Clean Code Development

Stefan Kert

BACHELORARBEIT

Nr. S1310307019

eingereicht am
Fachhochschul-Bachelorstudiengang

Software Engneering

in Hagenberg

im Juli 2016

Diese Arbeit entstand im Rahmen des Gegenstands

Gegenstand??

im

Semester??

Betreuer:

Josef Pichler, Dr.

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 14. Juli 2016

Stefan Kert

Inhaltsverzeichnis

Vorwort

Dies ist **Version 2015/09/19** der LaTeX-Dokumentenvorlage für verschiedene Abschlussarbeiten an der Fakultät für Informatik, Kommunikation und Medien der FH Oberösterreich in Hagenberg, die mittlerweile auch an anderen Hochschulen im In- und Ausland gerne verwendet wird.

Das Dokument entstand ursprünglich auf Anfragen von Studierenden, nachdem im Studienjahr 2000/01 erstmals ein offizieller LaTeX-Grundkurs im Studiengang Medientechnik und -design an der FH Hagenberg angeboten wurde. Eigentlich war die Idee, die bereits bestehende *Word*-Vorlage für Diplomarbeiten „einfach“ in LaTeX zu übersetzen und dazu eventuell einige spezielle Ergänzungen einzubauen. Das erwies sich rasch als wenig zielführend, da LaTeX, vor allem was den Umgang mit Literatur und Grafiken anbelangt, doch eine wesentlich andere Arbeitsweise verlangt. Das Ergebnis ist – von Grund auf neu geschrieben und wesentlich umfangreicher als das vorherige Dokument – letztendlich eine Anleitung für das Schreiben mit LaTeX, ergänzt mit einigen speziellen (mittlerweile entfernten) Hinweisen für *Word*-Benutzer. Technische Details zur aktuellen Version finden sich in Anhang **??**.

Während dieses Dokument anfangs ausschließlich für die Erstellung von Diplomarbeiten gedacht war, sind nunmehr auch *Masterarbeiten*, *Bachelorarbeiten* und *Praktikumsberichte* abgedeckt, wobei die Unterschiede bewusst gering gehalten wurden.

Bei der Zusammenstellung dieser Vorlage wurde versucht, mit der Basisfunktionalität von LaTeX das Auslangen zu finden und – soweit möglich – auf zusätzliche Pakete zu verzichten. Das ist nur zum Teil gelungen; tatsächlich ist eine Reihe von ergänzenden „Paketen“ notwendig, wobei jedoch nur auf gängige Erweiterungen zurückgegriffen wurde. Selbstverständlich gibt es darüber hinaus eine Vielzahl weiterer Pakete, die für weitere Verbesserungen und Finessen nützlich sein können. Damit kann sich aber jeder selbst beschäftigen, sobald das notwendige Selbstvertrauen und genügend Zeit zum Experimentieren vorhanden sind. Eine Vielzahl von Details und Tricks sind zwar in diesem Dokument nicht explizit angeführt, können aber im zugehörigen Quelltext jederzeit ausgeforscht werden.

Zahlreiche KollegInnen haben durch sorgfältiges Korrekturlesen und kon-

struktive Verbesserungsvorschläge wertvolle Unterstützung geliefert. Speziell bedanken möchte ich mich bei Heinz Dobler für die konsequente Verbesserung meines „Computer Slangs“, bei Elisabeth Mitterbauer für das bewährte orthographische Auge und bei Wolfgang Hochleitner für die Tests unter Mac OS.

Die Verwendung dieser Vorlage ist jedermann freigestellt und an keinerlei Erwähnung gebunden. Allerdings – wer sie als Grundlage seiner eigenen Arbeit verwenden möchte, sollte nicht einfach („ung'schaut“) darauf los werken, sondern zumindest die wichtigsten Teile des Dokuments *lesen* und nach Möglichkeit auch beherzigen. Die Erfahrung zeigt, dass dies die Qualität der Ergebnisse deutlich zu steigern vermag.

