

C# mit .NET

Das umfassende Handbuch

DAS INHALTS- VERZEICHNIS

» Hier geht's
direkt
zum Buch

Auf einen Blick

1	Allgemeine Einführung in .NET	33
2	Grundlagen der Sprache C#	67
3	Das Klassendesign	155
4	Vererbung, Polymorphie und Interfaces	235
5	Delegaten, Ereignisse und Lambda-Ausdrücke	301
6	Strukturen, Records und Enumerationen	339
7	Fehlerbehandlung und Debugging	359
8	Auflistungsklassen (Collections)	399
9	Generics – generische Datentypen	429
10	Weitere C#-Sprachfeatures	461
11	LINQ – Language Integrated Query	539
12	Arbeiten mit Dateien und Streams	575
13	Multithreading	631
14	Die Task Parallel Library (TPL) und das Task-based Async Pattern (TAP)	687
15	Grundlegende .NET-Klassen	715
16	Einführung in das Entity Framework Core	761
17	Desktop- und Cross-Platform-Entwicklung	819
18	Windows Presentation Foundation	845
19	Einführung in .NET MAUI	911
20	2D-Grafik in WPF und .NET MAUI	945
21	Komponententests (Unit-Tests)	967
22	Einführung in ASP.NET Core	1027
23	Einführung in Blazor	1089
24	Cloud-Entwicklung mit Azure	1135
25	Softwareentwicklung in Zeiten von KI	1179

Inhalt

Materialien zum Buch	30
Über diese Auflage	31
1 Allgemeine Einführung in .NET	33
1.1 Warum .NET?	33
1.1.1 Ein paar Worte zu diesem Buch	36
1.1.2 Die Beispielprogramme	38
1.2 .NET unter die Lupe genommen	39
1.2.1 Das Entwicklerdilemma	39
1.2.2 .NET – ein paar allgemeine Eigenschaften	40
1.2.3 Das Sprachenkonzept	42
1.2.4 Die Common Language Specification (CLS)	43
1.2.5 Das Common Type System (CTS)	45
1.2.6 Das .NET-Framework	46
1.2.7 Neue Entwicklungsparadigmen und Technologien	46
1.2.8 Die Common Language Runtime (CLR)	47
1.2.9 Die .NET-Klassenbibliothek	48
1.2.10 Das Konzept der Namespaces	49
1.3 Assemblies	51
1.3.1 Die Metadaten	52
1.3.2 Das Manifest	53
1.4 Die Entwicklungsumgebung	53
1.4.1 Editionen von Visual Studio 2022	54
1.4.2 Die Installation	55
1.4.3 Die Entwicklungsumgebung von Visual Studio 2022	57
1.5 Git – Versionskontrolle leicht gemacht	62
1.5.1 Die Grundbefehle von Git	62
1.5.2 Git in Visual Studio	64

2	Grundlagen der Sprache C#	67
2.1	Konsolenanwendungen	67
2.1.1	Ein erstes Konsolenprogramm	68
2.1.2	Top-Level-Statements als neuer Standard	72
2.2	Grundlagen der C#-Syntax	74
2.2.1	Kennzeichnen, dass eine Anweisung abgeschlossen ist	74
2.2.2	Anweisungs- und Gliederungsblöcke	75
2.2.3	Kommentare	76
2.2.4	Die Groß- und Kleinschreibung	77
2.2.5	Die Struktur einer Konsolenanwendung	77
2.3	Variablen und Datentypen	79
2.3.1	Variablendeklaration	79
2.3.2	Der Variablenbezeichner	80
2.3.3	Der Zugriff auf eine Variable	81
2.3.4	Ein- und Ausgabemethoden der Klasse »Console«	82
2.3.5	Die elementaren Datentypen von .NET	88
2.3.6	Ausgabe ganzzahliger Datentypen	96
2.3.7	Elementare Typkonvertierung	97
2.4	Operatoren	105
2.4.1	Arithmetische Operatoren	105
2.4.2	Vergleichsoperatoren	108
2.4.3	Logische Operatoren	109
2.4.4	Bitweise Operatoren	113
2.4.5	Zuweisungsoperatoren	115
2.4.6	Stringverkettung	116
2.4.7	Sonstige Operatoren	117
2.4.8	Operator-Vorrangregeln	117
2.5	Datenfelder (Arrays)	118
2.5.1	Die Deklaration und Initialisierung eines Arrays	118
2.5.2	Der Zugriff auf die Array-Elemente	120
2.5.3	Array-Bereiche und der Operator ^	121
2.5.4	Mehrdimensionale Arrays	122
2.5.5	Festlegen der Array-Größe zur Laufzeit	123
2.5.6	Bestimmung der Array-Obergrenze	125
2.5.7	Die Gesamtanzahl der Array-Elemente	125
2.5.8	Verzweigte Arrays	126

2.6	Kontrollstrukturen	127
2.6.1	Die »if«-Anweisung	128
2.6.2	Das »switch«-Statement	133
2.7	Programmschleifen	138
2.7.1	Die »for«-Schleife	139
2.7.2	Die »foreach«-Schleife	149
2.7.3	Die »do«- und die »while«-Schleife	150

3 Das Klassendesign 155

3.1	Einführung in die Objektorientierung	155
3.2	Die Klassendefinition	158
3.2.1	Klassen in Visual Studio anlegen	158
3.2.2	Das Projekt »GeometricObjectsSolution«	159
3.2.3	Die Deklaration von Objektvariablen	161
3.2.4	Zugriffsmodifizierer einer Klasse	162
3.2.5	Splitten einer Klassendefinition mit »partial«	163
3.2.6	Arbeiten mit Objektreferenzen	163
3.3	Referenz- und Wertetypen	166
3.3.1	Werte- und Referenztypen nutzen	166
3.4	Die Eigenschaften eines Objekts	167
3.4.1	Öffentliche Felder	167
3.4.2	Datenkapselung mit Eigenschaftsmethoden (Properties)	169
3.4.3	Auto-Properties (automatisch implementierte Properties)	174
3.4.4	Properties mit »init«	175
3.4.5	Unterstützung von Visual Studio	175
3.5	Methoden eines Objekts	176
3.5.1	Methoden mit Rückgabewert	176
3.5.2	Methoden ohne Rückgabewert	180
3.5.3	Methoden mit Parameterliste	180
3.5.4	Methodenüberladung	182
3.5.5	Variablen innerhalb einer Methode (lokale Variablen)	185
3.5.6	Modifizierer eines Parameters	187
3.5.7	Besondere Aspekte einer Parameterliste	193
3.5.8	Zugriff auf private Daten	199
3.5.9	Die Trennung von Daten und Code	200

