Einführung

Der Begriff *adaptiver Code* aus dem Untertitel dieses Buchs liefert ~~einen~~ gute Beschreibung für das Ergebnis, das Sie erhalten, wenn Sie die Prinzipien des Buchs anwenden: die Fähigkeit von Code, sich an alle neuen Anforderungen oder unvorhergesehenen Szenarien anpassen zu lassen, ohne dass er dafür in weiten Teilen umgearbeitet werden muss. Dieses Buch hat das Ziel, viele der aktuellen Best Practices aus dem Bereich der C#-Programmierung mit dem Microsoft .NET Framework in einem Band zusammenzufassen. Ein Teil des Inhalts wird zwar auch in anderen Büchern behandelt, aber diese Bücher konzentrieren sich entweder stark auf die Theorie oder sind nicht spezifisch auf die .NET-Entwicklung zugeschnitten.

Programmierung kann ein langsamer Prozess sein. Wenn Ihr Code adaptiv ist, sind Sie in der Lage, Änderungen schneller, einfacher und mit weniger Fehlern vorzunehmen, als wenn Sie mit einer Codebasis arbeiten, die Änderungen behindert. Wie jeder Entwickler weiß, ändern sich Anforderungen. Auf welche Weise mit diesen Änderungen umgegangen wird, macht den Unterschied zwischen erfolgreichen Softwareprojekten und denen aus, die aufgrund eines völlig außer Kontrolle geratenen Umfangs steckenbleiben. Entwickler können auf unterschiedliche Arten auf Änderungen der Anforderungen reagieren, wobei zwei gegensätzliche Standpunkte die Extreme des Spektrums bilden.

Erstens können Entwickler einen inflexiblen Standpunkt einnehmen. Bei diesem Ansatz ist das Projekt angefangen beim Entwicklungsprozess bis hinunter zum Klassenentwurf so inflexibel, als wäre es vor 50 Jahren mithilfe von Lochkarten implementiert worden. Wasserfall-Methodologien sind bekannte Vertreter dieser Zunft, der es darum geht sicherzustellen, dass sich Software nicht nach Belieben verändern darf. Ihr Beharren darauf, dass die Phasen von Analyse, Entwurf, Implementierung und Testen voneinander isoliert bleiben und nur in einer Richtung nacheinander abgearbeitet werden, macht es schwierig oder zumindest teuer für den Kunden, irgendwelche Anforderungen zu verändern, nachdem die Implementierung erst einmal begonnen hat. Der Code *braucht* daher gar nicht so beschaffen zu sein, dass er leicht zu ändern ist: Der Prozess verbietet praktisch Änderungen.

Der zweite Ansatz, agile Methodologie, besteht nicht etwa darin, lediglich eine Alternative zu solch inflexiblen Methodologien zu finden. Er will eine *Reaktion* auf sie sein. Agile

Prozesse haben sich das Ziel gesetzt, Änderungen als unerlässlichen Teil des Vertrags zwischen Kunde und Entwickler willkommen zu heißen. Wenn Kunden etwas an dem Produkt, für das sie bezahlen, verändern wollen, sollten der Zeitaufwand und die finanziellen Kosten den Umfang der Änderung widerspiegeln, aber nicht abhängig von der Phase sein, in der sich der Prozess momentan befindet. Im Unterschied zu anderen technischen Disziplinen arbeitet Software-Engineering mit einem äußerst elastischen Werkzeug: Quellcode. Die Steine und Mörtel, aus denen ein Haus entsteht, werden fest miteinander verbunden, während der Bau fortschreitet. Der Aufwand, um den Entwurf eines Hauses zu verändern, hängt natürlich davon ab, in welcher Phase sich der Bauprozess befindet. Hat der Bau noch gar nicht begonnen (wenn also bisher nur Pläne vorliegen), ist eine Änderung relativ billig. Sind die Fenster eingebaut, die elektrischen Leitungen verlegt und die Wasserrohre angeschlossen, wäre es unglaublich teuer, das obere Badezimmer nach unten neben die Küche zu verlegen. Bei Code sollte es nicht so aufwendig sein, Features zu verschieben und die Navigation einer Benutzeroberfläche umzugestalten. Leider ist das aber nicht immer der Fall. Allein der Zeitaufwand verhindert oft solche Änderungen. Daran ist meiner Ansicht nach vor allem eine fehlende Anpassungsfähigkeit des Codes schuld.

