

# Programmierkonzepte in der Physikalischen Chemie

## 6. (externe) Dokumentation

Albert-Ludwigs-Universität Freiburg



**UNI  
FREIBURG**

Dr. Till Biskup

Institut für Physikalische Chemie  
Albert-Ludwigs-Universität Freiburg  
Wintersemester 2017/18



## Zentrale Aspekte



- ❏ Grundlegende Ideen und Konzepte lassen sich schwer durch den Quellcode selbst dokumentieren.
- ❏ Dokumentiert werden sollte das Was und Warum (bzw. Was warum nicht). Das Wie beantwortet der Quellcode.
- ❏ Eine minimale externe Dokumentation (readme, install, erste Schritte) ist für größere Projekte unumgänglich.
- ❏ Externe Dokumentation lässt sich nur mit viel Disziplin mit dem Quellcode synchron halten.
- ❏ Einfach nutzbare Dokumentationswerkzeuge helfen, Dokumentation und Quellcode synchron zu halten.

“ *You can download our code from the URL supplied.  
Good luck downloading the only postdoc  
who can get it to run, though*

– Ian Holmes auf Twitter, 8. Jan. 2013

<https://twitter.com/ianholmes/status/288689712636493824>

Warum ist Dokumentation wichtig?

Vorurteile gegenüber Dokumentation

Arten von Dokumentation

Probleme mit (externer) Dokumentation

# Warum ist Dokumentation wichtig?

Ein Blick zurück – und nach vorne

(minimale) Anforderungen an Software zur Datenauswertung

- ▶ wiederverwendbar, zuverlässig, überprüfbar

		#			
wiederverwendbar	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>
selbstdokumentierend		<input checked="" type="checkbox"/>			
zuverlässig				<input checked="" type="checkbox"/>	
überprüfbar			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
nutzerfreundlich			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
erweiterbar	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>
reproduzierbar	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		

- ▶ geringe durchschnittliche Verweildauer von Mitarbeitern
  - maximal ca. vier Jahre
  - meist kein langfristiger Mitarbeiter, der Codepflege betreibt
- ▶ Ziel: schnell funktionierendes Programm
  - Nachhaltigkeit höchstens zweitrangig
  - Neuschreiben oft einfacher (und schneller), als Code zu verstehen, weiterzuverwenden, zu erweitern
- ☞ viel zu viel schlechter Code in den Wissenschaften
  - mangelhafte Dokumentation (auf allen Ebenen)
  - erschwert (unnötig) die Nachvollziehbarkeit
- ☞ mangelnde Effizienz
  - Code wird (notgedrungen) viel zu oft neu geschrieben.

# Warum ist Dokumentation wichtig?

Eine Liste guter Gründe

- ▶ Absichten/Konzepte nur schwer im Code dokumentierbar
  - Ziel des Projekts, Fähigkeiten
  - Grund für die verwendete Architektur
- ▶ Nachweis der Urheberschaft an Gedanken/Konzepten
  - grundlegende Gedanken frühzeitig schriftlich festhalten
- ▶ Nachvollziehbarkeit der Entwicklung eines Konzeptes
  - setzt Zugang zu alten Versionen (z.B. VCS) voraus
- ▶ Voraussetzung für Bewältigung größerer Projekte
  - externe Projektdokumentation (Details folgen)
- ▶ Gedankenstütze
  - besonders hilfreich für Programmierung „nebenher“

# Warum ist Dokumentation wichtig?

Was soll dokumentiert werden?

