Writing Technical Reports

Providing an overview that code alone cannot provide

Oleks Shturmov <oleks@oleks.info>

Version 1; May 8, 2018

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 a programming course.

The software you write, in a programming 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 you deem it necessary. However, please respect that this guide is based on the hard-earned experience of those who wrote software before us.

This guide was originally prepared for the course in Software Development 2017 at the Department of Computer Science at the University of Copenhagen (DIKU). It was then revamped to be more general, during the course on Distributed Objects 2018 at the Department of Computer Science at the University of Oslo (IFI/UiO). It is applicable in other programming courses going forth.

1 Structure

Beyond the front matter and appendices, your report should contain the following sections (in order): Introduction, Background, Analysis, Design, Implementation, User's Guide and/or Examples, Evaluation, and Conclusion. On occasion, you can omit certain sections, depending on your programming assignment text. These sections, among other details, are covered in-depth in the following subsections.

1.1 Front Matter and Paging

The front page of your report should present the reader with a title, list the course / activity and affiliation, 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 technical report should begin with an abstract, stating the purpose, and scope of your report. The purpose is usually to present a piece of software that solves a particular problem — state this problem *clearly*, and clearly state the extent of your solution.

It can be a good idea to write an abstract up-front, and to revise it right before submitting your report. An abstract can help you stay on track as you write your report, and will tell the reader whether reading the report is worth their time and effort.

The abstract can be followed by a longer introduction, motivating and explaining the problem further. However, depending on your task, the problem may already have been clearly presented to you, and there is not much leeway in how to interpret the given task. In this case, a longer introduction is likely to be unnecessary.

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 to conclude the introduction.

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

1.3 Background

This section serves to fill the gap between what you expect of your readers, and what they need to know to understand your solution. In a longer report, it can be a good idea to state your target audience in the introductory matters. In a shorter report, you can assume that the reader is mildly familiar with your programming task, and the context of your work.

Show your understanding of the main elements of the problem, and what goes into making a solution. For instance, if you are tasked with writing a game of Tic-Tac-Toe, this is a good point to recap the rules of the game.

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. It is at this point that we narrow in what sort of technology we will need to make a working solution.

For instance, if you are writing a game of Tic-Tac-Toe, explain how you intend for users to interact with the game, and what constitutes a game-frame, in a bullet-point list.

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 diagrams over the essential elements of your solution (e.g., using UML and entity-relation diagrams), give abstract descriptions and illustrations of the data structures, and present pseudo-code for the algorithms you used. Contrast this with other possible architectures, data structures, and algorithms.

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. Do not assume that the reader has read the background, analysis, and design sections. 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.
- Use proper academic citation (e.g., as in this report).

References

[1] ANDERSEN, R., AND SPORRING, J. Førstehjælp til Rapportskrivning. Department of Computer Science, University of Copenhagen (available at http://image.diku.dk/sporring/foerstehjaelp.pdf; last accessed on May 8, 2018), 2005.

[2] LARSEN, K. F., AND NISS, H. Kunsten at vejlede et konstruktionsprojekt. *Dansk Universitetspædagogisk Tidsskrift 2*, 4 (2007). At gøre de studerende til studerende (available at https://tidsskrift.dk/dut/article/view/5646/4939; last accessed on May 8, 2018).

[3] SESTOFT, P. Writing Reports. IT University of Copenhagen (available at https://www.itu.dk/~sestoft/itu/writing_reports.pdf; last access on May 8, 2018), 2002.

[4] SPORK, M. Et forsøg på redde jeres rapporter — eller — En tidlig julegave. Department of Computer Science, University of Copenhagen (available at http://www.diku.dk/OLD/undervisning/2000f/dat1f/rapportskrivning.ps; last accessed on May 8, 2018), 1996.