Dieses Buch demonstriert den zweiten Ansatz und erklärt anhand von Praxisbeispielen die Vorgehensweise beim Implementieren adaptiven Codes.

Wer sollte dieses Buch lesen

Dieses Buch soll die Lücke zwischen Theorie und Praxis schließen. Der Leser, für den das Buch geschrieben wurde, ist ein erfahrener Programmierer, der eher praktische Beispiele für Entwurfs-Patterns, SOLID-Prinzipien, Unit-Tests, Refaktorierung und andere Techniken sucht.

Fortgeschrittene Programmierer, die Wissenslücken stopfen wollen oder Zweifel und Fragen darüber haben, wie wichtige Best Practices der Branche zusammenpassen, profitieren am meisten von diesem Buch, besonders weil der Alltag beim Programmieren nur selten den oft simplen Beispielen oder gar der Theorie entspricht. Weite Teile von SOLID sind bekannt, aber die Feinheiten des Open/Closed-Prinzips (siehe Kapitel 6) und des Liskovschen Substitutionsprinzips (Kapitel 7) werden nicht vollständig verstanden. Selbst erfahrenen Programmierern ist manchmal nicht völlig klar, welche Vorteile die Dependency-Injection bietet (Kapitel 9). Genauso wird oft übersehen, welche Flexibilität – und somit Anpassungsfähigkeit – Schnittstellen dem Code verleihen (Kapitel 3).

Dieses Buch kann auch unerfahrenen Entwicklern helfen, weil sie hier von Grund auf lernen, welche Aspekte bekannter Patterns und Best Practices nützlich sind und welche langfristig eher schädlich. Der Code, den ich von Stellenbewerbern oft zu sehen bekomme, hat viele Gemeinsamkeiten. Der allgemeine Entwurf zeigt, dass der Bewerber bei vielen Fähigkeiten *fast* soweit ist, aber noch ein wenig Unterstützung braucht, um ein deutlich besserer Programmierer zu werden. Besonders das Entourage-Anti-Pattern (Kapitel 2) und das Service-Locator-Anti-Pattern (Kapitel 9) sind in den präsentierten Codebeispielen all-

gegenwärtig. Praktische Alternativen und Begründungen, warum sie sinnvoller sind, werden in diesem Buch beschrieben.

Erforderliche Vorkenntnisse

Im Idealfall haben Sie bereits einige praktische Erfahrung im Programmieren mit einer Sprache, deren Syntax C# ähnelt, zum Beispiel Java oder C++. Sie sollten außerdem eine solide Grundlage in Konzepten der prozeduralen Programmierung haben, beispielsweise bedingte Verzweigungen, Schleifen und Ausdrücke. Und Sie sollten etwas Erfahrung in der objektorientierten Programmierung mit Klassen haben und von Schnittstellen wenigstens schon gehört haben.

Dieses Buch eignet sich wahrscheinlich nicht für Sie, wenn ...

Dieses Buch eignet sich wahrscheinlich nicht für Sie, wenn Sie gerade erst anfangen, das Programmieren zu lernen. Dieses Buch behandelt fortgeschrittene Themen, die solide Kenntnisse grundlegender Programmierkonzepte voraussetzen.

Aufbau dieses Buchs

Dieses Buch besteht aus drei Teilen, die jeweils aufeinander aufbauen. Trotzdem müssen Sie das Buch nicht von Anfang bis Ende durchlesen. Jedes Kapitel ist in sich abgeschlossen und behandelt ein bestimmtes Thema. Wo es sinnvoll ist, finden Sie Querverweise auf andere Kapitel.

Teil I: Eine agile Grundlage

Dieser Teil schafft die Grundlage zum Erstellen von Software, die adaptiv ist. Er bietet einen Überblick über den agilen Prozess *Scrum*, für den Code benötigt wird, der sich einfach an Änderungen anpassen lässt. Die übrigen Kapitel in diesem Teil konzentrieren sich auf Details zu Schnittstellen, Entwurfs-Patterns, Refaktorierung und Unit-Tests.

