# Template for Bachelor, Master or Diploma Thesis

February 9, 2012

YOUR NAME

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

## Statement to the diploma thesis

I assure that I have written the 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, February 9, 2012 Signature

# Contents

# List of Figures

# List of Tables

# 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 implementating or programming – related topics. 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 maybe necessary to differ in minor or major parts from this template. Nevertheless all formulated advices of this template shall help you writing a well structured as well as understandable bachelor, master or diploma thesis.

## 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 analysed requirements by applying use case modeling techniques." Write instead: "Requirements were analysed by applying use case modeling techniques." So AVOID "I" in your thesis. You are not writing a diary!

## 1.2 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 toolsuite is hidden. You can use LyX almost like an Wordprocessor 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 toolsuite 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)<br>`http://www.tug.org/mactex/2011/` (Mac)<br>You should use your package manager of your distribution in order to install all necessary LaTeX packages and dependencies. Otherwise check<br>`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 Toolsuite

If you plan to use the above mentioned toolsuite 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.

## 1.3 Citing and referencing advices

### 1.3.1 Example: How to reference between sections

### 1.3.2 Example: How to cite

### 1.3.3 Example: How to quote

Quotes should be marked in the following way[1]:

> *"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."*

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

(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)

## 1.4 Text structuring advices

## 1.5 How to embed and reference tables and figures

Tables are primarilly used for presenting data in a structured way. Nevertheless a table should never exist in a text without a reflecting context paragraph.

Figures are primarilly 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 1.2 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 1.2 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 1.2 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.

Here are two examples how to embed and reference tables and figures within a text.

### 1.5.1 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

### 1.5.2 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.

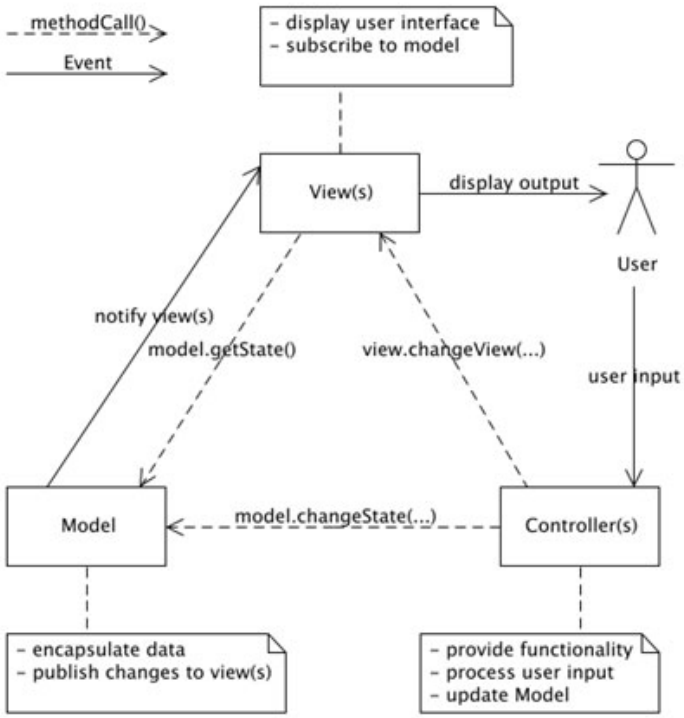


Figure 1.1: Model View Controller Pattern

## 1.6 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. This template file provided via GitHub[2] embeds the `listings` package[3] for these purposes and configures .

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

```
1 public class HelloWorld {
2
3   /**
4    * This is the main method where all begins
5    */
```

[2] `https://github.com/nkratzke/BAMA-Template`

[3] You can find intial documentation here: `http://en.wikibooks.org/wiki/LaTeX/Packages/Listings`

```
6   public static void main(String[] args) {
7     System.out.println("Hello, you have passed the following parameters:");
8     for(String a : args) {
9       System.out.println(a);
10    }
11  }
12
13 }
```

Listing 1.1: Another Hello World App

In LyX you have to open a TeX section and insert the following:

```
\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, you have passed the following parameters:");
        for(String a : args) {
            System.out.println(a);
        }
    }

}
\end{lstlisting}
```

# Chapter 2

# Literature Research and Requirements Analysis

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

In some cases it might be usefull to seperate 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 might 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 usefull in some cases - mark these parts clearly as cites or quotes (check section 1.3 how to do that).
- Use the citing features of the in section 1.2 mentioned typesetting systems to handle this automatically and frictionless.
- Regard the general text structuring advices in section 1.4 in order to find an appropriate section structure for this chapter.

# Chapter 3

# System Architecture

Within this chapter you should reflect 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 here. In both cases you should align the in chapter 2 derived requirements 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 usefull in some cases - mark these parts clearly as cites or quotes (check section 1.3 how to do that).
- Use the citing features of the in section 1.2 mentioned typesetting systems to handle this automatically and frictionless.
- Regard the general text structuring advices in section 1.4 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.

# 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 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 1.4 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.

# 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.

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 you regarded architecture principles on implementation level.

If you use code generation techniques (e.g. GUI Builder use code generators) 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 artefacts are generated and if these generated artefacts has to be handled in a special manner – especially when you have to deal with round trip engineering apsects.

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 1.4 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 1.6.

# Chapter 6

# {Extensibility}

This chapter is defenitely an optional chapter and only necessary if a thesis deals with extandable (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 howto/tutorial manner.

Be aware of:

- Regard the general text structuring advices in section 1.4 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 1.6.

# Chapter 7

# Test and Evaluation

# Chapter 8

# Conclusions and Outlook

# Appendix I

If you want to provide detailled information to some aspects, e.g. detailled 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.

# Bibliography