How to write a Bachelor, Master, Diploma Thesis?
Commented Template for Bachelor, Master or Diploma Thesis

August 21, 2012

Nane Kratzke (please replace it with your name)

Lübeck University of Applied Sciences
Department of Electrical Engineering and Computer Science
Mönkhofer Weg 239
23562 Lübeck
Germany

Statement to the bachelor, master or diploma thesis

I assure that I have written the bachelor, master or diploma thesis independently, without outside help.

Only the given sources have been used for the writing of the diploma thesis. Literally or the sense after taken parts are marked as those.

I agree that my work is published, in particular that the work is presented to third parties for inspection or copies of the work are made to pass on to third parties.

Lübeck, August 21, 2012 Signature

Contents

List of Figures

List of Tables

Acknowledgement

This document is for students and is primarily used by students. Nevertheless it is also improved by students feedback and proposals. So let me thank all my students who have used this document so far to write a thesis.

- Summer 2011: Yinlong Xie, Kangping Liu and Jiewei Wang
- Summer 2012: Daniel Schultz, Philip Schikora, Michael Schultze, Ilja Antipov and Florian Hübner

Your valuable inputs improved this document over time. And I hope this is going on.

Let me especially thank **Florian Hübner** who wrote an astonishing bachelor thesis in summer 2012 [Hüb12]. A lot of examples are taken directly from his outstanding work. Thank you very much. It was a pleasure to work with him.

Chapter 1

Introduction

This document provides some general advices and requirements which should be regarded while writing a bachelor, master or diploma thesis with a clear focus on computer science – especially implementing or programming – related topics. If you have to deal with a complete other domain maybe the general advices of this template are of some interest for you. But it is very likely that all aspects described in the chapters 2 and following are only of minor relevance for you – especially if you have no software specific problem to solve.

The here mentioned structures, advices and hints are recommendations and do not have to be followed word by word in a dogmatic interpretation. Each thesis has special aspects making it necessary to differ in minor or major aspects from this template. Nevertheless all formulated advices of this template shall help you writing a well structured, understandable and traceable bachelor, master or diploma thesis.

You should be aware that this template summarizes only some essential aspects on technical writing. It is recommended to account other sources on technical writing as well, e.g.

- Barrass [Bar96] (especially English speaking students) or
- Rechenberg [Rec06] (especially German speaking students).

This template provides general writing advices in section 1.1 and additional hints how to write an introduction for your thesis document (see section 1.2).

The following chapters concentrate how to general structure a software intensive bachelor, master or diploma thesis. Therefore chapter 2 provides some tips for your initial literature research and requirements analysis. How to derive and structure a profound system architecture is described in chapter 3. Chapter 4 gives advices how to use these system architecture outcomes to derive and document your software as well as data architecture. The documentation of your detail implementation is covered in chapter 5. Chapter 6 might be of interest for you, if you have to document extensible features of your solution. You can skip this chapter, if you do not have to deal with extensibility. Some hints how to document your test and evaluation efforts can be found in chapter 7.

This template provides finally some additional hints how to summarize your work, derive some conclusions of major relevance and provide an outlook (if possible) in the last chapter 8.

1.1 General writing advices

Do not forget to write your thesis in parallel to your software modeling, programming and testing activities. Thesis which are written after the complete modeling, programming and testing activities are often hardly readable and understandable.

Be aware of the following points:

- With ongoing progress you are becoming more and more an expert within your thesis domain. Nevertheless your surrounding does not run through your very individual learning process. So – at some point – it is very likely that you assume non basic knowledge as basic. Think back! If you had this knowledge before you start your thesis it might be basic knowledge. If not – it is very likely that this knowledge could be hardly assumed basic knowledge. If unsure – assume that it is no basic knowledge. Be aware of this common pitfall.
- You should write your thesis for someone who has no detail knowledge about your thesis domain. Do not write your thesis for your supervisor. You should write it for someone who might want to continue your work (maybe another student of the following semester) and needs your knowledge and experience but starts to get into touch with your work from a low experience level.
- Your target audience may be computer scientists but it is likely that they do not have detail knowledge about your thesis domain.
- Especially your second supervisor might be no domain expert. Keep him oder her in mind while writing. But be aware it is very likely that you first get to know your second supervisor after you have finished writing your thesis. This is challenging but also very disciplining for writing.
- Remember that you write a technical documentation. Your style of writing should reflect this. Try to write in a formal and professional style you are used to know from software or other technical manuals. If unsure you should try to copy this style of manual writing (do not copy the texts).
- Never write sentences like: "I analyzed requirements by applying use case modeling techniques." Write instead: "Requirements were analyzed by applying use case modeling techniques." So AVOID "I" in your thesis. You are not writing a diary!

In the following sections you get some advices regarding the use of typesetting systems used in academia (see section 1.1), how to cite and reference (see section 1.1.2), how to structure texts (see section 1.1.3), how to embed and reference tables and figures (section 1.1.4) and finally how to handle the documentation of source code (see section 1.1.5).

1.1.1 Use typesetting systems which are common in academia

Typical word processors like MS Word or Open Office are not made for long texts with a lot of references, footnotes, tables, figures, etc. So sometimes you get into trouble if your texts are getting longer and longer as well as your referencing gets more and more complicated. This can be very frustrating especially in your last finalizing steps of your thesis. Nevertheless this is typical for technical documentation and it is typical for a thesis document as well. So you should think about more professional word processors like LaTeX which are very common in the technical and academical domain. These tools provide the following features which will make your every day live of thesis writing a lot simpler:

- Reliable image and table handling
- Reliable list handling (list of figures, tables as well as contents)
- Reliable cross reference handling
- Reliable mathematical formula handling
- Reliable source code handling
- Reliable footnote handling
- Reliable cite and bibliography handling

- Reliable PDF generation

You are advised (but not forced) to use the following freely available and approved TEX toolset.

- LYX is a graphical front end to LATEX. With LYX a lot of technical details of the very complex LATEX tool suite is hidden. You can use LYX almost like an Word processor like MS Word or Open Office. Nevertheless it provides all detail features of LATEX if someone needs it. LYX runs on Linux/UNIX, Windows or Mac based systems.
- LATEX is a document markup language and document preparation system for the TEX typesetting program. TEX is a popular means by which to typeset complex mathematical formulae or complex technical or academical texts; it has been noted as one of the most sophisticated digital typographical systems in the world. TEX is popular in academia, especially in mathematics, computer science, economics, engineering, physics, statistics, and quantitative psychology. LATEX runs on Linux/UNIX, Windows or Mac based systems.
- BibTEX is a reference management software for formatting lists of references. The BibTEX tool is typically used together with the LATEX document preparation system. BibTEX makes it easy to cite sources in a consistent manner, by separating bibliographic information from the presentation of this information, similarly to the separation of content and presentation/style supported by LATEX itself. BibTex runs on Linux/UNIX, Windows or Mac based systems.
- Like LYX for LATEX there exist a lot of front ends for BibTEX which help you to manage your references beyond a text file basis. One of the most popular BibTEX front ends JabRef. JabRef runs on Linux/UNIX, Windows or Mac systems.