- **Kapitel 1: Einführung in Scrum**
  Dieses Kapitel eröffnet das Buch mit einer Einführung in Scrum, eine agile Projektverwaltungsmethodologie. Das Kapitel bietet einen detaillierten Überblick über die Artefakte, Rollen, Kennzahlen und Phasen eines Scrum-Projekts. Es zeigt außerdem, wie Entwickler sich selbst und ihren Code organisieren sollten, wenn sie in einer agilen Umgebung tätig sind.
- **Kapitel 2: Abhängigkeiten und Schichten**
  Dieses Kapitel beschäftigt sich mit Abhängigkeiten und Schichtarchitektur. Code kann nur adaptiv sein, wenn die Struktur der Lösung dies zulässt. Es werden die unterschied-

lichen Arten von Abhängigkeiten beschrieben: Eigenabhängigkeiten, Fremdabhängigkeiten und Frameworkabhängigkeiten. Das Kapitel beschreibt, wie Sie Abhängigkeiten verwalten und organisieren, von Anti-Patterns (die Sie vermeiden sollten) bis zu Patterns (die Sie nutzen sollten). Außerdem steigt es mit fortgeschrittenen Themen wie aspektorientierter Programmierung und asymmetrischer Schichtung tiefer in wichtige Programmiertechniken ein.

- **Kapitel 3: Schnittstellen und Entwurfs-Patterns**
  Schnittstellen sind in der modernen .NET-Entwicklung allgegenwärtig. Allerdings werden sie oft falsch verwendet, missverstanden und missbraucht. Dieses Kapitel zeigt einige bekannte und nützliche Entwurfs-Patterns und beschreibt, wie vielseitig eine Schnittstelle sein kann. Während das Kapitel den Leser durch die Schritte zur simplen Extrahierung einer Schnittstelle führt, zeigt es, wie Schnittstellen auf viele unterschiedliche Arten genutzt werden können, um ein Problem zu lösen. Mixins, Duck-Typing und flüssige Schnittstellen unterstreichen weiter die Vielseitigkeit dieser zentralen Waffe im Arsenal des Programmierers.
- **Kapitel 4: Unit-Tests und Refaktorierung**
  Zwei Techniken, die heutzutage jeder Programmierer beherrschen muss, sind Unit-Tests und Refaktorierung. Die beiden hängen eng miteinander zusammen und ihre Kombination produziert adaptiven Code. Ohne das Sicherheitsnetz von Unit-Tests ist die Refaktorierung fehlerträchtig. Und ohne Refaktorierung wird Code unhandlich, inflexibel und schwierig zu verstehen. Dieses Kapitel arbeitet ein Beispiel für Unit-Tests von bescheidenen Anfängen bis zu fortgeschrittenen, aber praktischen Patterns durch und stellt flüssige Assert-Verkettung, testorientierte Entwicklung und Mocking vor. Im Bereich der Refaktorierung demonstriert das Kapitel anhand praxisorientierter Beispiele, wie Sie die Lesbarkeit und Wartungsfreundlichkeit von Quellcode verbessern.

Teil II: Schreiben von SOLID-Code

Dieser Teil baut auf den Grundlagen auf, die in Teil I geschaffen wurden. Jedes Kapitel widmet sich einem Prinzip von SOLID. Dabei erklären diese Kapitel nicht einfach die trockene Theorie, sondern demonstrieren anhand von Praxisbeispielen die Umsetzung der Prinzipien. Indem jedes Beispiel in einen realen Kontext gesetzt wird, stellen die Kapitel in diesem Teil des Buchs den Nutzen von SOLID eindrucksvoll unter Beweis.

- **Kapitel 5: Das Single-Responsibility-Prinzip**
  Dieses Kapitel zeigt, wie Sie das Single-Responsibility-Prinzip in der Praxis mithilfe von Decorator- und Adapter-Pattern umsetzen. Die Anwendung des Prinzips führt dazu, dass mehr Klassen entstehen, diese Klassen aber kleiner werden. Das Kapitel zeigt, dass sich diese kleineren Klassen im Unterschied zu monolithischen Klassen, die einen großen Funktionsumfang enthalten, auf eine bestimmte Aufgabe beschränken und sich darauf konzentrieren, lediglich einen kleinen Teilaspekt eines größeren Problems zu lösen. In der Kombination mehrerer dieser Klassen zeigt sich dann die beeindruckende Leistungsfähigkeit der Einzelteile und des Gesamtsystems.