Der Quelltext zu diesem Dokument sowie das zugehörige LaTeX-Paket sind in der jeweils aktuellen Version online verfügbar unter

https://sourceforge.net/projects/hgbthesis/.

Trotz großer Mühe enthält dieses Dokument zweifellos Fehler und Unzulänglichkeiten – Kommentare, Verbesserungsvorschläge und passende Ergänzungen sind daher stets willkommen, am einfachsten per E-Mail direkt an mich:

Dr. Wilhelm Burger, Department für Digitale Medien,
Fachhochschule Oberösterreich, Campus Hagenberg (Österreich)
wilhelm.burger@fh-hagenberg.at

Übrigens, hier im Vorwort (das bei Diplom- und Masterarbeiten üblich, bei Bachelorarbeiten aber entbehrlich ist) kann kurz auf die Entstehung des Dokuments eingegangen werden. Hier ist auch der Platz für allfällige Danksagungen (z. B. an den Betreuer, den Begutachter, die Familie, den Hund, ...), Widmungen und philosophische Anmerkungen. Das sollte allerdings auch nicht übertrieben werden und sich auf einen Umfang von maximal zwei Seiten beschränken.

Kurzfassung

An dieser Stelle steht eine Zusammenfassung der Arbeit, Umfang max. 1 Seite. Im Unterschied zu anderen Kapiteln ist die Kurzfassung (und das Abstract) üblicherweise nicht in Abschnitte und Unterabschnitte gegliedert. Auch Fußnoten sind hier falsch am Platz.

Kurzfassungen werden übrigens häufig – zusammen mit Autor und Titel der Arbeit – in Literaturdatenbanken aufgenommen. Es ist daher darauf zu achten, dass die Information in der Kurzfassung für sich *allein* (d. h. ohne weitere Teile der Arbeit) zusammenhängend und abgeschlossen ist. Insbesondere werden an dieser Stelle (wie u. a. auch im *Titel* der Arbeit und im *Abstract*) normalerweise *keine Literaturverweise* verwendet! Falls unbedingt solche benötigt werden – etwa weil die Arbeit eine Weiterentwicklung einer bestimmten, früheren Arbeit darstellt –, dann sind *vollständige* Quellenangaben in der Kurzfassung selbst notwendig, z. B. [Zobel J.: *Writing for Computer Science – The Art of Effective Communication.* Springer-Verlag, Singapur, 1997].

Weiters sollte daran gedacht werden, dass bei der Aufnahme in Datenbanken Sonderzeichen oder etwa Aufzählungen mit „Knödellisten“ in der Regel verloren gehen. Dasselbe gilt natürlich auch für das *Abstract*.

Inhaltlich sollte die Kurzfassung *keine* Auflistung der einzelnen Kapitel sein (dafür ist das Einleitungskapitel vorgesehen), sondern dem Leser einen kompakten, inhaltlichen Überblick über die gesamte Arbeit verschaffen. Der hier verwendete Aufbau ist daher zwangsläufig anders als der in der Einleitung.

Abstract

This should be a 1-page (maximum) summary of your work in English.

Im englischen Abstract sollte inhaltlich das Gleiche stehen wie in der deutschen Kurzfassung. Versuchen Sie daher, die Kurzfassung präzise umzusetzen, ohne aber dabei Wort für Wort zu übersetzen. Beachten Sie bei der Übersetzung, dass gewisse Redewendungen aus dem Deutschen im Englischen kein Pendant haben oder völlig anders formuliert werden müssen und dass die Satzstellung im Englischen sich (bekanntlich) vom Deutschen stark unterscheidet (mehr dazu in Abschn. **??**). Es empfiehlt sich übrigens – auch bei höchstem Vertrauen in die persönlichen Englischkenntnisse – eine kundige Person für das „proof reading“ zu engagieren.

Die richtige Übersetzung für „Diplomarbeit“ ist übrigens schlicht *thesis*, allenfalls „diploma thesis“ oder „Master's thesis“, auf keinen Fall aber „diploma work“ oder gar „dissertation“. Für „Bachelorarbeit“ ist wohl „Bachelor thesis“ die passende Übersetzung.

