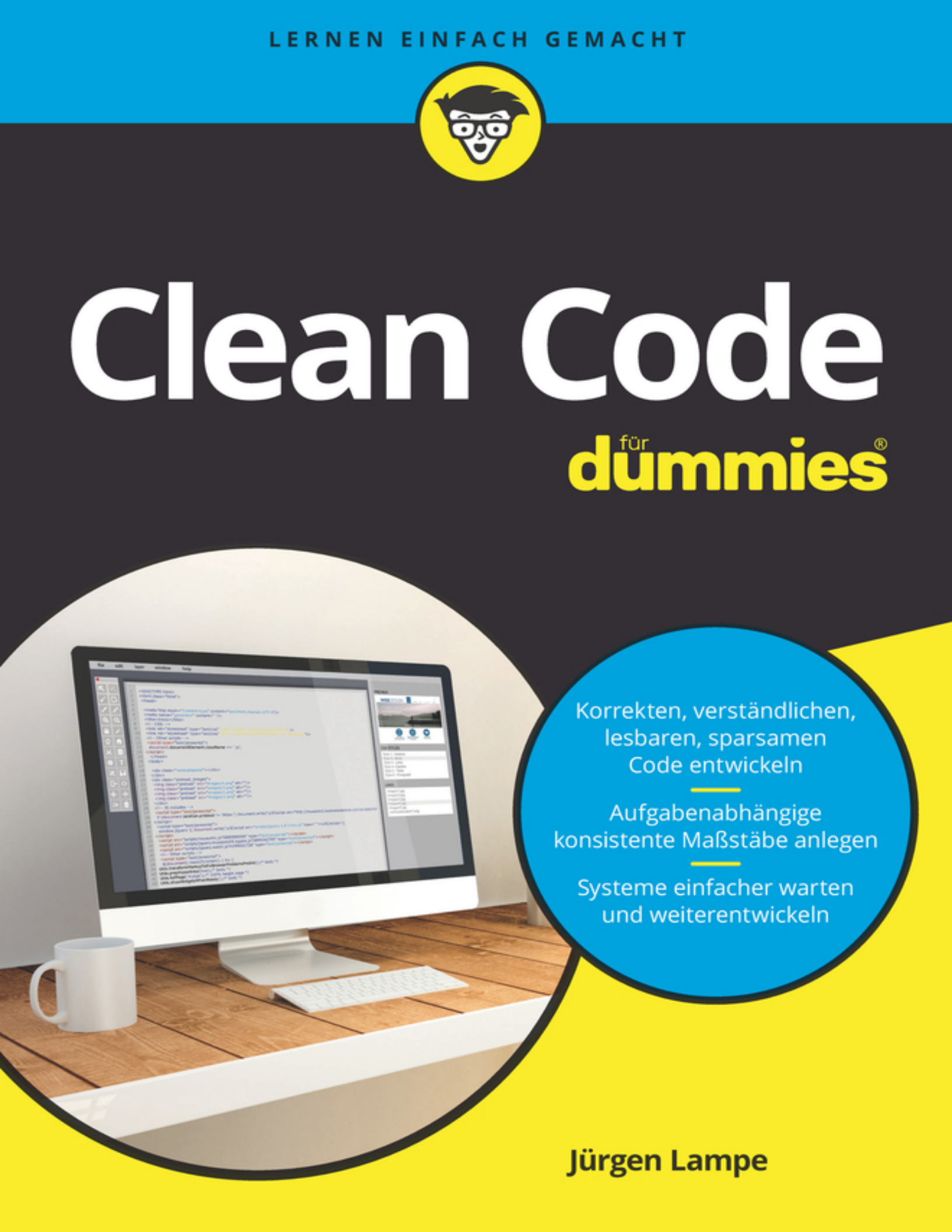
LERNEN EINFACH GEMACHT



Clean Code

für dummies®

Korrekten, verständlichen, lesbaren, sparsamen Code entwickeln

Aufgabenabhängige konsistente Maßstäbe anlegen

Systeme einfacher warten und weiterentwickeln

Jürgen Lampe

Clean Code für Dummies

Schummelseite

DAS CLEAN-CODE-PRINZIP

Software ist in unserem Alltag unverzichtbar geworden. Software ist aber vor allem der Code, der Funktionen definiert und aus dem lauffähige Systeme erzeugt werden. Clean Code ist eine wichtige Methode, um durch lesbaren Code zu langfristig pflegbarer Software zu kommen – wenn der Anspruch umfassend verstanden wird:

