

Bernhard Wurm

# Programmieren lernen!

Schritt für Schritt zum ersten Programm



# Auf einen Blick

<b>1</b>	<b>Einführung .....</b>	<b>15</b>
<b>2</b>	<b>Algorithmisches Denken .....</b>	<b>31</b>
<b>3</b>	<b>Die Wahl der Programmiersprache .....</b>	<b>85</b>
<b>4</b>	<b>Vom Text zum Algorithmus .....</b>	<b>111</b>
<b>5</b>	<b>Erste Algorithmen .....</b>	<b>125</b>
<b>6</b>	<b>Erweiterte Konzepte .....</b>	<b>141</b>
<b>7</b>	<b>Notepad selbst gemacht .....</b>	<b>193</b>
<b>8</b>	<b>Gekonnt grafische Anwendungen erstellen .....</b>	<b>209</b>
<b>9</b>	<b>So gestalten Sie Benutzeroberflächen .....</b>	<b>239</b>
<b>10</b>	<b>Was ist noch hilfreich? .....</b>	<b>263</b>
<b>11</b>	<b>War das schon alles? .....</b>	<b>277</b>
<b>12</b>	<b>Ein Wort zum Schluss .....</b>	<b>291</b>
<b>A</b>	<b>Lösungen der Kontrollfragen und Aufgaben .....</b>	<b>293</b>
<b>B</b>	<b>Literaturverzeichnis .....</b>	<b>333</b>

# Inhalt

Wie Sie mit diesem Buch lernen .....	13
<b>1 Einführung</b> .....	<b>15</b>
1.1 Programmieren macht Spaß! .....	15
1.2 Was ist überhaupt Programmieren? .....	20
1.3 Welche Bereiche der Softwareentwicklung gibt es? .....	24
1.4 Zusammenfassung .....	28
1.5 Aufgabe .....	29
1.6 Kontrollfragen .....	30
<b>2 Algorithmisches Denken</b> .....	<b>31</b>
2.1 Computer sind dumm! .....	31
2.2 Algorithmische Formulierung .....	31
2.3 Einführung in Sprachelemente .....	35
2.3.1 Eine erste Algorithmus-Formulierung .....	36
2.3.2 Elemente eines Algorithmus .....	41
2.4 Zusammenfassung .....	81
2.5 Aufgabe .....	83
2.6 Kontrollfragen .....	83
<b>3 Die Wahl der Programmiersprache</b> .....	<b>85</b>
3.1 Kriterien zur Auswahl einer Programmiersprache .....	85
3.2 Syntaxvergleich .....	96
3.3 Einführung in Visual Studio .....	99
3.4 Arbeiten mit Visual Studio (Microsoft Visual C# Express Edition) .....	100
3.4.1 Debuggen .....	105
3.4.2 IntelliSense .....	107
3.5 Zusammenfassung .....	108
3.6 Aufgabe .....	109
3.7 Kontrollfragen .....	109
<b>4 Vom Text zum Algorithmus</b> .....	<b>111</b>
4.1 Vom Text zum Algorithmus .....	111
4.2 Schrittweise verfeinern .....	115
4.2.1 Die Idee .....	115
4.2.2 Ein Beispiel .....	116

4.2.3	Vorteile des Vorgehens .....	121
4.2.4	Nachteile des Vorgehens .....	122
4.2.5	Conclusio beim schrittweisen Vorgehen .....	122
4.3	Zusammenfassung .....	123
4.4	Aufgabe .....	123
4.5	Kontrollfragen .....	124
<b>5</b>	<b>Erste Algorithmen</b> .....	<b>125</b>
5.1	Hello World .....	125
5.2	Balkendiagramm .....	126
5.3	Zusammenfassung .....	136
5.4	Aufgaben .....	137
<b>6</b>	<b>Erweiterte Konzepte</b> .....	<b>141</b>
6.1	Enumerationen .....	142
6.2	Strukturen .....	144
6.3	Exceptions .....	145
6.4	Klassen .....	151
6.4.1	Namespaces .....	155
6.4.2	Klassenmethoden vs. Objektmethoden .....	158
6.4.3	Das Geheimnisprinzip .....	160
6.4.4	Eigenschaften .....	164
6.4.5	Der Konstruktor .....	168
6.5	Objektorientierte Programmierung .....	170
6.5.1	Vererbung .....	171
6.5.2	Polymorphismus .....	175
6.5.3	Abstraktion .....	177
6.5.4	Interfaces .....	186
6.6	Zusammenfassung .....	190
6.7	Aufgaben .....	191
6.8	Kontrollfragen .....	192
<b>7</b>	<b>Notepad selbst gemacht</b> .....	<b>193</b>
7.1	Einschub: Partielle Klassen .....	196
7.2	Die Gestaltung der Benutzeroberfläche .....	196
7.3	Programmieren der Funktionalität .....	201
7.3.1	Dateizugriff und Systemdialoge .....	201
7.4	Nachtrag: Ereignisse .....	203
7.4.1	Erstellen von Ereignissen .....	204

7.4.2	Auf Ereignisse reagieren .....	206
7.4.3	Ereignisse bei Windows Forms-Anwendungen .....	206
7.5	Zusammenfassung .....	207
7.6	Aufgabe .....	207
7.7	Kontrollfragen .....	208
<b>8</b>	<b>Gekannt grafische Anwendungen erstellen .....</b>	<b>209</b>
8.1	Das Storyboard .....	210
8.1.1	Gestaltung der MainForm .....	212
8.2	Der Prototyp .....	218
8.3	Programmieren der Funktionalität .....	219
8.3.1	Menü-Event-Handler implementieren .....	223
8.3.2	Datenhaltung .....	224
8.3.3	Aufgaben anlegen und bearbeiten .....	227
8.3.4	Darstellung der Aufgaben in der ListView .....	232
8.4	Zusammenfassung .....	237
8.5	Aufgabe .....	238
8.6	Kontrollfragen .....	238
<b>9</b>	<b>So gestalten Sie Benutzeroberflächen .....</b>	<b>239</b>
9.1	Gruppen von Benutzern – oder die Frage nach dem idealen Wetter ...	239
9.2	Steuerelemente und ihr Einsatz .....	241
9.2.1	Checkbox .....	241
9.2.2	CheckedListBox .....	242
9.2.3	Button .....	243
9.2.4	Textbox & Label .....	243
9.2.5	Treeview .....	244
9.2.6	Listview .....	245
9.2.7	Menü & Kontextmenü .....	247
9.2.8	Track Bar (auch Slider genannt) .....	249
9.2.9	Listbox & Combobox .....	249
9.3	Standarddialoge .....	250
9.3.1	Laden & Speichern .....	251
9.3.2	Drucken .....	253
9.3.3	Farbwähler .....	254
9.3.4	Ordner auswählen .....	255
9.3.5	Schriftart auswählen .....	256
9.4	Gutes Design .....	257
9.4.1	Automatisches Anpassen der Größe .....	258
9.5	Wie hilft die Entwicklungsumgebung? .....	258

9.6	Zusammenfassung .....	259
9.7	Aufgabe .....	260
9.8	Kontrollfragen .....	260
<b>10</b>	<b>Was ist noch hilfreich?</b> .....	<b>263</b>
10.1	Ein Blatt Papier und ein Bleistift .....	263
10.2	Repository .....	263
10.3	Unit-Tests .....	265
10.4	Freunde zum Testen .....	269
10.5	Open Source und die Community .....	270
10.6	Verwenden Sie Bestehendes .....	270
10.7	Planung .....	271
10.8	Zusammenfassung .....	273
10.9	Aufgabe .....	274
10.10	Kontrollfragen .....	275
<b>11</b>	<b>War das schon alles?</b> .....	<b>277</b>
11.1	Nullable-Typen .....	277
11.2	Das Schlüsselwort var .....	278
11.3	LINQ .....	279
11.4	Extension Methods .....	280
11.5	Lambda-Expressions .....	281
11.6	Anonyme Typen .....	283
11.7	Alternativen zu Arrays .....	284
	11.7.1 Liste .....	285
	11.7.2 HashSet .....	285
	11.7.3 SortedList .....	286
	11.7.4 Dictionary .....	286
	11.7.5 SortedDictionary .....	287
11.8	Zusammenfassung .....	288
11.9	Aufgaben .....	289
11.10	Kontrollfragen .....	290
<b>12</b>	<b>Ein Wort zum Schluss</b> .....	<b>291</b>

<b>A</b>	<b>Lösungen der Kontrollfragen und Aufgaben</b> .....	293
A.1	Kontrollfragen .....	293
A.2	Aufgaben .....	304
<b>B</b>	<b>Literaturverzeichnis</b> .....	333
	Index .....	335

*EDV-Systeme verarbeiten, womit sie gefüttert werden. Kommt Mist rein, kommt Mist raus.*  
– André Kostolany