- **Kapitel 6: Das Open/Closed-Prinzip**
  Das Open/Closed-Prinzip klingt einfach, kann aber erhebliche Auswirkungen auf den Code haben. Es soll sicherstellen, dass Code, der den SOLID-Prinzipien folgt, nur erweitert, aber niemals bearbeitet wird. Dieses Kapitel stellt außerdem das Konzept der prognostizierten Variation in Bezug auf das Open/Closed-Prinzip vor und beschreibt, wie es Entwicklern hilft, mithilfe von Erweiterungspunkten die Anpassungsfähigkeit zu erhöhen.
- **Kapitel 7: Das Liskovsche Substitutionsprinzip**
  Dieses Kapitel zeigt, welche Vorteile es bringt, das Liskovsche Substitutionsprinzip auf Code anzuwenden. Besonders wird dabei auf die Tatsache eingegangen, dass diese Regeln helfen, das Open/Closed-Prinzip einzuhalten und versehentliche Beschädigungen aufgrund von Veränderungen zu verhindern. Es wird gezeigt, wie Verträge mit Vorbedingungen, Nachbedingungen und Dateninvarianten durch den Einsatz von Codeverträgen definiert und überprüft werden. Außerdem beschreibt das Kapitel wichtige Regeln zur Ableitung von Typen, zum Beispiel Kovarianz, Kontravarianz und Invarianz, und demonstriert die Auswirkungen, falls diese Regeln gebrochen werden.
- **Kapitel 8: Interface-Segregation**
  Nicht nur Klassen sollten kleiner sein, als sie oft sind, dieses Kapitel zeigt, dass auch Schnittstellen oft zu groß sind. Interface-Segregation ist eine simple Technik, die oft übersehen wird. Dieses Kapitel zeigt, welche Vorteile es hat, Schnittstellen möglichst klein zu halten und mit diesen kleinen Schnittstellen zu arbeiten. Außerdem nennt es unterschiedliche Gründe, die für das Auftrennen von Schnittstellen sprechen, zum Beispiel Anforderungen des Clients oder der Architektur.
- **Kapitel 9: Dependency-Injection**
  Dieses Kapitel beschäftigt sich mit dem Klebstoff, der die übrigen Techniken dieses Buchs zusammenhält. Ohne Dependency-Injection wäre vieles gar nicht möglich, sie ist wirklich derart wichtig. Dieses Kapitel enthält eine Einführung in Dependency-Injection und vergleicht die unterschiedlichen Methoden für ihre Implementierung. Das Kapitel behandelt die Verwaltung von Objektlebensdauern, die Arbeit mit Inversion-of-Control-Containern, das Vermeiden häufiger Anti-Patterns im Bereich von Diensten und das Analysieren von Composition-Roots und Resolution-Roots.

Teil III: Entwickeln von adaptivem Code in der Praxis

Dieser Teil demonstriert anhand einer Beispielanwendung, wie die im Buch beschriebenen Techniken kombiniert werden. Diese Kapitel enthalten eine Menge Code, er wird aber ausführlich erklärt. Weil dieses Buch sich mit der Arbeit in einer agilen Umgebung beschäftigt, orientieren sich die Kapitel an Scrum-Sprints und die durchgeführten Arbeiten sind das Ergebnis von Backlog-Einträgen und Änderungswünschen des Kunden.

- **Kapitel 10: Beispiel für die Entwicklung von adaptivem Code: Einführung**
  Dieses erste Kapitel beschreibt die Anwendung, die entwickelt werden soll: eine Online-Chat-Anwendung, die mit ASP.NET-MVC 5 entwickelt wird. Ein knapper Entwurf zeigt die geplante Architektur, außerdem werden die Features im Backlog erläutert.
- **Kapitel 11: Beispiel für die Entwicklung von adaptivem Code: Sprint 1**
  Unter Anwendung eines testorientierten Entwicklungsansatzes werden die ersten Features der Anwendung entwickelt, darunter das Anzeigen und Erstellen von Chaträumen und Nachrichten.
- **Kapitel 12: Beispiel für die Entwicklung von adaptivem Code: Sprint 2**
  Natürlich hat der Kunde einige Änderungen an den Anforderungen der Anwendung gemacht und das Team ist in der Lage, diese Änderungswünsche zu erfüllen, weil es adaptiven Code geschrieben hat.

Anhänge

In den Anhängen finden Sie einige Referenzinformationen, konkret eine Anleitung für die Arbeit mit der Quellcodeverwaltung Git und eine Erklärung, wie der Code für dieses Buch auf GitHub organisiert ist.