You find all necessary links for the above mentioned tool suite in the following table 1.1 but it is also worth to check `http://www.latex-project.org/`.

Tool	Description	Download-URL
LYX	LATEX Front End	`http://www.lyx.org/Download` (Linux/UNIX, Windows, Mac)
LATEX	TEX based typesetting program (BibTEX is included)	`http://www.tug.org/protext/` (Windows) `http://www.tug.org/mactex/2011/` (Mac) You should use your package manager of your distribution in order to install all necessary LATEX packages and dependencies. Otherwise check `http://www.tug.org/texlive/` (Linux/UNIX)
JabRef	Java Based Reference Manager for the BibTEX System	`http://jabref.sourceforge.net/` (Linux/UNIX, Windows, Mac)

Table 1.1: A proper TEX Tool suite

If you plan to use the above mentioned tool suite this template file can be accessed as a LYX file via GitHub (`https://github.com/nkratzke/BAMA-Template`). It should be a reliable starting point for your documentation process.

1.1.2 Citing and referencing advices

References are important for the reader because references enable to "jump" between chapters, sections and paragraphs of your thesis document. Typically you will develop and describe your solution step by

step: Requirements, high level architecture, low level architecture, test and evaluation. Each of these steps should be described in identifiable elements[1] of your thesis document. But these elements can not be seen isolated. They depend on each other and can not be seen as linear elements. To make this obvious for the reader you should use so called forward and backward references.

A forward reference "points" to a text element which is following the actual text element. Forward references are used to provide a look ahead for the reader in order to provide him some informations not already mentioned but which might be of interest for him. Section 1.1.2.1 shows how to do this (and this is simultaneously a forward referencing example).

A backward reference "points" to a text element which stands before the actual paragraph. Forward references are used to remind the reader that something has been already described in your thesis in order to provide some summarizing facts of these paragraphs which might be of interest for him in order to understand the actual paragraph better. For (forward as well as backward) referencing it is a good strategy to use the referencing features of academical typesetting systems already mentioned in section 1.1.1 (and this is simultaneously a backward referencing example).

So references are used to provide guidance within your own thesis document and your own thoughts. Nevertheless you have to deal with other form of sources in your thesis. You will use books, articles, websites, etc. in order to collect and reflect necessary knowledge/facts from other authors publications. To use their ideas or texts in an unmarked way is called plagiarism and will make you fail! It is therefore common academical practice to use their publications but mark every idea which originating in other authors publications. Especially your complete literature research (see chapter 2) depends on this principle.

You should distinct cites and quotes. A cite is the typical form referencing an external publication and has to be enlisted in the bibliography of your thesis. We can distinct several forms of citing. All of them are shown by example in section 1.1.2.2.

- Word by word cites of small length: This is used if you want to repeat another authors statement of minor length exactly in the author words. If some words are omitted this is marked by using the following sign [...]. You should provide the page where the cited paragraph can be found.
- non localizing cite: This is used if you want to refer generally to another source where your ideas originate in but it is not possible to localize a particular section, page or other identifiable element within this source.
- localizing cite: This is used if you want to refer specifically to an identifiable part of another source where your ideas originate in and it is possible and of importance to provide the reader with the necessary information. Nevertheless it is not important to reflect the authors ideas word by word. You should provide the page where the cited paragraph can be found.

A quote is a special form of a word by word cite of major relevance. You use quotes to repeat some statements of another author with major relevance. Often only the author is known but it was not published as a book, article, etc.. You should handle quotes with care and use them with caution. Section 1.1.2.3 shows some quoting examples. If a quote can not be found in a referenceable publication it has not to enlisted in the bibliography of your thesis.

1.1.2.1 Example: How to reference between sections

The referencing principle is made obvious by providing an so called outline as an example. An outline describes the general structure of a document in a summarizing and forward referencing way. The same

[1] Chapter, section, paragraph, appendix, etc.

works for backward referencing. The very idea is not to provide only a chapter, section or subsection number but also provide a very compact summary of the (forward or backward) referenced passages. The outline of this template for example could be the following:

> This template provides some general advices for writing diploma thesis in chapter 1. Chapter 2 provides some tips for your initial literature research and requirements analysis. How to derive and structure a profound system architecture is described in chapter 3. Chapter 4 gives advices how to use these system architecture outcomes to derive and document your software as well as data architecture. The documentation of your detail implementation is covered in chapter 5. Chapter 6 might be of interest for you, if you have to document extensible features of your solution . You can skip this chapter, if you do not have to deal with extensibility. Some hints how to document your test and evaluation efforts can be found in chapter 7.
> This template provides additional hints how to summarize your work, derive some conclusions of major relevance and provide an outlook (if possible) in the last chapter 8.

1.1.2.2 Example: How to cite

Here is a typical word by word cite example. The detailed publication information can be found in the bibliography at the very end of this template (and your thesis document) by using the key [Bar10].

> Barr defines the term "*[...] cloud as a coherent, large-scale, publicly accessible collection of compute, storage, and networking resources. These are allocated via web service calls (a programmable interface accessed via HTTP requests), and are available for short- or long-term use in exchange for payment based on actual resources consumed.*" [Bar10, p. 5]

Here is a typical non localizing cite. The detailed publication information can be found in the bibliography at the very end of this template (and your thesis document) by using the key [Bar10].

> The following considerations rely on Barrs [Bar10] description how to use Amazon Web Services cloud assets to host scalable web information systems.

Here is a typical localizing cite in combination with non localizing cites. The detailed publication information can be found in the bibliography at the very end of this template (and your thesis document) by using the keys [Bar10], [Kra11b] and [Kra11a].

> Barr mentions beside other use cases training, data storage, disaster recovery and business continuity as well as business and scientific data processing as economical interesting use cases [Bar10, p. 13 - 18]. This is also reflected by Kratzke [Kra11b, Kra11a].

1.1.2.3 Example: How to quote

Quotes should be marked in the following way[2]:

> *"Everything should be made as simple as possible, but not simpler."*
> (Albert Einstein)
>
> *"We can't solve problems by using the same kind of thinking we used when we created them."*
> (Albert Einstein)
>
> *"Any intelligent fool can make things bigger, more complex [...]. It takes a touch of genius – and a lot of courage – to move in the opposite direction."*
> (Albert Einstein)

[2] `http://rescomp.stanford.edu/~cheshire/EinsteinQuotes.html`

1.1.3 Text structuring advices

This template provides a general structure of bachelor, master or diploma thesis. Nevertheless it provides "only" a top level structure. The structure within the here mentioned chapters are up to you. That is because every thesis is different in its details, every thesis deals with other problems to solve and an appropriate documentation structure depends mainly on the given (and thesis individual) problem domain. Nevertheless there exist some tips how to structure readable and well structured technical documentations. Try to keep them in mind.

