Programmierkonzepte in der Physikalischen Chemie

15. Dokumentation im Code

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Dr. Till Biskup

Institut für Physikalische Chemie
Albert-Ludwigs-Universität Freiburg
Wintersemester 2018/19

Zentrale Aspekte

- Dokumentation im Code sollte auf das notwendige Minimum beschränkt werden.
- Veraltete Dokumentation schadet mehr als sie nützt.
- Formulierungen sollten so knapp und präzise wie möglich, die Formatierung konsistent und übersichtlich sein.
- Unklarer Code sollte nicht dokumentiert, sondern umgeschrieben werden.
- Dokumentationswerkzeuge erleichtern die Arbeit, setzen aber konsistente Formatierung voraus.

Dokumentation im Code

Zur Einführung

UNI FREIBURG

> *Real Programmers don't comment their code.*
> *It was hard to write, it should be hard to read.*
>
> – Ed Post

> *Real Programmers don't need comments—*
> *the code is obvious.*
>
> – Ed Post

Listing 1: linux-2.2.16/fs/buffer.c

```
1 /*
2  * After several hours of tedious analysis, the following
3  * hash function won.   Do not mess with it... -DaveM
4  */
```

Ed Post: Real Programmers Don't Use Pascal, Datamation 29, 263–265, 1983

Übersicht

UNI FREIBURG

Arten von Dokumentation im Code

Argumente für und gegen Kommentare

Regeln für Kommentare

Werkzeuge zur automatischen Erzeugung

Dokumentation im Code

Dokumentation direkt im Quellcode in Form von Kommentaren, die nicht ausgeführt (und vom Compiler/Interpreter ignoriert) wird

- Formatierung abhängig von der Programmiersprache
  - meist durch spezielles Zeichen vor dem Kommentar
  - mitunter unterschiedliche Zeichen für Beginn und Ende
- Unterscheidung von Zeilen- und Blockkommentaren
  - mitunter durch unterschiedliche Zeichen realisiert
  - dienen unterschiedlichen Einsatzgebieten

Dokumentation im Code

Unterscheidung von Zeilen- und Blockkommentaren

UNI FREIBURG

Listing 2: Beispiele für Block- und Zeilenkommentare in C

```c
/*
 * Blockkommentar in C
 */

// Zeilenkommentar in C
```

Listing 3: Beispiele für Block- und Zeilenkommentare in Python

```python
"""
Blockkommentar in Python
"""

# Zeilenkommentar in Python
```

- Block- und Zeilenkommentare haben unterschiedliche Einsatzgebiete.

Kommentare mit Daseinsberechtigung

Sinnvolle Einsatzmöglichkeiten für Kommentare im Code

UNI FREIBURG

- Kommentarköpfe
  - am Anfang von Klassen- und Funktionsdefinitionen
  - dienen u.a. der Schnittstellenbeschreibung
  - Hilfestellung zur Benutzung der Klasse/Funktion
  - werden häufig vom Hilfesystem automatisch verwendet
- Copyright-Hinweise
  - Autor und Jahr
  - ggf. Name der Lizenz
  - zusätzlich oft Datum der letzten Änderung
- (meist) einzeilige Kommentare im Code
  - Erläuterung einer Codezeile oder eines Codeblocks

☛ Der Kontext entscheidet, wie sinnvoll ein Kommentar ist.

Kommentare ohne Daseinsberechtigung

Was unter allen Umständen vermieden werden sollte

UNI FREIBURG

- Auskommentieren von Codeteilen
  - kurzzeitig während der aktiven Entwicklung erlaubt
  - sollte *nie* in die Versionsverwaltung wandern
  - Archivierung durch Versionsverwaltung obsolet geworden
- Änderungshistorie
  - vor allem in alten Programmen zu finden
  - komplette Historie der Änderungen mit Kurzkommentar
  - durch Versionsverwaltung obsolet geworden
  - wenn gewünscht, dann in externer Datei („`CHANGELOG`“)
- komplette Lizenztexte
  - meist viel zu lang
  - gehören in eigene Datei („`LICENSE`“, ggf. beim Quellcode)
  - im Code nur Verweis auf die Lizenz