## Wie Sie mit diesem Buch lernen

Mein Name ist Bernhard Wurm, und ich darf Sie sehr herzlich begrüßen. Sie wollen programmieren lernen? Ich darf Ihnen zu dieser herausfordernden Entscheidung gratulieren. Sie begeben sich auf einen etwas steinigen, aber sehr schönen Weg. Wie heißt es so schön: »Aller Anfang ist schwer.« Ich will Ihnen mit diesem Buch in elf Schritten oder besser gesagt elf Kapiteln dabei behilflich sein, die Hürden zu nehmen. Das Buch soll Sie dabei unterstützen, das Problem zu lösen, mit dem die meisten Personen kämpfen, die mit der Programmierung beginnen, und an dem sie am ehesten scheitern. Es handelt sich dabei, auf den Punkt gebracht, um das »algorithmische Denken«, also darum, wie Sie von einer Idee zu einem Programm kommen. Dabei spielen natürlich Elemente wie die Programmiersprache, die Idee, die Syntax und viele andere Punkte eine wesentliche Rolle. All diese Punkte werden beleuchtet, und Sie werden anhand praktischer kleiner Beispiele an das Thema herangeführt. Nachdem Sie in Kapitel 1 eine Einführung in die Welt der Softwareentwicklung erhalten und etwas Übersicht gewonnen haben, wird in Kapitel 2, »Algorithmisches Denken«, das analytische Denken für diesen Bereich geschärft. Sie werden mit verschiedenen Syntaxelementen einer Programmiersprache vertraut gemacht. In Kapitel 3, »Die Wahl der Programmiersprache«, gehe ich auf verschiedene Programmiersprachen ein, und eine Kurzeinführung in *Microsoft Visual C# Express Edition* wird Sie mit der Entwicklungsumgebung bekannt machen. Das sich anschließende Kapitel 4, »Vom Text zum Algorithmus«, gibt Ihnen Hilfestellung zu der Frage, wie Sie von einer textuellen Beschreibung ausgehend einen Algorithmus entwickeln können. Mit Kapitel 5, »Erste Algorithmen«, werden Sie selbst geführt Algorithmen entwickeln. Nach diesem großen Schritt erläutere ich weitere Konzepte, die für die Entwicklung von Computerprogrammen unabdingbar sind. Die Konzepte, die ich Ihnen in Kapitel 6, »Erweiterte Konzepte«, vorstelle, behandeln unter anderem Fehlerbehandlung, Namespaces, Klassen und objektorientierte Konzepte. Mit den Kapiteln 7, »Notepad selbst gemacht«, und 8, »Gekonnt grafische Anwendungen erstellen«, können Sie Ihre ersten grafischen Windows-Anwendungen program-

mieren, und in Kapitel 9, »Wie gestalte ich die Benutzeroberfläche?«, unterstütze ich Sie mit Hinweisen zur Gestaltung der Benutzeroberfläche, sodass sich ein Benutzer sehr einfach in Ihren Programmen zurechtfinden kann. Tipps aus der Praxis zur Softwareentwicklung will ich Ihnen dann in Kapitel 10, »Was ist noch hilfreich?«, ans Herz legen, und da dieses Buch natürlich bei Weitem nicht die gesamte Softwareentwicklung und auch nicht den gesamten Sprachumfang von C# abdeckt und abdecken kann, gibt Ihnen Kapitel 11, »War das schon alles?«, einen Ausblick auf Sprachelemente, die noch nicht betrachtet wurden, Ihnen aber das Leben als Softwareentwickler noch leichter machen können. Nach der Lektüre dieser Kapitel haben Sie ein solides Fundament von Techniken und eine Vorgehensweise, die Sie dabei unterstützt, Ihre Software erfolgreich zu entwickeln.

Sämtliche Kapitel sind nach folgendem Schema aufgebaut: Als Erstes gebe ich Ihnen einen kurzen Überblick über den Inhalt des Kapitels. Nachdem das Kapitel vollständig abgehandelt wurde, wird es mit einer Zusammenfassung abgerundet. Da Sie das Programmieren nur durch programmieren lernen werden, enthält jedes Kapitel ein paar Aufgaben. Ich empfehle Ihnen, sie durchzuarbeiten. Damit Sie sich selbst kontrollieren können, habe ich an das Ende jedes Kapitels Kontrollfragen gestellt, die die Inhalte des Kapitels abfragen.

Sowohl die Antworten auf die Kontrollfragen als auch Musterlösungen für die Aufgaben finden Sie im Anhang des Buches. Das Literaturverzeichnis am Ende des Anhangs hält einige Empfehlungen bereit, die Ihnen auf Ihrem zukünftigen Weg noch behilflich sein können.

Natürlich müssen Sie das Buch nicht vollständig von vorn bis hinten durcharbeiten. Es steht Ihnen frei, die Reihenfolge der Kapitel zu ändern, einzelne Kapitel zu überspringen oder Ähnliches. Die Kapitel bauen jedoch aufeinander auf, und daher möchte ich Ihnen empfehlen, das Buch in der angeführten Reihenfolge durchzuarbeiten (ja, *durchzuarbeiten*, denn das Buch nur zu lesen ist wahrscheinlich zu wenig).

Ich wünsche Ihnen nun viel Spaß mit diesem Buch und viel Erfolg bei der Entwicklung Ihrer Computerprogramme!

Peilstein,

**Bernhard Wurm**

»Wenn ich die Folgen geahnt hätte, wäre ich Uhrmacher geworden.«  
– Albert Einstein

# 1 Einführung

Dieses Kapitel bereitet Sie auf den Rest des Buches vor. Es stellt das Thema Softwareentwicklung und dessen Umfeld vor. Es gibt einen Überblick, wie groß dieses Thema ist, auf welchen Weg wir uns in diesem Buch begeben und was alles dazu benötigt wird. Programmieren ist kein triviales Thema. Dieses Buch legt jedoch den ersten Stein des Fundaments, das für einen Programmierer unabdingbar ist. Dieses Fundament wird in den nachfolgenden Kapiteln verstärkt und ausgebaut.

Das erste Kapitel hilft Ihnen dabei, zu errahnen, womit Sie konfrontiert werden, und soll Sie gleichzeitig in eine Richtung weisen, die Sie gern beschreiten möchten. Ich erläutere Grundbegriffe, die Sie als Entwickler kennen müssen, da Sie im täglichen Leben immer wieder damit konfrontiert werden, und ich stelle verschiedene Sparten der Softwareentwicklung mit ihren Vorzügen und Nachteilen vor.

## 1.1 Programmieren macht Spaß!

Bestimmt kennen Sie das Vorurteil, dass Programmierer nur im dunklen Kämmerlein hocken und absolut unsozial sind. Ich bin mir sicher, dass sich beim Lesen dieses Satzes ein entsprechendes Bild in Ihrem Kopf gebildet hat. Obgleich es derartige Programmierer auch geben wird, sind sie bestimmt die Ausnahme.

Sie haben auch mit Sicherheit schon davon gehört oder einmal daran gedacht, dass die Tätigkeit als Softwareentwickler bedeutet, den ganzen Tag vor dem Computer zu sitzen, was extrem langweilig sein muss.

Aber ich sage Ihnen: »Programmieren macht Spaß!« – und ich werde Ihnen auch nicht vorenthalten, warum:

- ▶ *Helpen Sie anderen Personen.* Es gibt nur einen Grund, ein Computerprogramm zu schreiben – um ein Problem zu lösen. Mit einem Computerpro-

gramm lösen Sie ein ganz bestimmtes Problem: zum Beispiel die Berechnung einer Primzahl. Ihr Programm könnte auch Kunden helfen, ihre Bestellung aufzugeben, oder man könnte mit ihm bestimmen, ob ein bestimmtes Produkt auch wirklich gefertigt werden kann. Meist lösen Sie als Programmierer nicht ein Problem, das Sie selbst haben, sondern das ein Kunde hat. Sie programmieren Auswertungen von Daten, die ein Mensch manuell nicht tätigen kann. Sie optimieren oder visualisieren Prozesse eines Unternehmens, um Schwachstellen aufzudecken oder um die Durchlaufzeit in einem Prozess zu verringern und somit Kosten zu sparen. Mit jedem Programm, das Sie schreiben, lösen Sie ein Problem, das ein Mensch bisher hatte.