Übrigens sollte für diesen Abschnitt die *Spracheinstellung* in LaTeX von Deutsch auf Englisch umgeschaltet werden, um die richtige Form der Silbentrennung zu erhalten, die richtigen Anführungszeichen müssen allerdings selbst gesetzt werden (s. dazu die Abschnitte **??** und **??**).

Kapitel 1

Einleitung

1.1 Überblick

In der modernen Softwareentwicklung geht es in erster Linie um das Umsetzen funktionaler sowie nicht funktionaler Anforderungen und um die Behebung von Fehlern welche bei der Umsetzung dieser Anforderungen häufig auftreten. Ein wichtiger Punkt dabei ist, den Code so zu gestalten, dass er nicht nur vom Programmierer, der ihn geschrieben hat, gelesen werden kann, sondern auch von anderen Programmieren und dies auch möglichst noch nach mehreren Monaten. Um dies zu erreichen, müssen gewisse Grundsätze angewendet werden. In den letzten Jahren hat sich zu diesem Thema eine Strömung ergeben welche sich als Clean Code Development (im folgenden nur noch als CCD abgekürzt) bezeichnet. Geprägt wurde diese Bewegung in erster Linie von Robert. C. Martin und seinem Bestseller Clean Coder.

1.2 Gliederung

Die Arbeit ist in drei größere Teile gegliedert. Der erste Teil beschäftigt sich in erster Linie mit der Erklärung des CCD Begriffes und des .....

Im zweiten Teil werden die einzelnen Regeln und Grundsätze anhand von Beispielen erläutert wie sie den Code verbessern bzw. welche Änderungen an Codestücken vorgenommen werden müssen, um diese Regeln und Grundsätze zu entsprechen.

Der dritte Teil beschäftigt sich mit der Analyse einiger Opensource Frameworks hinsichtlich der CCD Kriterien und deren Erfüllung. Es sollten dabei Codestücke gezeigt werden, welche die Kriterien erfüllen, sowie Codestücke die diese nicht erfüllen und dann abgeändert werden.

Kapitel 2

Einführung in CCD

2.1 Allgemeines zum CCD

Wie bereits in im Überblick erläutert, beschäftigt sich CCD in erster Linie mit Praktiken und Strategien, wie man Code so gestalten kann, dass er möglichst leicht zu Lesen, zu Warten und Anzupassen ist. Dabei sind diese Praktiken unabhängig von der Programmiersprache. Auf Grund der Tatsache, dass vor allem ältere Programmiersprachen gewisse Restriktionen wie zum Beispiel eine maximal Länge von Variablen aufweisen sind jedoch nicht alle Praktiken zur Gänze umsetzbar, jedoch kann sich trotzdem daran orientiert werden. Es ist weiterhin nicht relevant welches Programmierparadigma der Programmiersprache zu Grunde liegt. Die meisten Regeln gelten gleichermaßen für objektorientierte, funktionale und imperative Programmiersprachen. Einige Regeln sind auf Grund der unterschiedlichen Eigenschaften der einzelnen Paradigmen nicht zur Gänze umsetzbar, jedoch gibt es auch Abwandlungen der Regeln welche in den unterschiedlichen Paradigmen gelten.

2.2 Warum CCD?

Die Frage die sich im Zusammenhang mit CCD immer wieder ergibt ist: Warum CCD? Erst in den letzten Jahren hat sich CCD zu einem wirklich wichtigen Teil der Softwareentwicklung entwickelt. Bei einer Suche nach C̈lean Code Developmentërgeben sich 39.600.000 Ergebnisse (Stand November 2015). Auch bei Inseraten für Jobangebote ist Clean Code Development ein sehr wichtiger Punkt: Bei einer Suche nach dem selben Begriff auf stack overflow careers (http://careers.stackoverflow.com) ergeben sich hier 973 Treffer (Stand November 2015).