- **Anhang A: Adaptive Tools**
  Dies ist eine knappe Einführung in die Quellcodeverwaltung Git, die Sie zumindest in die Lage versetzen sollte, den Code von GitHub herunterzuladen und ihn in Microsoft Visual Studio 2013 zu kompilieren. Dies soll kein ausführliches Git-Handbuch sein – für diesen Zweck gibt es bereits hervorragende Informationsquellen, zum Beispiel das offizielle Git-Tutorial unter *http://git-scm.com/docs/gittutorial*. Mit einer Suchmaschine finden Sie bei Bedarf viele andere Quellen. Außerdem stellt dieser Anhang kurz einige andere Entwicklertools vor, zum Beispiel kontinuierliche Integration.
- **Anhang B (nur online und in englischer Sprache): GitHub-Codebeispiele**
  Indem ich den Code für dieses Buch auf GitHub bereitstelle, kann ich Änderungen an einer zentralen Stelle vornehmen. Das Repository ist schreibgeschützt, aber die Anhänge A und B verraten Ihnen, wie Sie den Code für ein Listing finden, herunterladen, kompilieren, ausführen und lokal verändern können. Wenn Sie denken, Sie haben einen Fehler gefunden, oder eine Änderung vorschlagen wollen, können Sie ein Pull-Request für das Repository schicken; ich sehe es mir gerne an. Anhang B finden Sie unter *microsoftpressstore.com* auf der Seite zu diesem Buch.

Konventionen in diesem Buch

In diesem Buch werden besondere Textelemente durch verschiedene Konventionen hervorgehoben. Wenn Sie bereits andere Bücher von Microsoft Press gelesen haben, sind Sie vermutlich damit vertraut. Falls nicht, können Sie sich in diesem Abschnitt darüber informieren.

Codelistings

Dieses Buch enthält zahlreiche Codelistings, deren Bedeutung kurz in einer Listingunterschrift erklärt wird (Listing 0–1).

```
public void MyService : IService
{

}
```

List. 0–1 Dies ist ein Codelisting. Sie finden etliche davon im Buch.

Wenn Ihre Aufmerksamkeit auf einen bestimmten Teil des Codes gelenkt werden soll, weil zum Beispiel Änderungen gegenüber dem vorherigen Beispiel vorgenommen wurden, sind die entsprechenden Abschnitte fett hervorgehoben.

Hinweise und Textkästen

Hinweiskästen werden für kurze Anmerkungen und Hinweise verwendet, während Textkästen umfangreichere Themen behandeln. Hier einige Beispiele:

HINWEIS Dies ist ein Hinweiskasten. Er enthält kurze Informationsbröckchen, die sich auf den Inhalt beziehen, aber besonders wichtig sind.

Dies ist ein Textkasten

Dieser Textkasten ist zwar relativ kurz, aber meist enthalten sie längere Beiträge zu Themen, die leicht vom Hauptthema abschweifen.

Systemvoraussetzungen

Sie brauchen die folgende Hardware und Software, um die Codebeispiele in diesem Buch zu verwenden:

- Windows XP Service Pack 3 (außer Starter Edition), Windows Vista Service Pack 2 (außer Starter Edition), Windows 7, Windows Server 2003 Service Pack 2, Windows Server 2003 R2, Windows Server 2008 Service Pack 2 oder Windows Server 2008 R2
- Visual Studio 2013, beliebige Edition (falls Sie Express Edition-Produkte verwenden, sind unter Umständen mehrere Downloads erforderlich)

- Microsoft SQL Server 2008 Express Edition oder höher (2008 oder R2 Release), mit SQL Server Management Studio 2008 Express oder höher (enthalten in Visual Studio; für Express Editions ist ein separater Download erforderlich)
- Computer mit einem Prozessor mit mindestens 1,6 GHz (2 GHz empfohlen)
- 1 GByte (bei einer 32-Bit-Version) oder 2 GByte (bei einer 64-Bit-Version) RAM (weitere 512 MByte, falls Sie einen virtuellen Computer verwenden oder SQL Server Express Editions einsetzen, mehr bei leistungsfähigeren SQL Server-Editionen)
- 3,5 GByte freier Festplattenplatz
- Festplattenlaufwerk mit 5.400 U/min
- DirectX 9-fähige Grafikkarte an einem Monitor mit mindestens 1024 x 768 Pixel Auflösung
- DVD-ROM-Laufwerk (falls Sie Visual Studio von DVD installieren)
- Internetverbindung zum Herunterladen von Software und Codebeispielen

Abhängig von Ihrer Windows-Konfiguration benötigen Sie unter Umständen Administratorrechte, um Visual Studio 2013 und SQL Server 2008-Produkte zu installieren oder zu konfigurieren.