- **Describe only the necessary.** Do not write sections because you think the mentioned facts are interesting. Every section has to be reasonable from the problem point of view. Your personal learning process has seldom something to do with the problem domain. So avoid to describe your learning process. Nobody is interested in that.
- **Number your sections hierarchically.** You should use the section handling features of an academical typesetting system to do this comfortable and frictionless (remember section 1.1.1 how to do this).
- **Introduce sections and subsections if they are appropriate from a problem point of view.** You should never introduce a section because you think that a heading would be "good looking". A heading should always be reasonable from a problem or object description point of view.
- **Do not over structure your thesis** in depth. Set you a maximum level of section depth before you start writing. A good depth would be three – for exceptional cases you can use an additional fourth section layer. Try to keep this level across your complete thesis documentation. If you recognize that you need additional section levels in one particular chapter it is time to think about a restructure of this chapter or even to split this chapter into two or more chapters. Maybe you also have to think about to partition your thesis into several main parts.
- **Do not introduce single subsections.** If you introduce only one subsection in a section this subsection is useless! Integrate it in your main section. When ever you introduce a section 1.x.1 you should have also a section 1.x.2. If not – integrate 1.x.1 with 1.x.
- **Align your section structure to the structure of the described object.** For example if you introduced a system architecture with five major building blocks on the same system level you could introduce five corresponding sections to describe them. Each section should document exactly one building block.
- **Use references to document dependencies** between described objects. If the described objects have dependencies you should reflect this between your corresponding sections as well (see section 1.1.2 how to reference efficiently).
- **Provide mini outlines for complex sections.** If sections are getting long and complex it is a good strategy to provide a starting mini outline reflecting the subsections 1.x.y of the main section 1.x (remember section 1.1.2.1 how to do this).
- **A headline stands never alone!** If you introduce a headline for 1.x which is directly followed by sub-headlines 1.x.1, 1.x.2, ..., 1.x.n you should think about integrating the section 1.x.1 as introduction for the sections 1.x.2, 1.x.3, ..., 1.x.n into section 1.x.
- **Use appendices to handle details.** If details are necessary you should describe the general and essential in the main document and reference a corresponding appendices section (see appendix A).

1.1.4 How to embed and reference tables and figures

Tables are primarily used for presenting data in a structured way. Nevertheless a table should never exist in a text without a reflecting context paragraph giving some additional information about the presented data.

Figures are primarily used for presenting data or complex aspects, relations, structures, etc. in a graphical way. A figure should never exist in a text without a reflecting context paragraph describing the intent and key information of the figure in words. In computer science and software modeling contexts these figures are often UML diagrams. Nevertheless the here mentioned advices are relevant for all kind of figures and diagrams.

Be aware of the following points:

- Number tables and figures. You should use the in section 1.1.1 mentioned typesetting table and figure handling features to handle this automatically and frictionless.
- Provide list of tables and figures. You should use the in section 1.1.1 mentioned typesetting features to handle this automatically and frictionless.
- Every embedded table or figure should be reflected and referenced from the written and surrounding text. You should use the in section 1.1.1 mentioned typesetting table and image embedding and referencing features to handle this automatically and frictionless.
- You should embed every table and figure in the same style (e.g. all tables and figures left-, right-adjusted or all centered, font-style and font-sizes in all tables the same, etc.).
- If you are using LyX/LaTeX you should embed all tables and figures in a floating style (that means the typesetting software decides where to place your tables and figures in text, you do not have to care about that).
- Keep your figures and tables readable. Be aware of to small fonts! Whenever possible try to avoid colors in your diagrams. Your thesis will be printed mostly on a greyscale basis.

Sections 1.1.4.1 and 1.1.4.2 show two examples how to embed and reference tables and figures within a text.

1.1.4.1 Example: How to embed and reference a table

Table 1.2 shows the grade scale of the last Object Oriented Programming exam. Only two students failed but most of them showed an astonishing good performance.

Grade	**1.0**	**1.3**	**1.7**	**2.0**	**2.3**	**2.7**	**3.0**	**3.3**	**3.7**	**4.0**	**5.0**
#	7	9	-	4	1	1	1	-	1	-	2

Table 1.2: Grade scale of last OOP exam

1.1.4.2 Example: How to embed and reference a figure

Figure 1.1 shows the general pattern of a Model View Controller (MVC) pattern, which is used in software engineering to separate user interface aspects from data handling (model) aspects in order to provide well structured as well as adaptable code.

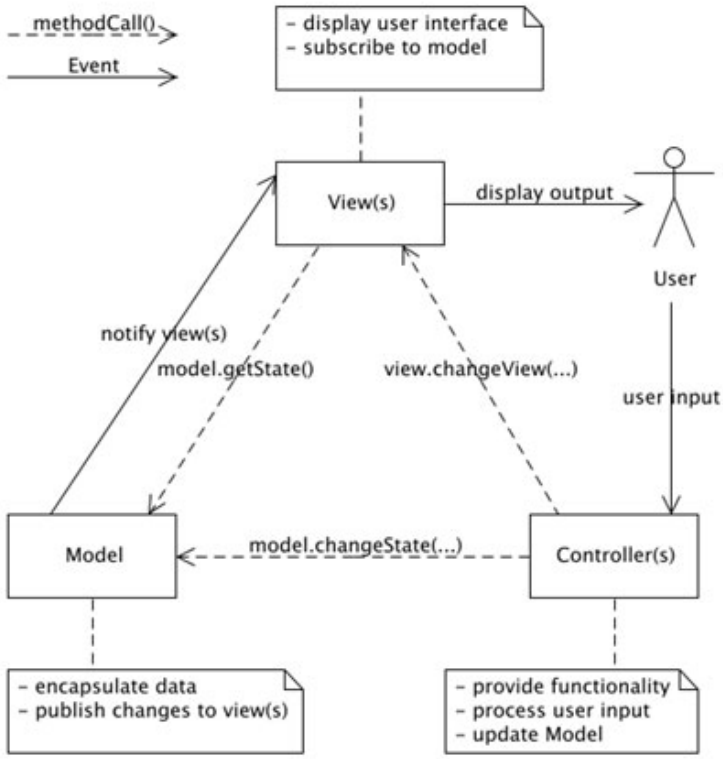

Figure 1.1: Model View Controller Pattern

1.1.5 How to handle source code

You will likely write a lot of code. But most of the code will not appear in your thesis document. Some crucial parts of your code might be referenced in an appendix or maybe in an implementation (see section 5) or test chapter (see section 7) if they are essential for understanding. Nevertheless you should embed as less source code as possible in your thesis document.

But if you need to (and you will typically in an implementation intensive thesis), you should do this by using special LaTeX packages. The LyX template file provided via GitHub[3] embeds the `listings` package[4] for those purposes and is preconfigured properly. You should use it.

This package provides to embed source code snippets like Listing 1.1 in the following style:

```
1  public class HelloWorld {
2
3    /**
4     * This is the main method where all begins
5     */
6    public static void main(String[] args) {
7      System.out.println("Hello, I got the following parameters:");
8      for(String a : args) {
9        System.out.println(a);
10     }
11   }
12
13 }
```

[3] https://github.com/nkratzke/BAMA-Template
[4] You can find initial documentation here: http://en.wikibooks.org/wiki/LaTeX/Packages/Listings

Listing 1.1: Another Hello World App

To use this in LyX you have to open a TeX section and insert the following TeX commands:

```
\lstset{language=JAVA,caption={Another Hello World App},label=listing}
\begin{lstlisting}
public class HelloWorld {

    /**
     * This is the main method where all begins
     */
    public static void main(String[] args) {

        System.out.println("Hello, I got the following parameters:");
        for(String a : args) {
            System.out.println(a);
        }
    }

}
\end{lstlisting}
```

It is also possible to embed complete source code listings by referencing the source code file. This is shown in appendix A.