- ✔ **Software ist Code, nicht nur – aber ohne Code keine Software.**
- ✔ **Code ist wichtig als ultimative Beschreibung dessen, was die Software macht.**
- ✔ **Jedes Projekt braucht ein angepasstes Code-Qualitätsziel.**
- ✔ **Guter Code ist das Ergebnis sorgfältig abgewogener Entscheidungen.**
- ✔ **Sauberer Code ist gut lesbar und verständlich und damit nachvollziehbar.**
- ✔ **Unsauberer Code verrottet leicht und bedroht langfristig die Wartbarkeit.**

PROGRAMMIEREN IST MEHR ALS HANDWERKSKUNST

Handwerkliche Fähigkeiten sind für eine professionelle Softwareentwicklung unverzichtbar, reichen allein aber noch nicht aus:

- ✔ **Solide handwerkliche Fertigkeit ist Voraussetzung für Professionalität.**
- ✔ **Code ist formalisiertes Wissen.**

Um guten Code schreiben zu können, brauchen Sie ein mentales Modell der Wirklichkeit.

- ✔ **Programmieren erfordert damit teilweise wissenschaftliche Arbeit, nämlich das Aufstellen eines schlüssigen Modells für einen Teil der Anwendungsdomäne.**

GRUNDPRINZIPIEN FÜR SAUBEREN CODE

Wichtige Regeln und Grundsätze für Clean Code:

- ✔ **Gute Namen**
  - Namen sind der einzige Weg, Programmelementen eine erfassbare Bedeutung zuzuordnen. Sie verbinden das mentale Modell mit der Implementierung.
  - Wählen Sie treffende Namen sorgfältig und konsistent!
- ✔ **Formatierung**
  - Macht die Struktur sichtbar und unterstützt so die Lesbarkeit
- ✔ **Kommentare**
  - Fast immer überflüssig und kein Mittel, unverständlichen Code besser zu machen
  - Wenn unumgänglich, dann kurz und präzise. Nicht für das *Wie* (das zeigt der Code), sondern für das *Warum* verwenden.
- ✔ **Methoden**
  - Eine Methode muss auf genau einer Abstraktionsebene angesiedelt sein und genau eine Funktion/Aufgabe erledigen
  - So wenige Parameter wie möglich verwenden
- ✔ **Schnittstellen**
  - Schmal und kohärent definieren
  - Fremdcode durch Adapter sichtbar abgrenzen
- ✔ **Objekte und Datenstrukturen**
  - Objekte und Datenstrukturen sind unterschiedliche Abstraktionen mit jeweils eigenen Anwendungsfeldern.
  - Objekte haben ein Verhalten und verbergen ihre Daten. Das erleichtert das Hinzufügen neuer Objektarten mit neuem Verhalten. Es ist jedoch schwierig, existierendes Verhalten zu ändern.
  - Datensätze haben kein Verhalten und zeigen ihre Daten. Das macht es einfach, neue Verarbeitungen zu realisieren, aber aufwendig, die Datenstruktur zu verändern.

- ✔ **Exceptions**
  - Strukturieren Sie Ihre Anwendung in try-catch-Blöcke so, dass am Ende immer wieder ein konsistenter Zustand erreicht wird
  - Exceptions so zur Fehlerbehandlung einsetzen, dass sie den Code vereinfachen
  - Verwenden Sie nur unchecked Exceptions (gegebenenfalls einpacken)

