Die zehn Methoden zum Schreiben unlesbarer Programme

B2

In diesem Kapitel

- Erfahren Sie, was Sie tun müssen, damit andere Programmierer mit Ihren Programmen nichts anfangen können

Die Bedeutung von Kryptographie wird immer stärker erkannt. Gehen Sie als moderner Software-Entwickler mit der Zeit und schreiben Sie Programme, die keiner außer Ihnen entziffern kann. Frei nach dem Motto: »It was hard to code, so it should be hard to read«. Die Flüche aller anderen Mitarbeiter (selbst schuld – hätten die was Anständiges gelernt) sind Ihnen sicher, wenn Sie die folgenden Ratschläge aufs Wort befolgen.

Wenn Sie jemand deswegen persönlich angreift oder bei Ihrem Vorgesetzten anschwärzt, benutzen Sie als Argument, dass die Programme dadurch schneller laufen, was den Kunden Geld sparen würde. Diese Methode ist besonders effektiv, falls Ihr Vorgesetzter BWL studiert hat.

Lügen Sie in den Kommentaren

Schreiben Sie in die Kommentare etwas rein, das überhaupt gar nichts mit dem darunter stehenden Code zu tun hat. Wenn Ihnen das zu aufwendig ist, »vergessen« Sie einfach, bei Codeänderungen den Kommentar entsprechend anzupassen. Der Effekt ist bereits nach kurzer Zeit der gleiche. Gehen Sie dafür im Gegenzug her und kommentieren Sie offensichtliche Tatsachen besonders ausführlich, zum Beispiel so:

```
i += 5; // addiere 5 auf i
```

Falls Sie einen Programmteil nicht mehr brauchen, dann kommentieren Sie ihn einfach aus. Lassen Sie ihn aber auf jeden Fall für immer im Quellcode drin, vielleicht braucht man ihn eines Tages wieder. Recycling tut bekanntlich Not. Wenn Sie das 1995 noch gebraucht haben, wieso sollte es heute falsch sein? Schließlich wollen Sie ja in der Zukunft das Rad nicht neu erfinden, behalten Sie daher den alten Code zur Sicherheit in Kommentaren. Kürzen Sie eine Funktion mit 120 Zeilen dadurch, dass Sie 115 Zeilen auskommentieren, aber noch 5 Zeilen in der Mitte der Funktion behalten. Dies erhöht die Übersichtlichkeit Ihrer Programme erheblich.

Verwenden Sie möglichst kurze Variablennamen

Nennen Sie grundsätzlich wichtige Variablen einfach nur `a`, `b` oder `kk`. Warum sollte eine Objektinstanz einer Datenbank nicht `d` heißen? Auf diese Weise beugt man schon Industriespionage vor, schließlich kann man nicht so leicht die Bedeutung erraten. Brechen Sie gleichzeitig diese Regel, indem Sie die normalerweise immer für `for`-Schleifen verwendeten Variablen `i`, `j` und `k` als Objektnamen für wichtige

Instanzen nehmen. Zum Beispiel wäre das Handle eines Fensters oder eine Datei doch ein gutes Beispiel, um das Objekt `i` zu taufen. Positiver Seiteneffekt dieser Regel ist, dass man nicht mehr einfach mit dem Editor nach den Instanzen suchen kann ... eine Suche nach `i` wird Abertausende Treffer erzeugen, aber nur wenige davon haben etwas mit dem Objekt zu tun.

Wenn Ihnen das zu trivial erscheint, wechseln Sie das Lager und verwenden Sie möglichst lange Variablennamen. Ideal ist es, wenn Sie hierbei im Namen Groß- und Kleinschreibung wechseln oder sich die langen Namen nur geringfügig unterscheiden. Auf den ersten Blick wird ein `DoCalculationOfMedians` sich nur wenig von einem `DocaculcationOfMedians` unterscheiden, aber machen Sie die Implementation grundverschieden, so dass der Aufrufer ständig nachsehen muss, welche Funktion nun die richtige war. Auch für Variablennamen ist das eine dankbare Spielwiese, speichern Sie alle wichtigen Daten in `Database`, den Rest in `DataBase`. C unterscheidet im Gegensatz zu Pascal schließlich Groß- und Kleinschreibung, also nutzen Sie diese neue Freiheit ausgiebig.