1.2 Advices to introduce your thesis

Within your introduction chapter you should

- provide some general information of your problem domain,
- provide some overview information of your thesis and how your thesis embeds in the problem domain overview,
- describe some innovative aspects or approaches of your work in comparison to other publications/solutions in this field,
- and reflect your thesis task description (or statement of work).

In alignment with your thesis task description you should finally develop an general outline of your thesis document which helps the reader to understand the general structure of your thesis. Therefore the outline should provide an overview of the ongoing chapters of your thesis (see section how to write an efficient outline 1.1.2.1). Regard the general text structuring advices in section 1.1.3 in order to find an appropriate section structure for the introduction chapter.

Be aware of:

- Do not provide to much details in the introduction chapter.
- Do not assume any background knowledge from the reader in the introduction chapter.
- It is very likely that your introduction chapter changes more than ones. A good strategy is to write your introduction in the very beginning and write it in the finalizing phase completely new!

Chapter 2

Literature Research and Requirements Analysis

Within this chapter you should reflect your task description as well as the state of the art formulated in literature to formulate key aspects in order to derive and refine necessary requirements which might be essential for the ongoing guidance of your software modeling and implementation steps.

In some cases it might be useful to separate literature research and requirements analysis in two distinct chapters. This might be an appropriate strategy for a literature research intensive thesis. For most bachelor or diploma thesis with a clear software implementation focus the provided template structure should be sufficient.

If you use special requirements engineering techniques like use cases you should explain the key concepts of the technique from a general point of view (provide references to your sources) and provide reasons why this technique is appropriate for your problem.

Be aware of:

- You should keep an eye on your references.
- Do not copy word by word. If you do - this might be useful in some cases - mark these parts clearly as cites or quotes (check section 1.1.2 how to do that).
- Use the citing features of the in section 1.1.1 mentioned typesetting systems to handle this automatically and frictionless.
- Regard the general text structuring advices in section 1.1.3 in order to find an appropriate section structure for this chapter.
- You should align the structure of this section more or less to the statement of work of your thesis you have already reflected in chapter 1.

2.1 Product Feature Analysis

It is a good strategy to derive some key features of the to be developed, enhanced or modified software. You should describe your analysis strategy, reflect your references and of course you have to analyse your task description to derive some key product features. It is very helpfull to refine these product features as a list of of identfiable feature entities. The following list is an example of a real world bachelor thesis [Hüb12].

- **PF1** Administration (register, modify, unregister) of one or more AWS accounts
- **PF2** Cross-Account and Cross-Region administration (list, modify, delete, operate on) of EC2 ressources
- **PF3** Cross-Account and Cross-Region administration (list, modify, delete, operate on) of S3 ressources
- **PF4** Cross-Account and Cross-Region creation of EC2 images and instances
- **PF5** Cross-Account and Cross-Region creation of S3 buckets
- **PF6** Cross-Account and Cross-Region administration (create, delete, modify, assign/modify group based rights) of IAM users

Of course you should have introduced and explained all relevant wordings used in the product feature list.

2.2 Software Requirements Analysis

Next step could be a more detailled software requirements analysis. You should derive more detailled requirements by analysing your product feature catalog.There exist several requirement engineering methodologies to do this. Most common might be the use case analysis methodology. But often a pure textual refinement of your software requirments would be sufficient. Nevertheless you should describe your used methodology (provide references) and why it is relevant for your problem. Do not use a methodology unreflected!

It is absolutely essential that you derive a list of identifiable software requirements which can be listed in a form like this list here (The following list is an example of a real world bachelor thesis [Hüb12]).

- ***[Req1]*** Accounts can be registered
- ***[Req2]*** Account data can be changed
- ***[Req3]*** Account data can be deleted
- ***[Req4]*** Account data is stored in a database
- ***[Req5]*** Account data can be retrieved from a database
- ***[Req6]*** Account data can be shown to the user in a convenient way
- ***[Req7]*** ...

In a final requirements analysis step you should map your derived software requirements to the initial product feature list in order to perform a completeness check. The most efficient way to do this is to provide something like a mapping table. An example is shown in table 2.1.

As you can see in the example of table 2.1, several product features are not covered by software requirements. This means in fact these features will not be implemented. This might be OK and in accordance with your task description. Nevertheless you should provide reasonable rationales in these cases. Arguments should be derived by the task description.

Requirement x Feature	PF 1	PF 2	PF 3	PF4	PF5	PF6
Req 1 (register account)	x					
Req 2 (change account)	x					
Req 3 (delete account)	x					
Req 4 (save account)	x					
Req 5 (load account)	x					
Req 6 (show account)	x					
Req 7 (switch accounts)	x					
Req 8 (cross-account operators)		x	x			
Req 9 (cross-region opertors)		x	x			
Req 10 (show instances)		x				
Req 11 (start instances)		x				
Req 12 (stop instances)		x				
Req 13 (terminate instances)		x				
Req 14 (how volumes)		x				
Req 15 (delete volumes)		x				
Req 16 (show buckets)			x			
Req 17 (delete buckets)			x			
Req 18 (show security groups)		x				
Req 19 (delete security groups)		x				
Req 20 (show snapshots)		x				
Req 21 (delete snapshots)		x				
Req 22 (synchronisation)		x	x			
Req 23 (perform account clean up)		x	x			

Tabelle 2.1: Example of feature to requirement mapping (according to [Hüb12])

2.3 Prototyping (often used for GUIs)

It is furthermore a often used technique to provide small prototypes to analyse, demonstrate and adjust software functionalities in early phases of software development. All kind of protoyping activities can be described in this chapter as well as relevant prototyping outcomes and findings. You should avoid to document every prototyping detail here. Concentrate on essential ideas and main solution strategies you have analysed by using prototypes. Several screenshots (see figure 2.1) of your application provide typically a good initial "feeling" what the software is about from an end users point of view.

