

## 19 Guter Code – ein Vorschlag

Es ist erstaunlich: Obwohl in sehr vielen Firmen in Microsoft Office VBA-Makros aufgezeichnet, erstellt und programmiert werden, wird man bei der Suche nach Programmierrichtlinien für VBA-Code im Internet fast nicht fündig.

Das folgende Kapitel fasst die wichtigsten „Unschönheiten“ zusammen und macht Vorschläge zu firmeninternen Programmierrichtlinien. Im Anschluss an jeden Absatz werden Gründe für die Einhaltung der Forderung genannt.

### 19.1 Code

#### 19.1.1 Kommentare konsequent

Kommentieren Sie Ihren Code konsequent. Schreiben Sie zu jeder verwendeten Variablen einen Kommentar über ihren Verwendungszweck. Schreiben Sie zu jeder Funktion und Prozedur eine kurze Erläuterung. Schreiben Sie in den „Einstiegspunkt“ des Programms, wer den Code geschrieben hat, wer der Auftraggeber ist, welche Version vorliegt, wann die letzten Änderungen vorgenommen wurden (auch bei Codeänderungen: Notieren Sie das Datum.). Schreiben Sie zu jedem Codeblock einen kurzen Kommentar, der die Funktion dieser Stelle erläutert.

**Begründung:** Häufig müssen Programme nachgebessert oder erweitert werden – manchmal wünscht der Kunde nach Jahren Korrekturen, weil sich firmeninterne Berechnungen, Abläufe oder andere Strukturen geändert haben. Kommentare erleichtern das Verständnis für eigenen oder fremden Code.

Ich habe schon ab und zu erlebt, dass Auftraggeber behaupteten, mein Programm würde falsch rechnen oder falsche Dinge tun. Als Beweis konnte ich meine Kommentare anführen, in denen ganz klar stand, wer wann was gewünscht hatte.

#### 19.1.2 Namen

Prozeduren, Funktionen, Variablen und Konstanten müssen mit einem Buchstaben beginnen. Sie dürfen Buchstaben, Ziffern und den Unterstrich enthalten. Dies schreibt VBA vor.

Alle selbst definierten Namen müssen aussagefähig sein. Verwenden Sie für Prozeduren das Schema SubstantivVerb oder VerbSubstantiv. Verwenden Sie CamelCasing, also die Verwendung von Groß- und Kleinschreibung, beispielsweise NamenEintragen, DateinamenAuslesen oder BenutzerInformationenAnzeigen.

Verwenden Sie keine Umlaute in Variablen, Modulnamen, Klassennamen ...

Benennen Sie Module mit dem Präfix „bas“, Klassen mit „cls“ und Formulare mit „frm“.

Verwenden Sie die Reddick-Konvention für Steuerelemente, Variablen und Konstanten.

**Begründung:** Mehrere Großbuchstaben in Prozedurnamen, Variablenamen und so weiter erleichtern das Lesen von Code. Wird eine Variable mit mehreren Großbuchstaben deklariert, dann wird sie, wenn sie bei der Verwendung kleingeschrieben wurde, automatisch in die richtige Schreibweise verändert. Dies erleichtert das Auffinden von Tippfehlern.

Auch wenn VBA-Code mit deutschen Umlauten in Frankreich und den USA läuft, musste ich einmal in einem Projekt mit vielen tausend Codezeilen mühevoll mit Suchen und Ersetzen nach deutschen Umlauten suchen, da auf asiatischen Rechnern aufgrund von Ersetzungsprogrammen „ä“, „ö“, „ü“ und „ß“ durch chinesische, koreanische, japanische ... Zeichen ersetzt werden.

