Proseminar „C-Grundlagen und Konzepte“, 2013
Programmierstile

Minh Nguyen Nhu

Gliederung

1. Allgemein

Mit Programmierstil meint man das Erstellen von Quellcode nach bestimmten vorgegebenen Regeln, was ein Teilaspekt der Softwarequalität ist. Es Bezieht sich eher auf die nichtfunktionalen und formalen Anforderungen, wie die Gestaltung des Codes. Also betrachtet man hier das "wie" und nicht das "was".

Ein guter Programmierstil zeichnet sich durch ein klar definiertes und von außen sichtbares Erscheinungsbild aus. Es erleichtert die Arbeit am Projekt für alle beteiligten Teammitglieder, insbesondere die Lesbarkeit, Verständlichkeit und Wartbarkeit werden durch die formale Sauberkeit und einer verständlichen Struktur deutlich verbessert, damit auch die Fehlerbeseitigung. Außerdem zwingt man sich durch einen gewissen Stil selbst zum sauberen Programmieren.

In größeren Projekten werden häufig Richtlinien für den Programmstil festgelegt. Dies dient nicht nur der Entwicklung, sondern ist besonders wichtig für die Wartung. So entfallen 80% der Lebenszeit einer Software auf die Wartung. Nur selten wird ein Produkt vom ursprünglichen Programmierer gewartet. Daher ist es hilfreich schon zu beginn einen guten Programmierstil zu verwenden. Jedoch sollte man den Stil des zu wartenden Programms beibehalten.

2. Kommentare

Man muss immer davon ausgehen das der Code auch von anderen gelesen und verstanden werden muss. dafür sind kommentare unerlässlich. Stellen die nicht selbsterklärend sind, bestimmtes Vorwissen erfordern oder für andere Stellen im Code wichtig sind, sollten kommentiert werden. Diese sollten sich jedoch nur darauf beschränken was eine Funktion macht und nicht wie.

3. Left-hand- vs. Right-hand-Comparison

Vergleich

```
if (antwort == 42) {...}
```
Right-hand

```
if (42 == antwort) {...}
```
Left-hand

Zuweisung

```
if (antwort = 42) {...}
```
Right-hand

```
if (42 = antwort) {...}
```
Left-hand

Bei der Right-hand-Comparison steht der zu vergleichende Ausdruck auf der rechten Seite, bei der Left-hand-Comparison auf der linken Seite. Beim Vergleich würde es keine auswirkungen haben, aber wenn man ausversehen eine Zuweisung macht, würde `antwort` auf `42` gesetzt werden und der folgende Block durchlaufen werden. In der anderen Richtung ist es nicht möglich, da der `42` `antwort` nicht zugewisen werden kann.

4. Namenskonventionen

Namenskonventionen sind Vereinbarungen zur Vergabe von Bezeichnern für Objekte, Variablen und Konstanten nach einem bestimmten System. Diese sollten denVerwendungshinweis hinreichend erklären oder zumindestens andeuten. Daher sollte man sie in der Regel "schön" ausschreiben. Ausnahmen sind Kürzel bei denen Missverständnisse vollkommen ausgeschlossen sind. Außerdem sollten sich die Schreibweisen zwischen Variablen, Funktionen, Dlassen, Konstanten usw unterscheiden.
Lokale Variablen sollten einen kurzen Namen haben. Einbuchstabige Bezeichner sollten nur für loakale indizes und Schleifen verwendet werden. So ist einer der verbreitesten Schleifenzähler `i`.

Damit Bezeichner besser lesbar sind sollten man CamelCase(Klein-/Großschreibung) oder Unterstriche benutzen. Eine Mischung von beiden ist dringend abzuraten.

Ein andere Aspekt ist welche Sprache man nutzen sollte. Wenn es Open Source ist sollte man unbedingt englische Bezeichner nehmen. In einem Team voller Deutschsprechende ist detusch vllt die besser Wahl. Da aber heutzutage jeder ein Hauch von Englisch versteht, würde ich diese Sprache empfehlen. Nicht nur das viele Fachbegriffe der Informatik ihren Ursprung im englischen haben, es ist auch manchmal wesentlich kürzer.

5. "Ungarische Notation"

Verdankt ihren Namen Charls Simonyi, aufgrund seiner ungarischen Herkunft als auch das "exotische" Aussehen der Bezeichner. Seine Idee war es die Aufgabe und den `type` einer Variabel in deren Namen zu verdeutlichen.
Mit `type` meinte Simonyi jedoch die Art einer Variabel im spezifischen Kontext. Microsoft interpretierte dies jedoch als Datentyp weshalb 2 Arten von Notationen entstanden. Apps Hungarian, im Sinne von Simonyi und Systems Hungarian, die Fehlinterpretation Microsofts. Letztere ist für den schlechten Ruf verantwortlich, weil die Bennenung einer Variabel nach dem Datentyp wenig zum Verständnis beiträgt und trotzdem viel Aufwand verursacht.

Apps Hungarian

Um wenig aussagekräftige Bezeichner zu vermeiden, hat er klar definiert, welche Attribute ein Name enthalten darf. Da alle Attribute jedoch optional sind, kann man Laufvariablen auch ein- bzw zweibuchstabig bezeichnen.