Argumente für und gegen Kommentare

Warum man kommentieren sollte – und warum nicht

UNI FREIBURG

> *Nothing can be quite so helpful as a well-placed comment. Nothing can clutter up a module more than frivolous dogmatic comments.*
> *Nothing can be quite so damaging as an old crufty comment that propagates lies and misinformation.*
>
> – Robert C. Martin

drei Argumente

- Kommentare können extrem hilfreich sein.
- Kommentare können Code unnötig aufblähen.
- Kommentare können lügen.

Robert C. Martin: Clean Code, Prentice Hall, Upper Saddle River, NJ, 2008, S. 53

Kommentare können hilfreich sein

Beispiele für hilfreiche Kommentare

UNI FREIBURG

- Kommentarkopf von Funktionen und Klassen
  - kurze Beschreibung: warum? wie zu nutzen?
  - sollte nicht lediglich die Namen wiederholen
  - oft vom integrierten Hilfesystem genutzt
  - für alle vom Nutzer aufrufbaren Funktionen/Klassen
- Aspekte, die sich nicht in Code ausdrücken lassen
  - Code beschreibt das „Was“, selten das „Warum“.
  - Leser kennen die Gedanken des Autors nicht (immer).
- Begründung für eine bestimmte Implementierung
  - bewahrt andere vor vorschneller „Optimierung“

☛ Erfahrung und Fähigkeiten des Programmierers spielen ebenfalls eine Rolle.

Kommentare können Code aufblähen

Dokumentation ist kein Selbstzweck

UNI FREIBURG

- Code und Kommentare sollten im Verhältnis stehen.
  - Code hat eine wesentlich höhere Informationsdichte.
  - Code ist exakt – ein Kommentar in der Regel nicht.
  - Kommentare ergänzen den Code, nicht umgekehrt.
  - Zu viele Kommentare schränken die Lesbarkeit ein.
- Kommentare sollten einen echten Mehrwert haben.
  - Nur das kommentieren, was nicht im Code steht.
  - Immer überlegen: Warum steht es nicht im Code?
- Code sollte für sich sprechen.
  - Das Ziel ist selbstdokumentierender Code.

☛ Typisches Problem unerfahrener Programmierer:
Erst wird gar nicht, dann viel zu viel kommentiert.

Kommentare können lügen

Die Folgen können mitunter dramatisch sein.

UNI FREIBURG

- Gründe für inkorrekte Kommentare
  - Code ändert sich.
  - Kommentare werden nicht automatisch überprüft.
  - Programmierer sind Menschen:
    Synchronität von Code und Kommentaren ist illusorisch.
- Folgen – wenn der Fehler erkannt wird
  - Verwirrung – Was wollte der Autor?
  - Verunsicherung – Kann man dem Code vertrauen?
- Folgen – wenn der Fehler unentdeckt bleibt
  - falsche Verwendung des Codes
  - unerklärliches Verhalten des Programms

☛ Vertrauensverlust durch zu viele falsche Kommentare

Kommentare sind immer nur eine Notlösung

Das Ziel ist selbstdokumentierender Code.

UNI FREIBURG

> *The proper use of comments is to compensate for our failure to express ourself in code. [...] Comments are always failures.*
>
> – Robert C. Martin

- Kommentare sind viel zu oft falsch.
  - Synchronität zwischen Code und Kommentaren ist eine unerreichbare Illusion.
- Kommentare lenken vom Code ab.
  - Widersprüche zwischen Kommentaren und Code sind im besten Fall verstörend, im schlimmsten Fall irreführend.
- Die Wahrheit liegt im Code.
  - Kommentare werden nicht automatisch überprüft.