- ▶ *Kein Problem gleicht dem anderen!* Als Programmierer haben Sie sehr viel Abwechslung im Job. Sie lösen Probleme. Probleme von Kunden. Und da kein Mensch und keine Firma wie der bzw. die andere ist, ist auch kein Problem wie das andere. Sie werden täglich mit neuen Herausforderungen und Kundenwünschen konfrontiert, und Sie werden mit diesen Herausforderungen wachsen. Obwohl es so viele Standardprogramme gibt, werden immer wieder neue Softwareprodukte programmiert, da viele Firmen eben nicht »Standard« sind und eigene, spezielle Lösungen benötigen. Immer wieder höre ich von Kunden auf die Frage, warum sie hierfür keine Standardlösung verwenden oder zur Not eine solche adaptieren: »Wir sind nicht Standard, bei uns ist alles viel zu speziell, um diese Anforderungen mit einem Standardprodukt abdecken zu können.« Sie können solche neuen Herausforderungen nutzen, um sich selbst zu beweisen, an sich selbst zu arbeiten und um selbst kreativ zu sein.
- ▶ *Kreativität ist gefragt.* Softwareentwicklung ist, auch wenn viele Nicht-Softwareentwickler ein anderes Bild im Kopf haben, eine sehr kreative Tätigkeit. Durch die ständige Neuartigkeit der Anforderungen an die Software und somit an Sie, ist Ihre Kreativität gefordert. Sie müssen sich auf Lösungssuche begeben. Das Suchen bzw. Erarbeiten einer adäquaten Lösung ist eine Aufgabe, die Ihre Kreativität fordert und fördert! Wenn Sie Software schreiben und stundenlang über einer Aufgabe oder einem Fehler in der Software brüten, werden Sie sich wohl so manches Mal wundern, wie andere Personen hinter einen solchen Fehler kommen. Das kann ich Ihnen verraten: Auch andere, erfahrene Programmierer haben über diesem oder einem ähnlichen Fehler schon gebrütet. Wahrscheinlich haben sie dafür genauso lange gebraucht wie Sie. Aber wenn Sie die Lösung gefunden haben, haben Sie sich den Lösungsweg so gut eingepägt, dass Sie einen ähnlichen Fehler in Zukunft in einem Bruchteil der Zeit finden werden – oder idealerweise gar nicht erst machen werden. Das heißt, zu Beginn verhalten Sie sich problemlösend. Beim zweiten Mal verhalten Sie sich routiniert. So erfordert der Weg der Problemlösung Abstraktion und Kreativität.

- ▶ *Sie lernen ständig neue Technologien kennen.* Die Welt der Softwareentwicklung ist wohl die am schnellsten voranschreitende Branche. Daher – und da keine Anforderung der anderen gleicht – werden Sie ständig mit neuen Technologien konfrontiert. Sie haben ständig die Möglichkeit, sich weiterzubilden. Es macht Spaß, sich mit neuen Technologien zu beschäftigen. Sie werden Freude verspüren, wenn Sie durch neue Technologien Herausforderungen viel einfacher, schneller und eleganter lösen können als bisher.
- ▶ *Sie werden nicht den ganzen Tag vor dem Computer sitzen.* Auch mit diesem Gerücht möchte ich aufräumen! Programmieren heißt nicht nur, dass Sie den ganzen Tag vor dem Computer sitzen! Programmieren heißt nicht, dass Sie allein mit Ihrem Computer und von der Welt abgeschnitten sind. Programmieren ist Team-Arbeit! Sie werden mit Kunden deren Probleme besprechen und Ihre erarbeiteten Lösungen präsentieren. Sie werden Papier und Bleistift benötigen, um Konzepte oder Skizzen von Benutzeroberflächen für Ihr Programm zu entwickeln. Sie werden mit Ihren Team-Kollegen Besprechungen abhalten, um den Code von Ihnen oder von Kollegen zu besprechen. Sie werden Fehler, neue Lösungsansätze und neue Technologien im Team durcharbeiten. Sie werden viel mit Ihren Kollegen arbeiten! Sie werden recherchieren, in Büchern und im Internet. Sie werden auf Konferenzen fahren und Schulungen belegen oder halten. Aber natürlich werden Sie auch vor dem PC sitzen, um Programme zu schreiben.

Das sind einige Punkte, die Sie als Softwareentwickler motivieren, Sie vorantreiben und immer wieder bestätigen werden. Jedoch ist das Programmieren nicht nur Spaß. Programmieren hat wie alles andere im Leben zwei Seiten. Programmieren ist auch Arbeit, und es wäre unfair, Ihnen nicht auch die Kehrseite der Medaille vor Augen zu führen. Immerhin gibt es lediglich drei Motive, die Sie bewegen, dieses Buch zu lesen:

- ▶ Sie haben noch nie programmiert und möchten in das Thema einsteigen.
- ▶ Sie haben erste kleine Schritte getan, sind jedoch (mit anderen Büchern oder dergleichen) noch auf keinen grünen Zweig gekommen.
- ▶ Sie sind bereits im IT-Bereich tätig und möchten auch einmal Einblick in die Entwickler-Seite der Branche erhalten.

Daher folgen hier, wie versprochen, einige Worte zur »dunklen Seite« der Softwareentwicklung:

- ▶ *Der Weg ist das Ziel.* Wie ich weiter oben gesagt habe, werden Sie sich in dieser Branche – wohl schneller und häufiger als in einer anderen Branche – mit neuen Technologien auseinandersetzen müssen. Dieser Prozess wiederholt sich immer wieder. Der Punkt, an dem Sie »Ich weiß jetzt alles« sagen, kann

nicht erreicht werden. Sie müssen immer wieder Neues lernen und können sich nie auf Ihren Lorbeeren ausruhen. Wahrscheinlich schreitet die Entwicklung der Branche viel schneller voran, als Sie nachlernen können. Das wird Sie dazu zwingen, sich auf entsprechende Bereiche zu spezialisieren und in diesen Bereichen top zu sein.

- ▶ *Perfektion gibt es nicht.* Ich habe noch nie einen Softwareentwickler getroffen, der mir bestätigt hätte, dass ein Programm zu seiner vollen Zufriedenheit ist. Selbst wenn Software ausgeliefert und als fertiggestellt deklariert wurde, können Softwareentwickler eine Unzahl von Dingen aufzählen, die sie beim nächsten Mal anders machen würden. Dies hängt unter anderem mit dem Lernprozess zusammen. Sie verlassen jedes Projekt mit viel mehr Wissen, als Sie zu Beginn des Projekts hatten. Sie gewinnen Erfahrung, und beim nächsten Projekt setzen Sie diese Erfahrung um. Möglicherweise waren Sie sich während der Entwicklung bereits über den einen oder anderen Knackpunkt im Klaren, und Sie haben das Problem aufgrund von Zeit und Kostendruck dennoch nicht optimal gelöst. Wird absolute Perfektion angestrebt, so wird das Projekt wahrscheinlich nie fertig. Daher kommt der Spruch »Software wird nicht released, sie entkommt«.
- ▶ *Softwareentwickler sind immer im Stress.* Immer, wenn ich mit einem Kunden bezüglich eines neuen Features oder eines neuen Entwicklungsprojekts im Gespräch bin, frage ich ihn, nachdem er mir das Problem geschildert hat, bis wann er die Lösung braucht. Die Antwort, dass noch Zeit sei, kommt leider sehr selten. Meist bekomme ich als Antwort ein Lächeln und das Wörtchen »gestern«. Wer ein Haus im klassischen Stil baut, geht nicht davon aus, dass dies in drei Tagen erledigt ist. Bei Software denken dies jedoch noch viele Personen.
- ▶ *Freud und Leid können nahe beieinander liegen.* Obgleich die Entwicklung von neuen Funktionen und Programmen sehr viel Spaß machen kann, kann die Fehlersuche äußerst langweilig und langwierig sein. Sie werden viel Zeit damit verbringen, nach Fehlern zu suchen. Das betrifft sowohl Fehler in Programmen, die Sie selbst geschrieben und zu verantworten haben, als auch Fehler, die Kollegen verursacht haben. Dabei haben Sie noch Glück, wenn Sie den Fehler vor dem Kunden finden und ihn im besten Fall bereits behoben haben, bevor der Fehler beim Kunden auftreten kann – was wieder mit dem oben erwähnten Stress zu tun hat.

Nach den angeführten Schattenseiten hoffe ich, Sie haben noch Interesse an dem Thema. Aber da Sie sich die Mühe gemacht haben, das Buch auszuwählen, zu erwerben und (bis hier) zu lesen, gehe ich davon aus, dass schon mehr passieren muss, bevor Sie die Flinte ins Korn werfen.

Ich habe dieses Buch geschrieben, damit Sie möglichst einfach programmieren lernen. Ich werde immer wieder Tipps und Erfahrungen aus der Praxis anführen. Und daher will ich auch zu diesen Punkten meine Tipps und bildlichen Vergleiche vortragen. Als Erstes möchte ich Ihnen einen Vergleich vorstellen, der Ihnen helfen soll, den Spaßfaktor möglichst maximieren und den Frust auf ein Minimum zu reduzieren.