WICHTIGE REGELN

Einige bewährte Regeln für das Schreiben sauberen Codes

- ✔ **Wiederholungen vermeiden** – *Don't repeat yourself*
- ✔ **Nur liefern, was verlangt wird** – *You ain't gonna need it*
- ✔ **Anliegen sortieren und nach Gruppen getrennt erledigen** – *Separation of concerns*
- ✔ **Nur eine Verantwortung pro Klasse** – *Single responsibility principle*
- ✔ **Offen für Erweiterungen und abgeschlossen gegenüber Änderungen** – *Open closed principle*
- ✔ **Klasse soll durch Unterklassen ersetzbar sein** – *Liskov substitution principle*
- ✔ **Schmale und kohärente Schnittstellen** – *Interface segregation principle*
- ✔ **Umkehr der Abhängigkeiten** – *Dependency inversion principle*
- ✔ **Einfach halten** – *Keep it simple*

CLEAN CODE LEBEN

Mit dem Schreiben von sauberen Programmzeilen ist es nicht getan. Der gesamte Softwareentwicklungsprozess und die Kultur der Softwareentwicklung müssen sich ändern:

- ✔ **Code Reviews zur gegenseitigen Weiterbildung**
- ✔ **Fehler als Quelle neuer Kenntnisse sehen und analysieren**
- ✔ **Ohne gute Testabdeckung kein sauberer Code**

✔ **Refactoring ist der zentrale Weg zu gutem und langlebigem Code**

Jürgen Lampe

Clean Code

für dummies®

Fachkorrektur von Maud Schlich

WILEY

WILEY-VCH Verlag GmbH & Co. KGaA

Clean Code für Dummies

Bibliografische Information der Deutschen Nationalbibliothek

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

Print ISBN: 978-3-527-71634-0
ePub ISBN: 978-3-527-82392-5

Coverfoto: © georgejmclittle – stock.adobe.com
Korrektur: Claudia Lötschert, Neuss

Über den Autor

Jürgen Lampe ist promovierter Mathematiker. Nach dem Studium arbeitete er mehrere Jahre als Entwickler für Prozessrechner. Die dabei zwingende Notwendigkeit, mit sehr begrenzten technischen Mitteln für den Kunden einen messbaren ökonomischen Nutzen zu erreichen, hat ihn dauerhaft geprägt. Natürlich blieb ihm auch das zeitraubende und mühselige Suchen und Beheben von Programmierfehlern nicht erspart, sodass nahezu zwangsläufig der Wunsch reifte, Exaktheit und Effektivität der mathematischen Verfahren auf die Softwareentwicklung zu übertragen. Insbesondere die teilweise bereits formalisierten Fachsprachen schienen dafür ein guter Ausgangspunkt zu sein. Über Jahre beschäftigte er sich dann mit der Definition von Fachprogrammiersprachen und deren Implementierung. Diese Forschungen zeigten ihm unter anderem die Wichtigkeit der Anwendungsmodellierung für eine saubere Softwareentwicklung.

Mit diesen Voraussetzungen war der Weg zu Clean Code einfach eine gern genommene logische Fortsetzung. Sie bestärkte ihn auch in der Ansicht, dass große und komplexe Systeme nicht aus dem Stand, sondern nur in einer evolutionären Folge von kleinen Veränderungen – Irrtümer eingeschlossen – gebaut und lebensfähig erhalten werden können.

Inhaltsverzeichnis

Illustrationsverzeichnis

Einleitung

Software ist wichtig und wird immer wichtiger. Vielleicht haben Sie angesichts dieser Tatsache auch, so wie ich, das flaue Gefühl, dass die Qualität der Programme und die Kosten ihrer Entwicklung viel zu oft nicht den Erfordernissen entsprechen.

Es ist offensichtlich, dass die Erstellung von Software in vielerlei Richtungen entscheidend verbessert werden muss. Wenn das nicht gelingt, drohen Ausfälle mit gewaltigem Schadenspotenzial.

Berichte über – oft nach einem Update – nicht oder nur noch eingeschränkt funktionierende Bank- oder Buchungssysteme vermitteln einen kleinen Vorgeschmack darauf, was in Zukunft möglicherweise alles passieren kann.