Robert C. Martin: Clean Code, Prentice Hall, Upper Saddle River, NJ, 2008, S. 54

Allgemeine Regeln für Kommentare

Eine erste Übersicht

UNI FREIBURG

- nicht das Offensichtliche kommentieren
  - nicht den Code wiederholen
  - gilt auch für Schnittstellen von Funktionen
- Funktionen und globale Daten kommentieren
  - wenn der Kontext nicht aus dem Namen klar wird
- schlechten Code nicht kommentieren – neu schreiben!
  - Kommentare kompensieren nicht mangelnde Qualität.
- dem Code nicht widersprechen
  - auf Aktualität und Korrektheit der Kommentare achten
- verdeutlichen, nicht verwirren
  - Code ist präzise, Kommentare sollten es ebenso sein.

Kernighan, Pike: The Practice of Programming, Addison Wesley, Boston 1999, Kapitel 1

Formulierungen: knapp und präzise

Auch der Leser hat nicht unbegrenzt Zeit.

UNI FREIBURG

- Code wird viel häufiger gelesen als geschrieben.
  - Daseinsberechtigung nur für relevante Kommentare
  - Code hat eine hohe Informationsdichte und ist präzise.
  - Kommentare sollten ähnlich präzise sein wie Code.
- Akt der Höflichkeit gegenüber dem Leser
  - Code zu lesen erfordert Konzentration.
  - Konzentration sollte nicht durch unwichtige Informationen oder unnötig lange Kommentare verringert werden
- Wertschätzung: Gute Kommentare kosten Zeit.
  - ähnlich wie die Benennung von Variablen, Funktionen etc.
  - Der erste Versuch ist selten gut genug.
  - Zeit für gute Kommentare zahlt sich vielfach aus.

☛ Sprachbeherrschung ist eine notwendige Voraussetzung.

Zur Sprache von Kommentaren

Möglichst auf Englisch

UNI FREIBURG

- Idealfall: geschliffenes Englisch
  - erfordert sehr gute Sprachbeherrschung
  - präzise und einfache Formulierungen
- Normalfall: einfaches Englisch
  - Die Audienz von Code ist oft international.
  - auch von Nicht-Muttersprachlern erreichbar
- Notfall: geschliffene Muttersprache
  - sehr viel besser als schlechtes Englisch
  - Es gibt Übersetzer...
- unbedingt vermeiden: schlechte sprachliche Qualität
  - gilt für Fremd- und Muttersprache
  - Zeichen für mangelnde Professionalität

Formatierung von Kommentarköpfen

Konsistent und entsprechend der Konventionen

UNI FREIBURG

- Erster Satz beschreibt die Funktion
  - *nicht* nur die Signatur (wörtlich) wiederholen
  - hilfreich für den korrekten Zuschnitt der Funktion
- Konsistenz
  - Voraussetzung für den Einsatz von Werkzeugen zur automatischen Erzeugung von Dokumentation
  - Hilfestellung durch Vorlagen im Editor
- sprachspezifische Konventionen beachten
  - gilt für Formatierung und Inhalte
  - Beispiel: PEP 257 in Python
- keine unnötigen Formatierungsstrings
  - stören (empfindlich) die Lesbarkeit und Übersichtlichkeit
  - Kommentare sollten *immer* als reiner Text lesbar sein.

PEP 257: `https://www.python.org/dev/peps/pep-0257/` (abgerufen am 29.11.2018)

Schlechten Code neu schreiben

Kommentare sind keine Kompensation

UNI FREIBURG

> *Don't comment bad code – rewrite it.*
>
> – Brian W. Kernighan, P. J. Plauger

- Kommentare sind keine Lösung für schlechten Code.
  - Guter Code braucht keine erläuternden Kommentare.
  - Guter Code ist lesbar und selbsterklärend.
- Guter Code ist das Ergebnis harter Arbeit.
  - setzt Bewusstsein, Wissen und Disziplin voraus
  - Fähigkeiten nehmen mit der Erfahrung zu.
- Kommentare sind eine temporäre Notlösung.
  - Schlechter Code *ohne* Kommentare ist noch schlimmer.

