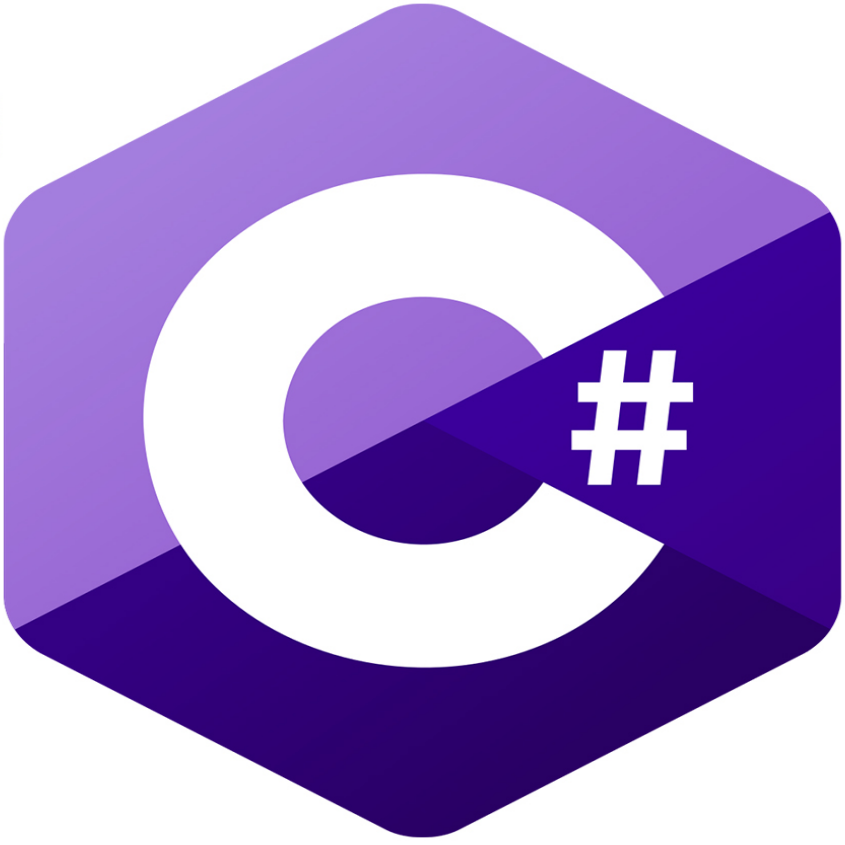
KOMPLETT IN FARBE

Robert Schiefele

C# KOMPENDIUM

PROFESSIONELL C# PROGRAMMIEREN LERNEN

OHNE VORWISSEN LOSLEGEN



Eine umfassende Einführung in C#!

- Alle Grundlagen der Programmierung verständlich erklärt
- OOP, LinQ, Datenbanken, ASP.NET, WPF, MS-Test, u.v.m.
- Übungsaufgaben mit Musterlösungen nach jedem Kapitel

Inklusive eBook zum Download

{BMU VERLAG}

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Informationen sind im Internet über http://dnb.d-nb.de abrufbar.

www.bmu-verlag.de
info@bmu-verlag.de

Lektorat: Lektormeister
Einbandgestaltung: Pro ebookcovers Angie
Druck und Bindung: Wydawnictwo Poligraf sp. zo.o. (Polen)

Taschenbuch: 978-3-96645-076-8
Hardcover: 978-3-96645-077-5
E-Book: 978-3-96645-075-1

C# Kompendium

Inhaltsverzeichnis

Kapitel 1
Einleitung

Dieses Buch beschäftigt sich mit der Programmiersprachsprache C# und dem .NET-Framework. Zudem werden die folgenden Programmierschwerpunkte umfassend behandelt:

- Erstellen von Konsolen-Apps
- Objekte verarbeiten mit Linq to Objects
- Fehlerbehandlung mit Exceptions
- Dateizugriff mit C#

Wie der Titel schon sagt, ist der Hauptschwerpunkt dieses Buchs die Programmiersprache C# selbst. Diese möchte ich möglichst umfassend mit all ihren Facetten behandeln. Des Weiteren möchte ich Ihnen noch ein paar Programmierthemen vorstellen, die für die Programmierung mit C# und dem .NET-Framework von großer Bedeutung sind. Allerdings sind diese Themen so umfangreich, dass eine vollständige Behandlung jedes einzelnen Aspekts mit all seinen Details ein eigenes Buch rechtfertigen würde. Daher werde ich die folgenden Themen nur jeweils mit einem eigenen Kapitel anreißen.

- Datenbankzugriff am Beispiel des Microsoft SQL-Servers
- ASP.NET MVC-Anwendungen fürs Web
- Grafische Benutzeroberflächen mit WPF
- Testautomatisierung mit MS-Test

1.1 Über den Autor

Der Autor Robert Schiefele hat 1981 im Alter von 15 Jahren damit begonnen, die Programmierung im Selbststudium zu erlernen. Die ersten Schritte machte er mit der Programmiersprache Basic, welche damals noch in der Form eines Kommandozeilen-Interpreters vorlag. Auch die Grundlagen der Assembler-Programmierung brachte er sich als Schüler selbst bei. Während seines Informatik-Studiums erlernte er dann die Programmiersprache Pascal. In studentischen Nebentätigkeiten kam er mit Turbo Pascal zum ersten Mal mit der Objektorientierten Programmierung in Berührung. Während dieser Nebentätigkeiten beschäftigte er sich mit Hilfe von Microsoft Access und dem Microsoft SQL-Server auch das erste Mal mit relationalen Datenbanken. Nach dem Studium erweiterte er seine Programmierkenntnisse als festangestellter

Datenbankentwickler um Microsoft Visual Basic und die Programmierung von Oracle-Datenbanken. Später nahm er eine Position als Berater für DevOps, was zu dieser Zeit noch ALS hieß, an. Dort lernte er auch die Programmiersprache Java. Ab der Jahrtausendwende arbeitete er als Berater für DevOps und Business Process Management. Während dieser Tätigkeit lernte er auch den Microsoft .NET Framework kennen. Zunächst arbeitete er mit VB.NET, stieg dann aber wegen der syntaktischen Ähnlichkeit mit Java auf die Programmiersprache C# um, welche er seitdem als seine bevorzugte Programmiersprache betrachtet.

Seit November 2016 arbeitet er als freiberuflicher Software-Entwickler ausschließlich in der .NET Programmierung – sowohl im Bereich der Web-Applikationen mit ASP.NET MVC als auch im Bereich der Desktopapplikationen mit WPF. Im Juli 2018 gründete er die Firma Contigo UG & Co. IT-Consulting KG, welche sich die Software-Entwicklung mit .NET auf die Fahnen geschrieben hat.

Privat lebt der Autor im Großraum München, ist verheiratet, hat zwei Kinder und beschäftigt sich in seiner Freizeit am liebsten mit seinem Garten.

1.2 Warum C#?

Wikipedia bezeichnet C# als eine streng typisierte, objektorientierte Allzweck-Programmiersprache. Diese Bezeichnung liefert uns schon mal drei gute Gründe, warum Sie mit C#-Kenntnissen bestens gerüstet sind, egal ob Sie in Ihrer Freizeit oder in Ihrem Beruf in Teil- oder in Vollzeit Software entwickeln.

Streng typisiert bedeutet, alle Konstrukte der Programmiersprache, mit denen man in irgendeiner Form Daten verarbeiten kann, haben einen fest zugeordneten Typ. Das heißt vereinfacht ausgedrückt: Ein Text ist ein Text und keine Zahl und zu einem Text können Sie auch nicht die Zahl 5 addieren. Strenge Typisierung ist ein wichtiges Hilfsmittel, wenn es darum geht, den sogenannten „Spaghetti-Code“ zu vermeiden. „Spaghetti-Code“ ist ein Ausdruck, der sagen möchte, dass ein Programm logisch so durcheinander ist wie ein Teller Spaghetti. Wenn Sie an solchen Programmen Veränderungen, Korrekturen oder Erweiterungen vornehmen müssen, ist das eine sehr mühsame und zeitraubende Angelegenheit.

Der zweite gute Grund: C# ist objektorientiert. Heutzutage kann man die Objektorientierung als State of the Art in der Programmierung bezeichnen. Sämtliche Programmiersprachen, die heute von Bedeutung sind, wie C++, Java, Python, TypeScript, JavaScript, VB.NET und natürlich C#, sind objektorientiert. Gerade wenn Sie sich beruflich mit der Programmierung beschäftigen wollen oder müssen, kommen Sie um dieses Konzept nicht herum. Die Objektorientierung ist zur Vermeidung von Spaghetti-Codes das wichtigste Hilfsmittel. Wie Objektorientierung genau funktioniert, werden wir in diesem Buch noch ausführlich erörtern.

