Software Engineering Projekt

eCourse

Codierungsrichtlinien und Konfigurationsmanagement

des Studiengangs Informatik

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

3819525, 4679471, 5247876

6499003, 6504782, 7182188

7750470, 8538336, 9654562

Studienjahrgang, Kurs	2019, INF19IN
Bearbeitungszeitraum	18.03.2021 bis 20.05.2021

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Motivation

Dieses Dokument dient als grobe Richtlinie um einen Überblick über die Codierungsrichtlinien und das Konfigurationsmanagement im Projekt eCourse zu erhalten.
Im Dokument werden die wichtigsten und erwähnenswerten Punkte zusammengefasst. Für weiterführende Informationen wird an gegebener Stelle auf die Originalquellen verwiesen. Diese hier zusammengefassten Informationen dienen der Konsistenz des entstehenden Programmcodes.
Ebenso soll dadurch eine einheitliche Versionsverwaltung garantiert werden.

2 Codierungsrichtlinien

In diesem Abschnitt sind die wichtigsten Codierungsrichtlinien, die im Projekt gelten, zusammengefasst. Es handelt sich dabei um eine Auswahl der wichtigsten und am häufigsten vorkommenden Situationen.

Das Projektteam hat sich für jede Programmiersprache bereits gültige Codierungsrichtlinien gesucht. Diese sollen im Projekt umgesetzt werden.

Damit für die am häufigsten auftretenden Situationen nicht immer der geltende Standard gelesen werden muss, sind an dieser Stelle die wichtigsten Informationen zusammengefasst.

Im jeweiligen Unterkapitel der Programmiersprache wird auch nochmals explizit auf den verwendeten Standard verwiesen.

Wie auch in allen hier verwendeten Standards wird Wert auf die Lesbarkeit des Codes gelegt.
Wirkt also eine Regel an einer bestimmten Stelle der Lesbarkeit des Codes entgegen, so wird immer die Lesbarkeit des Codes bevorzugt gegenüber der Befolgung der Regel.
Um ein Beispiel für eine solche Situation zu nennen: Dateinamen sollen nach Richtlinie immer im snake_case geschrieben sein. Allerdings repräsentiert eine Datei einen bestimmten Eigennamen. Hier wird dann bevorzugt, den Eigennamen korrekt zu verwenden, als ihn an die Richtlinie anzupassen.

2.1 Allgemein

Vorab einige Allgemeine Richtlinien, auf die sich das Projektteam geeinigt hat.

- Dateinamen werden im snake_case geschrieben (Bsp.: das_ist_snake_case)
- Jede Datei erhält einen Dateiheader, in dem die Inhalte der Datei kurz zusammengefasst werden
- Der Programmcode wird auf Englisch geschrieben. Von dieser Regel sind alle Inhalte die für den Nutzer der Software sichtbar sind ausgenommen.
- Dateien werden mit UTF-8 codiert

2.2 Python

Für die Programmiersprache Python wird die Codierungsrichtlinie PEP 8 verwendet. Eine ausführliche Version der Richtlinie findet sich hier.

Einrückung

Für die Einrückung werden 4 Leerzeichen verwendet, keine Tabs.
Weitergeführte Zeilen werden ebenfalls durch Einrückung oder durch Klammern gekennzeichnet

```
foo = long_function_name(var_one, var_two,
              var_three, var_four)

def long_function_name(
    var_one, var_two, var_three,
    var_four):
  print(var_one)
```

Listings 2.1: Weitergeführte Zeilen gekennzeichnet durch Einrückungen

Werden weitergeführte Zeilen durch Klammern gekennzeichnet, kann die schließende Klammer entweder unter dem ersten Zeichen der letzten Zeile stehen oder unter dem ersten Zeichen der Zeile, in der das Konstrukt begonnen wurde.

```
result = some_function_that_takes_arguments(
  'a', 'b', 'c',
  'd', 'e', 'f'
  )

my_list = [
  1, 2, 3,
  4, 5, 6,
]
```

Listings 2.2: Weitergeführte Zeilen gekennzeichnet durch Klammern

Maximale Zeilenlänge

Die maximale Zeilenlänge für Programmcode beträgt 79 Zeichen.
Davon ausgenommen sind Kommentare, für diese wird die maximale Zeilenlänge auf 72 Zeichen begrenzt.