- ▶ „Was“ und „Warum“ dokumentieren
  - Anforderungsanalyse
  - Übersicht über grundlegende Architektur
  - Begründung konkreter Entscheidungen (z.B. Sprache)
- ▶ konkretes „Wie“ im Quellcode selbst
  - Der Quellcode ist das Design.
  - (externe) Dokumentation nicht zu detailliert
- ▶ nicht gewählte Alternativen
  - bewahrt die Übersicht
  - vermeidet wiederholte (Fehl-)Entwicklungen
- ☞ Dokumentation oft parallel zur Implementierung
  - kein Problem, solange dokumentiert wird

## Dokumentation ist die Zeit nicht wert, die sie benötigt.

- ▶ Verständnis kostet viel mehr Zeit.
  - Code wird viel häufiger gelesen als geschrieben.
  - Hauptentwickler meist größter Nutznießer
- ▶ Dokumentation skaliert.
  - Dokumentation muss nur einmal geschrieben werden.
  - Persönliche Einweisung in die Nutzung skaliert nicht.
- ▶ genauso wichtig wie der Methodenteil einer Arbeit
  - wird vermutlich (leider) genauso wenig wertgeschätzt...
- ▶ zwingt zum Nachdenken über den Code
  - Erst überlegen, was das Programm können soll.
  - Gedanken/Ideen *vorher* schriftlich festhalten erspart Arbeit.

## Dokumentation ist (zu) zeitaufwendig.

- ▶ komplettes Nutzerhandbuch ist tatsächlich aufwendig
  - eher seltener Anwendungsfall
  - Fokus auf Konzepte und große Linien (wenig Änderung)
- ▶ Dokumentation nahe an der Implementierung ist eine Frage von Disziplin und Übung
  - Übung: selbstdokumentierender Code ultimatives Ziel
  - Disziplin: synchrone Änderung von Dokumentation und Code muss Routine werden
- ▶ Entwicklerdokumentation weitestgehend automatisierbar
  - setzt Kenntnis der Werkzeuge voraus
  - erfordert Disziplin in der Umsetzung
  - Details in späterer Lektion („Dokumentation im Code“)

## Dokumentation ist meist unvollständig und veraltet.

- ▶ (leider) häufig beobachtbare Tatsache
  - oft fehlt grundlegende Information (Ziel, Fokus, Installation)
  - Frage von Bewusstsein und Disziplin
  - Dokumentation skaliert, Lebenszeit nicht
- ▶ Hinweise für Nutzerhandbücher u.ä.
  - großes Aktualisierungsintervall (größere Versionssprünge)
  - vorab Gliederung erstellen
  - nur Abschnitte mit Inhalt veröffentlichen
- ▶ anders und besser statt nicht dokumentieren
  - selbstdokumentierender Code
  - feste Strukturen und Vorlagen
  - klare, eingeübte Abläufe

- ▶ unterschiedliche Kategorisierung möglich
  - Adressaten (Nutzer, Administratoren, Entwickler)
  - Ort (im Quellcode, separat/extern)
  - äußere Form (elektronisch/gedruckt; knapp/umfassend)
- ▶ Kontext wissenschaftlicher Software
  - theoretische Abhandlungen
  - Nutzer- und Entwicklerdokumentation
  - Kommentare im Quellcode
  - selbstdokumentierender Code

## externe Dokumentation

Dokumentation, die nicht direkt im Quellcode vorliegt, sondern in separaten Dateien neben oder getrennt vom Quellcode

### ▶ Charakteristika

- höchster Abstraktionsgrad von Dokumentation
- feste Form (These, Weißbuch, Bericht, Publikation)
- zusammenhängende Darstellung inkl. Literatur

### ▶ Vorteile

- beschreibt wissenschaftliche Ziele und Herkunft des Codes
- meist hohe Qualität und gute Übersicht
- (i.d.R.) begutachtet
- (i.d.R.) zitierfähig

### ▶ Nachteile

- Erstellung aufwendig
- statisch
- (meist) keine Beschreibung der konkreten Implementierung (keine Entwicklerdokumentation)