Downloads: Codebeispiele

Soweit möglich, habe ich sichergestellt, dass die Codelistings aus einem größeren Beispiel stammen, das entweder als eigenständige Anwendung oder als Unit-Test ausgeführt werden kann. Viele der einfacheren Unit-Tests habe ich mit MSTest geschrieben, daher wird kein externer Test-Runner dafür benötigt, aber für die komplexeren Unit-Tests habe ich auf NUnit zurückgegriffen. Den gesamten Code habe ich in Visual Studio 2013 Ultimate geschrieben. Einige Lösungen habe ich mithilfe der Preview-Version erstellt, aber sie wurden alle unter der fertigen Version kompiliert und getestet. Nach Möglichkeit habe ich keine Features verwendet, die in den Express Editions von Visual Studio 2013 nicht verfügbar sind, aber bei manchen Themen war das nicht möglich. Leser, die diesen Code ausführen wollen, müssen eine kommerzielle Version dafür installieren.

Der Code selbst steht auf GitHub unter der folgenden Adresse zur Verfügung:

http://aka.ms/AdaptiveCode_CodeSamples

Anhang A enthält eine Anleitung zur Benutzung von Git, während Anhang B (nur online) auflistet, in welchen Unterordnern des Git-Repositorys Sie die Listings aus den einzelnen Kapiteln finden.

Wenn Sie eine Anmerkung zum Code haben, können Sie mir in meinem Blog eine Nachricht hinterlassen:

http://garymcleanhall.wordpress.com

Danksagungen

Der Autorenname im Impressum vermittelt einen falschen Eindruck. Ich wäre nie in der Lage gewesen, dieses Buch ohne die folgenden Personen zu schreiben, die mir auf unterschiedliche Weise geholfen haben. Ich möchte danken:

- Victoria, meiner Frau, dafür, dass sie dieses Buch möglich gemacht hat. Das ist keine leere Floskel, sondern die reine Wahrheit.
- Amelia, meiner Tochter, dafür, dass sie in jeder Hinsicht perfekt ist.
- Pam, meiner Mutter, dafür, dass sie korrekturgelesen und mich mit wundervoll übertriebenem Lob angespornt hat.
- Les, meinem Vater, für all seine harte Arbeit.
- Darryn, meinem Bruder, für seine unermüdliche Hilfe und Unterstützung.
- Kathy Krause bei Online Training Solutions, für ihre hervorragende Arbeit, die dieses Buch lesbar gemacht hat.
- Devon Musgrave, für seine anscheinend grenzenlose Geduld.

Errata und Support

Wir haben uns sehr um die Richtigkeit der in diesem Buch enthaltenen Informationen bemüht. Fehler, die seit der Veröffentlichung bekannt geworden sind, werden auf der Microsoft Press-Website (in englischer Sprache) aufgelistet:

http://aka.ms/Adaptive/errata

Sollten Sie einen Fehler finden, der noch nicht aufgeführt ist, würden wir uns freuen, wenn Sie uns auf der gleichen Seite darüber informieren (in englischer Sprache).

Mit Anmerkungen, Fragen oder Verbesserungsvorschlägen zu diesem Buch können Sie sich auch an den dpunkt.verlag wenden:

hallo@dpunkt.de

Bitte beachten Sie, dass über unsere E-Mail-Adresse kein Software-Support angeboten wird.

Für Supportinformationen bezüglich der hier verwendeten Microsoft-Produkte besuchen Sie die Microsoft-Website *http://support.microsoft.com*.

Kostenlose E-Books von Microsoft Press

Von technischen Einführungen bis zu detaillierten Informationen über Spezialthemen decken die nur in Englisch verfügbaren kostenlosen E-Books von Microsoft Press eine große Themenbandbreite ab. Diese E-Books stehen in den Formaten PDF, EPUB und Mobi für Kindle zur Verfügung. Sie können sie herunterladen unter:

http://aka.ms/mspressfree

Schauen Sie ab und zu vorbei, um sich zu informieren, was es Neues gibt!