Nutzen Sie Copy&Paste ausgiebig

Moderne Editoren verfügen Gott sei Dank über die Copy&Paste-Funktion. Einfach einen Codebereich markieren und woanders hin kopieren. Verschwenden Sie Ihre intellektuellen Fähigkeiten niemals darauf, eine Funktion für eine mehrfach vorkommende Aufgabe zu erzeugen. Sondern kopieren Sie jedes Vorkommen einfach wieder. Sollte in dem Originalcode, der nun mehrfach kopiert wurde, noch ein Fehler sein, dann ist das eben Pech. Korrigieren Sie den Fehler von Hand in drei oder vier der gemachten Kopien, aber niemals in allen. Wenn Sie jemand auf diese Vorgehensweise anspricht, kontern Sie mit der gesparten Rechenzeit – jeder unnötige Funktionsaufruf kostet schließlich 0,3 Millisekunden Zeit.

Toppen Sie diese Vorgehensweise, indem Sie gleichzeitig auf die Nutzung der Template-Möglichkeiten von C++ verzichten. Schreiben Sie Klassen, die völlig identisch sind bis auf den verwalteten Grundtyp, und nutzen Sie dabei Copy&Paste, wo auch eine Template-Klasse möglich gewesen wäre. Weisen Sie in den Kommentaren keinesfalls darauf hin, dass es noch eine ähnliche Klasse wie diese gibt und daher alle Fehler mehrfach korrigiert werden müssen – das ist für andere Entwickler eine willkommene Detektivaufgabe, die von der langweiligen Entwicklungstätigkeit ablenkt. Dankbar werden Ihre Kollegen bemerken, dass ein Copy&Paste weniger Spuren hinterlässt als ein Einbruchsdiebstahl.

Seien Sie kreativ im Umgang mit dem Thesaurus

Sie wissen, was ein Thesaurus ist? Damit kann man synonyme Wörter zu einem Begriff finden. Besonders die englische Sprache ist da eine wahre Fundgrube, wo im Englischen doch sowohl keltische, romanische und angelsächsische Einflüsse zu finden sind – für fast jedes Wort gibt es dort noch zwei andere Wörter mit gleicher Bedeutung. Nennen Sie eine Methode einmal `showResults`, eine ähnliche Methode in einer anderen Klasse dann `displayValues`. Auch `presentResults` wäre eine nette Variante. Spielen Sie mit den sich daraus ergebenden Möglichkeiten konsequent. Im Umkehrschluss sollten Sie offensichtliche Bedeutungen vermeiden, verwenden Sie also `printValues` für eine Ausgabe auf dem Bildschirm und nehmen Sie dafür `showValues` als Methode, die die Werte auf dem Drucker ausdruckt.

Legen Sie sich niemals fest

Verschleiern Sie die Bedeutung von Parametern in Funktionen. Ein guter Weg ist einfach die Durchzählung der Parameter, so wie hier:

```
Book& searchTitle(int param1, int param2,
                  Book param3);
```

Innerhalb der Funktion wird man nur durch genaue Analyse des Codes erkennen können, dass hier in einem Intervall nach einem Buch gesucht wird. Legen Sie den Variablennamen auch immer so allgemein an, dass Sie den Typ austauschen können, ohne den Namen ändern zu müssen. Je genauer Sie sind, desto leichter kann man Ihnen Fehler nachweisen. Wem man Fehler nachweisen kann, der wird nie befördert.

So kann man ruckzuck aus

```
void addSomething(Book value);
```

ein

```
void addSomething(Author value);
```

machen, ohne dass man es gleich merkt. Wenn Sie jemand darauf anspricht, behaupten Sie, durch die kleinen Änderungen des Typs sind Sie schneller im Programmieren und Sie würden dadurch der Firma Geld sparen.

Richtig interessant wird es, wenn der Name der Funktion *so tut, als* würde der Wert des Parameters unverändert übernommen werden.