Der dritte gute Grund: C# ist eine Allzweck-Programmiersprache. Mit C# können Sie alles Mögliche programmieren. Folgende Arten von Programmen können mit C# realisiert werden:

- Konsolen-Apps für Windows, Linux und MacOS
- Apps mit grafischer Benutzeroberfläche für Windows, Android und iOS
- Web-Applikationen
- Dienste, die im Hintergrund laufen für Windows und Linux
- Spiele für Windows, Android, iOS, Xbox, PS4, Nintendo Switch, Steam und Web

Ein weiterer Grund für C# hat in den letzten Jahren zunehmend an Bedeutung gewonnen. C# ist unabhängig von Plattformen. Damit ein C#-Programm auf einem Computer laufen kann, benötigt es lediglich ein .NET-Framework, das für das jeweilige Betriebssystem dieses Computers geschrieben wurde. Für die Betriebssysteme Windows, Linux, Android, MacOS und iOS gibt es bereits angepasste Versionen des .NET-Frameworks. Mit C# lassen sich auch Programme schreiben, die auf kleinen, günstigen Einplatinencomputern wie dem Raspberry Pi laufen. Das ist möglich, da es für den Raspberry Pi speziell angepasste Linux- und Windows-Versionen gibt. Allerdings gibt es für Linux-Betriebssysteme eine Einschränkung für den Einsatz von C#: Desktop-Programme, die eine grafische Benutzeroberfläche verwenden, konnten zum Zeitpunkt der Erstellung dieses Buches noch nicht für Linux in C# erstellt werden, da noch kein .NET-Framework für Linux zur Verfügung steht, das die grafischen Fähigkeiten von Linux unterstützt. Eventuell ändert sich das im November 2021, wenn die Version 6.0 des .NET-Framework erscheint.

1.3 Die Historie von C#

Die Geschichte von C# ist sehr eng mit der Geschichte der konkurrierenden Programmiersprache Java verknüpft. Als die Plattform-unabhängige Programmiersprache Java aus dem Hause Sun Microsystems in den 90er Jahren des letzten Jahrtausends ihren Siegeszug antrat, erkannte auch Microsoft das Potential der neuen Programmiersprache und erwarb 1995 eine Lizenz für Java von Sun Microsystems und integrierte Java in seine Entwicklungsumgebung Developer Studio unter dem Namen J++. Als Microsoft aber spezielle Erweiterungen, die nur unter Windows lauffähig waren, auf den Markt brachte, sah Sun Microsystems die Plattform-Unabhängigkeit von Java gefährdet und intervenierte 1998 mit einem Gerichtsverfahren, das zugunsten von Sun Microsystems ausging. Aufgrund dieses Verfahrens musste Microsoft sämtliche Java-Technologien in seinen Produkten kompatibel zu Sun-Java halten. Microsoft brachte 1999 eine neue Version von J++ heraus, die alle gerichtlichen Auflagen erfüllte. Im Jahr 2000 gab Microsoft bekannt, dass es seine Java-Lizenz die 2001 ausläuft, nicht mehr verlängern wird und J++ in der kommen Version seiner Entwicklungsum-

gebung, die künftig unter dem Namen Visual Studio erscheinen wird, nicht mehr enthalten sein wird. Microsoft entwickelte dann seine Java-artige Programmiersprache für Visual Studio von Grund auf neu.

Dieses Projekt bekam den Codenamen Cool und wurde zur Markteinführung in C# (gesprochen C-Sharp) umbenannt. Sharp ist die englische Bezeichnung des Doppelkreuz-Zeichens. Das Doppelkreuz hat verschiedene symbolische Bedeutungen.

Zum einem wird es in der Notenschrift der Musik nach einer Note geschrieben, um anzuzeigen, dass die Note um einen Halbton erhöht ist. Somit drückt der Name aus, dass C# eine erhöhte Variante der Programmiersprache C ist, von der es einige grundlegende syntaktische Regeln geerbt hat.

Zum anderen kann das Doppelkreuz auch als Kombination von vier +-Zeichen, die in einem Quadrat angeordnet sind, interpretiert werden. Damit würde der Name C# bedeuten, dass man dem Namen C++ noch zwei weitere +Zeichen hinzugefügt hat und C# somit eine Weiterentwicklung von C++ ist.

C# wurde ursprünglich von Anders Hejlsberg im Auftrag von Microsoft entwickelt. Als es im Jahre 2002 mit Visual Studio 2002 und dem .NET-Framework 1.0 als C# 1.0 am Markt erschien, wurde der neuen Programmiersprache keine große Zukunft vorhergesagt.

„Es ist wie Java, kostet aber Geld, ist von Microsoft und läuft nur unter Windows." Dieser bissige Spruch kursierte damals unter Programmierern und entsprach zu diesem Zeitpunkt auch der Wahrheit. Aber das sollte sich im Laufe der Zeit Stück für Stück ändern.

Eine 2003 hastig nachgeschobene Version 1.1 des .NET-Frameworks mit einem Visual Studio 2003 brachte keine Verbesserung der Sprache C# an sich und auch nur eine geringe Verbesserung der Akzeptanz des Gesamtsystems.

Im Jahre 2005 erschien Visual Studio 2005 mit dem .NET-Framework 2.0 und mit C# in der Version 2.0. Diese Version zog nicht nur mit neuen, kurz zuvor releasten Sprachfeatures von Java gleich, sondern konnte auch noch ein paar zusätzliche Features bieten, die für Java bis heute nicht zur Verfügung stehen. Als Microsoft 2006 auch noch die kostenlosen Expressvarianten von Visual Studio 2005 veröffentlichte, konnten sich - bis auf die Plattform-Unabhängigkeit - C# und Java zum ersten Mal auf Augenhöhe begegnen.

Ebenfalls 2006 erschien die Version 3.0 des .NET-Frameworks, welche zwar weitere .NET Programmiermodelle brachte, aber keine Neuerungen in der Sprache C#.

2008 erschien Visual Studio 2008 und brachte den .NET-Framework 3.5 und C#3.0 mit. Das brachte weitere neue Sprachfeatures, die die Objektabfrage-Bibliothek Linq ermöglichten. Linq erfreut sich bis heute bei C#-Programmierern größter Beliebtheit. In diesem Buch habe ich Linq ein eigenes Kapitel gewidmet. Java-Entwickler mussten auf eine mit Linq vergleichbare Technik noch bis zum Erscheinen von Java 8 im Jahr 2014 warten. Ab C# 3.0 war C# als Programmiersprache bereits das mit Abstand bessere Java. Lediglich bei der Plattform-Unabhängigkeit konnte Java noch punkten.

Visual Studio 2010 brachte das .NET Framework 4.0 und C# 4.0. Neben ein paar kleineren Erweiterungen der Sprache C# konzentrierten sich die Microsoftentwickler hauptsächlich auf die Verbesserung von WPF, einer Technologie zum Erstellen grafischer Benutzeroberflächen.

Mit Visual Studio 2012 erhöhte sich die .NET-Framework Version auf 4.5 und die C#-Version auf 5.0. In C# kamen die sogenannten Async Features hinzu, die den Programmierer bei der Programmierung asynchroner Abläufe unterstützten. Diese Technik wird für Programme verwendet, die mehrere Dinge gleichzeitig tun.

Visual Studio 2013 erhöhte lediglich die Version des .NET-Frameworks auf 4.5.1, brachte aber signifikante Verbesserungen im Bereich Application Life Cycle Management.

Visual Studio 2015 brachte den .NET-Framework 4.6, die C#-Version 6.0 und zusätzliche Erweiterungen von C# und weitere Verbesserungen im Bereich Application Life Cycle Management. Zudem wurde für Visual Studio 2015 auch .NET Core 1.0 veröffentlicht. Das .NET Framework war von Anfang an auf Plattform-Unabhängigkeit ausgelegt, allerdings hat sich Microsoft vorerst nicht darum gekümmert, einen .NET-Framework für andere Plattformen außer Windows zur Verfügung zu stellen. Mit .NET Core sollte sich das nun ändern. Mit .NET Core 1.0 konnten in C# Konsolen-Apps und Webapplikationen erstellt werden, die auch auf Linux-basierten Computern liefen.

