A Review Guide for Reports in Software Construction. Revision 2.

Henrik Bærbak Christensen
Computer Science, Aarhus University
Aabogade 34, 8200 Århus N, Denmark
{hbc}@cs.au.dk

February 2020

Abstract

This document presents a checklist of issues to identify when reviewing technical reports like synopses, master reports, and theses.

1 Introduction

I have personally reviewed and graded literally hundreds of master theses, reports, and synopses. Some are really good, many are average, and some are really lousy. But what makes a good report good, and what makes a bad report bad? Well, if it is because the underlying technical work is bad you are in trouble, but often I find that what may have been solid technical work is not really appreciated because the report is ill structured, difficult to understand, or have other defects.

To take you into the minds of the examiner, I will put forward a series of checks you can do to your own writing in order to improve it. Basically it is the checks I do when I grade a report.

2 The Structure

All scientific papers and reports follow the same format!

This format is

- *Title and Authors.*
- *Abstract.* 8-16 lines explaining what you have done and your key results.
- *Motivation.* Broad introduction that sets the "stage" for the problem you are investigating. Why is it interesting? Often you also present key results of your work to set the context of the main body of text.

- *Problem statement / Hypothesis.* Describe the problem you are addressing as short and precise as possible. It is preferable if you can express it in a way that makes it into a statement whose validity we can examine—a hypothesis.
- *Method.* How will you study your problem and ensure your findings are valid?
- *Related work.* You should shortly outline research/results that are relevant for your problem statement. This can also be products that do similar stuff as you work on.
- *Results/"body" of the report.* Here you describe your main effort in your project - of course always tying back to your hypothesis / problem statement. Remember that this is "science in the small": your solution and/or what you have done is not nearly as interesting as the analysis and/or empirical observations that lead you to your result. Thus it is very important that you document process and decisions - and always keep a skeptical eye on yourself - try be your own 'devils lawyer'. Using the theories and concepts from the courses you have followed is also important.
- *Discussion.* What have we learned from your results?
- *Conclusion.* Restate the problem statement, describe in 10 lines your work, and present the results in short form.
- *References.* Lists all cited literature in alphabetic order. Sufficient information must be provided to allow a reader to find the references easily.
- *Appendix.* How to run your software, where to find the code, the interview guides you made, argumentations that are too long to make into the result section, etc.

Quite often (motivation, problem statement, and method) are subsections in a section called "Introduction".

Sometimes related work comes after the discussion or is merged into the discussion session.

> Verify that the report follows this structure and at least contains the information outlined.

3 The Problem

The most efficient way to make a really really bad report is to a) define a very vague problem (in problem statement section) and b) write about something else (in results section). Do not do that!

Writing a good problem statement is difficult but pays of because it is basically the one thing that tells you if a given paragraphs of writing is relevant or not.

Try to be as precise as possible. Broadly and ambitious problems statements are notoriously bad:

It is possible to develop a framework that supports all types of board games.

This is not a good problem statement for two reasons. First "it is possible" is vague and I can already say "yes, of course we can. The following Java code is the result: public interface Game { }". The second part "supports all types" is so ambitious that in order to validate this problem statement you really have to implement "all"—all board games known to man now and in the future. Thus I can already say that you will run out of man hours.

The solution is to be specific, delimit yourself, and make claims that can be validated by experiments! This is the good news because it means less work. Try this instead:

> *The "XYZ" framework provides 80% design and code reuse for the board games chess, checkers, and tic-tac-toe. A competent Java programmer can learn the framework and implement these games in less than 4 hours.*

Note that it is still a good exercise in developing a board game framework but now it is delimited (chess, checkers, and tic-tac-toe) and it is measurable (80% reuse, doable in 4 hours) and thus possible to validate by an experiment.

> Verify that the problem statement is a) present b) short (5-10 lines) c) well defined d) limited in scope so it is possible to do in the amount of time for the project e) as measurable as possible.

Often e) is pretty difficult to fulfill, I admit that.

Of all the check you can make to a report, this is the most important one because it is what natural science is about: finding answers to well defined questions.

4 Keep focus on the problem

Do not write or work on anything else but the problem!

> Verify that each section and each paragraph in the report directly contributes to the work and analysis of the problem statement.

Sometimes I see reports that have whole sections of interesting technical discussions that are unrelated to the problem statement. This is bad. I get annoyed: "Why am I reading this stuff about Eclipse configuration when the problem statement is about board game frameworks???"

IF, during your work, you find new problems that are more interesting than your original problem statement, then *rewrite your problem statement.* This is not a problem and much better than making the problem statement and the results section going out of sync.

5 Use the (problem, analysis, solution) template

A recurring problem of many reports is the use of a *narrative* writing style in the results section. By this I mean people tell their story of what they did when they worked on the problem:

> *First we coded the client and decided to use the visitor pattern to generate the game board. It really did not work out so we tried the builder pattern instead...*

This is boring to read and shows a process of trial-and-error rather than a scientific and analytical approach. Instead analyse and write it using a (problem, analysis, solution) template: concisely state the problem; list and analyse possible solutions, and finally pick the optimal solution.

> *We encountered the following problem:*
>
> ***Problem**: In our design we need to ....*
>
> *Analysis: According to GoF both the builder and visitor patterns are applicable. A closer analysis shows benefits and liabilities of both patterns. Below we discuss each pattern in turn.*
>
> *Visitor: ...*
>
> *Builder: ...*
>
> *Based upon the discussion above we choose the builder as it provides better architectural modifiability than visitor for the following reasons: ...*

Verify that text presenting analyses and results does not become narrative, but follows a (problem, analysis, solution) template.

