Advice on Preparing Reports in Softwareengineering

Paul Fischer, February 2021

1 Report

General

Avoid to only give a manual for or a description of the final product. Instead explain why the product looks as it is. Document the development process, decisions made on the way, and how you tackled problems.

The points below do not have to (but might) be the headlines/sections of the report and some might not be relevant for some projects. Below, the word "describe" has to be understood as "describe and justify your choice".

Remember that the person who is responsible for the course has to look at all reports, use appendices for lengthy illustrations. **Avoid to use internal "slang" expression, which you have developed during the project, without clearly defining them for the reader.**

Document, who is responsible for which part. It is **not** enough to say something like "we all contributed to the same extend". Best to set names at the start of sections/paragrahps.

Make sure that your names, study numbers, and group number appears on the cover page.

The problem

- Describe the problem and the context/domain in which it appears.
- Describe the scope of the project: Which sub-problems will be considered, which will not be considered.

The goals

- (Functional) What are the main goals of the project, what should be the properties of the final product?
- (Non-functional) For example: Usability, performance, maintainability, extensibility, simplicity, security, ...
- (Non-project)For example: Learn new technology, learn project management, ...

- What are the success criteria for the project?
- What is your level of ambition, both with respect to the problem and the grade?

Analysis of the problem

This is independent of the type of project (hard-/software) and the hardware platform or programming language (unless enforced by external factors).

- What are the main challenges for solving the problem.
- What are potential sub-problems.
- Are there know solutions to sub-problems.
- Propose solutions to the problem/sub-problems, name alternatives.
- Select the solutions you want to use and **justify you choice**.

Its is important that you explain why you selected or discarded a solution.

Design

The design should be largely independent of the implementation environment/paramming language. How ever the design paradigm (for example Model-View-Controler) has impact here.

- Describe the general architecture of the solution.
- Identify major system components (components, packages, interfaces, ...)
- Describe how the components work and interact, that is, specify *interfaces* [1]. Imprecise specifications of interfaces are one of the most common reasons for failure.
  - What does a component do, how does it manipulate data.
  - Which information does it deliver to other components.
  - How is the information presented/formattet.
  - Is the information sufficient for other components to work with.
- Describe the technologies/platforms used.

[1] Here, *interface* does not refer to the Java concept with the same name, but means the boundary between software components.

Both, Analysis and Design should be as independent of the programming language a s possible. Think of these parts as a guideline for an implementation where the platform and language are not yet decided.

Again, discuss alternatives and explain why you selected or discarded them.

I can make sense to merge *Analysis* and *Design*, for example, when the analysis leads to a specific solution and the design of the solution is best described close to the analysis.

In some cases there can be external factors which enforce to consider implementation details early in the process. This can for example be that the hardware platform is fixed, or the project has to be integrated into an existing software.

Implementation

This turns the design into software (or hardware).

- Describe the system components.
- Specific algorithms and data structures.
- Describe the program structure, e.g., describe detailed class hierarchies or data base schemes.
- Avoid to describe the whole code in detail (this can be done in an appendix and as comments in the code listing), however point out unconventional coding.

Test

- Which types of test were used and why were they chosen?
- Document the results and potential changes because of errors found.
- If applicable describe user feedback.

Project Management

- Did you use project management paradigms/tools, if not why?
- How did you assign tasks to the group members and how was the quality of the deliveries checked?
- How did it go? If there were conflicts, how were they handled?
- How did you manage time?

Conclusion

- What was achieved, which goals were reached?
- What was not achieved and why?
- What could have been done better ("if we could start again then ...")?
- Possible extension, ideas for the future.
- What did you learn in the project, both professionally and socially?
- Individual, personal experiences, "I .....".

Ethical Issues

Some projects will have a societal impact. In this case, ethical issues should be addressed. The impact on society can be indirect and hidden.

- Who will be personally affected by the results of the project and in which way?
- What is the impact on society in general?
- Can the results of the project be misused ("dual-use")?
- Does project follow ethical guidelines in case personal data is used or test persons are involved?

2 Additional Material

2.1 Citations and Use of External Code

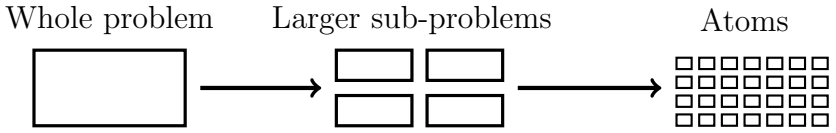
Here are some rules to avoid plagiarism. See also the relevant pages on DTUInside "Principper for god videnskabelig praksis"/"Principles for good scientific conduct".

- Citations have to be clearly marked. The author and source have to indicated.
- The last also holds for pieces of programming code, which has been created by others.
- If you use material (text, code, images, musik, video, etc) which is copyrighted, you have to get the permission in writing from the owner of the copyright.
- Depending on the course, you might be allowed to integrate external packages/APIs into your project. Again you have to indicate this and state the source.

Structuring Analysis etc.

There are many ways to structure Analysis, Design and Implementation. Two popular ones are *Top-down* are *Bottom-up*. To choice between the two largely depends on the individual preference.

In the top-down approach the problem is broken into smaller and smaller pieces.



In the bottom-up approach, the problem is gradually build from smaller pieces.