Diese Zahlen geben Aufschluss darüber, dass ein sehr großes Interesse an CCD besteht und dieses auch in sehr vielen Bereichen Anwendung findet.

Dies liegt in erster Linie an einer der

Kapitel 3

Grundlagen von CCD

In diesem Abschnitt sollten die Grundlagen für die unterschiedlichen Praktiken des CCD erläutert werden. Um den Rahmen dieser Arbeit nicht zu sprengen, werden hier nur die zentralsten Praktiken erläutert. Auf Grund der Tatsache, dass sich der dritte Teil dieser Arbeit in erster Linie mit dem Refactoring von Open Source Frameworks hinsichtlich der CCD Praktiken beschäftigt, werden die Beispiele in diesem Abschnitt kurz gehalten und sollten nur als kurze Einführung dienen.

3.1 Namensgebung

Eine der häufigsten Aufgaben eines Programmierers ist das vergeben von Namen. Variablen, Methoden, Klassen, Module, Bibliotheken, Programme. Alle diese Teile brauchen Namen und diese Namen sollten möglichst aussagekräftig sein. Eine Bibliothek die zum Loggen von Informationen wird, sollte im besten Fall das Wort Logging beinhalten, damit sich der Verwender dieser Bibliothek beim ersten Blick darauf schon eine Ahnung hat was sich darin befinden wird und er sich schon einen ersten Eindruck machen kann, was ihn in dieser Bibliothek erwartet.

Im CCD gibt es für die Namensgebung eine grundlegende Regel: Der Name sollte der Größe des Gültigkeitsbereich entsprechen. Wenn man z.B. eine Methode, die in einem globalen Kontext gültig ist, sollte einen dementsprechend deskriptiven Namen aufweisen. Zu beachten ist natürlich auch der Kontext in dem sich diese befinden, wofür die Methode verwendet werden sollte. Hier nun ein kleines Beispiel. In einem fiktiven Projekt gibt es folgende zentrale Klasse mit einer globalen Property.

```
1 public static class Globals {
2     public static string Information {get; set;}
3 }
```

Trotz der Kürze dieser Klasse ergibt sich bereits ein Problem. Aus dem Namen der Property kann man die Aufgabe dieser in keinster Weise ablesen. Es wird irgendeine Information gespeichert, doch welche Information genau wird gespeichert und wozu wird diese verwendet? Ein Blick auf die Verwendungen dieser Variable, führt zu der Erkenntnis, dass die Information die Version des Programmes darstellt. Der Programmierer hätte diese Verwirrung vermeiden können indem er die Namen anders gewählt hätte. Hier ein Beispiel zu einer besseren Variante:

```
1 public static class ApplicationInformation {
2     public static string Version {get; set;}
3 }
```

Durch eine Umbenennung der Klasse wird es auch um einiges klarer, auf welchen Teil des Programmes sich die Version bezieht. Man könnte jetzt noch den Datentyp Version implementieren, welcher die z.B. die Major-, Minor- und Patchnummer beinhaltet:

```
1 public static class ApplicationInformation {
2     public static Version Version {get; set;}
3 }
4
5 public class Version {
6   public int Major { get; set;}
7   public int Minor { get; set;}
8   public int Patch { get; set;}
9 }
```

Bei diesem letzten Beispiel kann man ohne viele Schritte erkennen, welche Aufgabe dieser Variable zu Teil wird. Durch die Erweiterung um einen eigenen Datentyp ist es jetzt auch leichter weitere Dinge zur Version hinzuzufügen. Angenommen eine zukünftige Anforderung für das Programm verlangt es, bei der Versionsnummer immer die Buildnummer mit anzugeben. Mit der letzten Variante ist dies kein Problem und es kann auch explizit darauf zugegriffen werden, ohne dass man wie bei der ersten Variante die Zeichenkette aufteilen muss.