- ▶ kurze Einführung (readme)
  - obligatorisch für jedes Projekt
  - Zielstellung und Fähigkeiten des Projekts (kurz!)
- ▶ Installationshinweise
  - insbesondere bei Quellcode-Distributionen
  - Verweis auf Abhängigkeiten und Voraussetzungen
- ▶ Beispiele
  - gut gewählt und hinreichend beschrieben
  - möglichst getestet/testbar
- ▶ Beschreibung der Nutzerfunktionen
  - Unterscheidung: öffentliche API/Entwicklerdokumentation
  - alle Nutzerfunktionen sauber beschreiben

- ▶ Details zur Implementierung
  - „Was“ und „Warum“
  - „Was warum nicht“:  
betrachtete, nicht gewählte Alternativen beschreiben
  - nicht zu detailliert: „Der Code ist das Design.“
  
- ▶ (automatisch generierte) API-Dokumentation
  - erfordert minimale Mehrarbeit und Disziplin beim Programmieren
  - vollständig automatisierbar
  - wird in späterer Lektion ausführlicher behandelt
  
- ▶ Projektdokumentation
  - Details folgen

## Bestandteile einer externen Projektdokumentation

- ▶ Anforderungsanalyse
  - kann sich ändern (gerade am Anfang)
- ▶ Architektur/High-Level-Design
  - nicht gewählte Alternativen benennen/begründen
- ▶ Roadmap
  - bei größeren Projekten bzw. limitierten Ressourcen
  - (flexible) Priorisierung einzelner Aufgaben
- ▶ Changelog
  - wichtig für Kompatibilität
- ▶ Konventionen
  - konsequente Umsetzung wichtiger als konkreter Inhalt

- ▶ **Asynchronität zwischen Code und Dokumentation**
  - Synchrone Änderung erfordert Disziplin.
  - veraltete Dokumentation ggf. schlimmer als fehlende
- ▶ **Zeitaufwand für die Erstellung von Dokumentation**
  - externe Dokumentation vom Code getrennt
  - Notwendige Vorausplanung ist Teil der Dokumentation.
  - nicht zu viele Details dokumentieren:  
„Der Code ist das Design.“
- ▶ **Unvollständige oder falsche Dokumentation**
  - großes Problem vieler „Nebenher-Projekte“
  - Ursache oft mangelnde Disziplin
  - falsche Dokumentation ggf. schlimmer als fehlende

- ▶ niederschwellig
  - Fokus: einfache Nutzbarkeit
  - Bsp.: Wiki
  
- ▶ Struktur/Konventionen
  - autoritative Antworten auf viele kleine alltägliche Fragen
  - hilft bei der Ausbildung von Routine
  - Konventionen sind nicht in Stein gemeißelt...
  
- ▶ Vorlagen (Templates)
  - befreiend, beschleunigend, vereinheitlichend
  
- ▶ klare Abläufe
  - Konventionen für den Akt des Dokumentierens
  - Automatisierung, ggf. Checklisten

- ▶ **Automatisierung**
  - vereinfacht und beschleunigt
  - sorgt für Konsistenz
  
- ▶ **ein Ort**
  - Informationen an einem Ort in einem (Meta-)Format
  - verhindert Inkonsistenzen
  - zentraler Speicherort mit gutem Zugriff (VCS, online)
  
- ▶ **Disziplin**
  - hilfreich: einfache Lösungen und Struktur/Konventionen
  
- ▶ **Vorausplanung**
  - Gliederung der gesamten Dokumentation
  - Priorisierung
  - ggf. nur Abschnitte mit Inhalt veröffentlichen



- ❏ Grundlegende Ideen und Konzepte lassen sich schwer durch den Quellcode selbst dokumentieren.
- ❏ Dokumentiert werden sollte das Was und Warum (bzw. Was warum nicht). Das Wie beantwortet der Quellcode.
- ❏ Eine minimale externe Dokumentation (readme, install, erste Schritte) ist für größere Projekte unumgänglich.
- ❏ Externe Dokumentation lässt sich nur mit viel Disziplin mit dem Quellcode synchron halten.
- ❏ Einfach nutzbare Dokumentationswerkzeuge helfen, Dokumentation und Quellcode synchron zu halten.