### **Wo Sie Hilfe bekommen**

Wissen Sie noch, wie Sie Fahrrad fahren lernten? Sie haben sich auf das Fahrrad gesetzt, und Ihre Eltern und Verwandten haben für Ihr Gleichgewicht am Stützrad gesorgt und Sie angeschoben. Nach einer Weile konnten Sie selbst in die Pedale treten und gewannen Selbstvertrauen in Ihre Fahrradkünste. Es dauerte eine Weile, bis Sie das Gleichgewicht auch ohne Stützräder halten konnten und endlich selbstständig Fahrrad fahren konnten. Das letzte Hindernis (zumindest war es bei mir so) war der Umstieg von der Rücktrittbremse auf die normalen Bremsgriffe. Ich habe mir einige Schrammen geholt.

Beim Programmieren ist es ähnlich. Vor allem zu Beginn ist es schwierig. Sie werden Fehler machen (nicht nur zu Beginn), und das Programm wird nicht tun, was Sie eigentlich von ihm verlangen. Wenden Sie sich mit Ihren Fragen und Problemen an Foren. Viele kompetente Programmierer werden Ihnen dort mit Rat und Tat zur Seite stehen. Nutzen Sie ein Forum, um nicht für alle Fehler »bluten« zu müssen. Nutzen Sie Foren, um die Sicherheit zu finden, die Sie bekommen haben, als Ihre Eltern Sie mit den Stützrädern geschoben haben. Sie können durch Foren viel lernen und auch anderen Personen helfen.

Sie werden sich in Geduld üben müssen, denn programmieren lernt man nun leider nur durch die Tätigkeit selbst. Wenn Sie nur Programmiercodes ansehen und zu verstehen versuchen, werden Sie zwar einen Einblick gewinnen, doch wenn Sie selbst nicht vor der Tastatur sitzen und versuchen, ein Programm zu schreiben, werden Sie nie selbstständig ein Problem lösen können. Genauso wie Sie das Programmieren nur in der Praxis lernen, lernen Sie auch die Fehlerbehebung und -vermeidung nur, indem Sie selbst Fehler machen und diese beheben. Die Fehlersuche ist jedoch im Lernprozess inbegriffen; Sie bekommen also zwei Schritte zur Erkenntnis zum Preis von einem – allerdings nicht ganz.

In einem Programm werden immer wieder Fehler auftreten. Erwarten Sie niemals von Beginn an ein vollständig fehlerfreies Programm von sich selbst oder von Ihren Kollegen. Seien Sie sich dessen immer bewusst! Ich kann Ihnen jedoch einen Tipp geben, der Ihnen ungemein bei der Fehlersuche und bei der Suche nach einer Lösung helfen kann: Fragen Sie einen Freund oder Kollegen, ob dieser kurz für Sie Zeit hat, und erklären Sie ihm, was das Programm genau macht oder

machen soll. Gehen Sie dabei den Programmcode Zeile für Zeile durch. Sie werden durch diese Methode sehr häufig den Fehler finden – und zwar ohne Zutun ihres Freundes oder Kollegen. Dabei ist es prinzipiell irrelevant, ob der Freund oder Kollege programmieren kann oder nicht, wobei es für ihn natürlich spannender ist (und auch für Sie), wenn er selbst programmieren kann. Dieser Tipp funktioniert wunderbar, und ich wende diese Methode selbst immer wieder mit einem Kollegen an. Diese Methode hilft, weil Sie durch das Erklären des Programms den ganzen Ablauf erneut durchgehen und jede Entscheidung, die Sie beim Schreiben des Programms getroffen haben, erneut hinterfragen müssen. In schlaun Büchern wird diese Methode als *Phantominspektor* bezeichnet. Versuchen Sie es – oftmals wird es Ihnen wie Schuppen von den Augen fallen, und Sie werden sich gar nicht erklären können, warum Sie es nicht bereits von Anfang an so gemacht haben.

## 1.2 Was ist überhaupt Programmieren?

Nachdem Sie jetzt wissen, worauf Sie sich einlassen, möchte ich kurz erläutern, was Programmieren eigentlich ist und was es nicht ist und warum es darauf ankommt, was Sie programmieren wollen.

Wenn Sie einen Text schreiben (eine E-Mail, einen Brief oder ein Buch), so schreiben Sie ihn in einer Sprache wie z. B. Deutsch oder Englisch. Dabei bestimmt die Grammatik der Sprache, wie welche Wörter aufeinander folgen dürfen, woraus ein Satz bestehen muss und so weiter. Die Grammatik bestimmt also die Möglichkeiten, die Sie haben. Die Grammatik ist ein Baukasten mit Regeln, den Sie verwenden, um Ihre Texte zu formulieren.

Wenn Sie ein Computerprogramm schreiben, schreiben Sie auch dieses in einer Sprache – in einer Programmiersprache. Diese Programmiersprache ist beinahe genauso weit von den Nullen und Einsen des Binärsystems entfernt wie die natürliche Sprache von den elektrischen Impulsen in den Synapsen unseres Gehirns. Die Grammatik einer Programmiersprache wird *Syntax* genannt, nach dem Begriff für den Satzbau. Sie Syntax beschreibt, wie die Sprache aufgebaut ist. Die Syntax bestimmt, was gültig ist und was nicht, welche Zeichen Sie benötigen und welche nicht verwendet werden dürfen. Diese Syntax ist viel eingeschränkter als die Grammatik der deutschen Sprache. Somit ist eine Programmiersprache auch leichter zu erlernen als eine natürliche Sprache. Eine Programmiersprache hat auch den Vorteil, dass sie lediglich geschrieben und nicht gesprochen existiert. Des Weiteren ist eine Programmiersprache im Gegensatz zu einer natürlichen Sprache eindeutig. Es gibt keine Ironie oder Mehrdeutigkeiten. Wie die Grammatik bei der natürlichen Sprache bestimmt die Syntax bei der Programmiersprache,

welches Konstrukt Ihnen zur Verfügung steht, um Ihre Programme zu schreiben. Wie in der natürlichen Sprache sind Sie gefordert, aus den einzelnen »Wörtern« entsprechende »Sätze« zu bilden.

### Definition »Syntax«

Die Syntax definiert, wie Sprachelemente zusammenhängen und verwendet werden dürfen.

Erst die Zusammenstellung der Wörter zu sinnvollen Sätzen und die Zusammenfassung von Sätzen zu einzelnen Abschnitten und Kapiteln verleihen den einzelnen Wörtern Sinn. Diese Zusammenfassung ist beim Programmieren das Schreiben eines *Algorithmus*. Der Algorithmus besteht aus vielen einzelnen Anweisungen (zum Beispiel »Gib einen Text aus«) und gibt dem Ganzen somit eine *Semantik*. Die Semantik ist also die Bedeutung des Programms. Dieser Zusammenhang ist in Abbildung 1.1 schematisch dargestellt.

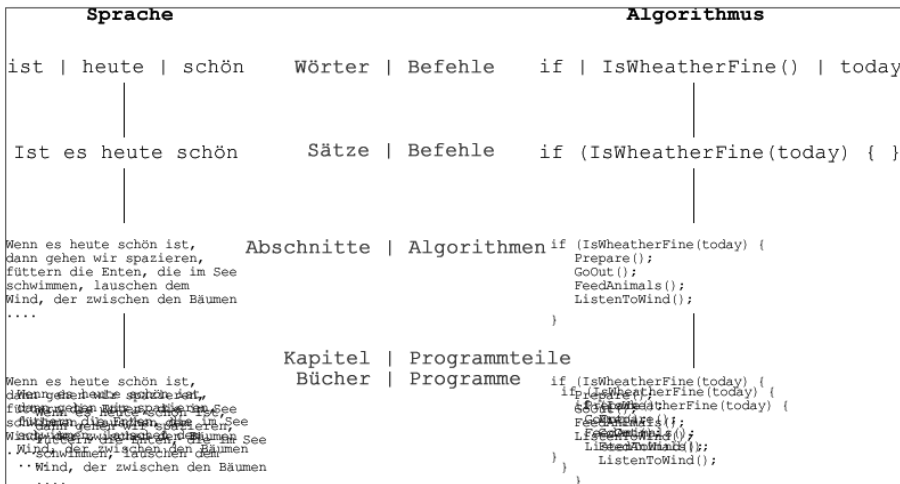


Abbildung 1.1 Schematische Darstellung von Semantik

### Definition »Algorithmus«

Ein Algorithmus ist eine Reihe von zusammengehörigen Befehlen, die einen bestimmten Zweck erfüllen.