3.5.10	Namenskonflikte mit »this« lösen	201
3.5.11	Methode oder Eigenschaft?	202
3.5.12	Umbenennen von Methoden und Eigenschaften	204
3.6	Konstruktoren	204
3.6.1	Konstruktoren bereitstellen	205
3.6.2	Die Konstruktoraufrufe	206
3.6.3	Definition von Konstruktoren	207
3.6.4	»public«- und »internal«-Konstruktoren	208
3.6.5	Primäre Konstruktoren	208
3.6.6	»private«-Konstruktoren	208
3.6.7	Konstruktoraufrufe umleiten	209
3.6.8	Vereinfachte Objektinitialisierung	210
3.7	Der Destruktor	211
3.8	Konstanten in einer Klasse	212
3.8.1	Konstanten mit dem Schlüsselwort »const«	213
3.8.2	Schreibgeschützte Felder mit »readonly«	213
3.9	Statische Klassenkomponenten	214
3.9.1	Statische Eigenschaften	214
3.9.2	Statische Methoden	217
3.9.3	Statische Klasseninitialisierer	218
3.9.4	Statische Klassen	219
3.9.5	Statische Klasse oder Singleton-Pattern?	219
3.10	Namensräume (Namespaces)	221
3.10.1	Zugriff auf Namespaces	222
3.10.2	Die »using«-Direktive	224
3.10.3	Globaler Namespace	224
3.10.4	Dateispezifische Namespaces-Deklarationen	224
3.10.5	Vermeiden von Mehrdeutigkeiten	225
3.10.6	Namespaces festlegen	226
3.10.7	Der »::«-Operator	228
3.10.8	Unterstützung von Visual Studio bei den Namespaces	229
3.10.9	Die Direktive »using static«	230
3.10.10	Die Direktive »global using«	231
3.11	Aktueller Stand der Klasse »Circle«	232

4	Vererbung, Polymorphie und Interfaces	235
4.1	Die Vererbung	235
4.1.1	Die Ableitung einer Klasse	236
4.1.2	Klassen, die nicht abgeleitet werden können	238
4.1.3	Konstruktoren in abgeleiteten Klassen	239
4.1.4	Der Zugriffsmodifizierer »protected«	240
4.1.5	Die Konstruktorverkettung in der Vererbung	241
4.2	Der Problemfall geerbter Methoden	244
4.2.1	Geerbte Methoden mit »new« verdecken	246
4.2.2	Abstrakte Methoden	248
4.2.3	Virtuelle Methoden	250
4.3	Typkonvertierung und Typuntersuchung von Objektvariablen	251
4.3.1	Die implizite Typkonvertierung von Objektreferenzen	252
4.3.2	Die explizite Typkonvertierung von Objektreferenzen	253
4.3.3	Typuntersuchung mit dem »is«-Operator	254
4.3.4	Typkonvertierung mit dem »as«-Operator	255
4.3.5	Pattern Matching (Musterabgleich) mit dem »is«-Operator	256
4.4	Polymorphie	257
4.4.1	Die »klassische« Methodenimplementierung	258
4.4.2	Abstrakte Methoden	259
4.4.3	Virtuelle Methoden	260
4.5	Weitere Gesichtspunkte der Vererbung	265
4.5.1	Versiegelte Methoden	265
4.5.2	Überladen einer Basisklassenmethode	266
4.5.3	Statische Member und Vererbung	267
4.5.4	Geerbte Methoden ausblenden?	267
4.6	Das Projekt »GeometricObjectsSolution« ergänzen	268
4.6.1	Die Klasse »GeometricObject«	268
4.7	Eingebettete Klassen	272
4.8	Interfaces (Schnittstellen)	272
4.8.1	Die Schnittstellendefinition	273
4.8.2	Die Schnittstellenimplementierung	274
4.8.3	Standardimplementierungen	280
4.8.4	Statische, abstrakte und virtuelle Mitglieder in Interfaces	282

4.8.5	Die Interpretation der Schnittstellen	283
4.8.6	Sortieren im Beispiel »GeometricObjectsSolution«	288
4.8.7	Weitere Anpassungen am Projekt »GeometricObjectsSolution«	290
4.9	Das Zerstören von Objekten – der Garbage Collector	292
4.9.1	Die Arbeitsweise des Garbage Collectors	292
4.9.2	Expliziter Aufruf des Garbage Collectors	294
4.9.3	Der Destruktor	294
4.9.4	Die »IDisposable«-Schnittstelle	296
4.9.5	Die »using«-Anweisung zum Zerstören von Objekten	298
4.10	Die Ergänzungen in den Klassen »Circle« und »Rectangle«	299

5 Delegaten, Ereignisse und Lambda-Ausdrücke 301

5.1	Delegaten	301
5.1.1	Einführung in das Prinzip der Delegaten	302
5.1.2	Verwendung von Delegaten	306
5.1.3	Vereinfachter Delegatenaufruf	306
5.1.4	Multicast-Delegaten	307
5.1.5	Kovarianz und Kontravarianz mit Delegaten	309
5.2	Ereignisse eines Objekts	311
5.2.1	Ereignisse bereitstellen	312
5.2.2	Die Reaktion auf ein ausgelöstes Ereignis	315
5.2.3	Allgemeine Betrachtungen der Ereignishandler-Registrierung	317
5.2.4	Wenn der Ereignisempfänger ein Ereignis nicht behandelt	318
5.2.5	Ereignisse mit Übergabeparameter	320
5.2.6	Ereignisse in der Vererbung	324
5.2.7	Ein Blick hinter die Kulissen des Schlüsselworts »event«	325
5.2.8	Die Schnittstelle »INotifyPropertyChanged«	327
5.3	Lambda-Ausdrücke	328
5.3.1	Anonyme Methoden	328
5.3.2	Lambda-Ausdrücke	330
5.3.3	Expression-bodied Member	332
5.3.4	Standardwerte für Parameter	333
5.3.5	Verwerfen von Parametern	333
5.4	Änderungen im Projekt »GeometricObjectsSolution«	334
5.4.1	Überarbeitung des Events »InvalidMeasure«	335
5.4.2	Weitere Ereignisse im Projekt »GeometricObjects«	336