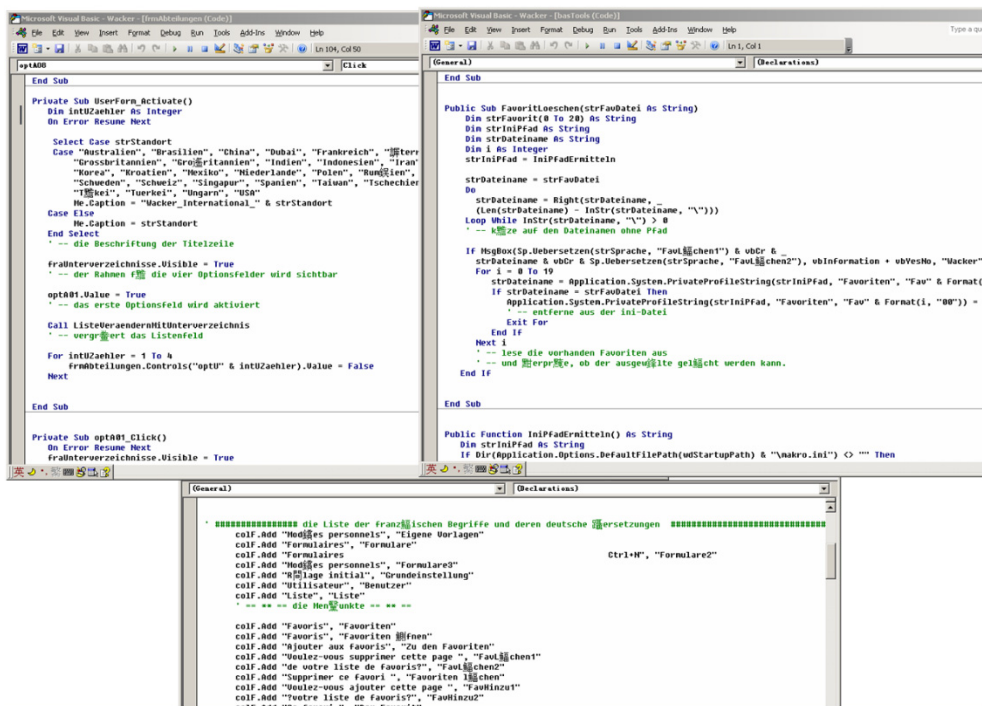


Abbildung 35: Asiatische Zeichenersetzungsprogramme können deutsche Umlaute „zerschießen“.

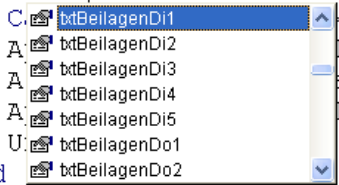
## 19. Guter Code – ein Vorschlag

Bei sehr vielen Steuerelementen erleichtert die Eingabe von

`Me.txt`

die Suche nach dem richtigen Textfeld. Ebenso können Sie auch „txt“ eingeben und sich die Auswahlliste mit [Strg]+[Leertaste] anzeigen lassen.

```
Private Sub cmdPlanerstellen_Click()  
    On Error Resume Next  
    me.tx|  
    C:\tdBeilagenDi1 kcel  
    A:\tdBeilagenDi2 lerts = False  
    A:\tdBeilagenDi3 s strPfad & "  
    A:\tdBeilagenDi4 lerts = True  
    U:\tdBeilagenDo1  
    End
```



**Abbildung 36: Die konsequente Verwendung von Konventionen erleichtert auch die Eingabe von Variablen und Objektnamen.**

### 19.1.3 Variablen, Variablennamen und ihre Deklaration

Verwenden Sie die Reddick- oder eine andere Konvention für die Namen Ihrer Variablen. Verwenden Sie sprechende Namen. Verwenden Sie nicht x, y, z, i, s oder ähnliche Namen für Variablen.

Kennzeichnen Sie globale Variablen durch das Präfix „g“, modulweit verwendete Variablen durch ein „m“.

Schreiben Sie Konstanten in Großbuchstaben.

Verwenden Sie interne VBA-Konstanten und nicht ihre dahinterliegenden Werte.

Verwenden Sie nicht eine Variable für verschiedene Aufgabenbereiche.

Deklarieren Sie jede Variable. Schreiben Sie

```
Option Explicit
```

an den Beginn jedes Moduls, jeder Klasse und jedes Formulars, beziehungsweise schalten Sie die Option „Variablendeklaration erforderlich“ ein.

In VBA ist es möglich, den Datentyp über das letzte Zeichen festzulegen, beispielsweise `Zuname$`, `Gehalt@` oder `ZeilenZaehler&`. Tun Sie das nicht.

Es gibt (fast) keinen Grund, den Datentyp `Variant` zu verwenden – vermeiden Sie ihn.

Es gibt keine Notwendigkeit, Zeichenketten mit fester Länge zu verwenden:

```
Dim strStrasse As String * 53
```

Unterlassen Sie dies in VBA!