Je mehr Software im Einsatz ist, desto bedeutender wird deren konsequente Verbesserung und Weiterentwicklung. Bereits heute entsteht ein großer Teil der Kosten nicht durch Neuentwicklungen, sondern durch die notwendigen Anpassungen.

Clean Code ist ein wichtiges Konzept, um Programme so zu schreiben, dass sie möglichst wenig Fehler enthalten und über lange Zeit stabil weiterentwickelt werden können. Jeder engagierte Entwickler sollte deshalb zumindest die wichtigsten Grundsätze kennen.

Über dieses Buch

Dieses Buch soll Ihnen die Gedankenwelt des Clean-Code-Konzepts nahebringen.

Das ist nicht mit der Darstellung einer Reihe von Regeln getan. Infolgedessen kann ich Ihnen auch nicht versprechen, dass Sie nach dem Studium dieses Buchs ein perfekter oder auch nur guter Clean-Code-Programmierer seien werden.

Ich kann Ihnen jedoch versprechen, dass Sie nach dem Lesen eine andere Sicht auf die Softwareentwicklung haben werden. Sie werden zumindest einiges besser verstehen und verständlicheren Code schreiben können. Und das ganz unabhängig davon, ob Sie das komplette Buch intensiv studiert oder nur Teile davon überflogen haben.

Softwareentwicklung kann gut mit der Expedition zu einem Ziel in schlecht erschlossenem Gelände verglichen werden. Je nachdem wie gut die Gegebenheiten bereits bekannt sind, lässt sich der erforderliche Aufwand mehr oder weniger genau vorhersagen. Je weniger Sie wissen, desto sorgfältiger muss die Unternehmung vorbereitet werden.

Mit diesem Buch will ich Ihnen helfen, sich zielstrebig auf Softwareprojekte einzustellen. Deshalb diskutiere ich nützliche handwerkliche Fertigkeiten und Verfahren ausführlich.

Daneben geht es mir aber auch um die mentale Vorbereitung. Anforderungen können sich stark unterscheiden und Schwierigkeiten beispielsweise durch den Umfang an Funktionen, Leistungsanforderungen oder ein unzureichendes Verständnis der Anwendungsdomäne verursacht werden.

Jeder dieser Punkte erfordert ein anderes Herangehen. Diese Differenzierung wird viel zu häufig unterlassen.

Schließlich noch ein dritter Punkt: Sie lernen (hoffentlich) jeden Tag etwas dazu. Ich zeige Ihnen

Wege, wie Sie das Gelernte nutzen können, um darauf aufbauend jeden Tag besser für Ihr Projekt zu arbeiten.

Mit dieser umfassenden Sicht hoffe ich, Ihnen auch dann zu helfen, wenn Sie die vorgestellten Techniken nicht oder nur stark eingeschränkt einsetzen können.

Und nicht zuletzt wünsche ich mir, dass dieses Buch auch dem einen oder anderen, der nicht selbst Code schreibt, aber eng mit IT-Entwicklungen verbunden ist, hilft, Softwareentwickler besser zu verstehen und ihnen den Freiraum zu gewähren, den sie für gute Ergebnisse brauchen.

Konventionen in diesem Buch

Wenn es konkret wird, verwende ich Java. Diese Programmiersprache ist sehr vielen Entwicklern zumindest bekannt und wird seit Jahren in zahlreichen Projekten eingesetzt. Allein aufgrund des vorhandenen Codebestands wird Java noch lange wichtig sein.

Allerdings ist Clean Code nicht an eine bestimmte Sprache gebunden. Fast alle Beispiele lassen sich ohne großen Aufwand in andere Sprachen übertragen. Und wahrscheinlich lernen Sie bei Ihren entsprechenden Versuchen sogar mehr, als wenn ich dieses Buch durch die Wiederholung von Beispielen in verschiedenen Programmiersprachen aufgebläht hätte.

Damit Sie sich gut zurechtfinden, erkläre ich Ihnen kurz die verwendeten Schriftarten und Hervorhebungen.