6 Strukturen, Records und Enumerationsen 339

6.1	Strukturen – eine Sonderform der Klassen	339
6.1.1	Die Definition einer Struktur	339
6.1.2	Initialisieren einer Strukturvariablen	341
6.1.3	Konstruktoren in Strukturen	342
6.1.4	Datensatzstrukturen (Record Structs)	344
6.1.5	Änderungen im Projekt »GeometricObjectsSolution«	344
6.2	Datensatztypen – Klassen mit Standards	348
6.2.1	Die Definition und Initialisierung eines Records	348
6.2.2	Konstruktoren in Records	349
6.2.3	Unveränderlich oder nicht?	350
6.2.4	Vererbungen und Schnittstellen	351
6.2.5	Automatische »ToString«-Methode	352
6.3	Klassen, Strukturen oder Records?	352
6.4	Enumerationsen (Aufzählungen)	355
6.4.1	Wertzuweisung an »enum«-Variablen	356
6.4.2	Alle Mitglieder einer Aufzählung durchlaufen	356
6.5	Boxing und Unboxing	357

7 Fehlerbehandlung und Debugging 359

7.1	Laufzeitfehler erkennen	360
7.1.1	Die »try ... catch«-Anweisung	362
7.1.2	Behandlung mehrerer Exceptions	364
7.1.3	Die Reihenfolge der »catch«-Zweige	367
7.1.4	Ausnahmen in einer Methodenaufkette	367
7.1.5	Ausnahmen werfen oder weiterleiten	368
7.1.6	Die »finally«-Anweisung	368
7.1.7	Ausnahmefilter	370
7.1.8	Die Klasse »Exception«	371
7.1.9	Benutzerdefinierte Ausnahmen	377
7.2	Debuggen mit Programmcode	383
7.2.1	Einführung	383
7.2.2	Die Klasse »Debug«	384
7.2.3	Die Klasse »Trace«	387
7.2.4	Bedingte Kompilierung	388

7.3	Fehlersuche mit Visual Studio	391
7.3.1	Debuggen im Haltemodus	391
7.3.2	Weitere Alternativen, Variableninhalte zu prüfen	396

8 Auflistungsklassen (Collections) 399

8.1	Collections im Namespace »System.Collections«	399
8.1.1	Die elementaren Schnittstellen der Auflistungsklassen	401
8.2	Die Klasse »ArrayList«	403
8.2.1	Einträge hinzufügen	403
8.2.2	Datenaustausch zwischen einem Array und einer »ArrayList«	406
8.2.3	Die Elemente einer »ArrayList« sortieren	407
8.2.4	Sortieren von Arrays mit »ArrayList.Adapter«	413
8.3	Die Klasse »Hashtable«	415
8.3.1	Methoden und Eigenschaften der Schnittstelle »IDictionary«	415
8.3.2	Beispielprogramm zur Klasse »Hashtable«	416
8.4	Die Klassen »Queue« und »Stack«	421
8.4.1	Die Klasse »Stack«	422
8.4.2	Die Klasse »Queue«	423
8.5	Eigene Auflistungen mit »yield« durchlaufen	424
8.6	Collection Expressions	427

9 Generics – generische Datentypen 429

9.1	Bereitstellen einer generischen Klasse	431
9.1.1	Mehrere generische Typparameter	433
9.1.2	Vorteile der Generics	433
9.2	Bedingungen (Constraints) festlegen	434
9.2.1	Constraints mit der »where«-Klausel formulieren	434
9.2.2	Typparameter auf Klassen oder Strukturen beschränken	436
9.2.3	Mehrere Constraints definieren	436
9.2.4	Der Konstruktor-Constraint »new()«	436
9.2.5	Das Schlüsselwort »default«	437
9.3	Generische Methoden	438
9.3.1	Methoden und Constraints	439

9.4	Generische Attribute	439
9.5	Generics und Vererbung	441
9.5.1	Virtuelle generische Methoden	442
9.6	Typkonvertierung von Generics	443
9.7	Generische Delegaten	444
9.7.1	Generische Delegaten und Constraints	445
9.8	»Nullable«-Typen	445
9.8.1	Konvertierungen mit »Nullable«-Typen mit dem »??«-Operator	446
9.9	Generische Collections	447
9.9.1	Die Interfaces der generischen Auflistungsklassen	448
9.9.2	Die generische Auflistungsklasse »List<T>«	448
9.9.3	Vergleiche mit Hilfe des Delegaten »Comparison<T>«	451
9.9.4	Parametrische Sammlungen (»params«-Schlüsselwort)	452
9.10	Kovarianz und Kontravarianz generischer Typen	452
9.10.1	Kovarianz mit Interfaces	453
9.10.2	Kontravarianz mit Interfaces	455
9.10.3	Zusammenfassung	456
9.10.4	Generische Delegaten mit varianten Typparametern	457
9.11	Generische Mathematik	457
9.12	Ergänzungen im Beispielprojekt »GeometricObjectsSolution«	459
10	Weitere C#-Sprachfeatures	461
<hr/>		
10.1	Implizit typisierte Variablen	461
10.2	Anonyme Typen	462
10.3	Erweiterungsmethoden	463
10.3.1	Die Prioritätsregeln	464
10.3.2	Generische Erweiterungsmethoden	466
10.3.3	Richtlinien für Erweiterungsmethoden	467
10.4	Spezielle Methoden	467
10.4.1	Partielle Methoden	468
10.4.2	Lokale Funktionen und Funktionszeiger	471
10.5	Operatorüberladung	474
10.5.1	Einführung	474
10.5.2	Die Syntax der Operatorüberladung	475