Zeilenumbrüche bei binären Operatoren

Zeilenumbrüche werden vor den binären Operatoren ausgeführt.

```
income = (gross_wages
      + taxable_interest
      + (dividends - quaified_dividends)
      - ira_deduction
      - student_loan_interest)
```

Listings 2.3: Zeilenumbrüche bei binären Operatoren

Leerzeilen

Funktionen und Klassendefinitionen werden durch zwei Leerzeilen eingerahmt.
Methoden in Klassen werden durch eine Leerzeile umschlossen.
Durch eine Leerzeile können außerdem Funktionen gruppiert werden und logische Abschnitte in Funktionen gekennzeichnet werden.

Imports

Jeder import-Befehl wird in eine separate Zeile geschrieben.
import-Befehle stehen in den Dateien immer auf der höchsten Ebene, direkt unter Modulkommentaren und Docstrings.

import-Befehle sind gruppiert nach folgender Reihenfolge:

1. Standard-Bibliotheken
2. verwandte Bibliotheken
3. lokale Applikationen und Bibliotheken

Jede dieser Gruppen wird durch eine Leerzeile von den anderen separiert.
Es werden wenn immer mögliche absolute imports verwendet.

```
# absoluter Import
import mypkg.sibling
from mypkg import sibling
from mypkg.sibling import example

# relativer Import
from . import sibling
from .sibling import example
```

Listings 2.4: absolute imports

Anführungszeichen für Strings

Für Strings werden einfach Anführungszeichen verwendet.
Innerhalb des Strings werden dann doppelte Anführungszeichen verwendet.
Innerhalb von Strings mit drei Anführungszeichen werden doppelte Anführungszeichen verwendet.

Leerzeichen

Nachlaufende Leerzeichen sollten vermieden werden.
Operatoren werden mit einem Leerzeichen auf jeder Seite umschlossen.
In verschiedenen Situationen sollen zusätzliche Leerzeichen vermieden werden:

```
# innerhalb von Klammern
  # Falsch
    spam( ham[ 1 ], { eggs: 2 } )
  # Richtig
    spam(ham[1], {eggs:2})

# zwischen nachlaufenden Kommas und der folgenden schliessenden
    Klammer
  # Falsch
    foo = (0, )
  # Richtig
    foo = (0,)

# direkt vor einem Komma, Semikolon, Doppelpunkt
  # Falsch
    if x == 4 : print x , y ; x , y = y , x
  # Richtig
    if x == 4: print x, y; x, y = y, x

# vor oeffnenden Klammern der Argumentenliste eines Funktionsaufrufs
  # Falsch
    spam (1)
  # Richtig
    spam(1)

# vor oeffnenden Klammern, die eine Indizierung oder eine Zuordnung
    starten
  # Falsch
    dct ['key'] = lst [index]
  # Richtig
    dct['key'] = lst[index]

# mehr als ein Leerzeichen um einen Operator
```

```
32    # Falsch
33      x       = 1
34      y       = 2
35      long_variable = 3
36    # Richtig
37      x = 1
38      y = 2
39      long_variable = 3
40
41  # bei Zuweisungen von default-Parametern oder bei Schluesselwort-
        Argumenten
42    # Falsch
43      def complex(real, img = 0.0):
44        return magic(r = real, i = img)
45    # Richtig
46      def complex(real, img=0.0):
47        return magic(r=real, i=img)
```

Listings 2.5: Situationen in denen zusätzliche Leerzeichen vermieden werden sollten

Kommentare

Kommentare müssen auf dem gleichen Stand sein, wie der Code, den sie beschreiben.
Kommentare sollten ganze Sätze sein. Das bedeutet, sie enden mit einem Punkt und starten mit einem Großbuchstaben, außer der erste Buchstabe ist ein Bezeichner, der mit einem Kleinbuchstaben anfängt.
Kommentare werden in englischer Sprache verfasst.

Blockkommentare gehören zu dem Code (bzw. einem Teil davon) der danach folgt. Sie sind genauso weit eingerückt wie der Code den sie beschreiben. Sie starten mit einem # und einem darauf folgenden Leerzeichen.

Inline Kommentare stehen in der selben Zeile wie ein Code-Statement. Sie sollten sparsam eingesetzt werden und werden weggelassen, wenn sie das offensichtliche Erklären. Sie sind mindestens ein Leerzeichen vom Code den sie beschreiben entfernt und beginnen mit einem # und einem darauf folgenden Leerzeichen.

Documentation Strings werden für alle sichtbaren Module, Funktionen, Klassen und Methoden erstellt. Sie sind nicht nötig für nicht-sichtbare Methoden. Diese sollten mit Blockkommentaren versehen werden. Documentation Strings beginnen mit drei Anführungszeichen und enden mit drei Anführungszeichen. Die schließenden Anführungszeichen stehen dabei in einer eigenen Zeile. Diese Regel gilt nicht für einzeilige Kommentare, bei diesen stehen die schließenden Anführungszeichen in der gleichen Zeile.

