Writing Technical Reports

Software Development 2017
Department of Computer Science
University of Copenhagen

Oleks Shturmov <oleks@oleks.info>
Alexander Christensen <alch@di.ku.dk

Version 1;
March 10, 2017

This document contains suggestions on how to structure a technical report describing the implementation of a small programming project. This document *does not* intend to cover how to structure a technical report for a large software development project, beyond the scope of our Software Development course.

The software you write, in this course, and beyond, will typically address a particular problem. The job of a technical report is to give an overview of your solution, explain the structure of your program, and tell the reader how to work with it. Your report is an exhibit of in-how-far you have understood, and solved the given problem. Technical reporting is an indispensable part of *sustainable* software development.

This guide is heavily based on the many guides that came before it [1–4]. We encourage you to read these at your leisure. As with other guides, this guide is a social construct. It may be imprecise, and you may deviate when deemed necessary. However, please respect that this guide is based on the long-lived experience of those that came before you (and before us).

1 Structure

Beyond the front matter and appendices, your report should contain the following sections (in order). These sections, and more, is covered in-depth in the following subsections.

- Introduction
- Background
- Analysis
- Design
- Implementation
- Evaluation
- Conclusion

1.1 Front Matter and Paging

The front page of your report should present the reader with a title, list the course / activity and institution, list the authors, and state the final date of your report. You may also add author contact information, document and/or software version information, etc., depending on the context and requirements.

The pages of your report should be numbered, and the headers and/or footers should provide some contextual information. This is so that if pages are ripped out of context, they can be stitched back together, in order.

1.2 Introductory Matters

A report must begin with a preface and/or abstract stating the purpose, and scope of your report. It can be a good idea to write this up-front, and to revise it before submitting your report. It can help you stay on track as you write your report, and it will tell the reader if reading it is worth their time and effort.

The preface and/or abstract should be followed by an introduction motivating the problem, giving a glimpse at the extent of your solution. An introduction will let the reader know, if reading on is worth further time and effort.

A short report may well begin just there. There is little need for a "table of contents" for a 1–5-page report. For a longer report, the reader may want to quickly skip to the parts that they find interesting, and so a table of contents is in order. In either case, it can be a good idea to give an overview of your report in conclusion of the introduction.

The introduction may also briefly list acknowledgements, if others, beyond the list of authors, have helped along the way.

1.3 Background

In this section, you should show that you understand the elements of problem, and the elements of the course / activity that go into making a solution. For instance, this is a good point to recap the design patterns you used, telling how they work, and why they can be a good idea for a particular problem in object-oriented design.

1.4 Analysis

In this section, you should show that you understand the problem and how to solve it. The outcome of this section should be some goals for your design, and requirements for your implementation.

1.5 Design

In this section you should give a high-level overview of your solution, and argue why you chose this design over others. You should use high-level notation, without delving into implementation details.

For instance, draw a diagram over the essential parts of your program (e.g. using UML), write the pseudo-code for the algorithms you used, and give abstract descriptions and illustrations of the data structures you used. Contrast this with other possible architectures, algorithms, and data structures.

1.6 Implementation

This is the part where you delve into low-level implementation details. Discuss the concrete implementations of your algorithms and data structures. Discuss the low-level challenges you faced, and how you addressed them. Your discussion should suffice for another programmer to pick up your program and run with it.

Although you *can* include parts of your code here, code is better included as part of the appendices to your report. Refer to the relevant parts of your appendices from here.

1.7 User's Guide and/or Examples

This section should target the readers less interested in extending, or integrating your application, but more interested in *using* it to solve the problem motivated in your introduction. Tell the reader how to run your program, and showcase some examples.

Depending on the context of your report, an outright "user's guide" might not be relevant, but it is always a good idea to showcase that your program works for some sample instances of the problem.

1.8 Evaluation

In this section you should evaluate your work and give a quality assessment of your implementation. Discuss the running time and space complexity of the algorithms you used. Discuss the quality of your code and documentation. Discuss the technical debt. Is it maintainable? Is it ready to meet the end-user?