- In einem Buch über Code finden Sie erwartungsgemäß Quellcode-Listings. Zur Darstellung wird wie allgemein üblich eine Festbreiten-Schriftart verwendet:

```
public class DasIstEinBeispiel extends Nothing {}
```

- ✔ Wenn im Text auf konkrete in Java definierte Namen Bezug genommen wird, werden diese ebenfalls in der Schriftart für Code dargestellt, zum Beispiel, wenn es um die Basisklasse `java.lang.Object` geht.
- ✔ Neue Begriffe werden in der Regel *kursiv* gesetzt. Manchmal wird die Kursivschreibung auch zur Hervorhebung benutzt.

Was Sie nicht lesen müssen

Selbstverständlich bin ich der Meinung, dass es sich lohnt, das gesamte Buch zu lesen. Aber natürlich müssen Sie das nicht.

Die Texte in Kästen sind Hintergrundinformationen, die Sie überspringen können.

Was für Sie wichtig ist und was nicht, hängt stark davon, warum Sie sich für dieses Thema interessieren und was Sie dazu bereits wissen.

Wenn Ihnen das Thema Clean Code noch relativ neu ist und Sie überhaupt erst einmal verstehen wollen, worum es dabei geht, sollten Sie mit den Kapiteln 1, 2 und 4 starten und dann mit einem Ihnen interessant erscheinenden Thema aus Teil III fortfahren.

Wenn Sie Clean Code bereits kennen, können Sie eigentlich sofort mit Teil III beginnen. Trotzdem empfehle ich Ihnen, unabhängig davon auch die Kapitel 6 und 7 zu lesen.

Für alle, die keine oder wenig Programmiererfahrung haben, sind die Kapitel 1, 2, 6, 7 und 21 geeignet, um das Verständnis für den Softwareentwicklungsprozess zu erhöhen.

Der Aufbau des Buchs ist so gewählt, dass es von Anfang bis Ende einer Linie folgt. Trotzdem ist es so geschrieben, dass Sie grundsätzlich jedes Kapitel für sich allein lesen können.

Wenn einzelne Gesichtspunkte in einem anderen Kapitel detailliert behandelt werden, finden Sie entsprechende Verweise – denen Sie folgen können oder nicht.

Wenn Sie etwas schon wissen oder es Ihnen unwichtig erscheint, überspringen Sie es – oder lesen Sie es später. Wie ich Ihnen überhaupt empfehlen möchte, es nicht beim einmaligen Durchlesen zu lassen. Ich bin ziemlich sicher, dass Sie im Lichte Ihrer gewonnenen Erfahrungen bei jedem erneuten Lesen einige Dinge besser oder erst richtig verstehen werden, die Ihnen zuvor entgangen sind.

Törichte Annahmen über die Leser

In diesem Buch dreht sich alles um Softwareentwicklung und Code. Daher sollte mindestens einer der folgenden Punkte auf Sie zutreffen.

- ✔ **Sie können programmieren.**

  Für das vollständige Verständnis ist eine gewisse Vertrautheit mit einer im Idealfall objektorientierten Programmiersprache nötig.
- ✔ **Sie wirken an IT-Projekten als Entwickler mit.**

Praktisch erleben können Sie den Nutzen von Clean Code vor allem bei größeren und länger laufenden Softwareprojekten.

- ✔ **Sie wollen besseren Code produzieren.**

  Diese Motivation wird Ihnen ganz bestimmt helfen, schneller und gründlicher zu lernen.

- ✔ **Sie verfügen über Ausdauer und Stehvermögen.**

  Um sauberen Code zu schreiben, werden Sie möglicherweise lieb gewonnene Wege verlassen müssen. Eine solche Umstellung Ihrer Arbeitsweise ist nicht von heute auf morgen bewerkstelligt, sondern braucht Zeit und Training.

- ✔ **Sie sind eng mit der Entwicklung von Software verbunden.**

  Wenn Sie als Projektleiter, Product Owner oder in einer ähnlichen Rolle eng mit Entwicklern zusammenarbeiten, gehören Sie nicht unmittelbar zur angepeilten Zielgruppe. Sie können jedoch viel über deren Methoden und Probleme lernen und dadurch das gegenseitige Verständnis erheblich fördern. Abschnitte, die sich unmittelbar auf den Code beziehen, können Sie natürlich überspringen.