Die Visual Studio Version 2017 kam mit .NET Core 2.0 und der C#-Version 7.0. Neben weiteren Verbesserungen der Programmiersprache C# und der Bereinigung einiger Kinderkrankheiten von .NET Core brachte Visual Studio 2017 vor allem Xamarin-Tools. Nachdem Microsoft den Hersteller Xamarin übernommen hatte, integrierte es dessen Tools in Visual Studio. Mit den Xamarin-Tools konnte man nun Applikationen für Android und iOS mit Visual Studio entwickeln.

Die beim Erstellen dieses Buchs aktuelle Version Visual Studio 2019 umfasst .NET Core 3.1 und C# 8.0. Auch hier wurde C# wieder verbessert. Aber die wichtigste Neuerung war .NET Core 3.1, das es ermöglicht, grafische Desktop-Applikationen als .NET Core-Applikationen zu entwickeln. Desktop-Applikationen für .NET Core zeichnen sich durch ein schnelleres Laufzeitverhalten aus und können als eine einzige ausführbare

Datei an den Benutzer geliefert werden, ohne dass dieser ein installiertes .NET-Framework benötigt.

Noch während der Entwicklung von .NET Core bis zur aktuellen Version 3.1 wurde parallel der normale .NET-Framework bis zur Version 4.8 weiterentwickelt.

Im November 2020 vereinte Microsoft die beiden Frameworks und brache .NET 5.0 nur noch ein einziges aktuelle Framework auf den Markt, das für Windows, Linux, Android, MacOS und iOS zur Verfügung steht. Allerdings werden grafische Desktop-Anwendung für andere Plattformen als Windows immer noch nicht unterstützt.

Eventuell gibt es diese Unterstützung im November 2021, wenn .NET 6.0 erscheint.

1.4 Der .NET-Framework, CIL, die CLI und die CLR

Früher unterschied man Programmiersprachen grundsätzlich in zwei verschiedene Arten: Compiler-Sprachen und Interpreter-Sprachen. Bei beiden schreibt man das Programm in der jeweiligen höheren Programmiersprache. Bei Compiler-Sprachen wird das Programm mit einem sogenannten Compiler in die Maschinensprache des Computers umgesetzt. Und dieses kompilierte Maschinenprogramm kann dann auf dem Zielcomputer ausgeführt werden. Bei Interpreter-Sprachen gibt es auf dem Zielrechner ein sogenanntes Interpreter-Programm, das das Programm, ohne es vorher umzusetzen, direkt in der höheren Programmiersprache abarbeitet. Der Vorteil von Compiler-Sprachen ist, dass die kompilierten Programme wesentlich schneller laufen, da der Compiler bei der Erstellung des Maschinencodes zahlreiche Optimierungen vornehmen kann. Der Vorteil von Interpreter-Sprachen ist, dass sie plattformunabhängig sind. Es genügt ein Interpreter-Programm für die jeweilige Zielplattform zu entwickeln und schon sind alle Programme in der zugehörigen Hochsprache auf dieser Zielplattform lauffähig.

Bei der Entwicklung des .NET-Frameworks ist Microsoft einen Mittelweg gegangen und hat das Beste aus beiden Welten vereinigt. Daher gibt es einen Compiler, der C# oder eine andere .NET-Sprache in einen sogenannten CIL-Code umsetzt. CIL steht für Common Intermediate Language. Den CIL-Code muss man sich als Maschinencode für einen virtuellen (gedachten) Computer vorstellen, den es als echte physische Maschine gar nicht gibt. Wenn man jetzt auf einem echten physischen Computer so eine Art Simulationsprogramm hätte, das diesen virtuellen Computer simuliert, könnte dieses Simulationsprogramm ein CIL-Programm ausführen. Das klingt zwar etwas kompliziert und umständlich, hat aber den Charme, dass ein Programm nur einmal entwickelt und in ein CIL-Programm umgesetzt werden muss, und dann auf allen Plattformen lauffähig ist, für die es so ein Simulationsprogramm gibt. Bis jetzt haben wir mit unserem neuen Verfahren lediglich die Vorteile eines Interpreters erreicht und haben uns zusätzlich den Nachteil eingefangen, dass wir

unser Programm kompilieren müssen, was bei herkömmlichen Interpretern ja entfallen könnte. Aber durch das Kompilieren erhalten wir auch Vorteile. Ein CIL-Code hat Ähnlichkeit mit einem echten Maschinencode und da er mit einem Compiler erzeugt wird, kann dieser Compiler wieder Optimierungen einbauen. Das heißt, unser CIL-Code ist zwar langsamer als ein echter Maschinencode, aber schneller als interpretierter Code und genauso Plattform-unabhängig wie der interpretierte Code. Zudem hat sich Microsoft einen kleinen Trick einfallen lassen und das ist der JIT-Compiler oder einfach nur JIT. Das steht für Just in Time-Compiler. Der JIT kompiliert den CIL-Code in den Maschinencode der Zielplattform. Damit ist ein Simulationsprogramm, das CIL-Codes ausführt, überflüssig. Die Laufzeit-Umgebung für den CIL-Code, die auch den jeweiligen JIT enthält, wird allgemein als CLI (Common Language Infrastructure) bezeichnet. Eine konkrete Implementierung für eine bestimmte Zielplattform dagegen wird als CLR (Common Language Runtime) bezeichnet. In anderer Literatur oder in manchen Internetbeiträgen wird die CIL manchmal auch als MSIL bezeichnet, das steht für Microsoft Intermediate Language, bezeichnet aber dieselbe Sache. Mit der folgenden Grafik möchte ich den Weg von einem in C# geschriebenen Programm zu einem ausführbaren Computerprogramm noch einmal veranschaulichen.