Das klingt jetzt vielleicht etwas theoretisch. Die Begriffe werden Sie jedoch sehr oft hören und lesen, und daher ist es wichtig, dass Sie auch wissen, was die einzelnen Begriffe bedeuten. Wenn Ihr Programm einen Syntaxfehler liefert, dann wissen Sie, dass »die Satzstellung« nicht passt oder dass Sie ein »Wort« verwen-

den, das nicht definiert ist. Bei einem semantischen Fehler handelt es sich umgangssprachlich um einen Denkfehler. Syntaxfehler sind einfacher zu finden, da Sie auf den Fehler aufmerksam gemacht werden. Semantische Fehler stellen Sie fest, wenn das Programm nicht das macht, was Sie eigentlich erwarten. Um das Ganze mit Syntax und Semantik etwas zu verinnerlichen und zu veranschaulichen, hier ein kleines Beispiel:

- ▶ Der Satz »Heute ist ein schöner Tag.« ist ein korrekter Satz. Sowohl syntaktisch als auch semantisch ist er in Ordnung.
- ▶ Der Satz »Vorgestern ist morgen heute.« ist grammatikalisch korrekt, allerdings kann vorgestern unmöglich der gleiche Tag wie morgen sein. Es handelt sich hier um einen semantischen Fehler.
- ▶ Der Satz »Ich gehen nach Hause.« ist grammatikalisch falsch (»Ich gehe nach Hause.« wäre richtig). Somit handelt es sich um einen syntaktischen Fehler.

#### Definition »Semantik«

Die Semantik gibt dem Programm die Bedeutung.

Obwohl eine Programmiersprache nun bei Weitem nicht so komplex wie eine natürliche Sprache ist, ist auch die Programmiersprache für einen Computer *nicht* ohne Weiteres verständlich. Ihr Programm besteht ausschließlich aus Befehlen. (Ja, als Programmierer sind Sie stets Befehlsgeber.) Diese Befehle müssen dem Computer jedoch verständlich gemacht werden. Dies macht der sogenannte *Compiler*. Der Compiler ist also der Übersetzer von der Programmiersprache in die Maschinensprache, die der Computer nun endlich ausführen kann. Es gibt hier zwar auch einige Detailgrade und »Zwischensprachen«, darüber müssen Sie sich jedoch aktuell keine Gedanken machen. Der Compiler kennt also sowohl die Maschinensprache als auch die Programmiersprache. Da ein Compiler genau eine Programmiersprache kennt, ist für jede Programmiersprache ein anderer Compiler nötig. Die Maschinensprache ist weitgehend gleich, doch Programmiersprachen gibt es viele verschiedene (dazu erfahren Sie später in Kapitel 3, »Die Wahl der Programmiersprache«, mehr). Wenn Sie sich nun die Frage stellen, warum Sie nicht gleich Ihr Programm in Maschinensprache schreiben, so sollten Sie diesen Gedanken gleich wieder verdrängen. Es ist für einen Menschen kaum mehr möglich, adäquate Programme in Maschinensprache zu formulieren. Diese ist nicht programmiererfreundlich.

Jede Programmiersprache hat ihren eigenen Compiler. Da der Compiler die Programmiersprache genau kennen muss, nimmt dieser auch eine syntaktische Überprüfung vor. Das heißt, dass der Compiler Ihr Programm nach Syntaxfehlern durchforstet und Ihnen gegebenenfalls den Fehler mit einer mehr oder weniger

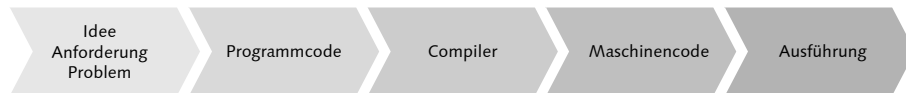
guten Fehlerbeschreibung aufzeigt. Was der Compiler nicht kann, ist, eine semantische Überprüfung vorzunehmen. Immerhin ist es ja nur ein Computer. Dieser kann zwar Regeln prüfen, den Inhalt versteht er jedoch nicht. Zu dem Thema, was ein Computer »versteht«, erfahren Sie mehr in Kapitel 2, »Algorithmisches Denken«.

Programmieren könnten wir also wie folgt definieren:

#### Definition »Programmieren«

Programmieren ist das Formulieren einer Problemlösung in Befehlsanweisungen mithilfe einer Sprache, die automatisch in Befehle übersetzt wird, die ein Computer ausführen kann.

Nach dem, was wir bisher behandelt haben, lässt sich der in Abbildung 1.2 dargestellte grundlegende Entwicklungsprozess festhalten:



**Abbildung 1.2** Grundlegender Entwicklungsprozess

Wenn man programmieren lernt, treten die größten Schwierigkeiten relativ früh auf, und zwar dann, wenn es darum geht, von einer Idee, einer Anforderung oder von einem Problem zum Programmcode (engl. *Sourcecode*) zu gelangen. Wie kann ich mein Problem, das ich als Mensch in natürlicher Sprache formuliere, in einen Programmcode gießen und somit dem Computer verständlich machen? Ist diese Hürde erst einmal genommen, so haben Sie den schwersten Teil bereits hinter sich. Sobald Sie dieses Hindernis überwunden haben, kann der Spaß am Programmieren beginnen, und nichts kann Sie mehr aufhalten. Da dieser Abstraktionsprozess, wenn Sie mit dem Programmieren beginnen, das Schwierigste ist, wird die Vorgehensweise vor allem in Kapitel 2, »Algorithmisches Denken«, genau behandelt. Das Gute an diesem Thema ist, dass es praktisch programmiersprachen-unabhängig ist.

Das Schöne an dem in Abbildung 1.2 abgebildeten einfachen Prozess ist, dass Sie sich primär nur um die ersten beiden Schritte bzw. um die Übergangsphase von Schritt 1 zu Schritt 2 kümmern müssen. Der 3. Schritt des Kompilierens wird vom Compiler übernommen. Dieser wird zwar Ihren Code aufgrund von Syntaxfehlern immer wieder mal verweigern, aber wenn der Compiler den Programmiercode annimmt, so erzeugt er vollautomatisch den Maschinencode, der vom Computer ausgeführt werden kann.

Da Sie sicher bereits auf Programmiercode warten, hier ein kleines Beispiel in *Pseudocode*:

```
If (2010 - YearOfBirth > 18)
    Write("Sie sind älter als 18 Jahre");
Else
    Write("Sie sind jünger oder genau 18 Jahre alt");
```

So kann zum Beispiel ein Programmteil aussehen. Dieser würde vom Jahr 2010 das Geburtsjahr abziehen und je nachdem, ob das Ergebnis größer als 18 ist, »Sie sind älter als 18 Jahre« oder, wenn das Ergebnis kleiner oder gleich 18 ist, »Sie sind jünger oder genau 18 Jahre alt« ausgeben. – Keine Angst, Sie werden im nächsten Kapitel erfahren, wie man derartige Zeilen interpretiert.

Der angeführte Programmcode ist sogenannter *Pseudocode*. Das heißt, er entspricht keiner echten Programmiersprache. Er ist an einige Programmiersprachen angelehnt, ist jedoch etwas leichter zu lesen als eine echte Programmiersprache.

#### Definition »Pseudocode«

Mit Pseudocode werden exemplarisch bestimmte Teile eines Algorithmus oder Konzepte aufgezeigt, wobei der Pseudocode lediglich Elemente einer Programmiersprache enthält, die für dieses Konzept von Bedeutung sind. Pseudocode ist daher eine vereinfachte Art von Programmiersprache, die für die Darstellung eines Sachverhaltes zweckmäßig ist.

Das nächste Kapitel behandelt den Schritt, wie Sie von der Idee zum Algorithmus und somit zum Programmcode kommen, inklusive den damit verbundenen Kenntnissen. Erst später gehe ich auf verschiedene Programmiersprachen und Algorithmen ein.

### 1.3 Welche Bereiche der Softwareentwicklung gibt es?

Bevor ich Sie in die algorithmischen Denkmuster entlasse, möchte ich Ihnen noch kurz vor Augen führen, in welchen Bereichen überall Softwareprogramme existieren und wo die Schwerpunkte bei verschiedenen Programmierunternehmungen liegen.

- ▶ *Computerspiele und Computergrafik*: Die Entwicklung von Computerspielen ist der Bereich, in dem wohl sehr viele Programmierer gern tätig wären. Die Entwicklung eines Computerspiels ist mit Sicherheit eine große Herausforderung mit großem Spaßfaktor. Doch als leidenschaftlicher Programmierer werden Sie sich wohl nicht mit der Verwendung eines Spiele-Frameworks (engl.

*Der Mensch ist immer noch der beste Computer.*  
– John F. Kennedy

## 2 Algorithmisches Denken

Dieses Kapitel hilft Ihnen bei Ihren ersten Schritten. Ich bringe Ihnen algorithmische Denkmuster bei, mit denen Sie aus einer Problembeschreibung einen Algorithmus entwickeln können. Zur Formulierung eines Algorithmus verwende ich Pseudocode. Durch diesen Pseudocode lernen Sie die grundlegenden Programmier-elemente kennen, mit denen Sie in Zukunft Ihre Programme schreiben werden. Nach Abschluss dieses Kapitels können Sie Algorithmen in Pseudocode formulieren und diesen Pseudocode mithilfe der Programmier-elemente in Programmcode umsetzen.

