

Programmierkonzepte in den Naturwissenschaften

13. Namen

PD Dr. Till Biskup

Physikalische Chemie und Didaktik der Chemie
Universität des Saarlandes
Sommersemester 2020





- 🔑 Gut gewählte Namen sind eine essentielle Voraussetzung für lesbaren Code – und alles, was davon abhängt.
- 🔑 Ein guter Name beantwortet die großen Fragen: Warum existiert etwas? Was tut es? Wie nutzt man es?
- 🔑 Programmierer sind Autoren („Schriftsteller“): Der Code sollte so offensichtlich wie möglich sein.
- 🔑 Ein Name, der in einem Kommentar erklärt werden muss, ist (meist) ein schlecht gewählter Name.
- 🔑 Gute Namen erfordern gute beschreibende Fähigkeiten und einen gemeinsamen kulturellen Hintergrund.

Warum sind Namen wichtig?

Allgemeine Regeln für Namen

Spezifische Regeln für Variablen, Funktionen, Klassen, Objekte

Ausblick: Namen lassen sich ändern

- “ *Die Grenzen meiner Sprache bedeuten die Grenzen meiner Welt.*
– Ludwig Wittgenstein, Tractatus 5.6
- ▶ Gute Namen sind wesentlich für die Lesbarkeit von Code.
 - Lesbarkeit ist zentral für Wiederverwendbarkeit, Nachvollziehbarkeit und Überprüfbarkeit.
 - ▶ Namen strukturieren unser Denken.
 - Namen repräsentieren (komplexe) Konzepte.
 - Systematisierung ist ein altes „Hobby“ der Menschheit.
 - ▶ Namen spiegeln unser Verständnis einer Sache wider.
 - je besser das Verständnis, desto treffender der Name

Warum sind Namen wichtig?

Was professionelle Programmierer dazu zu sagen haben.



“ *You should name a variable using the same care with which you name a first-born child.*

– James O. Coplien

“ *One difference between a smart programmer and a professional programmer is that the professional understands that clarity is king. Professionals use their powers for good and write code that others can understand.*

– Tim Ottinger

👉 **Namen sind einer der wichtigsten Aspekte lesbaren Codes.**

“ *The name of a variable, function, or class, should answer all the big questions. It should tell you why it exists, what it does, and how it is used. If a name requires a comment, then the name does not reveal its intent.*

– Tim Ottinger

- ▶ Antworten auf drei Fragen
 - Daseinsberechtigung
 - Fähigkeiten
 - Nutzung

- ▶ Qualitätskriterium
 - Ein Name sollte für sich sprechen.

Listing 1: FORTRAN77-Code eines realen Programms

```
do 300 i=1, nmd
  do 300 j=1, nmd
    tD(i, j) = tDr(i, j)
    tg1(i, j) = tg1r(i, j)
    tg2(i, j) = tg2r(i, j)
    ta1(i, j) = ta1r(i, j)
    ta2(i, j) = ta2r(i, j)
300 continue
```

Listing 2: Mehr FORTRAN77-Code eines realen Programms

```
1          rinty = st*BinhI(kB)*hfI1(k1)*hfI2(k2)
           *hfIu1(k3)*hfIu2(k4)*poptp
c
1          sigmaz1 = sigmaz + aal*ta1r(3, 3) + aa2*ta2r(3, 3)
           + aul(k3) + au2(k4)
1          deltax1 = deltax + aal*ta1r(3, 3) - aa2*ta2r(3, 3)
           + aul(k3) - au2(k4)
```

Gute Namen sollten...

- ▶ die Absicht offenbaren
 - Der Code sollte offensichtlich sein.
 - Ein Name, der erklärt werden muss, ist schlecht gewählt.
- ▶ aussagekräftige Unterscheidungen treffen
 - verschiedene Namen nur bei unterschiedlicher Bedeutung
 - Programmierer müssen auf kleinste Details achten.
 - Zufällige/inkonsistente Unterschiede lenken ab.
- ▶ aussprechbar sein
 - Unterhaltung über den Code ermöglichen
- ▶ suchbar sein
 - eindeutige, nicht zu kurze Namen
 - keine einzelnen Buchstaben oder Zahlen

Gute Namen sollten...