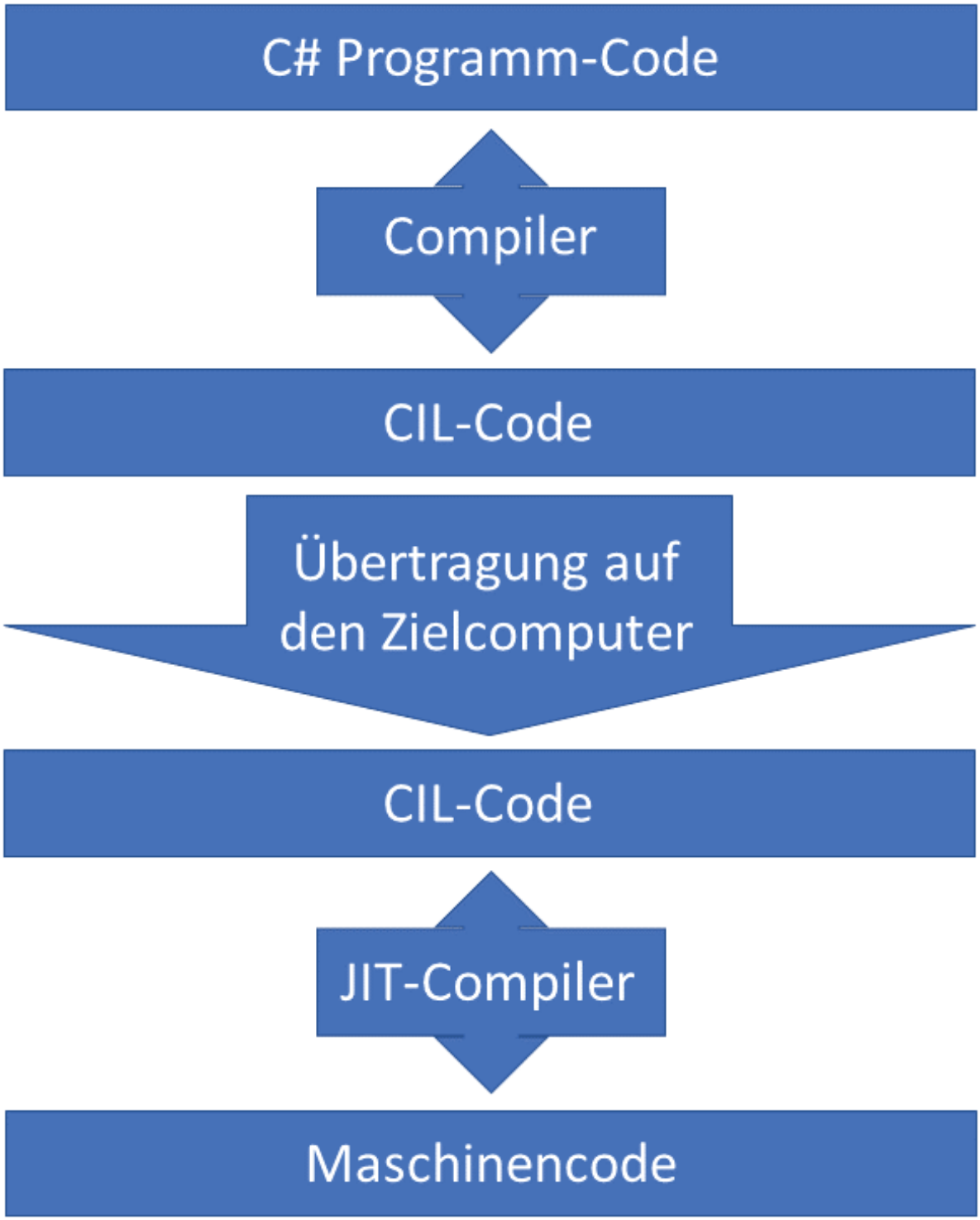


Abb.1.3.1 Kompilierprozess des .NET-Frameworks

Der Entwickler schreibt ein Programm in C#, kompiliert es auf seinem Computer in CIL Code. Er kann es dort auch ausführen und testen. Wenn das Programm in Ordnung ist, wird es auf den Zielcomputer beziehungsweise den Computer des Benutzers kopiert. Auf dem Computer des Benutzers befindet sich ein installiertes .NET-Framework und damit eine CLR. Die CLR enthält einen JIT-Compiler, der den CIL-Code in Maschinencodes kompiliert, der dann auf dem Zielcomputer ausgeführt wird.

Kapitel 2

Visual Studio: die professionelle Entwicklungsumgebung für C#

Eine Entwicklungsumgebung ist ein Softwarepaket, das uns hilft, möglichst komfortabel Software zu entwickeln. Eine Entwicklungsumgebung enthält einen Editor, mit dem ein Programmcode geschrieben werden kann und der vor allem auf die verwendete Programmiersprache zugeschnitten ist. Ein guter Editor sollte „Source Code Highlighting" und „Intellisense" beherrschen. „Source Code Highlighting" bedeutet, dass der Editor verschiedene Komponenten der jeweiligen Programmiersprache verschiedenfarbig darstellen kann, was die Lesbarkeit enorm erhöht. „Intellisense" ist eine Technologie, die Microsoft bereits in den 90er Jahren erfunden hat. Java-Programmierer, die die Entwicklungsumgebung Eclipse verwenden, kennen „Intellisense" bereits unter dem Namen „Code Complete". Ein Editor, der diese Technologie beherrscht, macht dem Programmierer Vorschläge, wie sein Programmcode weitergehen könnte, statt weiter zu tippen, kann der Programmierer aus einer Liste auswählen. Des Weiteren enthält eine Entwicklungsumgebung einen oder mehrere Compiler und verschiedene Projektvorlagen. Je nach Projekt, das Sie verwirklichen wollen, müssen Sie das Verzeichnis, in dem Sie Ihre Projektdateien anlegen, anders strukturieren. Eine geeignete Projektvorlage erledigt das automatisch für Sie.

Die professionelle Entwicklungsumgebung für die C#-Programmierung ist Visual Studio von Microsoft. Derzeit liegt sie in der Version Visual Studio 2019 vor. Visual Studio gibt es in verschiedenen Editionen, die je nach Leistungsumfang auch sehr stark im Preis variieren. Zudem gibt es eine kostenlose Community Edition. Die Community Edition dürfen Sie generell zur Entwicklung von Software, mit der Sie kein Geld verdienen, verwenden. Für kommerzielle Zwecke darf Sie nur verwendet werden, wenn im Unternehmen nicht mehr als 5 Personen Visual Studio nutzen und das Unternehmen nicht mehr als 1.000.000$ umsetzt und nicht mehr als 250 PCs betreibt. Das heißt, wenn Sie sich im Eigenstudium zu Hause C# beibringen wollen, können Sie auf jeden Fall die kostenlose Community Edition verwenden.

2.1 Die Installation von Visual Studio

Um Visual Studio 2019 Community Edition zu installieren, öffnen Sie einen Web-Browser und geben folgende URL ein:

https://visualstudio.microsoft.com/de/

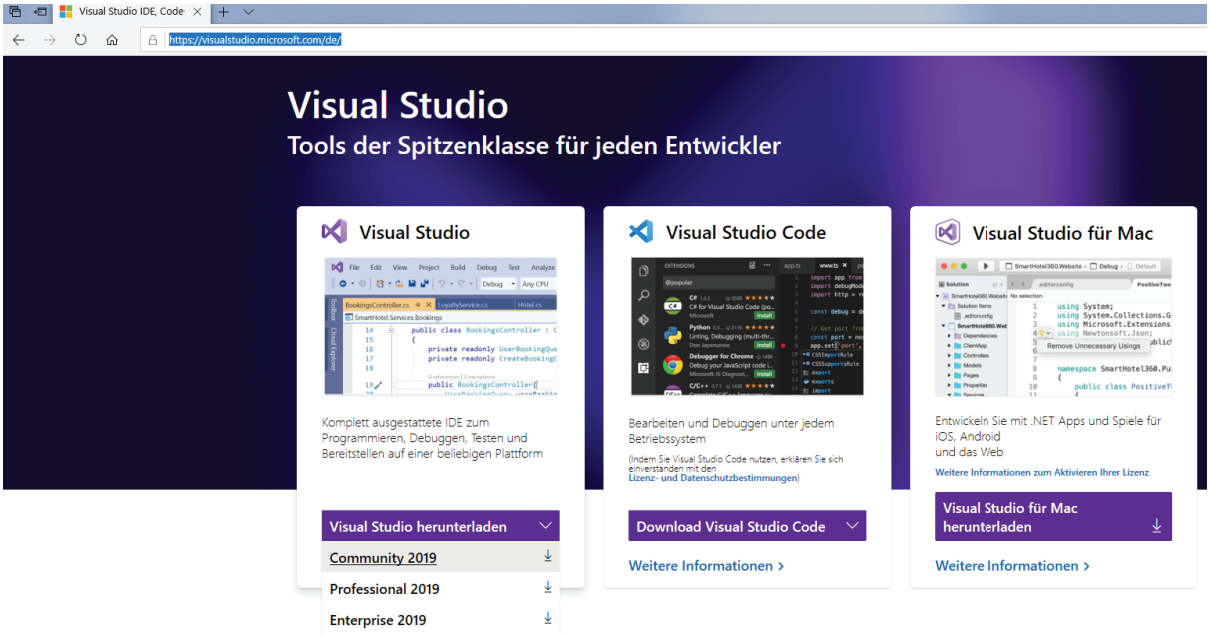


Abb. 2.1.1 Download Seite für Visual Studio

Klicken Sie auf „Visual Studio herunterladen/Community 2019“ und der Download des Installationsprogramms erscheint.

Abb. 2.1.2 Download des Visual Studio Installers

Je nachdem, welchen Browser Sie mit welchen Einstellungen verwenden, bietet Ihr Browser Ihnen das direkte Starten des Installationsprogramms an oder nicht. Wenn möglich, klicken Sie auf ausführen, wenn nicht, laden Sie die Installationsdatei herunter und starten Sie das Programm.

Windows fragt Sie nun, ob Sie dieses Programm wirklich ausführen wollen. Bestätigen Sie dies mit einem Klick auf die Schaltfläche „Ja“. Beim Begrüßungsdialog klicken Sie auf die Schaltfläche „fortfahren“. Danach wird das Visual Studio Installationsprogramm installiert und gestartet.