10.5.3	Die Operatorüberladungen im Projekt »GeometricObjectsSolution«	476
10.5.4	Benutzerdefinierte Typkonvertierung	482
10.6	»Nullable«-Referenztypen	485
10.6.1	»Nullable«-Referenztypen im Beispiel »GeometricObjects«	486
10.7	Die »with«-Anweisung	490
10.8	Indexer	491
10.8.1	Überladen von Indexern	493
10.8.2	Parameterbehaftete Eigenschaften	495
10.9	Attribute	498
10.9.1	Das »Flags«-Attribut	500
10.9.2	Benutzerdefinierte Attribute	503
10.9.3	Attribute für Lambda-Ausdrücke	507
10.9.4	Attribute auswerten	508
10.9.5	Festlegen der Assembly-Eigenschaften in »AssemblyInfo.cs«	510
10.10	Der bedingte NULL-Operator	511
10.11	Der »nameof«-Operator	513
10.11.1	Einsatz in der Anwendung »GeometricObjects«	513
10.12	Dynamisches Binden	514
10.12.1	Eine kurze Analyse	516
10.12.2	Dynamische Objekte	516
10.13	Tupel	518
10.13.1	Benannte und unbenannte Tupel	518
10.13.2	Zuweisungsoperationen	519
10.13.3	Projektionsinitialisierer	520
10.13.4	Tupelvergleiche mit »==« und »!=«	520
10.13.5	Tupel als Methodenrückgabewerte	521
10.13.6	Dekonstruktion von Tupeln	522
10.13.7	Einzelne Tupelelemente ausschließen	523
10.14	Pattern Matching (Musterabgleich)	523
10.14.1	»switch« ohne »case«	525
10.14.2	Weitere Muster	527
10.15	Rückgabewerte mit »ref«	529
10.16	Unsicherer (unsafe) Programmcode – Zeigertechnik in C#	532
10.16.1	Einführung	532
10.16.2	Das Schlüsselwort »unsafe«	532
10.16.3	Die Deklaration von Zeigern	533

10.16.4 Die »fixed«-Anweisung	534
10.16.5 Zeigerarithmetik	535
10.16.6 Der Operator »->«	536

11 LINQ – Language Integrated Query 539

11.1 Einstieg in LINQ	539
11.1.1 Verzögerte Ausführung	541
11.1.2 LINQ-Erweiterungsmethoden an einem Beispiel	541
11.2 LINQ to Objects	545
11.2.1 Musterdaten	545
11.2.2 Die allgemeine LINQ-Syntax	547
11.3 Die Abfrageoperatoren	549
11.3.1 Übersicht der Abfrageoperatoren	549
11.3.2 Die »from«-Klausel	550
11.3.3 Mit »where« filtern	552
11.3.4 Die Projektionsoperatoren	554
11.3.5 Die Sortieroperatoren	555
11.3.6 Gruppieren mit »GroupBy«	557
11.3.7 Verknüpfungen mit »Join«	559
11.3.8 Die Set-Operatoren-Familie	562
11.3.9 Die Familie der Aggregatoperatoren	564
11.3.10 Quantifizierungsoperatoren	568
11.3.11 Aufteilungsoperatoren	569
11.3.12 Die Elementoperatoren	571
11.3.13 Die Konvertierungsoperatoren	574

12 Arbeiten mit Dateien und Streams 575

12.1 Einführung	575
12.2 Namespaces der Ein- bzw. Ausgabe	576
12.2.1 Das Behandeln von Ausnahmen bei E/A-Operationen	577
12.3 Laufwerke, Verzeichnisse und Dateien	577
12.3.1 Die Klasse »File«	578
12.3.2 Die Klasse »FileInfo«	584

12.3.3	Die Klassen »Directory« und »DirectoryInfo«	587
12.3.4	Die Klasse »Path«	592
12.3.5	Die Klasse »DriveInfo«	593
12.4	Die »Stream«-Klassen	594
12.4.1	Die abstrakte Klasse »Stream«	595
12.4.2	Die von »Stream« abgeleiteten Klassen im Überblick	598
12.4.3	Die Klasse »FileStream«	599
12.5	Die Klassen »TextReader« und »TextWriter«	606
12.5.1	Die Klasse »StreamWriter«	607
12.5.2	Die Klasse »StreamReader«	610
12.6	Die Klassen »BinaryReader« und »BinaryWriter«	612
12.6.1	Komplexe binäre Dateien	615
12.7	Serialisierung	621
12.7.1	Serialisierungsverfahren	622
12.7.2	Binäre Serialisierung mit »BinaryFormatter«	623
12.7.3	Serialisierung mit »XmlSerializer«	624
12.7.4	XML-Serialisierung mit Attributen steuern	626

13 Multithreading 631

13.1	Einführung in das Multithreading	632
13.2	Threads – allgemein betrachtet	633
13.3	Mit der Klasse »Thread« arbeiten	635
13.3.1	Die Entwicklung einer einfachen Multithreading-Anwendung	635
13.3.2	Der Delegat »ParameterizedThreadStart«	637
13.3.3	Zugriff eines Threads auf sich selbst	638
13.3.4	Einen Thread für eine bestimmte Zeitdauer anhalten	638
13.3.5	Beenden eines Threads	639
13.3.6	Abhängige Threads – die Methode »Join«	642
13.3.7	Threadprioritäten festlegen	643
13.3.8	Vorder- und Hintergrundthreads	646
13.4	Der Threadpool	647
13.4.1	Ein einfaches Beispielprogramm	647
13.5	Synchronisation von Threads	648
13.5.1	Möglichkeiten der Synchronisation	650

13.5.2	Die Klasse »WaitHandle«	651
13.5.3	Sperren mit »Monitor«	655
13.5.4	Die Klasse »Mutex«	663
13.5.5	Die Klasse »Semaphore«	665
13.5.6	Der »System.Threading.Er«-Typ	670
13.5.7	Das Attribut »MethodImpl«	671
13.5.8	Die Klasse »Interlocked«	672
13.5.9	Synchronisation von Threadpool-Threads	672
13.6	Grundlagen asynchroner Methodenaufrufe	673
13.6.1	Asynchroner Methodenaufruf	675
13.6.2	Asynchroner Aufruf mit Rückgabewerten	679
13.6.3	Eine Klasse mit asynchronen Methodenaufrufen	682

14 Die Task Parallel Library (TPL) und das Task-based Async Pattern (TAP) 687

14.1	Die wichtigsten Klassen der TPL	688
14.2	Die Klasse »Task«	688
14.2.1	Die Konstruktoren eines Tasks	690
14.2.2	Das Erzeugen eines Tasks	691
14.2.3	Tasks mit Rückgabewert	692
14.2.4	Daten an einen Task übergeben	692
14.2.5	Auf das Beenden eines Tasks warten	694
14.2.6	Abbruch eines Tasks von außen	695
14.2.7	Bei Abbruch eine Callback-Methode aufrufen	697
14.2.8	Fehlerbehandlung	699
14.3	Die Klasse »Parallel«	701
14.3.1	Die Methode »Parallel.Invoke«	702
14.3.2	Schleifen mit »Parallel.For«	702
14.3.3	Den Grad der Parallelität beeinflussen	707
14.3.4	Auflistungen mit »Parallel.ForEach« durchlaufen	708
14.4	Asynchrone Programmierung mit »async« und »await«	708
14.4.1	Die Arbeitsweise von »async« und »await« verstehen	709
14.4.2	Asynchrone Operationen mit Rückgabewert	713