Überprüfen Sie, ob alle deklarierten Variablen verwendet werden.

Umgekehrt: Initialisieren Sie jede Variable, bevor Sie sie verwenden.

**Begründung:** Eine deklarierte Variable

```
Dim i As Integer
```

```
Dim Projekte%
```

oder schlimmer

```
Dim x As Double
```

sagt im Code nichts über ihren Gültigkeitsbereich aus.

intMonatsZaehler, lngAnzahlZeilenVerkaufsdaten oder dblRohrinnenDurchmesser sagt alles.

```
If MsgBox("Möchten Sie die Datei wirklich löschen?", _  
    vbYesNo + vbDefaultButton2 + vbInformation) = vbYes Then
```

ist leichter zu lesen als:

```
If MsgBox("Möchten Sie die Datei wirklich löschen?", 324) = 6 Then
```

#### 19.1.4 Codezeilen

Verwenden Sie pro Zeile maximal einen Befehl. Der Lesbarkeit halber darf eine lange Codezeile mit „\_“ umbrochen werden.

Umgekehrt: Strukturieren Sie Ihr Programm durch Leerzeilen und Einrückungen.

**Begründung:** Spaghetticode

#### 19.1.5 Verkettungen

Für die Zeichenverkettung sollte „&“ anstelle von „+“ verwendet werden.

**Begründung:** Es gibt gemischte Ausdrücke, in denen Zahlen nicht korrekt in Text konvertiert werden. In der Zeile

```
xlZelle.FormulaR1C1 = "=SUBTOTAL(" & intTeilergebnisFunktion & _  
    ",R[-" & lngProjektZaehler & "]C:R[-1]C)"
```

wird deutlich, dass die Zahlen Teil der Funktion sind, dagegen führt

```
xlZelle.FormulaR1C1 = "=SUBTOTAL(" + intTeilergebnisFunktion + _  
    ",R[-" + lngProjektZaehler + "]C:R[-1]C)"
```

zu einem Fehler.

### 19.1.6 Verzweigungen

Schreiben Sie If-Verzweigungen mehrzeilig.

Verwenden Sie für mehrere Fälle `Select Case` anstelle von `If`. Bei Fällen, die sich gegenseitig ausschließen, sollten Sie `ElseIf` den Vorzug gegenüber

```
If ... Then
```

```
End If
```

```
If ... Then
```

```
End If
```

geben.

### 19.1.7 Rücken Sie konsequent ein!

Verschachteln Sie maximal vier Ebenen.

**Begründung:** Durch die mehrzeilige If-Anweisung ist das Programm leichter erweiterbar:

```
If Me.txtJahresgehalt.Value = "" Then MsgBox "Geben Sie einen Wert ein."
```

```
    Me.txtJahresgehalt.SetFocus
```

```
Exit Sub
```

wird immer abgebrochen und nicht nur, wenn kein Wert eingegeben wurde – ein typischer Fehler, der aufgrund einer nachträglichen Korrektur entstand.

```
If strGeschlecht = "m" Or strGeschlecht = "M" Or _
```

```
    strGeschlecht = "w" Or strGeschlecht = "W" Or _
```

```
    strGeschlecht = "f" Or strGeschlecht = "F" Then
```

ist länger und umständlicher als:

```
Select Case strGeschlecht
```

```
    Case "m", "M", "w", "W", "f", "F"
```

### 19.1.8 Schleifen

Verwenden Sie ganzzahlige Variablen (Byte, Integer oder Long) für Zählerschleifen.

Ziehen Sie `For Each` bei Iterationen über Objekten gegenüber Zählerschleifen vor.

Verwenden Sie nicht `While ... Wend`. Geben Sie `Do ... Loop` den Vorzug.

Rücken Sie sauber ein (siehe Verzweigungen).

Verschachteln Sie maximal vier Ebenen (siehe Verzweigungen).

