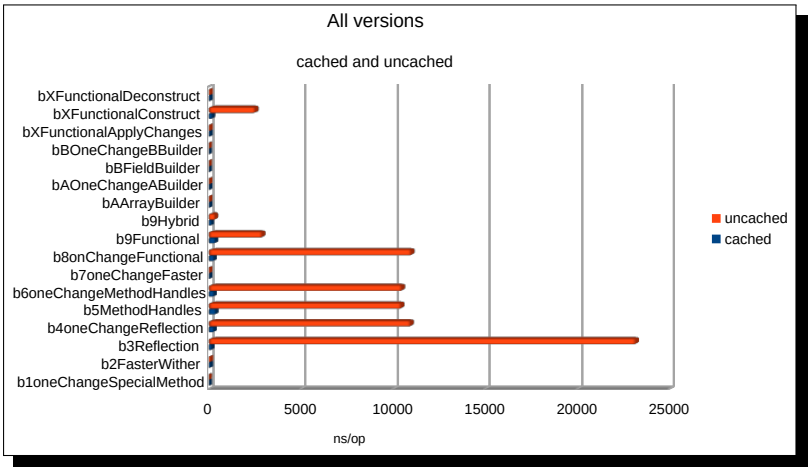


Figure 9.1: Benchmark results, LibreOffice calc, cropped

“The errors you made and are about to make have already been made by me”
said the experienced man to the newbie, selling it as wisdom.

You, in the future

10 | Tips and Tricks

- Either install LaTeXon your own machine, maybe in a docker-image, or use overleaf.
  On your own machine, the compilation will commence more quickly. There is also no compilation timeout, which you may run into with overleaf.

- actively use \include (for chapters) and \input to break you big document into smaller parts. If you then have a file called IncludeOnly.tex in your root dir, only the chapters in that file will be included. In this way it is easy to make something for your reviewers, but also speeds up the compilation process drastically, which is nice while you are editing and you work (re)view-driven.

- If you add pictures or tables, always choose vector format. Never jpeg unless it is a photo, png only for screenshots, which you should try to avoid. In the drawing tool (Visual Paradigm, Umlet, drawio) you can most likely export in pdf format which you can include with \includegraphics. If you have svg, which the other popular vector format, you can use a conversion tool to turn it into a pdf. Inkscape, which is available on all relevant platforms, does the trick for me. For tables, a spreadsheet(excel libreoffice calc) is a reasonable choice. Export the selection as pdf with no borders. Use a tool to clip/crop off the white borders. pdfcrop works for me there. Include the pdf with \includegraphics, but put in in table environment.

References

[BibLaTex project, 2024] biblatex. *CTAN BibLaTeX site*. web. 2014. URL: https://ctan.org/pkg/biblatex (cited on p. 17).

[Companion, 2004] Frank Mittelbach and Michel Goossens. *The LaTeXCompanion Second Edition*. Addison-Wesley, 2004 (cited on pp. 2, 4, 19).

[GoF 1994] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1st ed. Addison-Wesley Professional, 1994. ISBN: 0201633612. URL: http://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612/ref=ntt_at_ep_dpi_1 (cited on p. 18).

[Lamport, 1986] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley, 1986 (cited on pp. 2, 4).

[Nobody] Nobody Jr. *My Article*. 2006 (cited on p. 17).

[TeXbook, 1984] Donald E. Knuth. *The TeXbook*. Addison-Wesley, 1984 (cited on p. 2).

Appendices

A Some macro examples

The first macro defines a new environment. An environment is something with \begin{environmentname} and \end{environmentname}, like an itemize list. I wanted to tweak the standard lengths that are used, because I found the defaults a bit too spacy.

Listing A.1: Tighter itemize

```
\newenvironment{Itemize} {
  \begin{itemize}{}% start of envionment with all the settings
    \setlength\topsep{0ex}%
    \setlength\parskip{0ex}%
    \setlength\partopsep{0em}%
    \setlength\parsep{0em}%
    \setlength\itemsep{0em}%
  }{\end{itemize}% end of the environmet
}
```

B | Include Listings

The listing in listing 5.3 on page 11 was included with this code

```
\lstset{%first some settings
  numbers=right, % number the lines
  numberstyle={\tiny\color{gray}},frameround=tttt,framerule=1pt,↩
     rulesepcolor=\color{gray},
  framexrightmargin=5mm
}
```

followed by

```
  \lstinputlisting[language=java, firstline=1,firstnumber=1,
  lastline=39,numbers=right,basicstyle={\tiny\ttfamily},
  caption={The proper way to show your code.}
  ]{code/DAO.java}
```

C | Include using pdfpages

Include the whole pdf with all pages and some scale down to have our own page numbering. The pdf has been created from a website using the "print to pdf" feature of the **firefox** web browser which by the way has a very nice pdf viewer with annotation features.

Listing C.1: code to include next pages as pdf

```
\includepdf[scale=0.9,pages=-,pagecommand={\thispagestyle{\↩
    appendix}}]{appendix/sausageisnotsteel.pdf}
```

If you make the documents intended for the appendix in LaTeX too, you should of course simply \include them in your report, as all but the last appendix here have been done.