### 2.1 Computer sind dumm!

Ihr Computer kann Ihnen schöne Grafiken auf dem Bildschirm darstellen und viele mathematische Aufgaben schneller lösen als Sie? Dennoch ist die Darstellung einer schönen Grafik nicht der Verdienst des Computers, sondern der Verdienst des Programmierers, der den entsprechenden Algorithmus dazu geschrieben hat. Gleiches gilt natürlich auch für Lösungen von mathematischen Aufgaben, für den perfekten Ausdruck am Drucker und für das Abspielen Ihrer gespeicherten Musikstücke. Computer können selbstständig keine Algorithmen entwickeln. Sie können auch selbstständig keine Musik komponieren und schon ganz und gar nicht denken! Die Zeit, in der sich der Mensch Sorgen machen muss, dass sich die Welt der Maschinen gegen ihn erhebt, ist noch lange nicht gekommen – also: Computer sind dumm!

### 2.2 Algorithmische Formulierung

Stellen Sie sich vor, Ihr Computer wäre ein Koch. Dieser Koch kann zwar selbst nicht kochen, er kann jedoch das Kochrezept ablesen und die darin stehenden Anweisungen akribisch befolgen. So sind Sie als Programmierer gefordert, das

entsprechende Kochrezept zu schreiben. Da Ihr Computer, so wie jeder andere Computer auf der Welt, lediglich Befehle ausführen kann und selbstständig keinen Funken an Intelligenz besitzt, ist es Ihre Aufgabe, dieses Kochrezept so genau zu definieren, dass der Koch jeden zu tätigen Schritt ablesen kann.

Ein Beispiel:

Sie wollen einen Kuchen backen und sind bereits so weit, dass der Kuchen für 30 Minuten bei 180 °C in den Ofen muss. Dann steht dies in einem Backrezept in etwa so:

*und den Kuchen nun für 30 Minuten bei 180 °C backen.*

Ein Mensch würde die Anweisung verstehen und ausführen. Selbstverständlich würden Sie das Backrohr erst öffnen und diese Aktion lediglich dann ausführen, wenn sich aktuell nichts im Backrohr befindet. Genau daran würde Ihr Computer jedoch scheitern, wenn Sie Ihr Programm so formulieren. Ist das Backrohr aus irgendwelchen Gründen zu dem Zeitpunkt bereits offen, wird Ihr Programm funktionieren. Ist es jedoch geschlossen (was wohl wahrscheinlicher ist), so müssten Sie anschließend den Boden sauber machen, da die Anweisung für das Öffnen des Backrohrs nicht gegeben wurde.

Um ein funktionierendes Programm zu schreiben, ist es erforderlich, jede mögliche Situation zu definieren. Möglicherweise wurden Sie bereits mit der folgenden Fehlermeldung in einem Programm konfrontiert: »Es ist ein unbehandelter Fehler aufgetreten ...« Immer wenn Sie so etwas sehen, wissen Sie nun, dass gerade etwas passiert ist, was der Programmierer im Programm nicht berücksichtigt hat.

Um bei unserem Beispiel zu bleiben: Die Ofentür war geschlossen, und der Computer hat versucht, die Teigmasse in den Ofen zu schieben. Im schlimmsten Fall ist nun auch die Glastür des Ofens kaputt, im besten Fall ist die Teigmasse nicht zu Boden gegangen, und es kann ein erneuter Versuch unternommen werden. Übertragen auf ein Computerprogramm bedeutet dies, dass im schlimmsten Fall nun Dateien beschädigt (engl. *corrupt*) und nicht mehr verwendbar sind. Im besten Fall wird dem Benutzer eine Meldung angezeigt, die ihm dabei hilft, das Problem zu lösen.

Sie sehen, dass dieser eine Satz im Backrezept nicht ausreicht, und daher müssen wir ihn etwas umformulieren. Vorerst formulieren wir diese Anweisung in natürlicher Sprache:

*Nun öffnen Sie die Ofentür und prüfen, ob das Backrohr aktuell leer ist. Wenn das Backrohr aktuell leer ist, nehmen Sie den Kuchen und geben ihn in den Ofen. Ist*

*das Backrohr nicht leer, nehmen Sie den darin vorhandenen Gegenstand heraus. Geben Sie anschließend den Kuchen in das Backrohr. Anschließend schalten Sie den Ofen für 30 Minuten auf 180 Grad Celsius ein.*

Durch diese Formulierung kann doch schon nichts mehr falsch gemacht werden, oder?

Haben wir nicht etwas in der Formulierung vergessen? Da war doch noch etwas: Die Ofentür muss natürlich auch wieder geschlossen werden, nicht dass noch die Küche abbrennt! Auch diesen Fehler haben Sie möglicherweise schon einmal erlebt! Kennen Sie die Fehlermeldung a là »Die Datei kann nicht gelöscht werden, da sie noch von einem anderen Prozess verwendet wird«? Auch wenn Sie auf eine Datei zugreifen, indem Sie diese lesend oder schreibend öffnen, müssen Sie diese am Ende des Vorgangs wieder schließen, da diese ansonsten für andere Programme (und möglicherweise auch für dieses Programm) bis zum nächsten Neustart des Betriebssystems gesperrt ist und somit nicht verwendet werden kann. Also formulieren wir unseren Teil der Zubereitung noch etwas weiter um:

*Nun öffnen Sie die Ofentür und prüfen, ob das Backrohr aktuell leer ist. Wenn das Backrohr aktuell leer ist, nehmen Sie den Kuchen und geben ihn in den Ofen. Ist das Backrohr nicht leer, geben Sie den darin vorhandenen Gegenstand heraus. Geben Sie anschließend den Kuchen in das Backrohr. Schließen Sie das Backrohr. Konnten Sie das Backrohr schließen, schalten Sie anschließend den Ofen für 30 Minuten auf 180 Grad Celsius ein.*

Die Abfrage, ob das Backrohr geschlossen werden konnte, ist in diesem Fall wichtig, da in dem Fall, dass dies nicht möglich war (weil zum Beispiel die Ofentür defekt ist), die Küche abbrennen könnte. Um unser Beispiel wieder in die Welt der IT (Informationstechnologie – oder umgangssprachlich in die »Computervelt«) zu übertragen: Falls etwas passieren kann, das unter Umständen schwerwiegende Fehler nach sich ziehen würde, so vergewissern Sie sich, dass Sie diese Aktion auch gefahrlos ausführen können! Sie können also, wenn Sie versuchen, etwas in eine Datei zu speichern, nicht davon ausgehen, dass genügend Speicherplatz für den Schreibvorgang vorhanden ist. Sie können nicht davon ausgehen, dass der Benutzer Schreibrechte auf das Verzeichnis besitzt, Sie können nicht davon ausgehen, dass die Datei nicht gerade von einem anderen Programm gesperrt ist, und so weiter. Sie müssen also derartige Fehler behandeln. Dazu folgt später mehr im Abschnitt 6.3, »Exceptions«.

Dieses einfache Beispiel macht Folgendes deutlicher:

- ▶ Sie müssen dem Computer jede Kleinigkeit befehlen, sonst macht das Programm nicht, was Sie von ihm verlangen.

- ▶ Sie müssen in Ihrem Computerprogramm alle möglichen Fehlerfälle behandeln (später wird gezeigt, wie Sie mit möglicherweise vergessenen Fehlerbehandlungen umgehen können).
- ▶ Versuchen Sie, Dinge, die für Menschen selbstverständlich sind, bei der Formulierung eines Algorithmus nicht wegzulassen, sondern zu formulieren. (Natürlich müssen Sie die Ofentür öffnen – dann schreiben Sie es auch!)
- ▶ Ein Programm besteht zu einem großen Prozentsatz aus Befehlen, die lediglich Ausnahmen behandeln!

#### So wichtig ist Fehlerbehandlung

Ein stabiles Programm besteht zu 40 % aus Programmcode zur Fehlerbehandlung!

Bisher haben wir lediglich das Vorgehen in natürlicher Sprache erarbeitet. Als Nächstes vereinfachen wir die Sätze, indem wir uns auf die Wörter beschränken, die uns tatsächlich Informationen darüber liefern, was zu tun ist. Elemente, die den Satz ausschmücken, werden weggelassen:

*Ofentür öffnen*

*Backrohr leer?*

*Wenn JA:*

*Gib Kuchen in Backrohr*

*Wenn NEIN:*

*Nimm Inhalt aus Backrohr*

*Gib Kuchen in Backrohr*

*Schließe Backrohr*

*Backrohr geschlossen?*

*Wenn JA:*

*Ofen einschalten (30 min, 180 °C)*

Wie Sie sehen, habe ich lediglich die Füllwörter entfernt. Natürlich sind es nun keine grammatikalisch korrekten Sätze mehr, allerdings beinhalten sie genug Informationen, damit Sie wissen, was zu tun ist. Durch die Einrückung bei Fragen und den möglichen Antworten sind auch Zusammenhänge erkennbar.