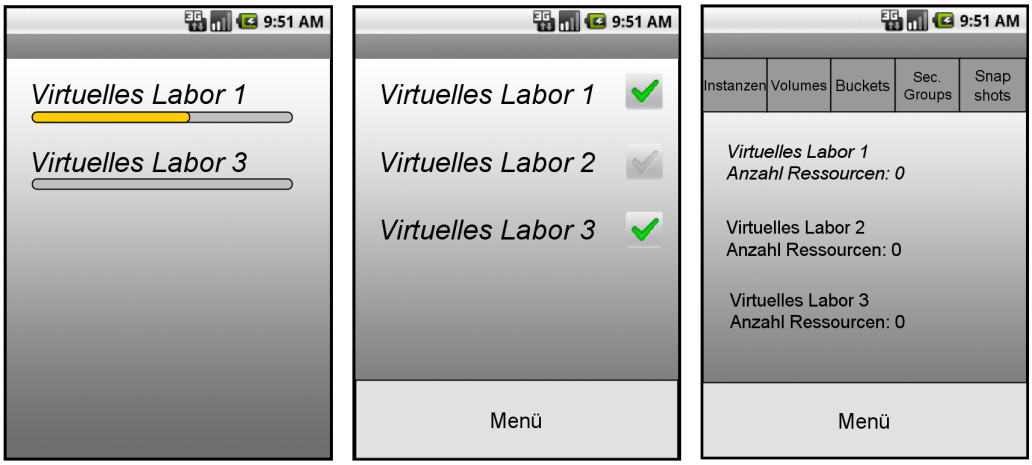

Abbildung 2.1: Example of several illustrating screenshots (here an Android application, [Hüb12])

2.4 General design decisions

A product can be realised in several ways. But you have to decide for exactly one way. This might be an appropriate section to describe existing and reflected design opportunities, their pros and cons and good reasons for your chosen approach which is guiding your system architecture development in the following steps.

Chapter 3

System Architecture

Use this chapter to reflect about the overall architecture of your problem domain and describe it as a general guidance structure to your solution. This might be the system and its interfaces you have to deal with (provide your sources you used to derive the system architecture). Or if the system architecture has to be developed you should use this chapter to document these steps and their relevant outcomes. In both cases you should align the derived requirements (see chapter 2) to the here presented system architecture. Do not forget to mention or develop architecture principles (e.g. loose coupling, publish-subscriber pattern, etc.) providing guidance to all following software engineering steps.

Whenever you have made architectural decisions influencing your solution you should reflect and document your reasons why you choose solution A instead of B or C. You should also document your decision criteria in these cases.

If you use special architecture modeling techniques like UML, SysML, etc. you should explain the key concepts of these techniques from a general point of view (provide references to your sources) and provide reasons why this technique is appropriate for your problem and how it fits to your used requirements engineering technique presented in chapter 2.

If you provide only a/some component(s) to a system/software you should use the here presented architecture to depict your part of responsibility (components of responsibility). It is a good strategy to do this in graphical way.

Be aware of:

- You should keep an eye on your references – especially when deriving your system architecture.
- Do not copy word by word. If you do - this might be useful in some cases - mark these parts clearly as cites or quotes (check section 1.1.2 how to do that).
- Use the citing features of the in section 1.1.1 mentioned typesetting systems to handle this automatically and frictionless.
- Regard the general text structuring advices in section 1.1.3 in order to find an appropriate section structure for this chapter. Your section structure should be aligned in some way to the structure of your top level architecture in order to support readability and understandability.

3.1 Architecture Principles

It is often possible to derive concrete architecture principles (e.g. loose coupling, publisher-subscriber pattern, layers, separation of concerns, abstraction, modularity, traceability, etc.) from your general design decisions (see section 2.4). If this is possible you should describe the relevant principles as well as their intents which are guiding your architecture development process.

3.2 Architecture Design Decisions

If there exist more than one usefull architecture you should discuss pros and cons of the several solutions and decide for the most appropriate one from your point of view.

3.3 System Architecture

To provide a general system understanding for the reader you typically have to develop or adapt/modify a system architecture. Do not mix your system architecture with your software architecture. A software architecture is typically more complex and is often UML intensive. System architectures should describe the principal structure of your solution from general point of view. The system architecture should be able to visualize your part of responsibilty if you have to provide only components for a bigger system (which is very often). Good system architectures can be developed by very primitive tools like Powerpoint. This is often absolutely sufficient. By developing your system architecture you provide the very general structure of your solution – do not try to describe and explain every detail in a system architecture. This is not the intent of a system architecture or a system context diagram. Figure 3.1shows an example for a good system architecture/context diagram.

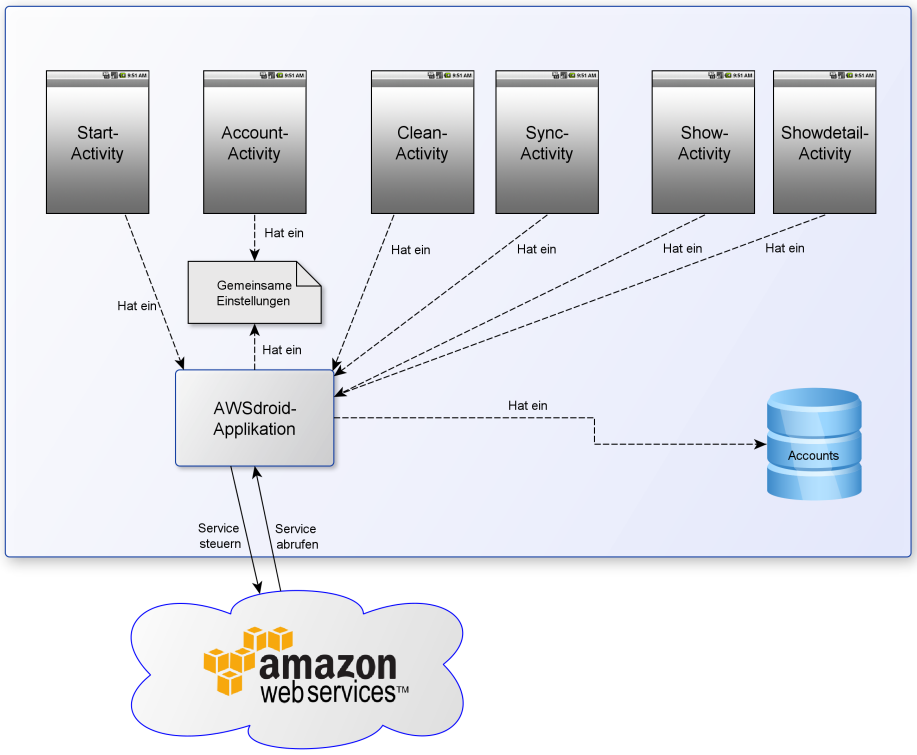

Abbildung 3.1: Example to provide a general system context, see [Hüb12]

It is furthermoe a good idea to map your architecture elements to your software requirements in order to check the completeness of your architecture. This can be done by providing a simple mapping table shown in table 3.1.

Requirement x System Architecture Entity	Account Activity	Start Activity	Sync Activity	Clean Activity	Show Activity	Show Detail Activity
Req 1 (register account)	x	x				
Req 2 (change account)	x	x				
Req 3 (delete account)	x	x				
Req 4 (save account)	x	x				
Req 5 (load account)	x	x				
Req 6 (show account)	x	x				
Req 7 (switch accounts)	x	x				
Req 8 (cross-account operators)		x	x	x		
Req 9 (cross-region opertors)		x	x	x	x	x
Req 10 (show instances)		x				x
Req 11 (start instances)		x			x	
Req 12 (stop instances)		x			x	
Req 13 (terminate instances)		x			x	
Req 14 (how volumes)		x				x
Req 15 (delete volumes)		x			x	
Req 16 (show buckets)		x				x
Req 17 (delete buckets)		x			x	
Req 18 (show security groups)		x				x
Req 19 (delete security groups)		x			x	
Req 20 (show snapshots)		x				x
Req 21 (delete snapshots)		x			x	
Req 22 (synchronisation)		x	x			
Req 23 (perform account clean up)		x		x		

Tabelle 3.1: Example of architecture entities to requirements mapping, see [Hüb12]

Chapter 4

Software Architecture and/or Software/Data Component Design

Within this chapter you should reflect the software architecture of your components of responsibility (defined in the chapter 3). Furthermore you should align the in chapter 2 derived requirements as well as the architecture principles and your components of responsibility (see chapter 3) to the here presented software architecture. The detail level depends on your thesis subject. It might be necessary to derive detail architectures for very complex components. It might be also necessary to provide some data models if you have an data intensive problem to solve.