Martijn Bonajo, Richard van den Ham, Pieter van den Hombergh, Linda Urselmans – V2.0 2021-01-10

Table of Contents

Currywurst is not made of steel and vice versa.

There seems to be a controversy about when to add cardinalities (https://en.wikipedia.org/wiki/Cardinality_(data_modeling)) to (class) diagrams and when not.

The answer is, as always: *it depends*.

The **purpose** of the diagram is what counts, so apply separation of concerns in the diagrams you make in analysis and design too. That does **not** mean that analysis is about data, and design is about the software system, but that the aspect you are dealing with in your model will decide what kind of information is needed in a diagram.

The purpose of modeling is communication

A model in any form is a *simplification* of reality. Model is the art of *leaving out details* that are not relevant. Not relevant for the *purpose* of the model. Modeling in this sense also implies that there can be more models of the same reality. Each of these models can then highlight different aspects of interest for what you want to communicate. You want to express these aspects to communicate a meaning, relation, dependency, composition etcetera. A model is always about abstraction because it never IS the real thing but only describe some of it's aspects.

Models can have different form. Formulas in math, screenplay or scripts for a movie, descriptive text, and diagrams. In UML we have many model types. The trick is to choose the correct model type for the aspects you want to discover. Drawing can be done with a pencil and paper is willing, so you can easily become confused about syntax and semantics of the modeling language, in particular on what to use when.

A natural order of modeling, diagram wise, is something like this:

- A sketch of the things involved, or a picture, which serves as an initial model. This is what you collect in an interview.
- The UML equivalent is an Object diagram (https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-object-diagram/), in which you can simply draw 6 dogs if that is what fits in a kennel.
- Or simplify it and draw 1 dog and and add 6 to say that you have six. It makes the model simpler and still have the same information.
- You want to model relations between objects, like ownership or space to live. In an object diagram that is a line between objects, and as far as dogs and owner goes the 'direction' of the line goes both ways, because *'you know your boss, don't you George?'*.
- Drawing many owners and dogs becomes tedious, so you need another simplification to leave details out. This gets you to an entity relation diagram or domain model. The diagram does what the name implies: objects and their associations. So one icon to represent dog and owner plus the connecting line suffices. Add multiplicites to express that an owner can have multiple dogs, but no less than one, less he/she not be an owner. As far as the dog is concerned there is only one real pack leader at any time. The graphical syntax you use (lines and boxes) can look a lot like an object diagram. The relevant details are in the relations and multiplicities or cardinalities at the ends.
- A next abstraction is wanting to reason about **types** a.k.a. classes in UML. The class diagram actually has the wrong name, it should have been called the **type diagram**. Its purpose is to find the relations between types, who uses whom or who determines what. It is about dependencies and further abstraction. For instance, both a dog and child should be able to listen, so they have a

commonality you want to express, like `Obeyor` which is not a word but expresses what you want the sub-types to do. It does not change dog nor child but extracts the detail you want to reason about.

Modeling serves to highlight the important aspects. Choose the proper kind of diagram and try to closely stick to the syntax and semantics of the diagram **and its purpose**, so it helps the understanding of the details you *DO* want to communicate.

Modeling is always about **leaving out details**. Making things simpler.
Do not add details to a diagram that make no sense.

To paraphrase someone famous Everything should be made as simple as possible, but no simpler (https://www.championingscience.com/2019/03/15/everything-should-be-made-as-simple-as-possible-but-no-simpler/).

The above implies that the situation is not **black** or white, but some shade of gray.
That grayness is the reason of the confusion, because there is not one answer that fits in all situations.
It also means that you do not have to draw all diagrams all of the time, but only the diagrams that are needed. The question of course is: How do you know that it is needed? The answer is: as soon as you want to discover or explain something. Discovering is explaining to yourself. In a course where you need to learn the syntax and semantics of a language you must of course practice it and use the "words" (icons, lines) to form proper sentences (models, diagrams).

A model is a kind of plan and help to keep oversight, both overall and in the details, where required.

Let me explain:

First you should consider the purpose of the diagram.

- If you want to play out a scenario, you need objects (instances), and a box and line or stick model actors suffices. Boxes often will do because they are easily drawn (simplification).
- Object diagram could be the next step.

Then

- If it is about **data** analysis, a.k.a. **domain modeling**, relations between entities, then multiplicities **DO** matter.
- A domain model show the relations between data elements. These relations often go two ways, and is natural, hence there is no prevalent direction.
- If it is about designing your system architecture, then the **directions** of the associations (**is-A**, **uses**, or **has-A**) take precedence.
- A proper class diagram of a software system is a directed graph (https://en.wikipedia.org/wiki/Directed_graph). The relations **need** to have a direction, so you can discover what should or can go where, or can't. The directions are especially important because in dependencies (which is what these arrows show), you are not allowed to have cycles (https://en.wikipedia.org/wiki/Circular_dependency) .

- In a **software system** class diagram, that serves to find potential reuse (using an already existing idea or type etc), cardinalities have no meaning.
  - Its suffices to know the **types** (class, interface) and their relation, and you rarely need to specify how many instances you will have.
  - If you *do need* to specify a number, then typically **enum** is the solution, which implicitly also serves as *Singleton* (https://dzone.com/articles/java-singletons-using-enum).