15 Grundlegende .NET-Klassen 715

15.1 Die Klasse »Object«	715
15.1.1 Referenzvergleiche mit »Equals« und »ReferenceEquals«	716
15.1.2 »ToString« und »GetType«	717
15.1.3 Die Methode »MemberwiseClone« und das Problem des Klonens	717
15.2 Die Klasse »String«	721
15.2.1 Das Erzeugen eines Strings	722
15.2.2 Die Eigenschaften von »String«	723
15.2.3 Die Methoden der Klasse »String«	723
15.2.4 Zusammenfassung der Klasse »String«	734
15.3 Die Klasse »StringBuilder«	736
15.3.1 Die Kapazität eines »StringBuilder«-Objekts	737
15.3.2 Die Konstruktoren der Klasse »StringBuilder«	738
15.3.3 Die Eigenschaften der Klasse »StringBuilder«	738
15.3.4 Die Methoden der Klasse »StringBuilder«	739
15.3.5 Allgemeine Anmerkungen	741
15.4 Der Typ »DateTime«	742
15.4.1 Die Zeitspanne »Tick«	742
15.4.2 Die Konstruktoren von »DateTime«	743
15.4.3 Die Eigenschaften von »DateTime«	744
15.4.4 Die Methoden der Klasse »DateTime«	745
15.5 Die Klasse »TimeSpan«	747
15.6 Die Klassen »DateOnly« und »TimeOnly«	750
15.7 Ausgabeformatierung	751
15.7.1 Formatierung mit der Methode »String.Format«	751
15.7.2 Formatierung mit der Methode »ToString«	755
15.7.3 Benutzerdefinierte Formatierung	756

16 Einführung in das Entity Framework Core 761

16.1 Was ist das Entity Framework Core?	761
16.1.1 Objektrelationale Abbildung (ORM)	763
16.1.2 Vorteile und Ziele	764
16.1.3 Migration von EF 6 nach EF Core	766

16.2 Grundkonzepte und Projekt-Setup von EF Core	767
16.2.1 »DbContext« und »DbSet«	768
16.2.2 Modellklassen (POCOs)	769
16.2.3 Konventionen bei Primär- und Fremdschlüsseln	770
16.2.4 Change Tracking	771
16.2.5 Migrationen	771
16.2.6 LINQ to Entities	772
16.3 Unser erstes Projekt	772
16.3.1 Anlegen des Datenmodells	773
16.3.2 Zugriff auf PostgreSQL	774
16.4 Datenabfragen mit LINQ	783
16.5 Weitere Datenbankzugriffe	786
16.5.1 Datenselektionen über Filter	786
16.5.2 Projektionen	787
16.5.3 Sortieren von Daten	788
16.5.4 Limitierungen und Aggregationen	788
16.5.5 Gruppierung und Existenzprüfung	789
16.5.6 »null«-Werte, COALESCE und DISTINCT	790
16.5.7 Roh-SQL für manuelle Abfragen	791
16.6 Eager, Lazy und Explicit Loading	791
16.6.1 Eager Loading	791
16.6.2 Lazy Loading	792
16.6.3 Explicit Loading	793
16.6.4 Tracking vs. No Tracking	793
16.7 Modellierung von Beziehungen	794
16.7.1 Eins-zu-viele-Beziehungen (1:n)	794
16.7.2 Eins-zu-eins-Beziehungen (1:1)	797
16.7.3 Viele-zu-viele-Beziehungen (m:n)	799
16.7.4 Fluent API vs. Data Annotations	801
16.7.5 Zusammengesetzte Schlüssel und Fremdschlüssel	802
16.8 Die Arbeit mit Migrationen	802
16.8.1 Grundprinzip: Code ist das Datenbankschema	803
16.8.2 Migrationen erzeugen und anwenden	803
16.8.3 Migrationen im Entwicklungsalltag	805
16.8.4 Migrationen rückgängig machen	805
16.8.5 Best Practices	806
16.9 Data Seeding und Datenbankinitialisierung	806
16.9.1 Grundlagen des Data Seeding	807

16.9.2	Seeding von Beziehungen	808
16.9.3	Manuelles Seeding zur Laufzeit	809
16.10	Modellierung von Vererbung und Polymorphie	809
16.10.1	Table-per-Hierarchy (TPH)	811
16.10.2	Table-per-Type (TPT)	811
16.10.3	Table-per-Concrete-Type (TPC)	812
16.11	Performance und Tuning	813
16.11.1	Der Einfluss von LINQ	813
16.11.2	Split Queries vs. Single Queries	814
16.11.3	Caching; Bewusst einsetzen	814
16.12	EF Core und moderne .NET-Features	815
16.12.1	Asynchrone Programmierung	815
16.12.2	Nullable Reference Types	816
16.12.3	Records, Init-only-Properties und C#-Features	816
17	Desktop- und Cross-Platform-Entwicklung	819
17.1	WinForms, WPF und .NET MAUI – ein Vergleich	819
17.2	Und was ist mit der UWP und WinUI 3?	821
17.3	XAML (Extended Application Markup Language)	823
17.3.1	Die Struktur einer XAML-Datei	823
17.3.2	Eigenschaften eines XAML-Elements in Attributschreibweise festlegen	826
17.3.3	Eigenschaften im Eigenschaftsfenster festlegen	826
17.3.4	Die Eigenschaft-Element-Syntax	827
17.3.5	Inhaltseigenschaften	828
17.3.6	Typkonvertierung	831
17.3.7	Markup-Erweiterungen (Markup Extensions)	833
17.3.8	XML-Namespaces	835
17.3.9	XAML-Spracherweiterungen	838
17.3.10	Die Direktive »#region« nutzen	840
17.4	Grundkonzepte von Windows Forms (WinForms)	840
17.4.1	Grundlegender Aufbau einer WinForms-Anwendung	841
17.4.2	Überblick über die Projektstruktur	843
17.4.3	Dialoge und Meldungsfenster in Windows Forms	844

