By Christof Lutteroth and Gerald Weber

Before you start you should look at similar documents, e.g. old reports/dissertations with a good grade or papers from good conferences. You can write a good document yourself by **imitating** them.

Academic writing is an iterative and incremental process. You start with an outline, adding bits as your project progresses. You should **get feedback on your writing whenever you can**, and improve it. Start small and simple. You can always add more detail later on.

If your supervisor is not allowed to read your full report, you should instead request feedback for a **detailed outline**. This outline should contain all the section and subsection headings, and ideally a heading for each paragraph. Creating such a detailed outline is also a good way to get started on your document.

Typical Structure

A typical structure for a paper/report/dissertation in Computer Science is the following:

0. Title and Abstract
1. Introduction
2. Related Work
3. Requirements
4. Design / Concepts / Specification
5. Implementation
6. Evaluation / Discussion
7. Conclusion

But the parts are written in a different order: Related Work, Requirements, Design, Implementation, Evaluation, Introduction, Conclusion, Title and Abstract.

Related Work

In every academic paper, there has to be a discussion of related work. You should search for related academic papers with keywords that are suitable for your project, e.g. using Google Scholar, Scopus or Web of Science.

- **Look for high-quality academic publications**. Preferably journal articles, books or conference papers. Technical reports, standard documents and industry white papers can also be used, but be aware that they may be biased (e.g. an industry whitepaper may try to

sell a product). Web pages such as Wikipedia articles should generally not be cited, as they may change anytime.

- **Select** the papers that are relevant for your project by reading the abstract and scanning over the paper. Usually only some of the papers are relevant for your project. However, it is possible to cite and briefly discuss papers that are only marginally relevant.
- **Read** the papers, or at least the parts that are relevant for your project. If a paper is about your topic, then read also the papers that are cited by it and the papers that cite it (some databases give you that information, e.g. ACM).
- **Cite** the papers in your own paper using a consistent citation style. For example, you could add a number in brackets such as [1] as a citation, and have a bibliography listing the papers with their numbers at the end. A paper should be cited in the first sentence it is referred to, e.g. "The ALM system [1]…" A citation should either point the reader to additional information about a something that is mentioned. It can also be used to support an argument, e.g. "As Lutteroth and Weber pointed out [2] …" A paper can be cited several times.

You should briefly describe what others are doing, i.e. summarize the most important points. **What** has been done? **Why**? **How**?

You can show that you are able to analyze other people's research by comparing several related works and pointing out the similarities and differences. You can also show that you are able to evaluate research objectively by pointing out advantages and/or disadvantages. To guide your own research project, you should look out for gaps in the research (unanswered questions) or potential improvements that you can do.

Requirements

In this part you come up with a list of requirements for your project, and explain each of your requirements. Requirements mean that you specify what functions your system is ideally supposed to fulfill. You are not saying how you will do it, just what you want in the end. For example, you may want your system to have certain important functions, to be fast, reliable, easy to maintain, easy to use by certain users, etc. You should also write why each of these requirements is important. If possible, prioritize the requirements: which ones are most important and which ones are less important? Why?

Your requirements should be supported by the related work you have read. That is, cite related work to show that you are targeting the right requirements. The requirements should address the research gaps or opportunities for innovation that you have identified while analyzing the related work.

Design

In this part you explore the design space of your project analytically and make well-founded design decisions. This means that whenever there are several possibilities something can be done in the

design, you describe and analyze all of them, possibly focusing on the most promising choices in your analysis. You should explain the pros and cons of each possibility, and based on that judgment, you make a choice for your project. This is how you come to a single design that is well thought-out. The design part may contain diagrams such as architecture diagrams or data models, and pseudocode for algorithms.

You should use figures whenever appropriate. Figures should be referred to in the text. Figures are usually not self explanatory, but should be either explained in the text or should be used to illustrate the text. Use a font that is not much smaller than the text font. If figures are taken from other sources, they must be referenced. This is generally to be avoided, except in Related Work.

Giving an example is usually very helpful for explaining a complex concept. The example should be small but illustrate the concept clearly. Sometimes the same example can be used as a running example throughout the document to explain several concepts.

Implementation

In this part you describe the implementation of your design, i.e. **what** you have implemented. You should describe the features (typically using screenshots) as well as the shortcomings. This implementation is usually a prototype, so it is natural that it is not perfect. You should also describe **how** you implemented the different parts, e.g. which tools and components were used in the development. Focus on the interesting bits, i.e. those that might be useful for others trying to implement a similar system or understand your program code. This part may contain short snippets of source code, if that helps to understand how non-trivial parts of the project were implemented. Long sections of technical information can also be included as an appendix if they are important. Make sure that you include your full source code on a CD when submitting your work to a supervisor.

Evaluation

In this part you are critically reflecting on your project outcome, but it is also a place to highlight the achievements of your project, i.e. how good and useful or how much better than other systems it is. You should evaluate your work based on your requirements, i.e. show that it fulfills the requirements. The type of evaluation depends on your project and should be discussed with your supervisor. Among the things that can typically be evaluated are the following: performance (e.g. speed, memory usage), usability (ideally from the perspective of typical users), and other types of quality (e.g. accuracy, deviation from some optimum). An evaluation may be done analytically by examining the project outcomes and explaining why they are valuable. It can also be done empirically, i.e. by using some sort of statistical data such as a benchmark or experimental measurements. It is always good to compare your outcomes to those of similar projects. An evaluation of some form is an essential component of each report and complements the analysis of options in the design section.

Introduction

In the introduction, you should answer the following questions: **what** is your project about (topic and scope) and **why** is it interesting (motivation). You should also briefly explain the context of your project, i.e. give some useful background information. To focus the attention of the reader, you can give a list of research questions that you are trying to answer in the project. Finally, you should give a brief outline of the rest of the document.

Conclusion

Here you sum up what you achieved in your project, highlighting the main achievements. Revisit the research questions and give a summary of the answers you came up with. You should also point out future directions of your work, e.g. possible extensions or unsolved problems and limitations of your work.

Title and Abstract

This comes last because then you know best how to sum things up. An abstract should be very brief (around 100-200 words), and should sum up **what** your project is about, a brief outline of the state-of-the-art, then your achievements and **how** you did it, and possibly **why** this is significant. An abstract is usually a single paragraph.

Proof-Reading

This is absolutely necessary! Give your document to your supervisor, your friends, or even to a professional proof reader (this is not very expensive).

Writing Style

Also within the sections your document should be well-structured. Typically, each idea should be expressed in a single paragraph. If a paragraph is too large (approx. more than 7 sentences), you should break it down into two paragraphs.

Try to avoid repetitions or redundancy. If there are redundant parts, try to regroup them so that each idea is explained only once.

Examples of How to Improve Sentences

The following gives some examples of sentences that can be improved, with explanations:

Today, software development projects are getting more and more complicated, and many version control systems can be used in software development projects.

The term "*software development projects*" appears twice, which seems redundant. Furthermore, there are actually two thoughts expressed in this sentence: "*software development projects are getting more and more complicated"* and *"many version control systems can be used"*. To make the text clearer, one should either express and explain the two ideas separately, i.e. in different

sentences, or establish a stronger connection between the two ideas (e.g. projects are getting more complicated because there are so many systems that can be used).

Choosing the right version control system will make project management work easily.

This sentence makes a claim that is too strong. To make project management work, there are many factors involved, and version control systems are just one. Tone down the sentence, for example by saying that version control systems provide important support for project management.