Whenever you have made architectural decisions influencing your solution you should reflect and document your reasons why you choose solution A instead of B or C. You should also document your decision criteria in these cases.

If you use special software or data modeling techniques like UML, ERM, etc. you should explain the key concepts of these techniques from a general point of view (provide references to your sources) and provide reasons why this technique is appropriate for your special component problem and how it fits to your used requirements engineering technique (see chapter 2) as well as system architecture approach (see chapter 3).

Be aware of:

- Regard the general text structuring advices in section 1.1.3 in order to find an appropriate section structure for this chapter.
- Your section structure should be aligned in some way to the components of responsibility of your top level architecture (see chapter 3) in order to support readability, understandability as well as traceability.
- Do not provide to detailed models in this section. If you need to provide details of crucial relevance you should document this in the following chapter 5.

4.1 Software Architecture Design Decisions

If there exist more than one usefull software architecture you should discuss pros and cons of the several solutions and decide for the most appropriate one from your point of view.

4.2 Software Architecture

You should use the software architecture to present the general structure of your software from a software developers point of view. Your software architecture should detail and refine your general system architecture introduced in section 3.3. Typically you introduce the general package structure of the software and essential main classes. A good example shows figure 4 using in an appropriate way standard UML modeling diagrams.

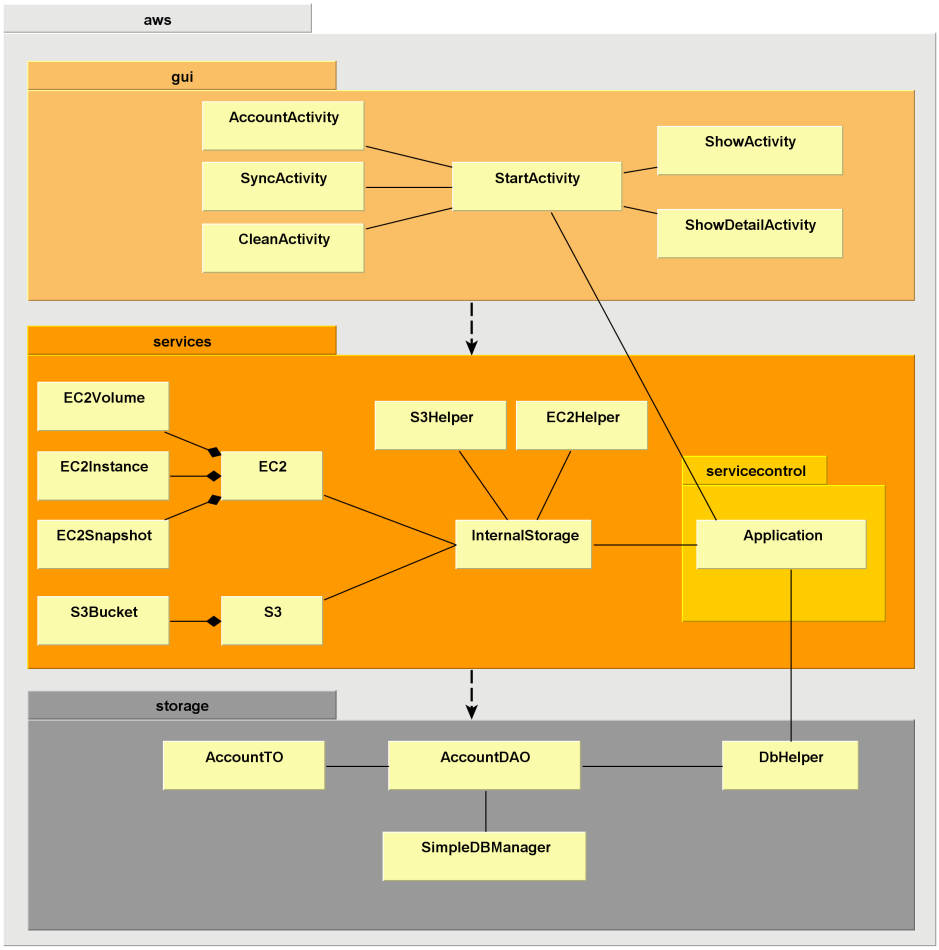

Abbildung 4.1: Example how to present a very general software structure, see [Hüb12]

For each package or class you introduce you should also provide some explaining paragraphs or sections. Do not assume that anyone can understand a software architecture by only viewing a picture.

Chapter 5

Implementation Description

Within this chapter you should reflect the implementation of your components of responsibility (defined in the chapter 3) and provide a general overview of your implementation as a general guidance. Describe recurring principles of your implementation.

You should align the general structure of this chapter according to your software architecture developed in section 4.2.

Whenever you have made implementation decisions influencing your solution, performance or your requirements you should reflect and document your reasons why you choose solution A instead of B or C. You should also document your decision criteria in these cases.

If you use different programming languages like JAVA, Python, C, C++, Haskell, PHP, etc. you should explain the key programming paradigms of these programing languages from a general point of view (provide references to your sources) and provide reasons why you used which language for which component. You should also describe how do you have handled architecture principles on implementation level.

If you use code generation techniques (e.g. often GUI Builder use code generators – it is not always obvious that in some aspects code generation is involved!) you should explain the key concepts of these techniques and describe how you applied these techniques in order to solve your problem. You should also reflect which artifacts are generated and whether these generated artifacts have to be handled in a special manner – especially when you have to deal with round trip engineering and testing aspects (see chapter 7).

Furthermore you should concentrate in your implementation description on very special aspects which are crucial for your solution. E.g. special algorithms, crucial interfaces, components of high criticality, interaction principles, etc.

Be aware of:

- Regard the general text structuring advices in section 1.1.3 in order to find an appropriate section structure for this chapter.
- Your section structure should be aligned in some way to the components of your software architecture (see chapter 4) in order to support readability, understandability as well as traceability.
- Do not provide detailed source code in this section. If you need to provide source code details of crucial relevance you should do this according to section 1.1.5.

Chapter 6

{Extensibility}

This chapter is definitely an optional chapter and only necessary if a thesis deals with extendable (plugin) software. In this case this chapter should reflect how your software is extendable. If you do not provide a plugin or a similar concept this chapter can be skipped.

It is a good strategy to identify and describe the relevant extension points of your solution and provide some general remarks how your solution can be extended.

I a second step you should demonstrate how your solution can be extended by a simple example in step by step how-to/tutorial manner.