Inzwischen ist aus einem einfachen Satz ein struktureller Pseudocode geworden. Dies ist der Ausgangspunkt zur Erstellung eines Algorithmus. Erstellen Sie immer entsprechenden Pseudocode, bevor Sie mit der Entwicklung eines Algorithmus beginnen – vor allem dann, wenn Sie mit Schwierigkeiten bei der Formulierung des Algorithmus in der Programmiersprache kämpfen. Der Pseudocode hilft Ihnen dabei, die Struktur des Algorithmus zu definieren, ohne dass Sie sich mit Syntax-Spitzfindigkeiten herumschlagen müssen. Später, wenn Sie erfahrener

und routinierter werden, werden Sie oftmals sofort mit der Algorithmenerstellung in der Programmiersprache beginnen. Folgende Tatsache soll Sie etwas motivieren, die Pseudocodeformulierungen zu verwenden: In Softwareprojekten werden oftmals Algorithmen und Programmstrukturen vor der eigentlichen Implementierung mithilfe von (UML-)Diagrammen (Ablaufdiagrammen, Flussdiagrammen, Klassendiagrammen usw.) erstellt. Diese Diagramme stellen den Datenfluss und die Struktur des Algorithmus dar. Die Formulierung in Pseudocode (auch wenn die Form eine etwas andere ist) ist also durchaus gängige Praxis.

Bevor wir nun zur Formulierung in einer Programmiersprache kommen, müssen Sie die Sprachelemente einer Programmiersprache kennenlernen. Damit befassen wir uns im Folgenden, und anschließend wird der Satz zum Kuchenbacken unser erster formulierter Algorithmus werden.

## 2.3 Einführung in Sprachelemente

Sie können nicht ohne eine Programmiersprache programmieren. Und da es sehr viele verschiedene Programmiersprachen gibt, haben Sie die Qual der Wahl, sich für eine zu entscheiden. Kapitel 3, »Die Wahl der Programmiersprache«, soll Sie bei der Wahl der Programmiersprache unterstützen. Dort bringe ich Ihnen die Vor- und Nachteile sowie Einsatzgebiete von verschiedenen Programmiersprachen nahe. In diesem Buch verwende ich als Syntax und zur Einführung die Programmiersprache C# (gesprochen »Si-Scharp«), die von Microsoft entwickelt wurde und weiterentwickelt wird. Wenn Sie C# programmieren können, so ist es für Sie ein Leichtes, auch in Java zu entwickeln. Sollten Sie sich später für C++ entscheiden, werden Sie auch hier sehr viele Ähnlichkeiten feststellen.

In Kapitel 1, »Einführung«, habe ich Ihnen gesagt, dass Sie beim Programmieren einen großen Sandkasten mit viel Spielzeug haben. Nun ist es so weit, Sie mit dem Spielzeug vertraut zu machen. Die Erklärung der Sprachelemente ist zwar nicht so prickelnd, aber ein notwendiges Übel. Sie werden jedoch lernen, Algorithmen zu formulieren und zu durchdenken, und anschließend werden Sie hier die nötigen Fähigkeiten erlernen, wie Sie den formulierten Algorithmus in einer Programmiersprache in Code umwandeln können. Danach werden Sie sich mit den syntaktischen Spitzfindigkeiten beschäftigen müssen. Da Sie allerdings den Algorithmus bereits formuliert haben, werden Sie sich nicht vom Weg abbringen lassen, und Ihr Fleiß wird von Erfolg gekrönt sein.

# Index

.NET 85

## A

---

Ablaufdiagramm 35  
Algorithmus 21  
  *Algorithmenstruktur* 34  
  *Denkmuster* 31  
  *Einfügesortieren* 117  
  *Elemente* 41  
  *Formulierung* 31, 111  
  *Sortieralgorithmus* 76  
  *Vorgehensweise* 41  
Anchor 218, 219  
Anforderungen 272  
Anonyme Methoden 94  
Anonyme Typen 279, 283  
Array 57, 118, 133, 181, 189, 279  
  *Alternativen* 284  
  *ausgezackte Arrays* 63  
  *CopyTo* 61  
  *Eigenschaften* 61  
  *GetUpperBound* 61  
  *Initialisierung* 61  
  *Jagged-Arrays* 63  
  *Length* 61, 129  
  *mehrdimensionale Arrays* 63  
  *nullbasierte Indizes* 59  
  *zweidimensionale Arrays* 62  
Assembly 157  
  *Verweis hinzufügen* 157  
Attribut 268  
Automatic Properties 92

## B

---

Balkendiagramm 126  
Bedingung 37, 40, 266  
Benutzer 239  
  *DAU* 240  
Benutzeroberfläche 17, 210, 211, 218,  
  219  
  *Größenanpassung* 258  
  *Windows Style Guides* 248  
Bibliothek 89

Binärsystem 20  
Boolescher Ausdruck 49  
Breakpoint 105  
Browser 25  
Button 243

## C

---

C 86  
C++ 86  
  *Managed C++* 89  
  *Templates* 87  
Call Stack 150  
call-by-reference 78  
call-by-value 78  
Character 128  
  *Character-Array* 128  
Checkbox 217, 241, 242, 248  
CheckedListBox 242  
Client 25  
Code-Behind-Datei 195, 207  
Codegerüst 102  
Combobox 245, 249, 250  
Compiler 22, 87  
Computergrafik 24  
Computerspiele 24  
Container → SplitContainer  
Control 176, 193  
CSV (Comma Separated Value) 57  
Cursor 230

## D

---

Datei  
  *corrupt* 32  
  *Dateizugriff* 201  
  *Speichern* 221  
Datenstruktur 141, 144  
  *Struktur anlegen* 144  
Datentyp 41, 222  
  *bool* 43, 277  
  *byte* 43  
  *char* 43  
  *DateTime* 145, 280, 281  
  *double* 43

Datentyp (Forts.)

- float* 43, 277
- int* 43, 222, 228, 277
- long* 43
- sbyte* 43
- short* 43
- Speicherplatz* 42
- string* 43, 222
- unsigned* 44
- Debuggen 98, 105, 106, 150
  - Ausführen bis Rücksprung* 107
  - Einzelschritt* 106
  - Prozedurschritt* 107
- Delegate 204, 205, 206, 208, 282, 298, 299
- Desktop-Softwareentwicklung 26
- Destruktor 168
- Dictionary 286, 287
- Dock 198

---

**E**

- Eigenschaft 177
- Eigenschaften-Fenster 197
- Einrückung 40
- else 38
- Embedded Systems 26
- Entscheidung 20
- Entwicklungsprozess 23, 272
  - Meilensteine* 272
- Entwicklungsumgebung 95
  - Microsoft Visual Studio* 98
  - Microsoft Visual Studio-Editionen* 98
  - Projektmappen-Explorer* 103
- Enumeration 141, 142
- Ereignis 94, 164, 203, 204, 205, 228, 235, 242, 243, 244, 245, 248, 250
  - EventArgs* 205
  - Event-Handler* 201, 202, 206, 208, 223, 224, 227, 229, 234, 235, 236, 299
  - registrieren* 200, 204, 229
  - selbst erstellen* 204
- Event → Ereignis
- Exception 142, 145, 164
  - Fehler auslösen - throw* 149
  - NullReferenceException* 205
  - try / catch / finally* 146
- exe-Datei 225
- Extension-Methods 280, 282, 283

---

**F**

- Falsch 37
- false 37
- Fehler
  - Absturz* 28
  - Fehlerbehandlung* 28, 34, 225
  - Fehlertoleranz* 28
  - Fehlervermeidung* 75
  - Syntaxfehler* 23
- File-Klasse 221
- Flussdiagramm 35
- Formular 194
  - Design* 195
- Framework 85
- Funktion 36, 73, 121
  - Ausgangsparameter* 79, 80
  - Eingangsparameter* 80
  - Funktionsaufruf* 36
  - FunktionsSignatur* 74
  - Hauptfunktion* 121
  - Hilfsfunktion* 121
  - Parameter* 40, 73, 76
  - return* 74
  - Rückgabewert* 73, 74
  - Übergabeparameter* 145, 167
  - Übergangsparameter* 80
  - Unterfunktion* 76
- Funktionszeiger 94

---

**G**

- Game Engine 25
- Garbage Collector 168
- Generics 91, 94, 222
- Geschwungene Klammern 37
- Grammatik 20
- GroupBox 215

---

**H**

- Hacker 25
- HashSet 285, 286, 287
- Hello World 125
- Hyperlink 243

**I**

---

if 37  
 Implementieren 37  
 int  
   *int.Parse* 130  
 IntelliSense 98, 107  
   *Eigenschaften* 108  
   *Ereignisse* 108  
   *Funktionen* 108  
 Interface 186, 189  
 Interface → Schnittstelle  
 Internet 25