Kernighan, Plauger: The Elements of Programming Style, McGraw-Hill, New York 1978, S. 144

Kommentare auf ein Minimum reduzieren

Was man kommentieren sollte – und was nicht

UNI FREIBURG

- Kommentarköpfe
  - hilfreich für alle vom Nutzer aufrufbaren Funktionen
  - selten wichtig für private Funktionen/Methoden
  - werden bei selbstdokumentierenden Signaturen überflüssig (gilt i.d.R. nicht für vom Nutzer aufrufbare Funktionen)
  - nicht nur Namen von Funktion und Parametern wiederholen
- Copyright-Hinweise
  - sinnvollerweise immer vorhanden
  - ggf. mit Datum der letzten Änderung, Hinweis auf Lizenz
- einzeilige Kommentare im Code
  - vermeiden: Code umschreiben und lesbarer machen
  - müssen zwingend einen Mehrwert haben
  - besonders auf Korrektheit und Aktualität achten

Automatische Erzeugung von Dokumentation

Übersicht

UNI FREIBURG

- Einsatzgebiete
  - automatische Erzeugung einer Entwicklerdokumentation
  - Übersicht über die Schnittstellen aller Funktionen
  - Übersicht über alle Klassen
- Vor- und Nachteile
  - immer so aktuell wie die Kommentare im Code
  - potentiell große Wirkung mit eher geringem Aufwand
  - automatische Erzeugung einer lesbaren Dokumentation
  - eher nicht für Nutzerdokumentation geeignet
- Voraussetzungen
  - konsistente Formatierung
  - Verwendung spezifischer Formatierungen

☞ Die Kommentarköpfe müssen selbst geschrieben werden!

Sprachspezifische Aspekte

Unterschiedliche Werkzeuge und Herangehensweisen

UNI FREIBURG

- Jede Sprache hat ihre Konventionen.
  - betrifft Struktur, Formatierung, verwendete Werkzeuge
  - Konventionen der jeweiligen Sprache verwenden
  - Bsp.: Java – JavaDoc; Python – Sphinx
- Unterschiede in Inhalt und Zielstellung
  - entwicklerzentriert: reine Schnittstellendokumentation
  - nutzerzentriert: inkl. Erklärungen und Beispiele
- unterschiedliche Möglichkeiten der Textauszeichnung
  - meist spezifische Formatierungsanweisungen
  - sollte die Lesbarkeit nicht beeinträchtigen
  - Beispiele: Markdown, reStructuredText

☛ Konventionen der jeweiligen Sprache zumindest kennen

Hinweise zur Nutzung

Integration in die Infrastruktur und mögliche Nachteile

UNI FREIBURG

- Integration in den Editor
  - Unterstützung sprachspezifischer Kommentarköpfe
  - Vorlagen, die nur noch ausgefüllt werden müssen
  - Vorlagen sollten anpassbar sein
  - erspart sehr viel Tipparbeit
  - sorgt für (weitgehend) konsistente Formatierung
- Integration in die Build-Infrastruktur
  - Generation der Dokumentation automatisierbar
  - weiterführende Werkzeuge: Aufrufgraphen (*call graphs*) etc.
- mögliche Nachteile
  - Kommentarköpfe nicht immer sinnvoll
    (Bsp.: nicht-öffentliche Methoden)

☛ überlegt verwenden – wie alle anderen Werkzeuge auch

Zentrale Aspekte

- Dokumentation im Code sollte auf das notwendige Minimum beschränkt werden.
- Veraltete Dokumentation schadet mehr als sie nützt.
- Formulierungen sollten so knapp und präzise wie möglich, die Formatierung konsistent und übersichtlich sein.
- Unklarer Code sollte nicht dokumentiert, sondern umgeschrieben werden.
- Dokumentationswerkzeuge erleichtern die Arbeit, setzen aber konsistente Formatierung voraus.