**Begründung:** Gerne wird beim Löschen eines Objektes übersehen, dass nun der Index nicht mehr korrekt ist. Zwar könnte man statt:

```
For intZaehler = 0 To xmlKnoten.ChildNodes.Count - 1
    If xmlKnoten.ChildNodes(intZaehler).Name = "Mitglied" Then
        xmlKnoten.ChildNodes(intZaehler).Delete
' -- führt zu Fehler!
```

schreiben:

```
For intZaehler = xmlKnoten.ChildNodes.Count - 1 To 0 Step -1
    If xmlKnoten.ChildNodes(intZaehler).Name = "Mitglied" Then
        xmlKnoten.ChildNodes(intZaehler).Delete
```

Flexibler ist sicherlich:

```
For Each xmlKnoten In xmlWurzelknoten.ChildNodes
    If xmlKnoten.Name = "Mitglied" Then
        xmlKnoten.Delete
```

While ... Wend ist weniger flexibel als die vier Varianten von Do ... Loop.

### 19.1.9 Sprünge

Vermeiden Sie Exit For, Exit Do ebenso wie Exit Function oder Exit Sub.

Verwenden Sie unter keinen Umständen End.

Vermeiden Sie GoTo (außer bei Fehlern – dort gibt es keine Alternative).

**Begründung:** Spaghetticode

## 19.2 Programmierstil und Programmierstrategien

### 19.2.1 Fehler abfangen

Fragen Sie alle Fehlerquellen konsequent ab.

```
On Error Resume Next
```

ist sehr riskant und nur bedingt einsetzbar.

Erzeugen Sie keine Fehler mit Err.Raise.

Verwenden Sie nicht Fehlerroutrinen, um Voraussetzungen zu überprüfen.

Fangen Sie Fehler mit aussagekräftigen Meldungen ab. Verwenden Sie nicht vbCritical. Tippfehler in Fehlermeldungen sind peinlich. Verwenden Sie stets den internen Befehl Err.Description.

## 19. Guter Code – ein Vorschlag

---

In Fehlerbehandlungsprozeduren sollten Sie alle von Ihnen vorgenommenen Einstellungen zurücksetzen: geöffnete Programme schließen, Datenverbindungen beenden, Mauszeiger zurücksetzen ...

Überprüfen Sie stets Objekte auf `Empty` und `Null` (besonders in der Datenbankprogrammierung).

**Begründung:** Es muss sicherlich nicht erwähnt werden, dass Menschen, aber auch Maschinen, manchmal eigenartige Dinge tun. Darauf sollte stets reagiert werden. Das Minimum lautet:

```
On Error GoTo Fehler
```

```
...
```

```
Exit Sub
```

```
Fehler:
```

```
Msgbox "Es trat ein Fehler auf:" & vbCrLf & Err.Description
```

Anders als in anderen Programmiersprachen (VB.NET, C#, Java ...) liegt in VBA keine strukturierte Fehlerbehandlung vor. Ein

```
Err.Raise 13
```

(oder Ähnliches) ist immer mit einem „Restrisiko“ behaftet.

Wenn Sie wissen möchten, ob Visio geöffnet ist, dann bitte nicht so:

```
On Error GoTo Fehler
```

```
Set wdApp = GetObject(, "Visio.Application")
```

```
wdApp.Visible = True
```

```
...
```

```
Exit Sub
```

```
Fehler:
```

```
Set wdApp = CreateObject("Visio.Application")
```

```
Err.Clear
```

```
Resume Next
```

### 19.2.2 Prozeduren und Routinen, Module und Klassen

Vermeiden Sie Prozeduren in Modulen. Verwenden Sie Klassen.

Bestimmen Sie den Gültigkeitsbereich exakt: `Private` oder `Public`.

Lagern Sie mehrfach verwendeten Code in Funktionen oder Prozeduren aus.

Funktionen müssen immer einen Wert zurückgeben – tun sie dies nicht, so sind Prozeduren zu verwenden.

Parameter werden als Wert `byVal` und nicht als Referenz übergeben.

Überschreiten Sie nicht 100 Zeichen pro Zeile.

Überschreiten Sie nicht 100 Codezeilen pro Prozedur.

Überschreiten Sie nicht 50 Codezeilen pro Funktion.

Überschreiten Sie nicht 10 Prozeduren pro Modul.

Überschreiten Sie nicht 50 Eigenschaften, Prozeduren und Funktionen (Methoden) pro Klasse.

Alle Prozeduren verfügen über eine Fehlerbehandlung.

**Begründung:** Klassen sind flexibler und leichter skalierbar als Module.

Zu lange Funktionen und Methoden werden unübersichtlich. Umgekehrt ist ausgelagerter Code leichter zu warten als Codeschnipsel, die an mehreren Stellen gepflegt werden müssen.

