# Writing Reports

Peter Sestoft, The IT University of Copenhagen

May 13, 2002

This document is a translation into English by Noah Torp-Smith of Peter Sestoft's 'Udformning af rapporter' from the course GP, May 2000 (If you want a Danish version, that document is recommended). It contains suggestions to the making of reports over smaller programming projects. When one generates a program to solve a certain problem, one has to describe and account for the construction of the program, explain how it is used, and argue that it solves the problem. The resulting report is a very important part of the project, as important as a working program.

## 1 The main sections of the report

The front page must indicate information such as author(s), title, date, course / activity, and the IT University of Copenhagen. Then these main sections should follow.

- Preface and introduction
- Background and description of the problem
- Problem analysis
- User's guide and examples
- Technical description of the program
- Test
- Conclusion
- References
- Appendixes: Test data, source code, images, etc.

In a four–week project with 3 participants, the sections can for example be 1, 5, 5, 5, 4, 4, 1 pages long, summing up to 25 pages, not including references and appendixes. There is no hard upper bound, but the longer the report, the harder it is to overview and understand it.

## 1.1 Preface and introduction

The preface indicates where, in what period, and in which connection the work has been performed. Write here, if the report is aimed at a special group of readers, and thank the people that have helped. The rest of the introduction describes and motivates the problem that is solved by the intended program, using no details.

Example:

> This report has been written in May 2000 in a four–week project at the IT University of Copenhagen under the supervision of .... We thank John Doe for assistance with the program for handling TIFF–images.
>
> During the project, we developed a Java–program for analysis of microscope images of lactic acid bacteria. The program analyses a TIFF file that is produced by a digital camera that in turn is connected to a microscope. It finds the contours of all bacteria above a certain size in the image, and prints these contours to a file in REG format. Files in REG can then be given as input to the standard program BactEdit from Yoyodyne Corp., and one therefore saves the tedious manual drawing of the bacteria's contours in BactEdit.

## 1.2 Background and description of problem

Here, one mentions the problem (administrative, biological, economical, ...) that the program deals with, and demands for the program are set. There must be enough background and explanations so that one can understand what it is that we want the program to do, and how it is supposed to do it. One can refer to relevant literature.

Use cases can be used to further specify the demands for the program. One describes how the program is used by a user: what possibilities must be present (e.g.: not only booking of a ticket, but also cancellation of it), what can go wrong (e.g.: try to book a ticket when there are no more seats left), how should the program react to such an error, etc.

The general design of the system is usually placed here. This can for example be the decision whether a system with a high server–load (the treatment of data takes place on the server, and the user only sees an HTML–based interface), or a system with high load on the client side (only data is treated by the server, all computations are performed by the clients), is chosen.

## 1.3 Problem Analysis

The previous section has determined what the program should be able to do. In this section, it is discussed how a program can be constructed so that it performs this task. The problem analysis can consist of a description of alternative solutions, e.g. different ways to represent data, and different ways to calculate desired results.

The possible solutions are presented at a notional level with the terminology and notation that is relevant for the problem, and with the present terminology for algorithms and data structures. One can use small pieces of actual code in the presentation where a certain precision is needed.

It is OK to describe both a simple and slow algorithm and a fast but complex one, and then decide to implement the slow one for reasons of time.

If a program utilizes a database, then the program analysis is the place to discuss database design: what tables are needed, what are the keys, and what are the relations between the tables?

## 1.4 User's guide and examples

In this section, the functionality of the program, without reference to its internal structure, is described. If a program is window–based, one should have pictures or drawings of the user interface and a description of the different components (buttons, menus, windows, etc.).

If the program reads and writes text files, the format and meaning of input and output data should be described, along with program parameters, if any.

If there are limits to the program, or certain requirements for the input data, they must be written in the user's manual. An example: "The program can only find outer contours, and thus does not work with shallow bacteria".

Explain what error messages the program can give, and their meanings.

Give a complete example of a use of the program, detailed enough for the reader to run the example herself (this is a good way to get to know a program). The example can, e.g., show the input, show how the example is executed, and show the output.

## 1.5 Technical description of the program

In this section, the most important modules, data structures, and algorithms are described. The description can be divided into subsections:

- Internal data structures: how are data represented in the program
- Internal algorithms: how are data treated in the program
- The user interface:
  - For window–based programs: what is the lay–out, what are the components, and what are the users;
  - For programs that work on text files: How are input and output read and written.

If you have made an object oriented design, then describe the purpose of each class. If a class is used to create objects with, then describe the most important (public) methods in the class. Normally, it is not recommended to go through each details of all classes. The technical description of the program is not supposed to be a repetition of what one can already read in the source code of the program, but rather a help to understand it.

### 1.6 Test

Here, it is shown to what extent the program solves the problem that is set. This can be done with a user test, or with a systematic (internal or) external test. One can do with a systematic test of only particularly interesting parts of the program. The example in the user's guide can sometimes be used as user test.

In any case, the given input data, the expected output data, the actual output data, and the observed discrepancies must be described. The section must contain a conclusion with an evaluation of the results of the test and its thoroughness: how great is the risk that there are no errors in the program that we have not found?

### 1.7 Conclusion

The conclusion must briefly sum up the work and its result. Is the program working? Is it practical to use? Is it fast enough? Should something have been done differently? Are there any extensions or improvements that ought to be made in future versions of the program?

### 1.8 Appendixes

The source code of the program must be enclosed as an appendix. It should be printed in a font with a fixed width, e.g. Courier, not Times. Make sure that the source code has a reasonable lay–out (indenting) so that it becomes readable.

## 2 The form of the report

Remember that the report is a mean of communication, not a literary experiment, and not a random selection of written pages.

- Write with a purpose: to communicate something, not to make a report longer.
- Write for a target group: have a certain reader in mind. Most sections are aimed at the person that has ordered the program, but the user's manual is aimed at persons that want to use the program, and the technical description is aimed at persons that wish to modify (correct or extend) the program.
- Keep it simple: Avoid unnecessary words; avoid pseudo–scientific formulations; avoid variations for the sake of variations; describe related ideas with related terms and formulations.
- Write correctly: slang, misspellings, errors, abbreviations and notions that are not explained, and wrong punctuation disturb the reader and impedes the communication.
- Make a table of contents, and number the sections and subsections, if it increases clarity.