- ▶ ein Wort pro Konzept verwenden
 - konsistent über alle Klassen etc.
 - genau ein Wort für genau ein Konzept (eindeutig/bijektiv)
- ▶ aus dem Lösungsraum stammen
 - Normalerweise lesen Programmierer den Code.
 - Wenn es keinen Begriff aus der Informatik gibt:
Begriff aus dem Problemraum verwenden
- ▶ aussagekräftigen Kontext hinzufügen
 - Die wenigsten Namen sprechen für sich selbst.
 - Kontext: gut benannte Funktionen, Klassen, Objekte
 - im Notfall Präfix vor den Namen setzen

Gute Namen vermeiden...

- ▶ Desinformation
 - Namen und Inhalt sollten übereinstimmen.
 - konsistente Schreibweise statt kleine Variationen
 - Unterschiede *klar* hervorheben

- ▶ Kodierung
 - Bsp.: Variablentyp durch Präfix
 - Typen können sich ändern und sind normalerweise ein unwichtiges Detail der Implementierung.

- ▶ Notwendigkeit mentaler Übersetzung
 - häufige Ursache: Namen weder aus dem Problem- noch aus dem Lösungsraum
 - anderes Beispiel: unnötig kurzer Name

Gute Namen vermeiden...

- ▶ unnötige Metaphern
 - häufig sehr kontextspezifisch
 - keine Redensarten oder Jargon
- ▶ Mehrdeutigkeiten
 - genau ein Wort für genau ein Konzept (eineindeutig)
 - Gerade subtile Unterschiede in der Bedeutung erfordern unterschiedliche Namen.
- ▶ unnötigen Kontext
 - offensichtlichen Kontext nicht im Namen wiederholen
 - keine „globalen“ Präfixe
 - Namensräume, Klassen, Objekte sorgen für Kontext.

- ▶ Faustregel: so kurz wie möglich, so lang wie nötig
 - Moderne Programmiersprachen schränken wenig ein.
 - Moderne Editoren helfen beim Verwenden langer Namen.
- ▶ Der Kontext entscheidet.
 - Je globaler der Kontext, desto mehr Information muss durch den Namen transportiert werden.
 - Namen sollten in ihrem jeweiligen Kontext eindeutig sein.
 - Der Kontext sollte im Namen nicht wiederholt werden.
- ▶ Ein Name ist eine Abstraktion.
 - Namen transportieren Konzepte.
 - Namen sind Etiketten, keine vollständigen Beschreibungen der dahinterstehenden Konzepte.

- ▶ Moderne Programmiersprachen schränken wenig ein.
 - Meist wird zwischen Groß- und Kleinschreibung unterschieden.
- ▶ Es gibt viele verschiedene Konventionen.
 - konsistente Umsetzung wichtiger als Inhalt
 - Konventionen *schriftlich* festhalten
- ▶ Konventionen transportieren Information.
 - Bsp.: GROSSBUCHSTABEN für Konstanten
 - können fehlende Aspekte von Sprachen emulieren
- ▶ Konsistenz ist entscheidend.
 - gleichartige Namen immer gleich schreiben
 - *nie* unterschiedliche Schreibweisen desselben Namens

zwei verbreitete Konventionen

▶ CamelCase

- großer oder kleiner Anfangsbuchstabe
- mitunter Unklarheit über Wortgrenzen (Bsp.: `filename` oder `fileName`)
- Konventionen für groß geschriebene Abkürzungen: groß oder klein fortfahren?

▶ Unterstriche

- Viele Programmiersprachen erlauben nur Unterstriche.
- Bindestrich wird meist als Minuszeichen (Operator) interpretiert

☞ Wichtig ist die konsequente Umsetzung, weniger die Wahl einer Konvention.

☞ Konventionen sollten *schriftlich* festgehalten werden.

Listing 3: Eine einfache, aber schwer verständliche Funktion

```
def get_them():
    list1 = ArrayList()
    for x in the_list:
        if x[0] == 4:
            list1.add(x)
    return list1
```

Listing 4: Die gleiche, aber diesmal verständlichere Funktion

```
def get_flagged_cells():
    flagged_cells = ArrayList()
    for cell in game_board:
        if cell[STATUS_VALUE] == FLAGGED:
            flagged_cells.add(cell)
    return flagged_cells
```

👉 Irgendeine Idee, um was es hier gehen könnte?