Präfix-Beispiele	
pointer	**p**
ma**p**	**mp**
domai**n**	**dn**
index	**i**
element	**e**
difference	**d**

Datentypen-Beispiele	
flag	**f**
char	**ch**
string	**st**
functio**n**	**fn**
fi**l**e	**fl**
void	**v**

Bezeichner-Beispiele	
`Min`	allererstes Element eines Arrays
`Last`	letztes Element eines Arrays
`Max`	tatsächliche Anzahl von Elementen eines Arrays

6. Einrückungsstile

Mit Einrückungsstil wird die Art und Weise bezeichnet, Code zur besseren Lesbarkeit einzurücken und umschließende Syntax-Elemente wie Geschweifte Klammern zu positionieren. Daher wird er auch manchmal Klammerstil genannt.

Viel diskutiert wird über die Einrückungstiefe der untergeordneten Blöcke. Besonders weit verbreitet sind Einrückungstiefen um 4 bis 8 Zeichen, aber auch 2 oder 3 werden meist verwendet. Während das eine die Sichtbarkeit der Einrückung priorisiert, ist die nutzbare Zeilenlänge bei der anderen größer.
Jedoch das umstrittenste Element eines Programmstils ist die Positionierung der Klammern.
So gibt es unterschiedliche Ansichten wie "Schönheit" definiert ist.

Beispiele für Einrückungsstile

1TBS

```
int f(int x, int y, int z) {
        if (x < foo(y, z)) {
                haha = bar[4] + 5;
        } else {
                while (z) {
                        haha += foo(z, z);
                        z--;
                }
                return ++x + bar();
        }
}
```

Der Name kommt aus der Hacker-Szene und steht für "one True Brace Style", übersetzt: "einzig wahrer Kalmmerstil". Bezieht sich darauf, dass dieser Stil von den C-Erfindern Brian W. Kernighan und Dennsi Ritchie definiert wurde.

Aufgrund der Einrückung erkennt man die Anweisung die den Block einleitet recht schnell. Jedoch bei Unachtsamkeit besteht die Gefahr, öffnende Klammer zu übersehen und schließende zu vergessen.
Dafür wird wesentlich mehr Code im Vergleich zu anderen Stielen in der gleichen Anzahl an Zeilen dargestellt und unselbstständige Folgeblöcke werden an der schließenden geschweiften Klammer unter einer Schließendengeschweiften Klammer in derselben Spalte erkannt.

Überwiegende Mehrheit der Javaprogrammierer verwenden diesen Stil.
C-Code verwendete früher 8 Leerzeichen Einrückung, heute sind auch 4 oder 3 Leerzeichen zu sehen. Bei Java und C++ ist es in der Regel eine Einrückung um 4.

K&R

```
int f(int x, int y, int z)
{
        if (x < foo(y, z)) {
                haha = bar[4] + 5;
        } else {
                while (z) {
                        haha += foo(z, z);
                        z--;
                }
                return ++x + bar();
        }
}
```

Das Original von den C-Erfindern Brian W. Kernighan und Dennis Ritchie, dass in ihrem Buch "The C Programming Language" verwendet wurde. Auch die Code-Conventionen von Linux schlagen diesen Stil vor.

Die Einrückung beträgt 8 Leerzeichen. Dies erleichtere die Lesbarkeit auch nach mehrstundiger Arbeit mit dem Quelltext. Dafür werden die Anweisungen stärker nach rechts geschoben. Da man jedoch nicht mehr als 3 Ebenen einrücken sollte, ist es noch im Rahmen. Trotzdem wird dies von manchen Programmierern als "unleserlich" bezeichnet.

Der Untschied von K&R im Vergleich zum 1TBS ist, dass es bei der Definition von Funktionen auf

dem Allman-Stil zurückgreift. Die öffnende Klammern befinden sich in einer eigenen Zeile Begründungen hierfür sind, dass Funktionen nicht geschachtelt werden können und es biete eine besonders hohe Lesbarkeit bei umbrochenen Funktionsdefinitionen.

Auf C++ wird dieser Stil auch nach dem C++ Erfinder Stroutrup bezeichnet, der den Stil um Elemente für C++ erweitert hat. Für alle neuen Elemente, z.B. Klassen, werdengeschweifte Klammen ans Ende gestellt.

Allman

```
int f(int x, int y, int z)
{
    if (x < foo(y, z))
    {
        haha = bar[4] + 5;
    }
    else
    {
        while (z)
        {
            haha += foo(z, z);
            z--;
        }
        return ++x + bar();
    }
}
```

Dieser Stil wurde nach Eric Allman bennant, der eine Vielzahl an BSD(Berkeley Software Distribution, eine Version des Unix-Betriebsystems an der Universität von Kalifornien in Berkeley) Werkzeugen geschrieben hat, weshalb er auch manchmal als BSD-Stil bezeichnet wird, was nicht ganz stimmt.

