



Programmierstil

Tobias Sielhorst

06 December 2006

chair for computer aided medical procedures

department of computer science | technische universität münchen

Was ist guter Stil?

- Übersichtlich/verständlich = debugbar → fehlerfrei
- Übersichtlich/verständlich = optimierbar → schnell, effizient
- Übersichtlich/verständlich = wiederverwendbar → mehr Zeit für wichtigeres + alter Code wurde vermutlich schon getestet

Funktionen

- Länge
 - Eine Funktion sollte nicht mehr als eine Seite beinhalten
 - Ein Aufruf sollte nicht länger als eine Zeile sein
 - Ein Aufruf sollte nicht viele andere Aufrufe schachteln
- Kommentare
 - Mindestens in jede dritten Zeile sollte ein Kommentar sein
 - Alle öffentlichen und geschützten Funktionen und Variablen sollten Kommentare im Header haben
 - Kommentare entweder vor dem Code oder in der selben Zeile – nie in folgenden Zeilen
- Variablendeklaration sollte in der Nähe des ersten Aufrufs sein

Namen

- Sollten nicht den Namen des Autors beinhalten
- Sinn machen (doIt, doThis, handleThat)
- Etwas bezeichnen (x, x1, a, b)
- Nicht ähnlich zu anderen Namen sein (myReference, myReferene)
- Auf keinen Fall Rechtschreibfehler enthalten
- Auf englisch sein (oder in der offiziellen Sprache des Projektes)

Datenfluss

- Const benutzen
- Es gibt keinen vernünftigen Grund für globale Variablen, Funktionen oder anderes
- Es gibt keinen vernünftigen Grund für `#define`
- Möglichst wenig Daten fließen lassen, indem sich der interne Zustand des Objektes ändert
(also in einer Funktion Sachen in eine Membervariable ablegen, die man auch mit einer Referenz übergeben kann)
- Man muss nicht immer alles mit Vererbungshierarchien lösen – nur wenn es die Aussage trifft

Schneller Code

- Stilvoll programmieren bedeutet nicht (effektiv) langsamerer Code:
- 80/20 Regel beachten: 80% der Zeit ist das Programm in 20 % vom Code
 - Erst schauen welche Zeile am meisten Zeit verbraucht, dann diese optimieren
 - Ansonsten standardmäßig Geschwindigkeit gegen Übersichtlichkeit und Einfachheit opfern
- Wenn man das durchzieht, hat man immer schnellen Code, denn man beschäftigt sich weniger mit Debuggen und Optimierung von Stellen, die es nicht Wert sind

Fehlersuche

- Din A4 Blatt mit seinen Lieblingsfehlern anlegen
- Compiler-Warnungen haben ihren Sinn!
- Linkerfehler habe nur 3 Ursachen
 - Die .lib-Datei wurde nicht im Projekt nicht (korrekt) angegeben
 - Der Pfad für die .lib Datei wurde nicht (korrekt) angegeben
 - Es gibt die Funktion im Header, aber nicht in der cpp Datei (mit der korrekten Signatur)

Bibliotheken

- Man muss nicht das Rad 2x erfinden
- STL sollte man kennen (gehört standardmäßig zu C++)
 - `vector` (`list`, `map`) statt Arrays
 - Effiziente Suchalgorithmen für jeden Container

<http://www.sgi.com/tech/stl/>
- Boost Bibliotheken (gehören *noch* nicht standardmäßig zu C++)
 - andere Container
 - Graphen und Algorithmen
 - Intelligente Pointer, etc...

<http://www.boost.org/>
- Vieles anderes gibt es schon in vernünftiger Qualität mit einer freundlichen Lizenz