Be aware of:

- Regard the general text structuring advices in section 1.1.3 in order to find an appropriate section structure for this chapter.
- Your section structure should be aligned in some way to the identified extension points of your solution.
- Typically you have to provide some source code examples in this section. You should do this according to section 1.1.5.

Chapter 7

Test and Evaluation

Within this chapter you should reflect your test and evaluation strategy and provide corresponding test and evaluation results for your components or responsibility in order to prove that you have fulfilled the guiding requirements derived in chapter 2.

Describe recurring principles of your test and evaluation strategy. You should define criteria when you rate a test successful. These criteria have to be derived by your requirements (see chapter 2), the system as well as the software architecture (see chapter 3 and 4).

Maybe you have to develop a (small) test architecture – especially when you have to deal with test mocks and external interfaces which are not belonging to your area of responsibility but are of crucial importance for the system.

If you use special test or evaluation techniques like unit testing, coverage, acceptance testing, mocks, etc. you should explain the key paradigms of these test and evaluation techniques from a general point of view (provide references to your sources) and provide reasons why you used which technique for which component/requirement. You should also describe how you regarded the architecture influence on test level.

If you use code generation techniques (e.g. GUI Builder use code generators) you should explain how these generated code parts have been tested and why.

Be aware of:

- Regard the general text structuring advices in section 1.1.3 in order to find an appropriate section structure for this chapter.
- Your section structure should be aligned in some way to
  - your requirements (see chapter 2),
  - the components of your system (see chapter 3) as well as
  - components of your responsibility (see chapter 4) in order to support readability, understandability as well as traceability.
- Do not forget your interfaces (between components of your responsibility but also between components of your and external responsibility) derivable from chapter 3. Your test strategy should also provide test results which proof that you interact correctly with interfaces that do not belong to your responsibility.
- Do not provide detailed source code in this section. If you need to provide source code details of crucial relevance you should do this according to section 1.1.5.

7.1 Test categories

Your test strategy should derive several test categories. Example of test categories could be (taken from [Hüb12]):

- **GUIT** Graphical User Interface Tests to test user interface behaviour
- **UT** Unit Tests to test classes (you might also think about the systematic use of code coverage tools in order to measure the unit testing quality by objective metrics)
- **ST** System Tests to test complex system behaviour
- **LT** Live Tests to test system behaviour in live environments

Describe each test category, their intent and why you introduce them.

7.2 Test cases

In a next step you should develeop and describe test cases for each test category introduced in section 7.1. All test cases should cover the whole application. Therefore it is relevant to derive test cases from your architecture and your requirements. You should document this coverage by providing one or mapping tables. Table 7.1 shows an example how to map test cases to requirements introduced in section 2.2. The same is possible for a mapping against architectural entities introduced in section 3.3 or 4.2.

7.3 Test results

Finally you should provide informations concerning your test results. Especially non successfull tests should be discussed in detail. You should derive a statement whether you have reached the goal of your initial task description by reflecting your test results from a critical point of view. Use your mappings (test cases -> requirements, test cases -> architecture, architecture -> requirements, requirements -> product features) to argue whether short comings of your tests are vital or not.

It is always a good idea to present your test results in a well-arranged (this is mostly a tabular) form (see table 7.2as an example).

		Requirements																			
Testlevel		1-2	3	4-5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
GUIT	GUI1	x																				
	GUI2	x																				
	GUI3		x																			
	GUI4				x																	
UT	U1	x																				
	U2	x																				
	U3			x																		
	U4			x																		
	U5								x				x				x		x			
	U6														x							
ST	S1	x																				
	S2	x																				
	S3		x																			
	S4				x																	
	S5					x																
	S6						x															
	S7							x														
	S8									x												
	S9										x											
	S10											x										
	S11													x								
	S12															x						
	S13																	x				
	S14																			x		
	S15																				x	
	S16																					x
LT	L1									x												
	L2										x											
	L3											x										
	L4													x								
	L5															x						
	L6																	x				
	L7																			x		
	L8																				x	
	L9																					x

GUIT GUI test **UT Unit test** **ST System test** **LT Live test**

Tabelle 7.1: Example of a test mapping, see [Hüb12]

Testlevel	Test-Nr.	Description of the test case	Test result
GUIT	GUI1	Dialog zum Registrieren eines Accounts aufrufen	successfull
	GUI2	Dialog zum Ändern eines Accounts aufrufen	successfull
	GUI3	Dialog zum Löschen eines Accounts aufrufen	successfull
	GUI4	Registrierte Accounts können angezeigt werden	successfull
UT	U1	Accountdaten prüfen (Vorbedingung für Req 1)	successfull
	U2	Accountdaten in der Datenbank aktualisieren	successfull
	U3	Accountdaten in die Datenbank speichern	successfull
	U4	Accountdaten aus der Datenbank laden	successfull
	U5	Synchronisation des EC2 Service (ohne Anzeige)	successfull
	U6	Synchronisation des S3 Service (ohne Anzeige)	successfull
ST	S1	Account kann über Menü registriert werden	successfull
	S2	Account kann über lange Auswahl geändert werden	successfull
	S3	Account kann über lange Auswahl gelöscht werden	successfull
	S4	Registrierte Accounts können angezeigt werden	successfull
	S5	Accounts können aktiviert / deaktiviert werden	successfull
	S6	Operationen erfolgen auf alle aktiven Accounts	successfull
	S7	Operationen erfolgen über alle Web Service Regionen	successfull
	S8	Instanzen können gestartet werden	successfull
	S9	Instanzen können gestoppt werden	successfull
	S10	Instanzen können terminiert werden	successfull
	S11	Volumes können gelöscht werden	successfull
	S12	Buckets können gelöscht werden	successfull
	S13	Security Groups können gelöscht werden	successfull
	S14	Snapshots können gelöscht werden	successfull
	S15	Manuelle Synchronisation kann ausgelöst werden	successfull
	S16	Manuelle Bereinigung kann ausgelöst werden	successfull
LT	L1	Instanzen können gestartet werden	successfull
	L2	Instanzen können gestoppt werden	successfull
	L3	Instanzen können terminiert werden	successfull
	L4	Volumes können gelöscht werden	successfull
	L5	Buckets können gelöscht werden	successfull
	L6	Security Groups können gelöscht werden	successfull
	L7	Snapshots können gelöscht werden	successfull
	L8	Manuelle Synchronisation kann ausgelöst werden	successfull
	L9	Manuelle Bereinigung kann ausgelöst werden	successfull

Tabelle 7.2: Example of a test result overlook, see [Hüb12]

Chapter 8

Summary, Conclusions and Outlook

This is primarily a recapitulating chapter. Nevertheless – beside your introduction chapter – it is likely that it becomes the most read part of your thesis. So do not underestimate your "finishing" chapter. It is an absolutely essential chapter of your thesis. You should

- summarize your major findings of your literature research and results of your requirements analysis (see chapter 2),
- reflect your overall architecture and guiding architecture principles (see chapter 3),
- pinpoint highlights and advantages of your software architecture (see chapter 4) and your implementation (see chapter 5),
- describe extension points and principles of your solution in order to summarize how your solution can be extended (only if extendable, see chapter 6),
- summarize your test/evaluation strategy and your test/evaluation results in context of your thesis requirements in order to emphasize your results (see chapter 7).