**J**

---

Java 85

**K**

---

Klasse 151, 154, 155, 158, 181, 183, 185, 222, 280  
   *abgeleitete Klasse* 160  
   *abstrakte Basisklasse* 177  
   *abstrakte Klasse* 180  
   *abstrakte Methode* 185  
   *Basisklasse* 172, 175, 178, 181, 182  
   *Console* 158  
   *Eigenschaft* 91, 154, 164, 166, 178, 181  
   *FileInfo* 155, 156  
   *Klassenmethode* 153  
   *Methode* 154, 167, 181  
   *Sichtbarkeit* 160, 164  
   *statische Methode* 158  
   *Überschreiben einer Methode* 173  
   *Zugriff auf die Basisklasse - base* 179  
 Klassenbibliothek 101  
 Klassendiagramm 35  
 Kommentar 46  
   *Einzeilig //* 46  
   *Mehrzeilig /\* \*/* 46  
   *XML-Kommentar* 183  
 Konsole 205  
 Konsolenanwendung 101, 194  
 Konstante 41, 45, 46, 74  
 Konstruktor 144, 162, 168, 169, 170  
   *überladen* 170  
 Kontextmenü 247

Konzepte 17  
 Kreativität 16

**L**

---

Label 215, 216, 217, 228, 243  
 Lambda-Expressions 92, 281, 282  
 Lastenheft 272  
 Lernprozess 18  
 LINQ 91, 279, 280, 281, 282, 283  
 Listbox 242, 249, 250  
 Liste 285, 286  
 ListView 213, 215, 217, 232, 234, 235, 236, 244, 245  
   *ListViewItem* 234  
 Logisches Oder - || 52  
   *Kurzschlussauswertung* 53  
 Logisches Und - && 51  
   *Kurzschlussauswertung* 53

**M**

---

Main 102, 194  
 Maschinencode 23  
 Maschinensprache 22  
 Maschinensteuerung 26  
 Menü 197, 212, 223, 247, 248  
   *MenuStrip* 197, 212  
   *Shortcuts* 197  
 MessageBox 226  
 Methode 222  
 Microsoft Visual C# Express Edition 13, 99, 100, 102, 109, 125, 154, 194, 260  
 Mobile Softwareentwicklung 25

**N**

---

Namespace 102, 155, 158, 268  
   *Namenskonflikte vermeiden* 102  
   *System* 158  
   *System.Collections.Generic* 285  
   *System.Configuration* 157  
   *System.Drawing* 157  
   *System.GlobalIZATION* 157  
   *System.IO* 155, 156, 157, 202, 252  
   *System.Linq* 279, 280, 283  
   *System.Net* 157  
   *System.Net.Mail* 157  
   *System.Text* 157

## Namespace (Forts.)

- System.Threading* 157
- System.Web* 157
- System.Windows.Forms* 157
- using* 156

Nullable-Types 94

**O**

Objekt 62, 154, 178, 181, 185, 281

- Methoden* 62
- Zustand* 62

Objektorientierte Programmierung 170

- Ableitung* 177
- Abstraktion* 177, 190
- Datenkapselung* 190
- Generalisierung* 177
- Konkretisierung* 177
- Polymorphie* 191
- Vererbung* 191

Objektorientierung 86, 94

- Eigenschaft* 94
- Geheimnisprinzip* 160
- Klasse* 94
- Methode* 94
- Objekt* 94

**P**

Partielle Klasse 196

Pflichtenheft 272

Phantominspektor 20

Polymorphismus 175

- Laufzeittyp* 176

Programmablauf 20

Programmblock 37

Programmcode 23

Programmieren 23

Programmiersprache 20, 22

- Entscheidungskriterien* 95

Projektmappe 103

Projektmappen-Explorer 223

Prototyp 210, 218

Prozedur 39, 74

Prozedurale Programmierung 86

Pseudocode 24, 34

**R**

Radiobutton 242

ReadOnly 167

Reference-Types 80

Repository 263

Repository → Versionsverwaltung

**S**

Schaltfläche 218

Schieberegler → TrackBar

Schleife 64

- break* 68
- continue* 71
- do* 69
- Endlosschleife* 69
- for* 65
- foreach* 70, 285, 287
- Laufbedingung* 64
- Laufvariable* 65
- verschachtelte Schleifen* 71, 131
- while* 66, 112

Schlüsselwort 60, 104

- abstract* 181
- base* 179
- const* 45
- get* 165
- new* 60, 144, 284
- null* 231
- out* 79
- override* 174, 175
- partial* 196
- ref* 78
- return* 74
- set* 166
- static* 159
- this* 280, 281
- using* 156, 283
- var* 278, 284
- virtual* 174, 181, 183
- void* 74
- where* 282

Schnittstelle 177, 186

- Eigenschaft* 187
- Methode* 188
- Sichtbarkeit in Schnittstellen - public* 188

Schrittweise verfeinern 115, 223

- Die Idee* 115

- Schrittweise verfeinern (Forts.)
    - Nachteile* 122
    - Vorteile* 121
  - Semantik 21, 22
  - Semantische Fehler 22
    - Denkfehler* 22
  - Serialisieren 220, 221
    - XmlSerializer* 221, 222
  - Server 25
  - Setter - set 167
  - Shellscript 239
  - Sichtbarkeit 185
    - internal* 161, 164
    - private* 160, 164, 185
    - protected* 160, 185
    - protected internal* 161
    - public* 161, 164, 167, 185
  - Silverlight 101
  - Slider → TrackBar
  - Solution 103
  - SortedDictionary 287
  - SortedList 286
  - Sourcecode 23
  - Spiele- Framework 25
  - SplitContainer 212, 213, 219
  - Stack Trace 150
  - Standarddialog 250
    - Ergebnis abrufen - DialogResult* 202
    - Farbwähler* 254
    - FolderBrowserDialog* 255
    - OpenFileDialog* 253, 255
    - PrintDialog* 253, 254
    - SaveFileDialog* 253
    - Schriftart auswählen - FontDialog* 256
    - ShowDialog-Methode* 252
  - Standardprogramme 16
  - Statement 38
  - Statische Methodenvariablen 94
  - Steuerelement 241
    - Ausrichtung* 198
    - Checkbox* 241
    - Positionieren - Anchor* 218, 219, 241
    - Positionieren - Dock* 198, 241
  - Storyboard 210, 216, 218, 219, 273, 299
  - Stream 221, 222
  - Stream-Klasse 221
  - Strichpunkt 36, 42
  - String
    - Split* 129
  - Struktur 141
  - switch 53, 112
    - break* 56
    - case* 55
    - default* 57
    - if/else-Kaskade* 54
  - Synchronisation 25
  - Syntax 20, 21
    - syntaktische Überprüfung* 22
    - Syntaxfehler* 22, 43
    - Syntax-Highlighting* 104
  - Systemdialog → Standarddialog
- ## T
- 
- Technologie 17
  - Teilprobleme 119
  - Templates 91
  - Test
    - Code-Abdeckung* 266
    - Regressionstest* 266
    - Unit-Test* 265, 266
    - Unit-Test mit NUnit* 267, 268, 269
  - Textbox 198, 199, 201, 202, 215, 243
    - MultiLine* 198, 215
    - PasswortChar* 244
    - ScrollBars* 198, 216
  - TrackBar 216, 217, 249
  - Treeview 244
    - Node* 245
  - true 37, 50
- ## U
- 
- UML 35
  - Unsicherer Code 94
  - Use-Case 126, 210, 273
  - User Experience 26
- ## V
- 
- Value-Types 80
  - Variable 41
    - Nullable* 277, 278
    - Referenzvariable* 78
    - Variablenwert* 42
    - Werteüberwachung* 105
    - Wertzuweisung* 42, 47

## Index

Vererbung 171  
    *abgeleitet* 172  
    *Interfaces statt Mehrfachvererbung* 186  
    *Mehrfachvererbung* 186  
Versionsverwaltung 264, 265, 270, 274  
    *Branch* 264  
    *Changeset* 264  
    *checkin* 264  
    *checkout* 264  
    *Client - AnkhSVN* 265  
    *Client - Tortoise SVN* 265  
    *commit* 264  
    *Merge* 264  
    *Revision* 264  
    *Subversion* 264  
    *Visual Studio Team Foundation Server*  
        264  
Verteilte Softwareentwicklung 26  
Verzweigung 49

Vielgestaltigkeit 175  
Visual Studio 85

## W

---

Wahr 37  
Wartung 45  
Webanwendungen 25  
Wenn 37  
Windows Forms-Anwendung 101, 194,  
    206  
Windows Presentation Foundation 258  
WPF-Anwendung 101  
WPF-Browser-Anwendung 101

## X

---

XBAP 101  
XML (Extensible Markup Language) 220