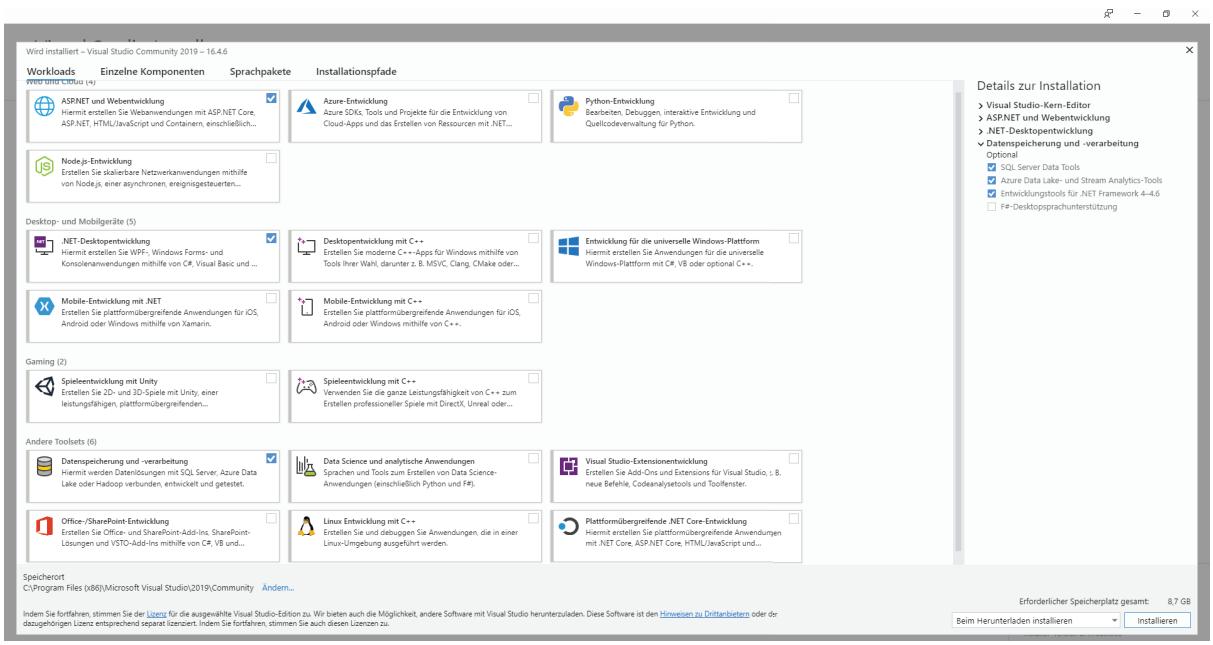


Abb. 2.1.3 Auswahl der Pakete für die Installation von Visual Studio

Für die Beispielprogramme dieses Buches benötigen Sie folgende Pakete:

- ASP.NET und Webentwicklung
- .NET Desktopentwicklung
- Datenspeicherung und -verarbeitung

Es steht Ihnen natürlich frei, weitere Pakete, die Sie interessieren, zu installieren. Um den Installationsprozess zu starten, Klicken Sie auf die Schaltfläche „Installieren“ rechts unten. Die Installation beginnt. Je nach Anzahl der ausgewählten Pakete und je nach Bandbreite Ihrer Internetverbindung kann die Installation zwischen ein paar Minuten und zwei Stunden dauern. Nach der Installation startet Visual Studio mit dem ersten Begrüßungsdialog.

Abb. 2.1.4 Visual Studio Anmeldung

Wenn Sie bereits über ein Visual Studio Konto verfügen, klicken Sie auf die Schaltfläche „Anmelden“. Wenn Sie noch nicht über ein Visual Studio Konto verfügen, können Sie ein Konto erstellen, in dem Sie auf „Erstellen Sie ein Konto!“ klicken. Falls Sie kein Konto erstellen wollen, klicken Sie auf: „Jetzt nicht, vielleicht später“. Nach der Anmeldung erscheint der Standard-Begrüßungsdialog von Visual Studio. Hier können Sie ein neues Projekt erstellen, was wir dann im nächsten Kapitel tun werden.

2.2 Ein Projekt in Visual Studio erstellen

Nachdem wir die Visual Studio Community Edition erfolgreich installiert haben, wollen wir uns ansehen, wie man in Visual Studio ein Projekt erstellt, bevor wir unser erstes Programm in C# schreiben.

Starten Sie Visual Studio. Das Startfenster von Visual Studio erscheint am Bildschirm.

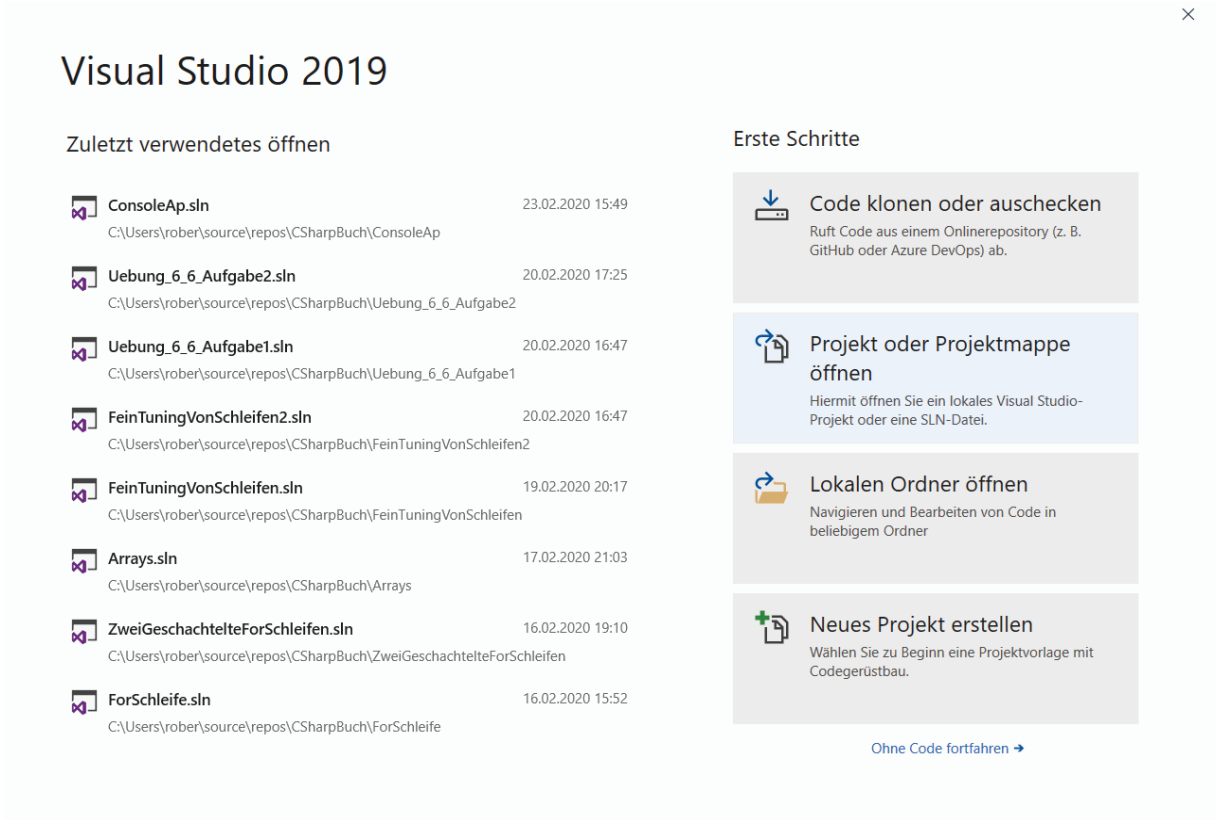


Abb. 2.2.1 Der Willkommensdialog von Visual Studio

Klicken Sie auf die Schaltfläche „Neues Projekt erstellen“.