Namenskonventionen

Neben der Beachtung der nachfolgenden Regeln sollte als erster Buchstabe in den Bezeichnern auf folgende Buchstaben verzichtet werden:

l (kleines L)
O (großes o)
I (großes I)

Module haben kurze Namen und sind im snake_case geschrieben.
Klassen haben Bezeichner die im CamelCase geschrieben sind.
Exceptions sind Klassen und sollten „Error“ als Suffix erhalten.
Funktionen haben Bezeichner die im snake_case geschrieben sind.
Methoden und Instanzvariablen werden wie Funktionen behandelt.
Konstanten werden für ein ganzes Modul definiert und sind in CONSTANT_CASE geschrieben.

Vererbung

Ist eine Klasse auf Vererbung ausgelegt, sollten alle Methoden und Instanzvaraiblen nichtöffentlich deklariert werden.
Öffentliche Attribute sollten keine führenden Unterstriche haben.
Sollten Attributsnamen mit reservierten Schlüsselworten kollidieren, soll ihnen ein nachlaufender Unterstrich angehängt werden.

weitere Empfehlungen

Vergleiche zu Singeltons (wie None) sollten mit is (bzw. is not) durchgeführt werden.
Bei Return-Statements sollte auf Konsistenz geachtet werden. Entweder geben alle Funktionsteile etwas zurück oder keiner. Soll ein Statement keinen Wert zurückgeben, sollte None als Rückgabewert verwendet werden.

```
1 def foo(x):
2   if x >= 0:
3     return math.sqrt(x)
4   else:
5     return None
```

Listings 2.6: konsistente return-Statements

2.3 HTML und CSS

Die Codierungsrichtlinien von HTML und CSS die im Folgenden zusammengefasst sind, beziehen sich auf die Codierungsrichtlinien für HTML und CSS von Google. Die Gesamtheit der Codierungsrichtlinien findet sich unter diesem Link.

2.3.1 Allgemein

Protokolle
Es sollte immer das https-Protokoll verwendet werden. http ist nur in Ausnahmefällen erlaubt, wenn das Dokument nicht mit https erreicht werden kann.
Auf die strikte Beachtung dieser Regel wird in diesem Projekt verzichtet, da das Projekt auf die Entwicklung der Anwendung abzielt und nicht auf die Entwicklung der Infrastruktur.

Einrückung
Jedes Level wird mit 2 Leerzeichen eingerückt. Tabs sollten nicht verwendet werden.

Groß- und Kleinschreibung
Der Programmcode sollte mit Kleinbuchstaben geschrieben werden. Großbuchstaben werden nur in Strings verwendet, die im späteren Dokument ausgegeben werden.

Nachlaufende Leerzeichen
Nachlaufende Leerzeichen sollten immer vermieden werden.

Dateicodierung
Dateien werden mit UTF-8 codiert. Die Codierung sollte in HTML Stylesheets und HTML-Dokumenten mit einem Metatag angegeben werden.

```
1 <meta charset="utf-8">
```

Listings 2.7: Metatag für die UTF-8 Codierung

Kommentare
Der Code sollte an den Stellen erklärt werden, an denen es nötig ist.

Kommentare sollten dann auftauchen, wenn sich der Leser folgende Fragen stellen könnte:

Was deckt der Code ab

Welchen Zweck hat der Code

Warum wurde diese Lösung verwendet

Kommentare werden nach folgenden Richtlinien in den Code integriert:

```
{# Einzeiliger Kommentar #}

{% comment %}
  Mehrzeiliger
  Kommentar
{% endcomment %}
```

Listings 2.8: Kommentare

Handlungspunkte

Kann an einer Stelle der Code noch nicht vollständig geschrieben werden bzw. ist man an einer gewissen Stelle auf einen anderen Entwickler angewiesen, sollte die Stelle mit einem Todo markiert werden.
In diesem Todo sollte vermerkt sein, was noch zu tun ist und eventuell falls bekannt, wer es tun muss.

```
{# TODO(john):revisit centering #}
<center>Test</center>
```

Listings 2.9: Kennzeichnung eines Handlungspunktes

2.3.2 HTML

Dokumententyp

HTML-Dokuemente werden mit HTML 5 geschrieben.
Dies sollte durch den DOCTYPE im Dokumentekopf angezeigt werden.

```
<!DOCTYPE html>
```