3.2 Methoden

Eine weitere wichtige grundsätzliche Regel im CCD ist es Funktionen und Methoden möglichst klein zu halten. Es sollte darauf geachtet werden, dass diese dabei das sogenannte Single responsibility principle (kurz SRP) befolgen. Dieses Prinzip besagt, dass alle Aspekte des Softwareentwurfs so geplant werden sollten, sodass sie nur eine Aufgabe erfüllen. Wenn dieses Prinzip für Funktionen und Methoden befolgt wird, sollten diese automatisch sehr klein werden. Wenn eine Methode oder eine Funktion sehr lange ist, deutet dies meist auf eine Verletzung dieses Prinzips hin. Ein weiterer guter Indikator ist die Schwierigkeit mit der man eine Methode oder Funktion testen kann.

3.3 Klassen

Kapitel 4

Analyse von Open-Source Frameworks

In diesem Abschnitt erfolgt die Analyse der unterschiedlichen Opensource Frameworks hinsichtlich der CCD Kriterien es sollten dabei folgende Frameworks betrachtet werden:

- Log4net (.NET)
- Hibernate (Java)
- Git (C)

Es werden zur Analyse gewisse Code Abschnitte aus diesen Frameworks entnommen, die entweder besonders gut zeigen, wie ein Refactoring zur Erfüllung der CCD Kriterien zu lesbareren Code führen kann, oder Code beinhalten der bereits den Kriterien entspricht.

4.1 Log4net

4.1.1 Allgemeines

Log4net ist wohl eines der bekanntesten Logging Frameworks für .NET und wird als solches von zahlreichen Programmierern verwendet. Es ist eine Portierung des im Java Bereich bekannten Log4j Das Framwork an sich steht unter der Apache License 2.0 bereit. Log4net ist dabei ein Teil der Apache Logging Services und somit auch ein Teil der Apache Software Foundation.

4.1.2 Überflüssige Kommentare

Eines der wohl am meisten diskutierten Themen hinsichtlich CCD ist das entfernen von überflüssigen Kommentaren. Auch in Log4net lassen sich diese finden. Da dies ein sehr heikles Thema ist, werde ich mich hier speziell nur auf die Kommentare beziehen, die keinen wirklichen Mehrwert hinsichtlich automatisiert generierter Dokumentation liefern. Wenn wir uns den

Konstruktor aus der Klasse *ConfigurationChangedEventArgs* im Namespace *log4net.Repository*ansehen können wir sofort feststellen, dass die Kommentare im Prinzip redundant sind.

```
1         /// <summary>
2         ///
3         /// </summary>
4         /// <param name="configurationMessages»</param>
5         public ConfigurationChangedEventArgs(ICollection
      configurationMessages)
6         {
7             this.configurationMessages = configurationMessages;
8         }
```

Er bringt für den Programmierer keinen wirklichen Mehrwert. Was genau in der übergebenen *Collection* übertragen wird, wird auch durch den redundanten Kommentar nicht näher erläutert. Da der <summary> Abschnitt sowieso leer ist, könnte man diesen getrost entfernen, da er nur den Quelltext aufbläht. Bei dem Konstruktor hätte man sich auch durch eine bessere Namensgebung für den Parameter behelfen können. Ein wohl viel besserer Name für die *configurationMessages* wäre . Mit diesem Namen dürfte bei einer Änderung, oder auch bei einer Verwendung dieser Klasse sofort klar sein, welchen Nutzen dieser Parameter hat.

4.1.3 Schlecht gewählter Klassenname

4.1.4 Fazit

Im großen und ganzen lässt sich sagen, dass bei der Entwicklung von Log4net sehr darauf geachtet wurde, die Codebasis so zu gestalten, dass man sich sehr leicht einarbeiten kann. Die Bibliothek ist sehr logisch aufgebaut und die Klassen bzw. Namespacenamen sind so gewählt, dass man sich gut zurecht findet. Es sollte also für keinen Programmierer ein gröberes Problem sein, sich in diese Bibliothek einzuarbeiten. Einige Dinge sind auf Grund des Alters von Log4net mit den modernen IDEs nicht mehr nötig, wie z.B. die Präfixe für Variablen, jedoch wird dies meist über Coding Conventions festgelegt und sollte daher auch so im ganzen Projekt durchgezogen werden.