### 19.2.3 Objekte

Verwenden Sie keine Standardeigenschaften von Objekten.

Verwenden Sie keine Objektkaskaden, sondern deklarieren Sie jedes einzelne Objekt.

Early Binding ist Late Binding vorzuziehen.

Leeren Sie am Ende des Programms alle verwendeten Objektvariablen.

**Begründung:** Sie wissen nicht, ob an die als Variant deklarierte Variable das Zellobjekt übergeben wird oder der Inhalt der Zelle.

```
Dim xlZelle As Range
```

```
Dim x
```

```
...
```

```
x = xlZelle
```

Codezeilen wie beispielsweise der folgende sind nicht nur schwer nachzuvollziehen, sondern gestatten keinerlei Zugriff auf die Datei oder das Tabellenblatt.

```
intZeilen = Application.ThisWorkbook.Worksheets(1). _  
    Cells(1, 1).CurrentRegion.Rows.Count
```

Early Binding erleichtert aufgrund der IntelliSense-Listen die Programmierarbeit.



## 19. Guter Code – ein Vorschlag

---

Der Garbage-Kollektor von Microsoft funktioniert in VBA nicht immer sauber – Sie sollten Objekte aus dem Arbeitsspeicher leeren, um sicher zu sein, dass der Arbeitsspeicher nicht unnötig belastet wird.

### 19.2.4 Programmierstil

Verzichten Sie auf knappen, „eleganten“ Programmierstil, wenn dies zu Lasten der Lesbarkeit geht. Schreiben Sie lieber ein paar Zeilen Code mehr, auch wenn dadurch möglicherweise einige Zeilen mehr abgearbeitet werden müssen.

**Begründung:** Versuchen Sie herauszufinden, was folgende Zeile berechnet:

```
dblWeihnachtsgeld = _  
    ((0.8 + Sgn(dblBetriebszugehoerigkeit - 15) * 0.05 _  
    + Sgn(dblGehalt - 4000) * 0.05)) * dblGehalt
```

Eine mehrzeilige If-Anweisung wäre hier sicherlich angebracht.

### 19.2.5 Strategien für die Benutzereingaben

In VBA erstellte Dialoge sollten sich an den Microsoft-Anwendungen orientieren. Verwenden Sie Steuerelemente im Sinne von MS Office, also Befehlsschaltflächen zur Aktivierung von Aktionen, Optionsfelder für Einzelauswahl, Kontrollkästchen für Mehrfachauswahl. Belegen Sie einen der Radiobuttons vor.

Benutzereingaben müssen komfortabel per Maus und per Tastatur zu erledigen sein. Mit Alt+[unterstrichener Buchstabe] sollte jedes Feld per Tastatur zu erreichen sein, die Tabulatortaste sollte eine logische Sprungreihenfolge erwirken.

Verwenden Sie das für Microsoft typisch Hintergrundgrau und keine bunten Farben. Wenn Sie etwas farblich hervorheben möchten, dann verwenden Sie lediglich eine Farbe. Diese wird auf allen Dialogen beibehalten.

Verwenden Sie keine hüpfenden, blinkenden oder sich bewegenden Steuerelemente. Verwenden Sie interne Dialoge, beispielsweise fürs Öffnen, zur Auswahl von Ordnern, Drucker, Schriften etc.

Verwenden Sie nur eine Schrift auf sämtlichen Dialogen. Verwenden Sie eine serifenlose Schrift. Verwenden Sie lediglich eine Schriftgröße. Heben Sie nur mit Fett hervor.

Die Leserichtung in unserer westlichen Welt ist von links nach rechts, von oben nach unten. Gestalten Sie entsprechend die Dialoge.

Überladen Sie Dialoge nicht mit Steuerelementen. Wenn Sie sehr viele Steuerelemente benötigen, programmieren Sie Assistenten, verwenden Sie mehrere Registerblätter oder mehrere Dialoge.

Verwenden Sie „dynamische Formulare“, um dem Benutzer zu helfen, das heißt: Deaktivieren oder verbergen Sie Buttons, damit der Benutzer in bestimmten Fällen keine falschen Angaben machen kann.

Gestalten Sie Masken ordentlich. Verwenden Sie die Ausrichten- und Verteilen-Funktionen, oder arbeiten Sie über die Eigenschaften Left, Top, Width und Height.