Klassen und Objekte: (zusammengesetzte) Substantive

- ▶ generelle Regel: (zusammengesetzte) Substantive
 - zu allgemeine Begriffe als Zusatz vermeiden
 - *keine* Verben
- ▶ Klassen tragen allgemeinere Namen als Objekte.
 - Klassen repräsentieren das allgemeine Konzept.
 - Objekte repräsentieren eine konkrete Instanz.
 - bei der Benennung von Klassen Objekte mitdenken
- ▶ Klassen und Objekte schaffen Kontext.
 - Kontext für Eigenschaften und Methoden

Funktionen und Methoden: (aktive) Verben

- ▶ generelle Regel: (aktive) Verben
 - aktive Verben zumindest am Anfang jedes Namens
 - nicht ausschließlich Substantive oder Adjektive
- ▶ spezielle Namen für spezielle Methoden
 - Lesen, Schreiben oder Überprüfen von Eigenschaften
 - festes Präfix vor der jeweiligen Eigenschaft
 - Konventionen: `get`, `set`, `is`
- ▶ Methoden von Klassen/Objekten
 - Kontext (Klassen-/Objektname) *nicht* wiederholen
 - ggf. Konventionen für private Methoden

- ▶ kurz für lokale, lang für globale(re) Variablen
 - je kürzer, desto übersichtlicher
 - Kriterium: im jeweiligen Kontext *eindeutig*
 - ggf. passenden Kontext schaffen (z.B. Objekt/Klasse)
- ▶ keine einzelnen Buchstaben
 - einzige Ausnahme: Laufindex in *kurzer* Schleife
 - *i* und *j* in mancher Programmiersprache reserviert
 - *niemals* `l` verwenden (!)
- ▶ Benennung magischer Zahlen
 - Faustregel: Alle Zahlen außer 0 und 1 sind „magisch“.
 - aussagekräftiger, suchbarer Name
 - ggf. komplett in Großbuchstaben (Konstanten)

“ *The hardest thing about choosing good names is that it requires good descriptive skills and a shared cultural background.*

– Tim Ottinger

- ▶ Beschreibung setzt (präzise) Beobachtung voraus.
 - Wissenschaftler sind (eigentlich) prädestiniert ...
 - Präzise und prägnante Beschreibung lässt sich trainieren.
 - Diese Fähigkeiten lassen sich vielfältig einsetzen.

- ▶ gemeinsamer kultureller Hintergrund
 - Konventionen für die Benennung von Konzepten
 - Wissen über Konzepte und Entwurfsmuster

“ *Choosing good names takes time but saves more than it takes. So take care with your names and change them when you find better ones.*

– Tim Ottinger

- ▶ Gute Namen sparen Zeit.
 - je besser der Name, desto (einfacher) lesbarer der Code
- ▶ Namen sollten gut überlegt werden.
 - Benennung ist eine der häufigsten Tätigkeiten.
 - Bewusstsein für die Bedeutung guter Namen ist wichtig.
- ▶ Namen lassen sich ändern.
 - Moderne Editoren helfen bei der Umbenennung.

“ *Our goal, as authors, is to make our code as easy as possible to understand. We want our code to be a quick skim, not an intense study.*

We want to use the popular paperback model whereby the author is responsible for making himself clear

and not the academic model where it is the scholar's job to dig the meaning out of the paper.

– Tim Ottinger



- 🔑 Gut gewählte Namen sind eine essentielle Voraussetzung für lesbaren Code – und alles, was davon abhängt.
- 🔑 Ein guter Name beantwortet die großen Fragen: Warum existiert etwas? Was tut es? Wie nutzt man es?
- 🔑 Programmierer sind Autoren („Schriftsteller“): Der Code sollte so offensichtlich wie möglich sein.
- 🔑 Ein Name, der in einem Kommentar erklärt werden muss, ist (meist) ein schlecht gewählter Name.
- 🔑 Gute Namen erfordern gute beschreibende Fähigkeiten und einen gemeinsamen kulturellen Hintergrund.