Listings 2.10: Kennzeichnung des Dokumententyps

Alternativer Text für Medien

Alle eingefügten Medien werden mit einem alternativen Text versehen. Von dieser Regel sind allerdings alle Bilder ausgenommen, die nur einen rein dekorativen Effekt erfüllen.

```
<!-- alternative text -->
<img src="spreadsheet.png" alt="Spreadsheet␣screenshot.">

<!-- no alternative text if it's only for decorative means -->
<img src="decorative.png" alt="">
```

Listings 2.11: Alternativtext

Trennung von Inhalten

Strukturelle, gestalterische und verhaltensbasierte Inhalte sollten strikt voneinander getrennt sein.

Generelle Formatierung

Jeder Block, jede Liste und jedes Tabellenelement erhält eine eigene neue Zeile.
Alle Kindelemente werden im Vergleich zu ihren Elternelementen um eine Einheit eingerückt.

```
<table>
  <thread>
    <tr>
      <th scope="col">Income
      <th scope="col">Taxes
</table>
```

Listings 2.12: Generelle Formatierung von HTML Dokumenten

Zeilenumbrüche

Lange Zeilen sollte umgebrochen werden.
Die weitergeführte Zeile sollte um mindestens 4 Leerzeichen weiter eingerückt sein, als die Zeile die sie weiterführt.

```
<md-progress-circular md-mode="indeterminate" class="md-accent"
  ng-show="ctrl.loading" md-diameters="35">
</md-progress-circular>
```

Listings 2.13: Zeilenumbrüche

Anführungszeichen

Es sollten immer doppelte Anführungszeichen (“”) verwendet werden.

2.3.3 CSS

ID- und Klassennamen

Die Namen sollten immer den Sinn des Elements widerspiegeln. Dadurch sollten entweder bedeutende Namen oder generische Namen verwendet werden.
Außerdem sollten die Namen so kurz wie möglich und so lang als nötig sein.
Die Namen sollten zur besseren Lesbarkeit durch Bindestriche getrennt sein.
Es wird also dash-case verwendet.

```
#video-id {}
.ads-sample {}
```

Listings 2.14: ID- und Klassennamen

Typselektoren

Qualifizierende ID- und Klassennamen sollten bei Typselektoren vermieden werden.
Ebenfalls sollte darauf verzichtet werden Elementbezeichner mit IDs oder Klassen zu vermischen.

```
/* Not recommended */
  u1#example {}
  div.error{}#

/* Recommended */
  #example {}
  .error {}
```

Listings 2.15: Kennzeichnung von Typselektoren

Shorthand Properties

Falls möglich sollte für alle Eigenschaften bei denen es möglich ist die Kurzschreibweise verwendet werden.

```
/* Not recommended */
  font-family: palatino, georgia, serif;
  dont-size: 100%
  line-height:1.6;

/* Recommended */
  font: 100%/1.6 palatino, georgia, serif;
```

Listings 2.16: Shorthand Properties

Verwendung von Nullen

Nach einer Null als Größenangabe darf keine Einheit stehen, außer sie ist ausdrücklich verlangt.
Ebenfalls sollten keine führenden Nullen in Werten verwendet werden.

```
flex: 0px; /* Here a unit is required */
margin: 0; /* The unit is not required */

font-size: .8em; /* No leading zero */
```

Listings 2.17: Verwendung von Nullen in Größenangaben

Hexadezimale Notation

Es sollten wann immer möglich alle hexadezimalen Angaben mit drei Zeichen anstatt mit den üblichen 6 Zeichen angegeben werden.

```
color: #ebc;
```

Listings 2.18: hexadezimale Angaben

Deklarationsreihenfolge

Alle Deklarationen sollten in alphabetischer Reihenfolge sortiert sein.
Weitere Suffixe, die auf eine Deklaration folgen, sollten ebenfalls in alphabetischer Reihenfolge an die Deklaration selbst folgen.

```
background: fuchsia;
border: 1px solid;
-moz-border-radius: 4px;
-webkit-border-radius: 4px;
border-radius: 4px;
color: black;
text-align: center;
text-indent: 2em;
```

Listings 2.19: Sortierung von Deklarationen

Einrückung von Blockinhalten

Blockinhalte werden um eine Einheit eingerückt.

```
@media screen, projection {
  html {
    background : #fff;
    color: #444
```

```
5     }
6 }
```

Listings 2.20: Einrückung von Blockinhalten

Anführungszeichen

Im Gegensatz zu HTML werden bei CSS einfache Anführungszeichen (' ') verwendet.
In URI-Werten werden keine Anführungszeichen verwendet.