4.2 Hibernate

4.2.1 Allgemeines

Die Bibliothek Hibernate ist einer der bekanntesten OR-Mapper für Java. Mit ihm ist es möglich ein relationales Datenbankmodell auf ein objektorientiertes Datenmodell zu mappen. ———————

4.2.2 Fazit

4.3 Git

Das von Linus Torvalds und anderen Entwicklern entwickelte dezentrale Versionsverwaltungsprogramm Git ist ein in C programmiertes Tool zur

Kapitel 5

Schlussbemerkungen[1]

An dieser Stelle sollte eine Zusammenfassung der Abschlussarbeit stehen, in der auch auf den Entstehungsprozess, persönliche Erfahrungen, Probleme bei der Durchführung, Verbesserungsmöglichkeiten, mögliche Erweiterungen usw. eingegangen werden kann. War das Thema richtig gewählt, was wurde konkret erreicht, welche Punkte blieben offen und wie könnte von hier aus weitergearbeitet werden?

5.1 Lesen und lesen lassen

Wenn die Arbeit fertig ist, sollten Sie diese zunächst selbst nochmals vollständig und sorgfältig durchlesen, auch wenn man vielleicht das mühsam entstandene Produkt längst nicht mehr sehen möchte. Zusätzlich ist sehr zu empfehlen, auch einer weiteren Person diese Arbeit anzutun – man wird erstaunt sein, wie viele Fehler man selbst überlesen hat.

5.2 Checkliste

Abschließend noch eine kurze Liste der wichtigsten Punkte, an denen erfahrungsgemäß die häufigsten Fehler auftreten (Tab. 5.1).

[1] Diese Anmerkung dient nur dazu, die (in seltenen Fällen sinnvolle) Verwendung von Fußnoten bei Überschriften zu demonstrieren.

Tabelle 5.1: Checkliste. Diese Punkte bilden auch die Grundlage der routinemäßigen Formbegutachtung in Hagenberg.

- ☐ **Titelseite:** Länge des Titels (Zeilenumbrüche), Name, Studiengang, Datum.
- ☐ **Erklärung:** vollständig Unterschrift.
- ☐ **Inhaltsverzeichnis:** balancierte Struktur, Tiefe, Länge der Überschriften.
- ☐ **Kurzfassung/Abstract:** präzise Zusammenfassung, passende Länge, gleiche Inhalte und Struktur.
- ☐ **Überschriften:** Länge, Stil, Aussagekraft.
- ☐ **Typographie:** sauberes Schriftbild, keine „manuellen“ Abstände zwischen Absätzen oder Einrückungen, keine überlangen Zeilen, Hervorhebungen, Schriftgröße, Platzierung von Fußnoten.
- ☐ **Interpunktion:** Binde- und Gedankenstriche richtig gesetzt, Abstände nach Punkten (vor allem nach Abkürzungen).
- ☐ **Abbildungen:** Qualität der Grafiken und Bilder, Schriftgröße und -typ in Abbildungen, Platzierung von Abbildungen und Tabellen, Captions. Sind *alle* Abbildungen (und Tabellen) im Text referenziert?
- ☐ **Gleichungen/Formeln:** mathem. Elemente auch im Fließtext richtig gesetzt, explizite Gleichungen richtig verwendet, Verwendung von mathem. Symbolen.
- ☐ **Quellenangaben:** Zitate richtig referenziert, Seiten- oder Kapitelangaben.
- ☐ **Literaturverzeichnis:** mehrfach zitierte Quellen nur einmal angeführt, Art der Publikation muss in jedem Fall klar sein, konsistente Einträge, Online-Quellen (URLs) sauber angeführt.
- ☐ **Sonstiges:** ungültige Querverweise (**??**), Anhang, Papiergröße der PDF-Datei (A4 = 8.27×11.69 Zoll), Druckgröße und -qualität.

Quellenverzeichnis