18	Windows Presentation Foundation	845
18.1	Eine WPF-Anwendung und ihre Dateien	845
18.2	Fenster in der WPF	847
18.3	Nachrichtenfenster mit »MessageBox«	848
18.4	Standarddialoge in der WPF	850
18.5	Die WPF-Layoutcontainer	850
18.5.1	Das »Canvas«	852
18.5.2	Das »StackPanel«	852
18.5.3	Der »Grid« Layoutcontainer	855
18.6	WPF-Steuerelemente	857
18.6.1	Die Gruppe der Schaltflächen	857
18.6.2	Das Steuerelement »Label«	859
18.6.3	Das Steuerelement »TextBox«	860
18.6.4	WPF-Listenelemente	862
18.7	Ereignisse in der WPF	866
18.7.1	Ereignishandler bereitstellen	866
18.7.2	Routing-Strategien	868
18.8	Die Ereignishandler	870
18.8.1	Die Klasse »RoutedEventArgs«	870
18.8.2	Die Quelle des Routing-Prozesses	871
18.8.3	Die Eigenschaft »Handled«	872
18.8.4	Registrieren und Deregistrieren eines Ereignishandlers mit Code	872
18.9	WPF-Datenbindung und WPF-Commands	873
18.10	Die Klasse »Binding«	876
18.10.1	Binden an eine Datenquelle	877
18.10.2	Mit »Path« an eine Eigenschaft der Datenquelle binden	880
18.10.3	Die Bindungsrichtung mit »Mode« festlegen	882
18.10.4	Das Aktualisieren mit »UpdateSourceTrigger« steuern	884
18.10.5	Die Ereignisse »SourceUpdated« und »TargetUpdated«	885
18.10.6	Beenden einer Bindung	886
18.11	WPF-Commands	887
18.11.1	Die Befehlsquelle	890
18.11.2	Die Arbeitsweise eines Befehls	893
18.11.3	Die Klasse »CommandManager«	894

18.12 »RoutedCommand«-Objekte und »CommandBindings«	898
18.12.1 Vordefinierte WPF-Commands	898
18.12.2 Die Klasse »RoutedCommand«	899
18.12.3 Befehlsbindungen mit »CommandBinding« einrichten	900
18.13 Das MVVM-Pattern	902
18.13.1 Die Theorie hinter dem Model-View-ViewModel-Pattern	903
18.13.2 Allgemeine Beschreibung des Beispielprogramms	904
18.13.3 Das Bereitstellen des Modells	906
18.13.4 Bereitstellen des ViewModels	908
19 Einführung in .NET MAUI	911
<hr/>	
19.1 Was ist .NET MAUI?	911
19.2 Architektur von .NET MAUI im Kontext von .NET	913
19.3 Struktur eines MAUI-Projekts	914
19.3.1 Überblick über die Projektdateien und -struktur	914
19.3.2 Wie MAUI mit XAML und C# kombiniert arbeitet	920
19.4 Eine erste Anwendung	921
19.4.1 »Hello World« mit MAUI	922
19.4.2 Live-Preview und Hot Reload	924
19.5 Zentrale Konzepte	925
19.5.1 Architekturmodelle: MVU und MVVM	925
19.5.2 Layouts: Strukturierung der Oberfläche	926
19.5.3 Controls: Grundbausteine der Oberfläche	927
19.5.4 Zusammenspiel dieser Konzepte	931
19.6 Datenbindung und MVVM mit .NET MAUI	932
19.6.1 Datenbindung: Grundlage für MVVM	932
19.6.2 Ein einfaches MVVM-Beispiel in .NET MAUI	933
19.6.3 Zwei-Wege-Bindung (Two-Way Binding)	936
19.6.4 Validierung	937
19.6.5 Navigation	938
19.6.6 MVVM mit dem »CommunityToolkit.Mvvm«	938
19.7 Plattformunabhängigkeit und plattformspezifischer Code	940
19.7.1 Zugriff auf native APIs: Beispiel mit Vibrationsfunktion (Android)	940
19.8 Debugging und Deployment	942
19.8.1 Debugging auf Emulatoren und echten Geräten	942
19.8.2 Deployment: Grundlagen für Android, Windows und iOS/macOS	943

20	2D-Grafik in WPF und .NET MAUI	945
20.1	Shapes	945
20.1.1	Allgemeine Beschreibung	945
20.1.2	»Line«-Elemente	946
20.1.3	»Ellipse«- und »Rectangle«-Elemente	947
20.1.4	»Polygon«- und »Polyline«-Elemente	947
20.1.5	Darstellung der Linien	947
20.2	»Path«-Elemente	949
20.2.1	Das Element »GeometryGroup«	950
20.2.2	Das Element »CombinedGeometry«	951
20.2.3	Geometrische Figuren mit »PathGeometry«	952
20.3	»Brush«-Objekte	953
20.3.1	»SolidColorBrush«	954
20.3.2	»LinearGradientBrush«	955
20.3.3	»RadialGradientBrush«	957
20.3.4	Muster mit »TileBrush«	959
20.3.5	Bilder mit »ImageBrush«	961
20.3.6	Effekte mit »VisualBrush«	962
20.3.7	Das Element »DrawingBrush«	964
20.4	2D-Grafik in .NET MAUI	965
20.4.1	Shapes – einfache Formen direkt in XAML	965
20.4.2	»Path« – komplexe Vektorformen	965
20.4.3	»GraphicsView« – flexibles, manuelles Zeichnen	966
21	Komponententests (Unit-Tests)	967
21.1	Was ist ein Unit-Test?	967
21.2	Ein erster Komponententest	970
21.2.1	Das »AAA«-Prinzip	972
21.2.2	Automatisches Generieren eines Testprojekts	974
21.2.3	Ausführen eines Unit-Tests	975
21.3	Komponententest schreiben und ausführen	977
21.3.1	Unit-Tests gruppieren	977
21.3.2	Merkmale festlegen	978
21.3.3	Wiedergabelisten	985
21.3.4	Exceptions testen	986