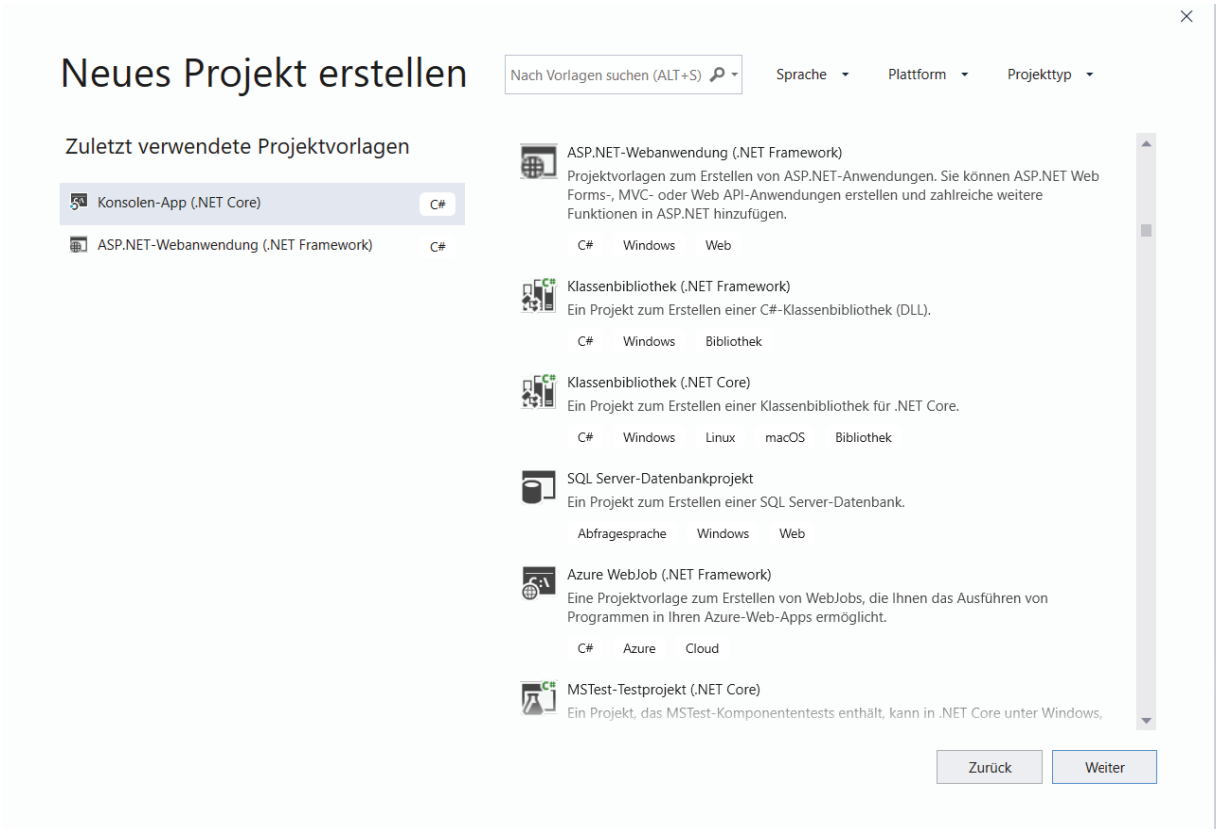


Abb. 2.2.2 Ein neues Projekt erstellen

Auf der linken Seite sehen Sie die von Ihnen zuletzt verwendeten Projektvorlagen und auf der rechten Seite eine lange Liste von allen Projektvorlagen in Visual Studio.

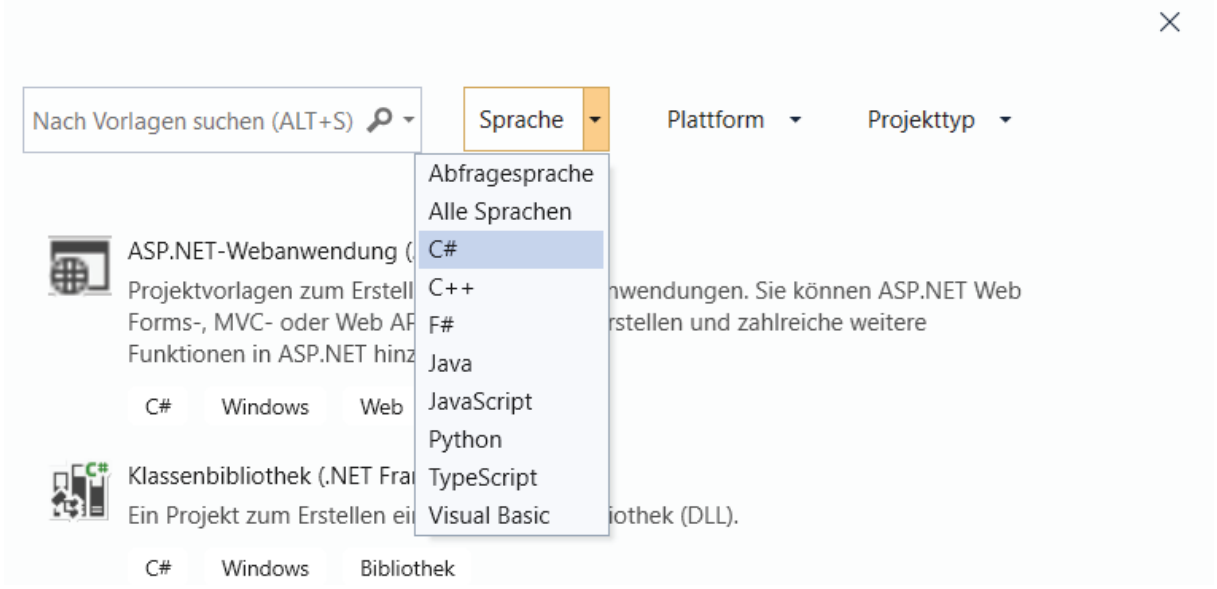


Abb. 2.2.3 Die Auswahl der Programmiersprache

In der Dropdownbox-Sprache können Sie eine Programmiersprache auswählen, um nur Projektvorlagen für diese Programmiersprache anzuzeigen. Hier wählen Sie natürlich C# aus.

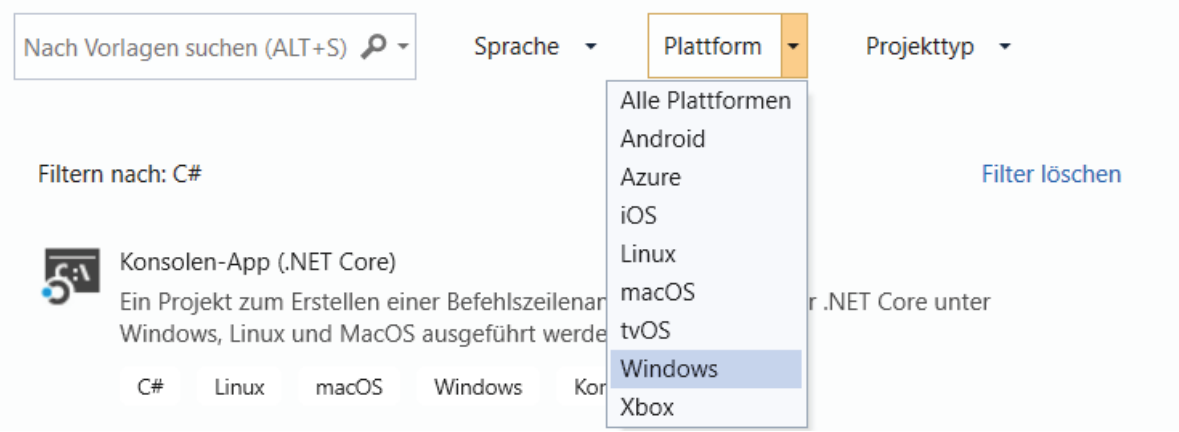


Abb. 2.2.4 Die Auswahl der Plattform

In der Dropdownbox Plattform können Sie eine Plattform beziehungsweise ein Betriebssystem auswählen, für das Sie ein Projekt erstellen wollen. Jetzt sehen wir nur noch Projektvorlagen für Windows und C# in der Liste.