Finally you should draw some conclusions in order to show what your solution is contributing to your problem context. If you have developed ideas during your thesis realization which could not be realized due to comprehensible or obvious reasons you could state them here as an outlook.

Be aware of:

- Do not provide to much details in this chapter.
- Do not assume any background knowledge from the reader (like in the introduction chapter).
- After you have finished you this chapter you should think about a complete rework of your introduction chapter in order to get both deeply aligned. Remember the introduction and this summary chapter are the most read chapters of your thesis. They are little bit like a business card of yours.

And finally: May be you want provide an additional foreword? Or some acknowledgments in own sections? This would give your thesis a personal touch and would enable to express your gratitude to persons who had significant intellectual impact to your thesis or supported you in a mentionable manner.

Appendix A

Detailed Information Example

If you want to provide detailed information to some aspects, e.g. detailed configuration options. You should describe the main and key aspects within your thesis document and reference all additional details to an appropriate appendix section. So you can keep your text well structured, understandable and readable as well as provide all necessary details.

May be you want to embed a long source code like listing A.1 of crucial importance? This could be done in the following style

```
\lstset{caption={WRSC Races - A googlemap drupal module},label=wrscraces}
\lstinputlisting[language=PHP]{wrscraces.module}
```

and would produce the following output:

```php
<?php
/**
 * Implementation of hook_menu().
 */
function wrscraces_menu() {
  $items = array();

  $items['wrsc/racemap'] = array(
    'title' => 'WRSC Races',
    'description' => 'Presents WRSC Races on a googlemap',
    'page callback' => 'wrscraces_map_menucallback',
    'page arguments' => array(),
    'access arguments' => array('access map'),
    'type' => MENU_NORMAL_ITEM,
  );

  $items['admin/wrscraces/settings'] = array(
    'title' => 'WRSC Admin',
    'description' => 'Setting up the worldserver for WRSC Races',
    'page callback' => 'drupal_get_form',
    'page arguments' => array('wrscraces_settings_form'),
    'access arguments' => array('admin world server access'),
  );

  return $items;
}

```

```
28 /**
29  * Implementation of hook_perm().
30  */
31 function wrscraces_perm() {
32   return array('admin world server access', 'access map');
33 }
34
35 /**
36  * Helper Function to embed javascript
37  * in a more convenient way
38  */
39 function _wrscraces_embed_script($path) {
40   $START = '<script type="text/javascript" src="';
41   $END = '"></script>';
42   drupal_set_html_head($START.$path.$END);
43 }
44
45 /**
46  * Embeds a google map with WRSC races
47  */
48 function wrscraces_map_menucallback() {
49
50   _wrscraces_embed_script("http://maps.google.com/maps/api/js?sensor=true");
51   _wrscraces_embed_script("/".drupal_get_path('module', 'wrscraces').'/js/updatewrscmap.
       js');
52
53   $lat = variable_get('wrsc.center.lat', 0);
54   $lng = variable_get('wrsc.center.lng', 0);
55
56   $out = "<div id='wrscrace_map_canvas' style='width:100%; height:600px'></div>\n";
57   $out .= "<script type='text/javascript'>wrsc_initialize($lat, $lng);</script>\n";
58
59   return $out;
60 }
61
62 /**
63  * Form for setting up the center position of the race area
64  */
65 function wrscraces_settings_form() {
66   $form['wrsc_settings'] = array(
67     "#type" => "fieldset",
68     "#title" => t("WRSC Center Position")
69   );
70
71   $form['wrsc_settings']['lat'] = array(
72     '#type' => 'textfield',
73     '#title' => t('latitude'),
74     '#size' => 10,
75     '#default_value' => variable_get('wrsc.center.lat', 0),
76     '#maxlength' => 20,
77     '#description' => t("Geographic latitude, e.g. 53.8725"),
78   );
79
80   $form['wrsc_settings']['lng'] = array(
81     '#type' => 'textfield',
82     '#title' => t('longitude'),
83     '#size' => 10,
84     '#default_value' => variable_get('wrsc.center.lng', 0),
```

```
85      '#maxlength' => 20,
86      '#description' => t("Geographic longitude, e.g. 10.7045"),
87    );
88
89    $form['wrsc_settings']['submit'] = array(
90      '#type' => 'submit',
91      '#value' => t('Set WRSC Race Area Center'),
92    );
93
94    return $form;
95  }
96
97  /**
98   * Validate hook for wrsc_settings_form
99   * Will be executed whenever the settings form is submitted
100  */
101 function wrscraces_settings_form_validate($form, &$form_state) {
102   $lat = $form_state['values']['lat'];
103   $lng = $form_state['values']['lng'];
104
105   if (!is_numeric($lat)) form_set_error('', t('Please provide a numeric value for
          latitude')." not $lat");
106   if (!is_numeric($lng)) form_set_error('', t('Please provide a numeric value for
          longitude')." not $lng");
107
108   if ($lat > 90.0 || $lat < -90.0) form_set_error('', t('Please provide a numeric value
          for latitude between -90.0 and +90.0'));
109   if ($lng > 180.0 || $lng < -180.0) form_set_error('', t('Please provide a numeric value
           for latitude between -180.0 and +180.0'));
110 }
111
112 /**
113  * Submit hook for wrsc_settings_form
114  * Will be executed whenever the settings form is validated without errors
115  * Stores the center of race area into the persistent variables
116  * wrsc.center.phi
117  * wrsc.center.lambda
118  */
119 function wrscraces_settings_form_submit($form, &$form_state) {
120   $lat = $form_state['values']['lat'];
121   $lng = $form_state['values']['lng'];
122
123   variable_set('wrsc.center.lat', $lat);
124   variable_set('wrsc.center.lng', $lng);
125
126   drupal_set_message(t('WRSC Racing Area is set to ')."lat: $lat lng: $lng");
127 }
128 ?>
```

Listing A.1: WRSC Races - A googlemap drupal module

Bibliography

[Bar96] R. Barrass. *Scientists Must Write: A guide to better writing for scientists, engineers and students*. UK: Oxford University Press, 1996.

[Bar10] Jeff Barr. *Host Your Web Site in the Cloud - Amazon Web Services made easy*. sitepoint, 2010.

[Hüb12] Florian Hübner. Mobile app zur verwaltung virtueller labore. Bachelor's thesis, 2012.

[Kra11a] Nane Kratzke. Cloud-based it management impacts - qualitative weaknesses and strengths of clouds. In *CLOSER 2011 - 1st International Conference on Cloud Computing and Services Science*, pages 145 – 151, Nordwijkerhout, 05/2011 2011.

[Kra11b] Nane Kratzke. Overcoming ex ante cost intransparency of clouds - using system analogies and a corresponding cost estimation model. In *CLOSER 2011 - 1st International Conference on Cloud Computing and Services Science (special session on Business Systems and Aligned IT Services - BITS 2011)*, pages 707 – 716, Nordwijkerhout, 05/2011 2011.

[Rec06] Peter Rechenberg. *Technisches Schreiben (nicht nur) für Informatiker*. Hanser Verlag, 2006.