21.3.5	Codeabdeckung (Code Coverage)	988
21.3.6	Testen von privaten Methoden	991
21.3.7	Die Attribute »TimeOut« und »Ignore«	993
21.3.8	Komponententests debuggen	994
21.4	Die Klasse »TestContext«	995
21.5	Data-Driven Unit-Tests (datengetriebene Tests)	998
21.5.1	Eine Datenbank als Datenquelle	1000
21.5.2	Data-Driven Test mit XML	1002
21.5.3	Data-Driven Test mit einer CSV-Datei	1004
21.5.4	Die Datenquelle in der »App.config«-Datei konfigurieren	1006
21.5.5	Testdaten mit dem Attribut »DataRow«	1008
21.5.6	Testdaten mit dem Attribut »DynamicData«	1010
21.6	Lebenszyklus-Attribute	1011
21.6.1	Die Attribute »TestInitialize« und »TestCleanup«	1012
21.6.2	Die Attribute »ClassInitialize« und »ClassCleanup«	1014
21.6.3	Die Attribute »AssemblyInitialize« und »AssemblyCleanup«	1015
21.7	Testen mit »Assert«	1016
21.7.1	Die Klasse »Assert«	1017
21.7.2	Die Klasse »StringAssert«	1020
21.7.3	Die Klasse »CollectionAssert«	1022
21.8	Test-Driven Development – TDD	1025
22	Einführung in ASP.NET Core	1027
<hr/>		
22.1	Was ist ASP.NET Core?	1027
22.1.1	REST-APIs	1029
22.1.2	ASP.NET Core Web App (MVC) als klassische Webanwendung	1029
22.1.3	Minimal APIs	1029
22.1.4	ASP.NET Core Web App (Razor Pages)	1030
22.1.5	Blazor	1030
22.1.6	Hintergrunddienste, WebSockets, SignalR	1031
22.1.7	Gemeinsamkeiten und Unterschiede	1031
22.2	Web-API-Projektstruktur und erste Schritte	1032
22.2.1	Projektstruktur am Web-API-Beispiel	1035
22.2.2	Der Lebenszyklus einer ASP.NET-Core-Anwendung	1039
22.2.3	Ordnerstruktur und Clean Architecture	1039

22.3 Minimal APIs vs. klassische Controller-Architektur	1040
22.3.1 Controller mit POST und Antwort	1041
22.3.2 Nachteile von Minimal APIs	1045
22.3.3 Nachteile von Controllern	1046
22.3.4 Auswahl eines Architekturstils	1046
22.4 Routing und Middleware	1047
22.4.1 Was ist die Middleware?	1047
22.4.2 Das Routing-System	1049
22.4.3 Zusammenspiel von Routing und Middleware	1050
22.4.4 Filter	1050
22.4.5 Route Groups	1052
22.4.6 Eigene Middleware erstellen	1053
22.5 Dependency Injection	1054
22.5.1 Was ist Dependency Injection?	1055
22.5.2 Registrierung von Diensten	1055
22.5.3 Nutzung im Code	1056
22.5.4 Erweiterte Nutzung und Testbarkeit	1057
22.6 Konfiguration und Umgebungen	1057
22.6.1 Das Konfigurationssystem von ASP.NET Core	1058
22.6.2 Das Options Pattern	1059
22.6.3 Umgebungen (Environments)	1060
22.6.4 Umgebungsabhängige Konfiguration	1061
22.7 Entity Framework Core Integration	1061
22.7.1 Nutzung in Controllern	1062
22.7.2 Nutzung in Minimal APIs	1066
22.8 Authentifizierung und Autorisierung	1068
22.8.1 Authentifizierung vs. Autorisierung – ein grundlegender Unterschied	1068
22.8.2 Authentifizierungsmechanismen in ASP.NET Core	1069
22.8.3 JWT-basierte Authentifizierung – der moderne Standard für APIs	1069
22.8.4 Autorisierung – feiner differenzieren, was erlaubt ist	1070
22.8.5 ASP.NET Core Identity – Benutzerverwaltung »aus der Box«	1071
22.9 Fehlerbehandlung und Logging	1072
22.9.1 Fehlerbehandlung: Kontrolliert statt chaotisch	1072
22.9.2 Exception Handling Middleware selbst gestalten	1073
22.9.3 Logging – strukturierte Informationen statt Konsolenausgabe	1074
22.9.4 Externe Logging-Tools	1075
22.9.5 Logging in der Produktion	1076

22.10 OpenAPI/Swagger	1076
22.10.1 Was ist OpenAPI?	1076
22.10.2 Was ist Swagger?	1077
22.10.3 Konfigurationen – Beschreibungstexte und Anmerkungen hinzufügen	1080
22.11 Deployment und Hosting	1081
22.11.1 Selbstgehostet: Kestrel und das neue Hosting-Modell	1082
22.11.2 Deployment unter Windows mit IIS	1082
22.11.3 Deployment unter Linux mit Nginx	1083
22.11.4 Self-contained Deployment	1083
22.11.5 Deployment mit Docker (Containerisierung)	1083
22.11.6 Continuous Deployment (CD) mit GitHub Actions	1084
22.12 Unit-Testing von ASP.NET-Core-Anwendungen	1085
22.12.1 Projektstruktur für Tests	1085
22.12.2 Einen Dienst testen: Beispiel mit Mocking	1086
22.12.3 Integrationstests mit »WebApplicationFactory«	1087
22.12.4 Datenbankabhängige Tests mit In-Memory-Datenbank	1088

23 Einführung in Blazor 1089

23.1 Was ist Blazor?	1089
23.1.1 Blazor Server	1090
23.1.2 Blazor WebAssembly (Blazor WASM)	1091
23.1.3 Blazor Hybrid	1092
23.1.4 Blazor Auto Render Mode (ehemals Blazor United)	1092
23.2 Wie funktioniert Blazor?	1093
23.2.1 Komponentenbasiertes Modell	1093
23.2.2 Blazor Server vs. WebAssembly – Laufzeitmodelle	1094
23.2.3 Der Lebenszyklus einer Komponente	1095
23.3 Projektstruktur und Build-System	1096
23.3.1 Tooling und Entwicklungserfahrung	1103
23.3.2 Debugging von Blazor-Anwendungen	1104
23.3.3 Build und Deployment	1105
23.4 Blazor-Komponenten im Detail	1106
23.4.1 Aufbau einer Komponente	1106
23.4.2 Parameter und Bindings	1107
23.4.3 Event-Handling	1108