Good experimental work in software engineering and architecture is incremental and iterative, so the above "process" is repeated: you state a problem, you analyze/experiment with it, you conclude on the analysis/experiment—which leads to yet another problem or problemset. You described and solved A, which leads to problem B, which leads to problem C, etc. Therefore the (problem, analysis, solution) template is often repeated two, three, or many times in the "body" text of the report.

6 Use what you have learned!

I get really really sad when people have followed one of my courses in which highly applicable techniques are *not* used in cases where they should.

Examples: An ill defined problem statement about availability of a service ala "my service should be highly available"—why not express it as a Quality Attribute Scenario? Two pages of text like "then the client calls method getInstance() that returns an object of type 'Data', next the client calls getName() in this object, which makes a call to the service Foo that..."—why not a sequence diagram?

Verify that the analyses and presentations use common terminology and techniques from your courses and computer science in general.

Almost all topics that we teach in engineering and architecture proposes some kind of notation (templates, diagrams, etc.) or concise way of expression whose purpose is to present data in a terse and concise format. By using it you avoid a lot of words and you demonstrate that you can actually use it.

> Verify that UML, architectural views, QAS, and other clear and concise notations are used to describe software designs and architectures.

And finally, do not just use what you have learned—tell the reader what you are using and refer to it, so the reader knows that you are on common and safe ground. We have to agree about what we are talking about. Do not just say "We use Bass et al. in our analysis", but be specific about which aspect and for what purpose, ala "We formulate the quality requirement using the QA technique from Bass §4-8 (2013)".

> Be specific about what techniques, notation, theory, results, you use by providing proper and specific references. Especially if you refer to a book, be sure to include section/chapter number.

I do *not* want to read a complete book in order to find that one line of reasoning from it, that you are using in your report.

7 Terminology and Clarity

Science is about being precise and clear.

You are only precise if you use well defined terms. This means that you either a) use commonly known terms correctly or b) define your own terms.

Do not write "object" when you mean "class". If you use the term "component" then it is the most ill-defined term in computer science and you have to write what you mean by it? Is component in your world a compilation unit, a run-time unit, a Jar file or DLL, or ???

> Verify that terms are well defined and used consistently throughout the report.

A report must be clear in its communication. This means it should be structured and presented in a way such that the reader is lead naturally from one problem and its analysis to the next problem and its analysis, and so forth. It is not OK if just "the information is somewhere in the report."

As an example of *how not to do it*, a specific report I reviewed had an hypothesis along the lines of "we have a present (bad) architecture, and based on a stakeholder workshop that outlined new requirements, we present a new and better architecture." However, in the first body/result section of the report, the present/bad architecture was presented only by six cross references to sections in the appendix without any summary. Likewise the established requirements were just another six cross references without summary. Thus the reader had absolutely no idea of the context in the following discussion of the new architecture.

> Verify that the report's main text (excluding appendices) is a clear and meaningful text that can be understood without reading appendices. Appendices can be used to provide further details but they must never be required reading in order to understand the main text.

Be also aware of assuming "obvious knowledge" because often it is only obvious to you. In particular, be aware of organization context that you assume ("obvious because we always have done like this in our company"), or assumptions about technology ("everbody knows how Windows ASP works") etc.

An important part of clarity is also that the reader can *find and overview your results*. The (problem, analysis, solution) template obviously ends in some kind of conclusion: the measured result, the verified architecture, the summary of the experiments, etc. It is important that you summerize these partical results/conclusions, so the reader can find it in one place—in contrast to read two pages again and combine those three important learnings by himself. ØUse some kind of typography, notation, tabular format, bullet list, or what ever suits the result, so reader can easily overview it and find it again.

> Summerize your (partial) results. Use typography to make it stand out from the text.

8 Typography

Make your report easy to read by using typography to make important text stand out. Notice how I have used typography to highlight important points to check in a report, and notice how easy they are to spot in all the rest of the text. Now try to imaging if I had merged all the text into one large ongoing text without using colors, bold, emphases, etc. and made the text look like a novel. You would have had much more trouble identifying the check points then.

In a same manner, use indentation and bold face to signal important writing in your report (definitions, major results, problem statement, etc.) and make it easy for the reader to find them.

> Verify that central text paragraphs (like problem statement, definitions, major results, summaries, etc.) clearly stand out in the text.

9 Use consistent referencing style

In the area of minor problems, some reports use citations, footnote and in text parentheses to cite related work. This makes for an uneven look. As a general rule, when you want to reference a book, a paper, a web site or anything external to the report, do so using a citation; not footnotes, not parentheses, and not inlined text.

You cite some work by a reference in square brackets. Some use numbers like [3] or author plus year like [Christensen2005], and then spell out the full book/paper/website in the references section. Review your teaching material or books for examples.

> Verify that a consistent citation style is used.

I prefer using the author plus year style, like [Bass2013]. Why? Because I know that book, I do not have to jump to the literature list to find out what [3] is...

10 Spelling and Grammar

I must admit that my own papers and even my books have a fair share of spelling and grammar defects. I am simply not really good as "seeing" my english errors. Therefore I am also not that picky about minor errors in students' reports: after all it is about software construction, not about English or Danish writing. Still there is a limit—a sentence must have a subject and a verb, not be too long and winding, and a comma may come in handy.

> Verify that the language is not so full of spelling and grammar errors that it becomes difficult to understand.