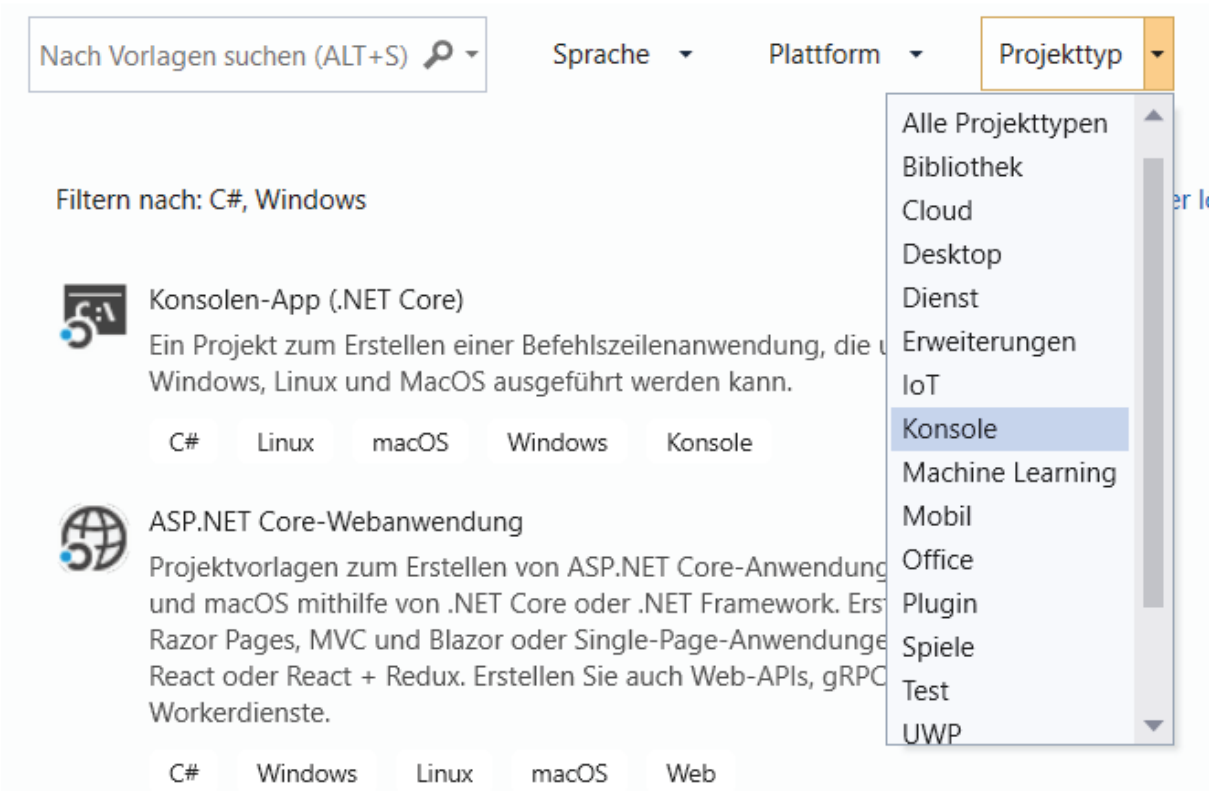


Abb. 2.2.5 Die Auswahl des Projekttyps

In der Dropdownbox Projekttyp können Sie die Projektart für das zu erstellen Projekt auswählen. Wählen Sie hier Konsole aus. Jetzt sind nur noch Projektvorlagen für Konsolenanwendungen in C# unter Windows in der Liste.

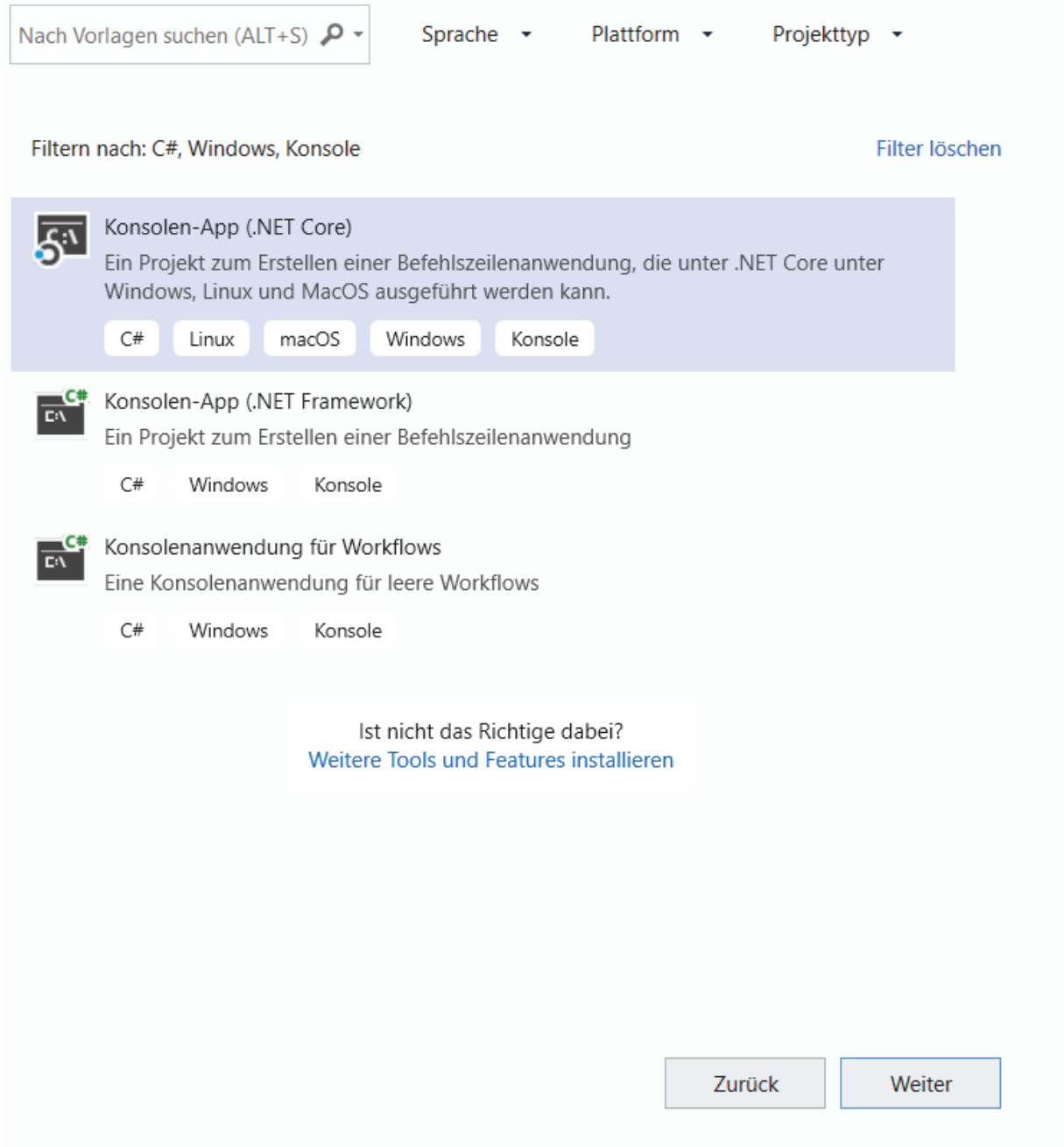


Abb. 2.2.6 Die Auswahl der Projektvorlage

Wählen Sie Konsolen-App (.NET Core) aus und klicken Sie auf die Schaltfläche „Weiter“ und das Fenster „Neues Projekt konfigurieren“ erscheint am Bildschirm.