23.4.4	Komponenten verschachteln und wiederverwenden	1109
23.4.5	Code-Behind: Logik auslagern	1109
23.5	Routing und Navigation	1110
23.5.1	Routing über die »@page«-Direktive	1110
23.5.2	Navigation mit »NavLink« und »NavigationManager«	1111
23.5.3	Parameter in Routen	1111
23.5.4	Navigation in Blazor Server vs. WebAssembly	1112
23.5.5	Fehlerseiten und Fallbacks	1113
23.6	Datenbindung und Formulare	1114
23.6.1	One-Way- und Two-Way-Binding	1114
23.6.2	Formulare mit »EditForm«	1114
23.6.3	Validierung mit »DataAnnotations«	1117
23.6.4	Validierungsereignisse	1118
23.6.5	Eigene Validierungslogik	1118
23.7	State Management	1118
23.7.1	Lokaler Zustand in Komponenten	1119
23.7.2	Gemeinsamer Zustand mit Services	1119
23.7.3	Zustand über Seiten hinweg erhalten	1120
23.7.4	»StateHasChanged« und UI-Aktualisierung	1120
23.7.5	Fortgeschritten: Zentraler Zustand mit Events	1121
23.8	Interop mit JavaScript	1122
23.8.1	JavaScript-Dateien einbinden	1122
23.8.2	JavaScript aus C# aufrufen	1122
23.8.3	C# aus JavaScript aufrufen	1123
23.8.4	Typkompatibilität und Marshalling	1124
23.8.5	Besonderheiten bei WebAssembly und Blazor Server	1124
23.8.6	Eine JS-Bibliothek einbinden	1125
23.9	Zugriff auf APIs und Backend	1125
23.9.1	Der HttpClient in Blazor	1125
23.9.2	»HttpClient« bereitstellen und konfigurieren	1127
23.9.3	Fehlerbehandlung und Statusausgabe	1127
23.9.4	Daten an das Backend senden	1128
23.9.5	Grundlagen der Authentifizierung und Autorisierung	1128
23.9.6	Eigenes Backend mit Minimal APIs	1129
23.10	Blazor Hybrid für den Desktop	1129
23.10.1	Was ist Blazor Hybrid?	1130
23.10.2	Blazor Hybrid mit .NET MAUI	1130
23.10.3	Blazor Hybrid mit der WPF	1131
23.10.4	Einschränkungen und Anwendungsfälle	1132

23.11 Fortgeschrittene Themen	1132
23.11.1 SEO mit Blazor	1133
23.11.2 Komponentenbibliotheken verwenden	1133
23.11.3 Testing von Blazor-Komponenten	1133

24 Cloud-Entwicklung mit Azure 1135

24.1 Was ist Microsoft Azure?	1135
24.2 Überblick über Cloud-Computing	1136
24.3 Cloud-Service-Modelle: PaaS, IaaS, und SaaS	1138
24.4 Vorteile der Cloud-Entwicklung mit Azure	1141
24.5 Azure-Ressourcenmanagement und -Architektur	1142
24.6 Schlüsseldienste: Azure Compute, Storage, Databases und KI	1147
24.7 Azure CLI und Azure PowerShell	1152
24.8 Azure SDKs für .NET	1155
24.9 Nutzung des Azure-Portals	1159
24.10 Azure SQL Database und Cosmos DB	1159
24.11 Azure Blob Storage	1164
24.12 Azure Functions und Logic Apps	1167
24.13 Azure Monitor und Application Insights	1172
24.14 Einrichten von Azure DevOps	1176

25 Softwareentwicklung in Zeiten von KI 1179

25.1 Warum KI in der Softwareentwicklung?	1179
25.2 Grundlagen der künstlichen Intelligenz	1180
25.2.1 Von KI zu Machine Learning und Deep Learning	1181
25.2.2 Lernarten: Überwacht, unüberwacht, verstärkend	1181
25.2.3 Arten generativer KI-Modelle und ihre Verfahren	1183
25.2.4 Generative Modelle im Bereich Text- und Codegenerierung	1184
25.3 KI-Plattformen und Tools von Microsoft	1185
25.3.1 Azure AI Services: KI aus der Cloud	1186
25.3.2 ML.NET: Machine Learning direkt in .NET	1187

25.3.3	Azure KI-Suche	1187
25.3.4	Azure KI Speech	1188
25.3.5	Azure KI Vision	1188
25.3.6	Azure OpenAI: Zugang zu leistungsfähigen Sprachmodellen	1188
25.4	Machine Learning mit ML.NET	1189
25.4.1	Der typische Ablauf in ML.NET	1189
25.4.2	Ein Beispiel: Kundenaussagen bewerten (Sentiment Analysis)	1190
25.5	Azure AI Services und Azure OpenAI in C# nutzen	1198
25.5.1	Bilderkennung mit Azure KI Vision	1198
25.5.2	Azure OpenAI: Zugriff auf generative KI	1203
25.6	Generative KI nutzen	1204
25.6.1	Typische Modelle: GPT, DALL-E, Codex	1205
25.6.2	Modelle von Microsoft	1206
25.6.3	Anwendungsfelder generativer KI in der Softwareentwicklung	1207
25.6.4	Integration von generativer KI in .NET-Anwendungen	1209
25.6.5	Prompt Engineering: Der neue Skill für Entwickelnde	1211
25.6.6	Generative KI lokal nutzen: ONNX, kleine Modelle und Hybridansätze	1212
25.7	Microsoft.Extensions.AI und Microsoft.Extensions.VectorData	1214
25.7.1	Was ist Microsoft.Extensions.AI?	1215
25.7.2	Was ist Microsoft.Extensions.VectorData?	1216
25.8	Einsatz von KI in der Softwareentwicklung selbst	1218
25.8.1	GitHub Copilot: KI als Programmierassistent	1219
25.8.2	GitHub Copilot in Visual Studio	1220
25.8.3	KI-gestützte Codeanalyse und Refactoring	1225
25.8.4	Automatisierte Tests und Testdatengenerierung	1226
25.8.5	Unterstützung bei Dokumentation und Codeverständnis	1226
25.9	Ethik, Datenschutz und Verantwortung in der KI-Entwicklung	1227
25.9.1	Fairness und Bias: Wenn Modelle Vorurteile lernen	1227
25.9.2	Datenschutz: Sensible Daten verantwortungsvoll nutzen	1228
25.9.3	Nachvollziehbarkeit und Transparenz	1228
25.10	Best Practices und typische Fallstricke	1229
25.10.1	Verstehen, was KI leisten kann – und was nicht	1229
25.10.2	Datenqualität: Fundament jedes KI-Projekts	1230
25.10.3	Modellwahl und Parametrierung mit Bedacht	1230
25.10.4	Evaluation ist keine Nebensache	1230
25.10.5	Skalierung, Performance und Betrieb	1231
25.10.6	Dokumentation und Wartbarkeit nicht vergessen	1231
Index	1233