Ändern Sie trotzdem hemmungslos die Inhalte, aber verschleiern Sie dies durch einen anders klingenden Funktionsnamen.

Wenn's mit dem Englisch hapert

Schließlich sind wir als Deutsche ohnehin von Anglizismen überhäuft, wieso sollten wir hier mit dem Englischen schonend umgehen? Falls Ihnen ein englischer Begriff nicht geläufig ist, dann schlagen Sie diesen niemals im Wörterbuch nach. Verwenden Sie stattdessen doch einfach das deutsche Wort, am besten gepaart mit den restlichen Begriffen in Englisch. Jeder versteht doch, was mit `addNewFahrzeug` gemeint ist. Greifen Sie notfalls auf eine andere Ihnen bekannte Sprache zurück, falls Sie zum Beispiel im letzten Urlaub in Spanien waren und noch einige Brocken Kneipenspanisch beherrschen, so werfen Sie Ihre Bildung in die Waagschale und nennen die Funktion einfach `addNewCoche`.

Als Software-Entwickler verdient man ja häufig ganz gut und kann daher auch teurere Urlaube machen als ausgerechnet in Spanien. Zeigen Sie Ihre Welterfahrung ... raten Sie mal, wo der Entwickler der Klasse `Nangsue` zuletzt im Urlaub war?

Sollten Sie dagegen StarTrek-Fan sein, dann sind auch einige Brocken klingonisch immer wieder ein guter Scherz in Variablennamen.

Ziehen Sie Schreibfehler konsequent durch

Kreatives Schreiben kennen wir schon ... aber kennen Sie bereits kreatives Verschreiben? Wenn Sie in einem Klassennamen einen Tippfehler eingebaut haben, korrigieren Sie niemals den Klassennamen. Halten Sie stattdessen die falsche Schreibweise konsequent im ganzen Programm durch. Dies gewinnt an Wirksamkeit, wenn Sie diese Klasse möglichst oft brauchen, so dass jeder andere Anwender der Klasse zähneknirschend Ihre falsche Schreibung übernehmen muss. Besonders gut gefallen hat mir einmal in einem Programm die Schreibweise `Specturm` statt `Spectrum`. Das konnte man noch sehr schön auch in den Methoden `FindMaxInSpecturm` und `TransformSpecturm` sehen – der Autor hat für dieses Wort einen dauerhaften Knoten in den Fingern. Wozu gibt es schließlich zwei Gehirnhälften? Machen Sie ausgiebig Gebrauch davon.

Vermeiden Sie auch die Korrektur einfacher Buchstabendreher – wenn da steht `setAuthro` statt `setAuthor`, dann halten Sie das eisern und stoisch durch. Irgendwie hat so ein Buchstabendreher ja etwas Schicksalhaftes an sich, wer sind wir denn, dass wir so was korrigieren?

Seien Sie modern und geben Sie Ihren Quellcode frei

Seit dem Siegeszug von Linux und der damit bekannt gewordenen GPL – Quelltexte werden ebenso wie das Programm veröffentlicht und dürfen nach gewissen Regeln verändert und erweitert werden – hat fast jeder eine Homepage, auf der er seine privaten Programme auch veröffentlicht. Erkennen Sie die Zeichen der Zeit und stellen Sie Ihre Quellcodes auch ins Netz! Achten Sie dabei darauf, dass Sie eine Headerdatei MYDEFINITIONS.H überall inkludieren. Innerhalb von MYDEFINITIONS.H fügen Sie die Zeile

```
#include "c:\\develop\\includes\\defs.h"
```

ein. Liefern Sie nun diese Datei DEFS.H *auf keinen Fall* mit den Quelltexten mit. Vernachlässigen Sie dabei die Tatsache, dass Sie eigentlich die GPL verletzen. Besonders effektiv wird das, wenn in DEFS.H nur eine Kleinigkeit definiert wird, so dass der andere Anwender Ihr Programm *fast* übersetzen kann – aber nur fast. Besonders effektiv ist diese Methode, wenn Sie eine E-Mail-Adresse im Quelltext angeben, die Sie nicht mehr abfragen.