Abschnittskommentare

Abschnitte werden durch Kommentare gruppiert.

```
1  /* Header */
2
3  #adw-header {}
4
5  /* Footer */
6
7  #adw-footer {}
8
9  /* Gallery */
10
11 .adw-gallery {}
```

Listings 2.21: Einteilung einer CSS-Datei durch Abschnittskommentare

weitere Gestaltungsrichtlinien

Jede Zeile wird mit einem Semikolon beendet.
Zwischen einer Eigenschaft und dem dazugehörigen Wert wird nach dem Doppelpunkt ein Leerzeichen eingefügt.
Zwischen dem letzten Selektor und der darauf folgenden öffnenden Klammer des Deklarationsblock steht ein Leerzeichen. Die öffnende Klammer des Deklarationsblocks steht dabei in der selben Zeile wie die Selektoren.
Jeder Selektor bzw. jede Deklaration und von dem nächsten seiner Art durch eine neue Zeile getrennt.
Verschiedene Regeln wrden durch Leerzeilen voneinander getrennt

```
1  h2,
2  h3 {
3    font-weight: bold;
4    line-height : 1.2;
5  }
6
```

```
7 html {
8   background: #fff;
9 }
```

Listings 2.22: Beispiel für die weiteren Gestaltungsrichtlinien

3 Konfigurationsmanagement

Alle Dateien werden über den Versionsverwaltungsdienst GitHub versioniert und unter den Projektteilnehmern ausgetauscht.
Das Repository befindet sich hier.

Zur Gestaltung des Repositorys sind folgende Regeln zu beachten: Jedes Themengebiet (Frontend, Backend und Dokumentation) hat seinen eigenen Branch. Der Zugriff auf diesen Branch kann nochmals intern eingeschränkt werden. Dies wird von den einzelnen Arbeitsteams aufgabenbezogen festgelegt.
Ein Commit auf den Master-Branch kann erst ausgeführt werden, wenn alle Punkte der Definition Of Done (siehe 3.1) abgearbeitet sind.
Dies beinhaltet, dass der Code getestet ist, den Codierungsrichtlinien in Kapitel 2 entspricht und von mindestens einer anderen Person des Developmentteams abgenommen wurde.

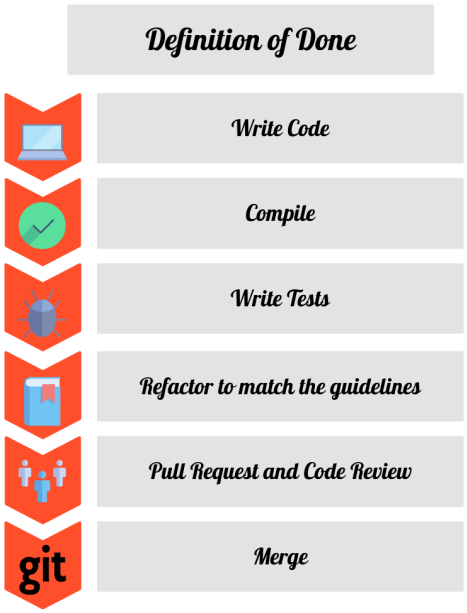

Abbildung 3.1: Definition of Done

Jeder Commit wird mit einer Zusammenfassung und einer tiefer gehenden Beschreibung versehen.
Die Zusammenfassung enthält dabei eine kurze Beschreibung des geänderten Merkmals. Die Beschreibung geht dann genauer auf das geänderte Merkmal ein.

Des Weiteren befinden sich im Repository noch ein Kanban-Board, ein Wiki und eine Issue-Liste.
Das Kanban-Board wird zur Organisation des aktuellen Sprints eingesetzt. Dafür werden alle Tasks des aktuellen Sprints gesammelt. Diese werden dann an die Arbeitsgruppen verteilt und von diesen Bearbeitet. Ist ein Task aus Sicht der Arbeitsgruppe fertiggestellt, verschieben sie diesen in die Review-Leiste. Dies signalisiert den anderen Gruppen, dass die Arbeit des Teams überprüft werden kann. Wird der Task als erledigt angesehen, kann er in die Done-Leiste geschoben werden.

Im Wiki können alle Informationen die für alle Teams wichtig sind gesammelt werden. Dabei handelt es sich vor allem um Informationen, die schnell und immer verfügbar sein sollten.

In der Issue-Liste können dann neben den aktuell laufenden Tasks auch während der Entwicklung auftauchende Bugs aufgenommen werden.