An essential part of software evaluation is **testing**: Give an overview of what you thought was relevant to test in the context of the problem. Give a high-level overview of the tests you conducted, and present your test results. Discuss your test results, what do they show? Employ the scientific method in your quality assessment: tell the reader how to reproduce your test results.

1.9 Conclusion

The conclusion should briefly sum up the report. As such, it should also sum up the work you've done. Does the program solve the problem at hand? Is it a good solution? What future work remains?

1.10 Appendices

Your report should not be too long, and should include little code. The report is supposed to give an overview the code alone cannot provide. Appendices are a good place to actually list your code, list long-winded test results, and include sections which go beyond the scope of your report.

When including source code, use proper source code listings: **do not use screen shots**. Screen shots cut in the eye of the reader, and make your code non-selectable, and non-copy/paste-able. All this impedes communication of code. (When using LaTeX, we can recommend using the `listings` package, or the `minted` package together with the `Pygments` Python package).

2 Writing Techniques

Your report is an exhibit of in-how-far you have understood, and solved the given problem. It should give an overview that your implementation alone cannot provide. As such, your report should be well-formed and easy to read.

The following expands on the items originally listed at the end of [3]:

- Everything in your report should serve a purpose, don't fill the report with superfluous or redundant content.
- Have a target audience in mind. It can be a good idea to state the target audience of your report in the front, or introductory matter. Some sections may also target a more technical audience, interested in extending, or integrating your program, while others may target the end-users.
- Keep it crisp and clear: Avoid unnecessary "colouring" words; avoid pseudo–scientific formulations; avoid variations for the sake of variations; describe related ideas with related terms and formulations.
- Write well: slang, bad grammar, bad spelling, unexplained abbreviations and notions (relative to the target audience), all impede communication.
- Give an overview of your report, and make your report easy to navigate. Number your pages, sections, subsections, figures, etc. Make a table of contents, if your report is sufficiently long.

3 Gluing it all Together

We encourage you, when writing a technical report, to design your report with a very strong, consistent mindset. The introduction is meant for presenting the contents of your report, and should not strife away from this by explaining issues unrelated to the work that you have done.

When you write the analysis it is important that you analyse *only*, what you have been presenting so far - the introduction and background present a context, which must be analysed so that the reader will be convinced that you have actually understood the problem prior to making any design decisions.

The analysis directly links together the design and implementation sections, in which everything that has been subject to any prior analysis will be touched upon. These later sections must also be carefully written such that no new topics should be introduced, which have not already been presented at the beginning of the report.

This naturally progresses to the point where you have an implementation section, which convincingly describes *how* everything mentioned so far has been done, and how the arising issues have been solved. Therefore, when reaching the evaluation, you can evaluate upon your design choices, extracted from analysing and understanding the problem, and carefully explained with only necessary, non-trivial implementation details - thus testing your design against rules of good software practice, code testing, and requirements validation.

This process of gluing together the sections of the report will be a natural progression from presenting a task, analysing the requirements for this task to be completed, describing what design you envisioned, and how this vision was realized. Much like taking the reader on a guided tour through the complete working process up to the point, where you can reflect upon the degree of completeness, the correctness of your implementation, the justification of your design choices, and even the process of analysing and understanding the task which was given to you.

It will, following from such a consistent working process, be a natural way of rounding off by concluding on your design choices, the degree to which you solved the task given, if some parts of the problem were not immediately solveable - explaining what went wrong, how you think the problem might have been tackled given more time, if you could have done something different. Or, if you consider the problem solved, then clearly state that the project has been successful and summing up your successful implementations.

References

[1] ANDERSEN, R., AND SPORRING, J. Førstehjælp til Rapportskrivning. Department of Computer Science, University of Copenhagen, 2005. 1

[2] LARSEN, K. F., AND NISS, H. Kunsten at vejlede et konstruktionsprojekt. *Dansk Universitetspædagogisk Tidsskrift 2*, 4 (2007). At gøre de studerende til studerende. 1

[3] SESTOFT, P. Writing Reports. IT University of Copenhagen, 2002. 1, 4

[4] SPORK, M. Et forsøg på redde jeres rapporter — eller — En tidlig julegave. Department of Computer Science, University of Copenhagen, 1996. 1