Durch die Umklammerung der Blöcke und ihrer Einrückung ist die Struktur besonders gut zu erkennen. Klammer-Paare lassen sich durch ihre gleiche Einrückungsebene leicht finden

Anders als im 1TBS, ist die öffnende Klammer in einer eigenen Zeile was zu einen hohen Zeilenverbrauch führt. Es wird eine zusätzliche Zeile mehr je Block als beim 1TBS benötigt. In C

und Sprachen mit ähnlichem Präprozessoren gibt es jedoch Situationen, in denen dieser Stil auch wesentliche Vorteile gegenüber 1TBS zeigt. Und zwar beim Einsatz von bedingtem Compiling für alternative Blockeinleitungen. Denn da bleibt die Anzahl der Klammern gleich, was wichtig für die korrekte Zuordnung von öffnenden und schließenden Klammen ist.

Horstman

```
int f(int x, int y, int z)
{    if (x < foo(y, z))
    {    haha = bar[4] + 5;
    }
    else
    {    while (z)
         {    haha += foo(z, z);
              z--;
         }
         return ++x + bar();
    }
}
```

Horstman ist eine Abwandlung des Allman-Sitls, bei dem die erste Anweisung nach der öffnenden Klammer in derselben Zeile geschrieben wird.

Für Versionsverwaltungs-Tools ist dieser Stil problematisch. Fügt man eine neue Anweisung zwischen der öffnenden Klammer und der ersten Anweisung ein, erscheint diese Änderung als "eine Zeile wird durch 2 Zeilen ersetzt" statt "eine neue Zeile wurde Eingefügt". So wird die ursprüngliche Zeile als geändert markiert, obwohl sie semantisch gleich geblieben ist.

Whitesmith-Stil

```
int f (int x, int y, int z)
        {
        if (x < foo (y, z))
                {
                haha = bar[4] + 5;
                }
        else
                {
                while (z)
                        {
                        haha += foo (z, z);
                        z--;
                        }
                return ++x + bar ();
                }
        }
```

Dieser Stiel wurde ursprünglich in der Dokumentation des ersten kommerziellen C-Compilers eingesetzt und war früher in den Windows-Programmierhandbüchern populär.

Hierbei sind die Block klar erkennbar und im Vergleich zu Allman hat es einen besseren Lesefluß, aber auch hohen Zeilenverbrauch.

GNU-Stil

```
int f (int x, int y, int z)
{
   if (x < foo (y, z))
     haha = bar[4] + 5;
   else
     {
       while (z)
         {
           haha += foo (z, z);
           z--;
         }
       return ++x + bar ();
     }
}
```

Vorallem in GNU-Projekten present und empfohlem. Kombiniert die Vorteile von Allman und Whitesmiths, in dem die geschweiften Klammern in der Einrückung eine eigene Ebene besitzen.
Dafür bildet es keine schöne Linie mit den Kontrollstrukturen.

Gegner des Stils argumentieren, dass Chaos bei Quelltextformatierung entstehe, die Lesbarkeit und konsequente Einrückung würde erschwert werden und es seie besonders fehleranfällig.

7. Tabs oder Leerzeichen?

Tabs besitzen den Vorteil, dass ihre Schrittweite einstellbar sind. So hängt die Einrückungtiefe je nach den Anzeige-Optionen des Editors des jeweiligen Programmierers ab. Die ist gleichzeitig ein Nachteil da bei Übertragung von Code mit anderen Projektmitgliedern es zu Verweirrung kommen kann.
Dies geschieht beim Leerzeichen nicht. Einrückungen mit Leerzeichen werden konstant dargestellt, unabhängig vom den anzeigedarstellungen des Editors.

Man sollte die Vermischung von Tabs und Leerzeichen vermeiden. Das Anzeigen von Tabs im Editor hilft dabei.

Die Konventionen von Java, Linux, Sun und weitere raten von der Nutzung von Tabs ab.

8. Style Checker und Beautifier

Beide helfen den Programmierer die gewünschten Programmierstil beizubehalten.
Style Checker ist Ein Tool welches die Einhaltung eine zuvor definierten Programmstils überprüft, jedoch formatieren diesen nicht um. Beispiele sind Checkstyle für Java.
Beautifier hingegen formatieren den Code um. Beispiele hierfür sind astyle und indent.

Jedoch Beautifier formatieren sehr schematisch und ausdruckslos um. Daher können sie den semantischen Hintergedanken nicht berücksichtigen. Ein anderer Nachteil ist, dass sie den persönlichen Stil eines Programmierers nicht beachten. Jedoch das Ego sollte in einem Team nicht existent sein.

9. Quellen

http://de.wikibooks.org/wiki/C-Programmierung:_Programmierstil
http://en.wikipedia.org/wiki/Naming_convention_%28programming%29
http://de.wikipedia.org/wiki/Programmierstil
http://de.wikipedia.org/wiki/Style_Checker
http://de.wikipedia.org/wiki/Beautifier
http://de.wikipedia.org/wiki/Code-Qualit%C3%A4t
http://en.wikipedia.org/wiki/Coding_conventions
http://de.wikipedia.org/wiki/Ungarische_Notation
http://en.wikipedia.org/wiki/Indent_style
http://www.planet-quellcodes.de/forum/index.php?showtopic=4052