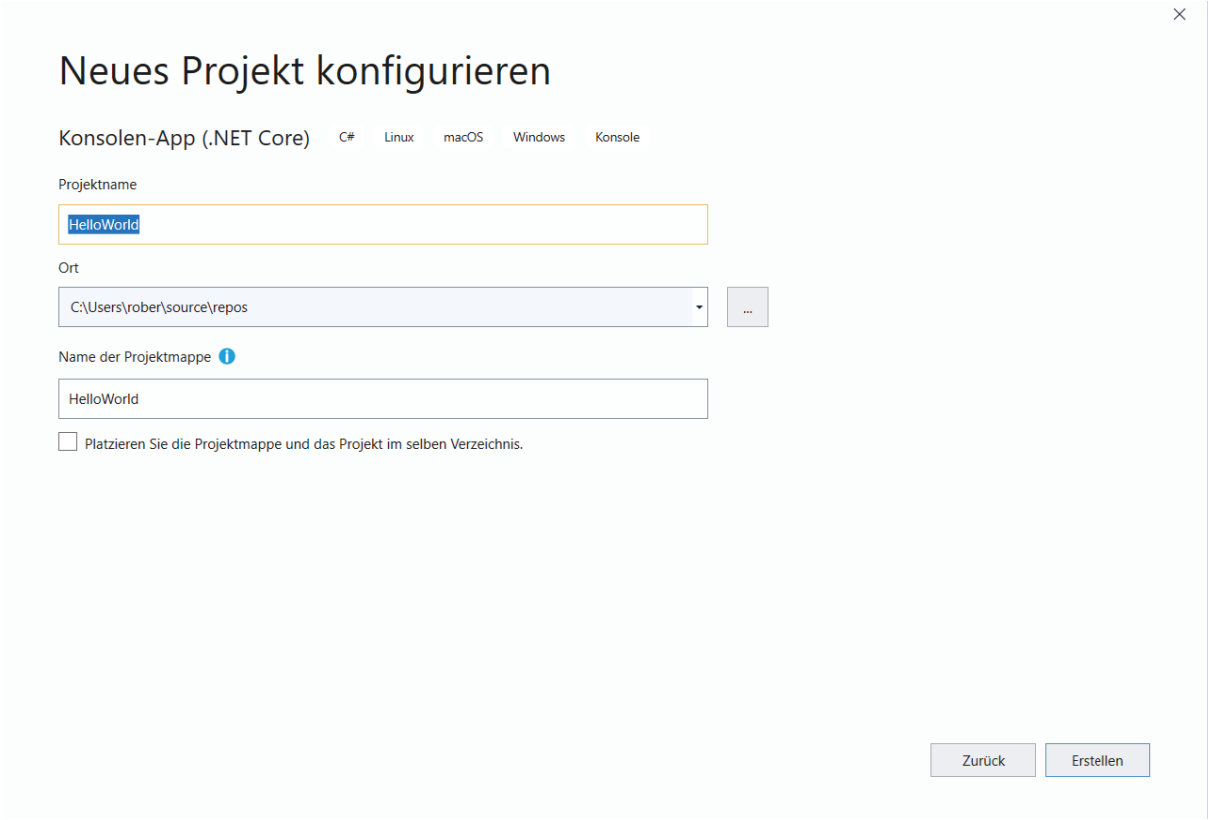


Abb. 2.2.7 Das neue Projekt konfigurieren

Hier können Sie den Namen des Projekts festlegen. In unserem Fall heißt es „HelloWorld“. Im Feld Ort können Sie einen Speicherort für Ihr Projekt festlegen. Für den Namen der Projektmappe verwendet Visual Studio standardmäßig denselben Namen wie für das Projekt. Sie könnten Ihn zwar ändern, aber wir lassen ihn vorerst, wie er ist. Die Checkbox „Platzieren Sie die Projektmappe und das Projekt im selben Verzeichnis“ lassen wir auch leer. Auf das Thema Projektmappen werde ich in einem späteren Kapitel eingehen. Wenn Sie jetzt auf die Schaltfläche erstellen klicken, erzeugt Visual Studio ein Konsolen-App-Projekt für C# für die Plattform .NET Core. Das heißt, das zu erstellende Programm ist unter Windows, Linux und MacOS lauffähig.

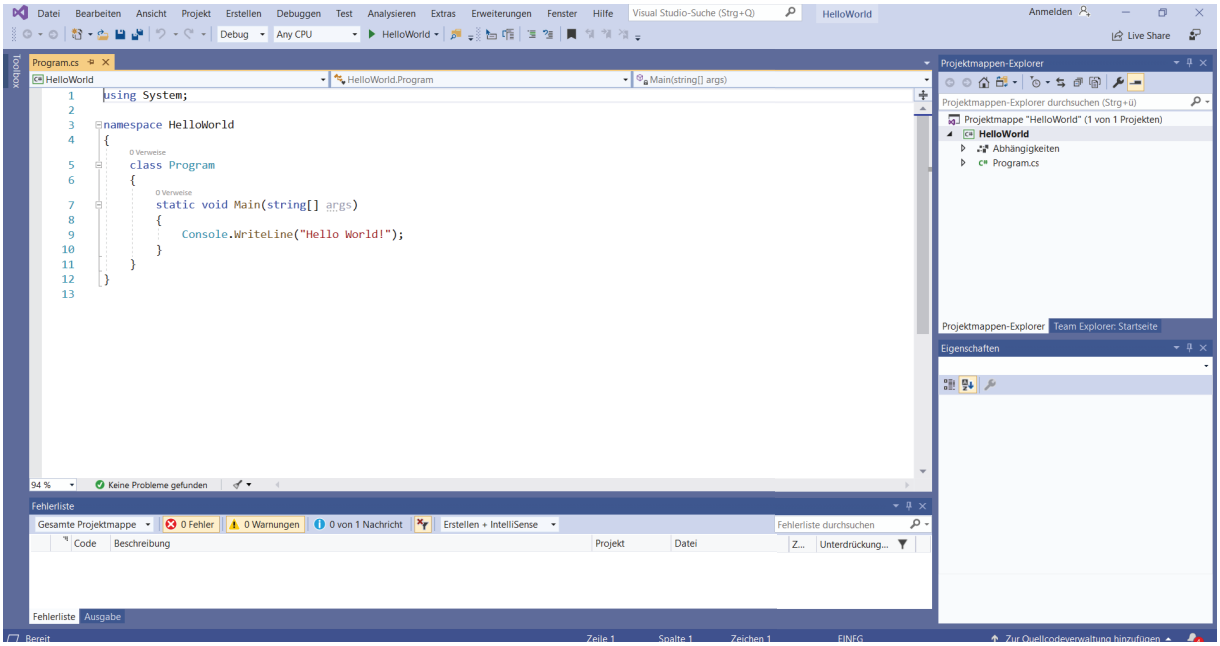


Abb. 2.2.8 Ein neu erstelltes Projekt

Im Hauptfenster sehen sie ein von Visual Studio erzeugtes Beispielprogramm und rechts oben sehen Sie den sog. „Projektmappen-Explorer“. Er stellt die Projektmappe und die darin enthaltenen Projekte dar. Eine Projektmappe ist eine Sammlung von zusammengehörigen Projekten, die man als Programmierer gemeinsam bearbeiten möchte. Unsere erste Projektmappe heißt „HelloWorld“ und das einzige darin enthaltene „Projekt“ heißt auch „HelloWorld“. In ernsthaften Softwareprojekten hat man üblicherweise mehr als ein Projekt in einer Projektmappe. In diesem Buch kommen wir vorerst mit einem Projekt in unserer Projektmappe aus. Im Kapitel: „Visual Studio reloaded: Funktionalitäten für Fortgeschrittene“ werden wir sehen, wann es hilfreich ist, mehr als ein Projekt in einer Projektmappe zu haben.

Jetzt müssen Sie das Programm nur noch starten. Klicken Sie dazu auf die Schaltfläche mit dem grünen Dreieck in der Menüleiste. Es erscheint ein Ausgabefenster mit dem Text „Hello World!“.

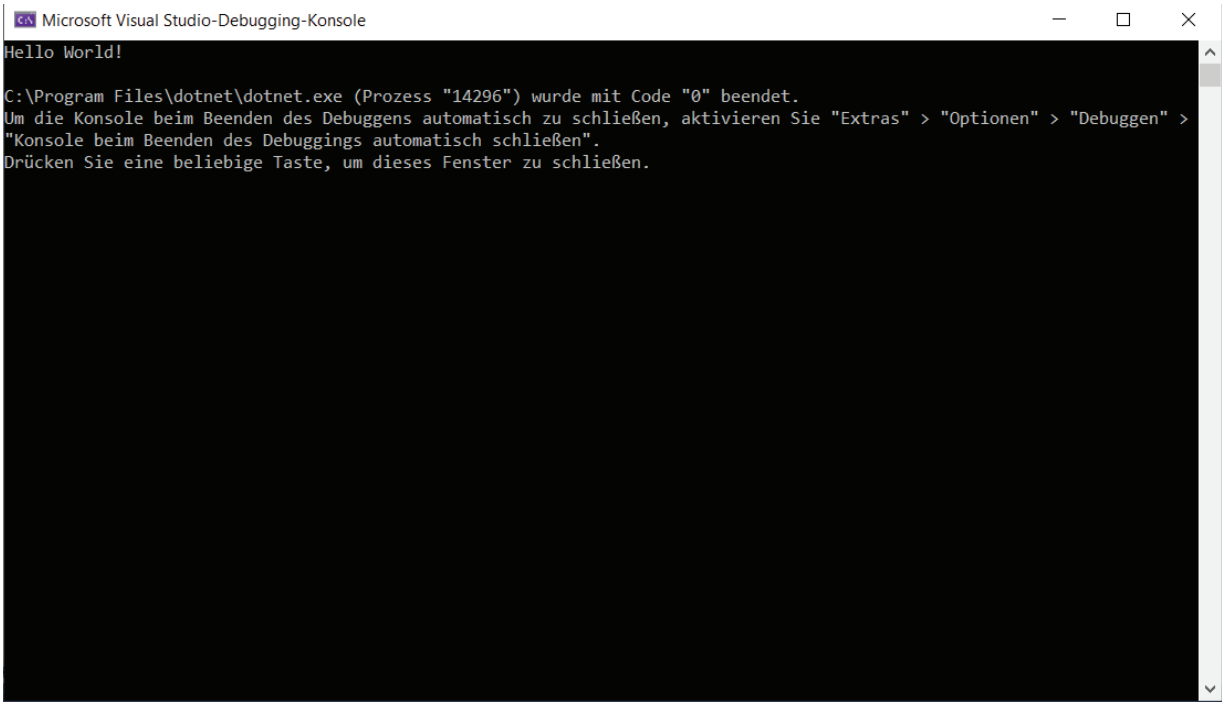


Abb. 2.2.9 Die Konsolen-App "Hello World!"

Im nächsten Kapitel werden wir dieses von Visual Studio automatisch erstellte Programm genauer untersuchen.