Finden Sie Workarounds für eigene Fehler

Falls Sie einmal einen Fehler in einer Klasse eingebaut haben, korrigieren Sie auf keinen Fall diesen Fehler. Sondern lösen Sie das in der Form, dass alle übergeordneten Klassen einen Workaround für dieses Problem einsetzen. Mir fällt da zum Beispiel eine Containerklasse ein, für die wir eine Methode brauchen, die eine Referenz auf das letzte Element zurückgibt. Wegen eines Fehlers gibt die Funktion nun aber eine Referenz auf das vorletzte Element zurück. Lösen Sie das so

```
Book& LastBook = Bookshelf.getLastBook();
LastBook = Bookshelf.getNext(LastBook);
```

Und zwar wie immer konsequent – rufen Sie bei jeder Verwendung von `getLastBook` eine Funktion auf, die noch eine Position weiterschaltet. Dies wird es sehr effektiv verhindern, dass jemand den Fehler jemals korrigieren kann, es sei denn, er will überall die obigen Zeilen ändern. Tut er es doch, wird er mit

großer Wahrscheinlichkeit ein nicht mehr lauffähiges Programm vorfinden. Gleichzeitig treiben Sie alle Leute in den Wahnsinn, die Ihre Methode `getLastBook` verwenden und sich darauf verlassen, dass Sie eine Referenz auf das letzte Buch erhalten. Wenn Sie schon Fehler machen, dann verschleiern Sie diese möglichst wirksam. Es ist eine Gnade, programmieren zu dürfen, kein Recht.

Verzichten Sie auf lesbare Codeformatierungen

Das Thema Einrückung in C ist sowieso nur Verursacher von Glaubenskriegen. Bevor Sie sich also festlegen, ob Sie

```
if (a == 0) {
   function();
}
```

oder

```
if (a == 0)
{
   function();
}
```

schreiben, mischen Sie das einfach völlig willkürlich ohne jede Systematik. Sie zeigen dadurch deutlich, dass Sie keiner der Glaubensrichtungen in C angehören, was sich für einen offenen Geist immer gut macht. Aber werden Sie nicht so neutral, Ihre Pascal-Wurzeln zu leugnen. Ein einfaches

```
#define Begin {
#define End }
```

wertet auch Ihr C deutlich auf – so schöne Konstrukte wie

```
if (a == 0)
Begin
   function();
End
```

werden dann möglich. Besonders ausbaufähig wird die Sache dann, wenn Sie sich eines Tages endlich an die geschweiften Klammern `{` und `}` gewöhnt haben – dann mischen Sie natürlich alte Bräuche mit neuen Erkenntnissen, zum Beispiel so:

```
if (a == 0)
Begin
   function();
End
else
{
   functionB();
}
```

Das alles muss ja weiterhin nicht dazu führen, dass Sie auf Funktionen verzichten, die `begin()` und `end()` heißen.

Einrückungen sind ohnehin was für Weicheier, rücken Sie je nach Lust des Tages ein, an Montagen und Donnerstagen in Monaten mit »r« drei Leerzeichen, an den anderen Tagen vier. Vor Wochenenden rücken Sie gar nicht mehr ein, weil Sie das am Montag nachholen. Schreiben Sie gleichzeitig kurze Abfragen in eine einzige Zeile – da die meisten Debugger zeilenorientiert arbeiten, kann in einer Zeile dieser Form

```
if (a == 0) functionA() else functionB();
```

der Breakpoint nicht auf einen der `if`-Zweige gesetzt werden. Man sollte ohnehin mit seinem Monitor angeben – wenn Sie auf einem Bildschirm mit 1600er-Auflösung arbeiten, dann machen Sie die Zeilen so lange wie möglich. Jemand mit einem normalen Monitor muss dann entweder eine 6-Punkt-Schrift nehmen, um die Zeile auf den Schirm zu quetschen, oder er muss in jeder Zeile scrollen. Dies trainiert entweder Hand oder Auge. Auch Ausdrucke auf A4-Papier werden dadurch zu grafischen Genüssen.