Let us use metaphores. **Sausage** and **Sausage Machines**.

- A **domain model** is about the data. It is about *sausage*.
  - In the domain you specify what proportions of meat, fat, spicing, and skin you use. This is called the recipe. The processing takes place in machines.
- A system design is about the *sausage machines*.
  - What type of bolts and nuts do we need. What is the type of steel for the grinder, and what is it's shape etc. You need **only one** design (==template, class) of each.
  - The size (diameter) and consistency of the sausage matter, but for the rest it is

metal, wood, plastic, rubber etc.
  - The temperature and preparation time of course also matter, but that is how you *use* the machine.
- The assembly process of the machine is the last part as far as producing machines goes.
  - In the assembly you need to know the number of bolts, grinder wheels etc. Meat nor sausage is involved, unless you hit yourselves with a spanner. The sausages **will** be involved in the test lab. (Think test scenarios). Compare the assembly process of the meat grinder to what a compiler does to a program. As far as the compiler is concerned, the program's source code is just data. And so are the components of the meat grinder during assembly. During use it becomes (part of) the *system*.
  - You will find places where counts matter, like a GUI that shows items of a varying number.
    But than also, that count comes from the data. (Show the customers in a GUI list).
  - In all other cases the GUI widgets will have *names*. These names can be applied to the system components. Those are specialized by for instance inheritance or composition but also by configuration, like the colour of the handle of the meat grinder or the text on a label. Or even applying plugins like in a meat grinder by replacing the grind wheel and extruder to get a different sausage type.

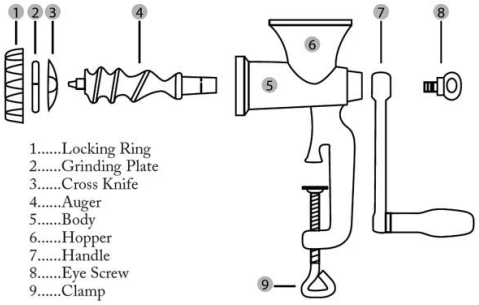
Transformation of the domain model into the system class diagram sounds reasonable, but is in fact a fallacy (https://en.wikipedia.org/wiki/Fallacy)._

It would be as if you could turn sausage into a sausage machine. I'm pretty sure your Butcher will tell you different when you would propose such a solution.

The domain and (part of) the system.

A. Data to process.

B. System plan.



▼ Quiz: What UML diagram type describes the images above best?

The sausages come close to an object diagram. Not much details is shown, which is proper for sausages. *You do not want to known*. The model is there to express **yummy** none-the-less.

The meat grinder picture is actually also an object diagram, it is the model to be used for assembly. It does have type names though, and luckily for my case only uses one of each. Ikea build plans are different.

Lucy, where is that Alan-wrench?

Sometimes it helps to go from the meta level to metaphors, to explain things, because $\text{meta} = \text{abstract}^2$.
Do so if you have to explain something, but once the abstraction has been understood by the reader, you can stop simplifying.

It is easier to model sausages than dogs, because the former sit still.

That concludes this intermezzo on currywurst (https://www.chefkoch.de/rezepte/1180961224164162/Currywurst.html).

Pieter van den Hombergh, March 2021.

At all costs, avoid complexity. That rule is a life saver.
A Seasoned Engineer.

D Counting words made simple

It is mandatory to include the word count of your thesis in the information page. There are many constructs shown on the Internet but we choose to go for the simplest solution possible.

D.1 What to count

It is resonable to only count the words in the report proper. This implies only counting the words in the chapters, which is easy enough by using the `texcount` command with a *glob* for the chapters: `chapters/*.tex`

glob

D.2 How to count

```
texcount  -inc -sum  -1 chapters/*.tex > wordcount.txt
```

This file is then simply read inside the info block using `\input`:

```
\begin{infoblock}
  word count: & \input{wordcount.txt} words.
\end{infoblock}
```

in the `partials/informationpage.tex` file.

D.3 When to count

To make sure that the word count in the file is updated in the whenever a chapter file changes, we use the Makefile.

In the Makefile we make the `wordcount.txt` file dependent on the chapter files with this simple rule:

```
wordcount.txt: chapters/*.tex
____texcount  -inc -sum  -1 chapters/*.tex > wordcount.txt
```

and make the main.pdf file dependent on the `wordcount.txt` file as well.

When that does not work for you, maybe because you cannot or will not use `make` in your environment, then execute the `texcount` as given above in the terminal.

Even simpler, delete the `wordcount.txt` file and run pdflatex on your main document again. That is arranged with the little bit of LaTeX code in the `configuration/thesis_config.tex` file. For that to work you need to run latex with the `-shell-escape` switch.

```
\IfFileExists{wordcount.txt}{}{
  % not exists
  \immediate\write18{texcount -inc -sum -1 chapters/*.tex > wordcount.txt}
}
```

By the way, we have if you are on Windows: we have good experience with git bash + make installed via chocolaty, so you might want to try that too.