Elemente, die logisch oder funktional zusammengehören, sollten in einem Rahmen oder Rechteck visuell gruppiert werden.

Fangen Sie Fehler mit aussagekräftigen Meldungen ab. Verwenden Sie nicht `vbCritical`. Tippfehler in Fehlermeldungen sind peinlich. Verwenden Sie stets den internen Befehl `Err.Description`.

Für den Anwender zur Verfügung gestellte Funktionalitäten müssen transparent sein. Doppelklick auf eine Schaltfläche, gedrückte [Strg]-Taste bei der Texteingabe oder Tastenkombinationen sind nur als zusätzliche Optionen zu verwenden. Alles muss über sichtbare Menüs und/oder Schaltflächen erreichbar sein.

Hierzu zählen nach SNI-Style-Guide beziehungsweise die ISO-Norm 9241 die sieben Grundsätze der Dialoggestaltung.

Begründung: Ergonomie! Dem Anwender sind in der Regel Arbeitsweise, Struktur und Oberfläche von Microsoft schon bekannt und vertraut. Bei selbst erstellten Dialogen erleichtert es das Zurechtfinden, wenn Sie in Anlehnung an Microsoft-Dialoge gestalten.

Der Anwender findet sich auf unterschiedlichen Dialogen schnell zurecht, wenn sie alle gleichförmig gestaltet sind. Hierzu gehören Gestaltung, Schrift, Farben und Funktionalität.

Manche Benutzer arbeiten gerne mit der Maus arbeiten, andere lieber mit der Tastatur. Berücksichtigen Sie deshalb beide Aspekte.

Serifenlose Schriften können besser gelesen werden als Schriften mit Serifen.

Benutzer erschrecken leicht, wenn sie den roten Kreis mit einer Fehlermeldung erhalten. Deshalb sollten Fehlermeldungen dezent, aber aussagekräftig sein.

### 19.2.6 Strategien für die Datenausgabe

Bei der Datenausgabe sind stets sämtliche Voraussetzungen zu prüfen.

## 19. Guter Code – ein Vorschlag

---

Wählen Sie den kleinsten Datenspeicher, der für den entsprechenden Zweck ausreicht.

Speichern Sie nur dann in der Registry, wenn dies wirklich nötig ist.

Lang andauernde Speichervorgänge müssen dem Benutzer angezeigt werden: Schalten Sie den Cursor auf „Sanduhr“, schreiben Sie eine Information in die Statuszeile, zeigen Sie die Speicheraktionen als Hinweis auf dem Dialog an. Vergessen Sie nicht, diese Einstellungen nach erfolgreich durchgeführten Transaktionen wieder zurückzusetzen.

Bei sehr lang andauernden Vorgängen muss der Benutzer über den Fortschritt informiert werden, entweder mittels einer Fortschrittsanzeige oder einer Information, wie viel Prozent bereits abgearbeitet wurde.

**Begründung:** Um Grundeinstellungen des Benutzers in einer Datenbank oder einer Excel-Tabelle zu speichern, würde man mit Kanonen auf Spatzen schießen. Hierbei genügt eine Textdatei (auch XML- oder \*.ini-Datei).

### 19.2.7 Vor der Auslieferung

Testen Sie gut. Lassen Sie das Programm vor der Auslieferung eine Nacht lang liegen, und testen Sie es dann erneut. Lassen Sie andere Personen „gegentesten“. VBA bietet den Befehl `Debuggen | Kompilieren von Projekt an`, mit dem Codefehler aufgefunden werden können.

## 19.3 Fazit

Der vorliegende Katalog versteht sich als Empfehlung und Hilfestellung, nicht als dogmatisch anzuwendendes Konzept. Vielleicht erscheint die eine oder andere Richtlinie oder Forderung etwas überzogen. Zugegeben: In dem Beispielcode dieses Buches habe ich mich auch nicht 100 prozentig an die hier aufgestellten Richtlinien gehalten.

Sicherlich muss im einen oder anderen Fall auch mal gegen eine dieser Regeln verstoßen werden. Jedoch würde ich als EDV-Leiter die Messlatte lieber etwas höher als zu niedrig aufhängen – sie wird automatisch von den Programmierern, die damit arbeiten, nach unten korrigiert.