- ✔ **Sie sind bereits ein perfekter Clean-Coder.**

  Dann brauchen Sie dieses Buch eigentlich nicht. Lesen Sie es bitte trotzdem kritisch und machen Sie mich auf Fehler und Vergessenes aufmerksam. Oder – noch besser – verfassen Sie gleich »Cleaner Clean Code für Dummies«.

Wie dieses Buch aufgebaut ist

Dieses Buch besteht aus fünf Teilen.

Teil I: Das Clean-Code-Prinzip

In diesem Teil erfahren Sie die Grundgedanken des Clean-Code-Prinzips. Warum ist Code als solcher so wichtig? Was ist überhaupt guter und sauberer Code? Daneben geht es um Fragen des Abwägens zwischen konkurrierenden Zielstellungen.

Teil II: An Herausforderungen wachsen

In Teil II lernen Sie wesentliche Herausforderungen kennen, die Sie als Softwareentwickler bewältigen müssen. Der Schwerpunkt liegt dabei auf jenen Problemen, die vor dem Beginn des Codeschreibens gelöst sein sollten.

Diese Differenzierung der Problembereiche hilft Ihnen, keine unrealistischen Erwartungen bezüglich des Effekts von sauberer Programmierung zu hegen.

Teil III: Sauberen Code schreiben

Teil III zeigt Ihnen die wichtigsten Regeln zum Schreiben von sauberem Code. Dabei erfahren Sie jeweils auch, unter welchen Bedingungen eine Regel verwendet werden sollte und wann es besser ist, sie nicht anzuwenden.

Sie lernen zudem, wie Sie mögliche Fehler klassifizieren und dementsprechend sauber behandeln können.

Schließlich geht es auch noch um neuere Sprachbestandteile, und Sie sehen, wie diese ohne Beeinträchtigung der Codequalität sinnvoll eingegliedert werden können.

Teil IV: Wege zum Ziel

In Teil IV steht die praktische Umsetzung des Clean-Code-Konzepts im Mittelpunkt. Sie erhalten Tipps, wie es Ihnen gelingen kann, das Schreiben von sauberem Code in den Softwareentwicklungsprozess zu integrieren.

Teil V: Der Top-Ten-Teil

Im letzten Teil des Buchs habe ich noch ein paar nützliche Hinweise für Sie versammelt:

- Ratschläge, die Sie auf dem Weg zum Clean-Code-Entwickler vor Fehlern bewahren können und Ihnen hoffentlich helfen, das Ziel zu erreichen.
- Eine Liste von Webseiten, die Ihnen detailliertere Informationen zu den aufgeworfenen Themen und praktischen Fragen geben.

Symbole, die in diesem Buch verwendet werden

Die Glühbirne markiert einen Tipp. Falls Sie nur Wissen erwerben wollen, können Sie solche Hinweise überspringen. Wenn es Ihnen aber auch um die praktische Umsetzung geht, sollten Sie diese Tipps nicht nur einmal sorgfältig lesen.

An dieser Stelle finden Sie Hinweise, die Ihnen helfen sollen, bei der praktischen Anwendung des Dargestellten Fehler zu vermeiden.

Daher können Sie diese Texte überspringen, wenn Sie nur das Thema verstehen wollen.

Hier schränke ich die Bedeutung einzelner Wörter durch eine genauere Abgrenzung ein, um im folgenden Text Missverständnisse zu vermeiden. Solche Definitionen sollten Sie besser nicht ignorieren.

Mit dem Wegweiser kennzeichne ich zusätzliche Erläuterungen beispielweise durch Analogien aus anderen Bereichen oder kurze Beispiele. Für das Verständnis sind diese Ausführungen zwar nicht zwingend erforderlich, aber hoffentlich hilfreich.

Wenn Sie wenig Zeit haben oder das Kapitel bereits zum zweiten Mal lesen, können Sie diese Texte überspringen.

Umfangreichere Beispiele, die unabhängig vom umgebenden Text verständlich sind, werden auf diese Weise gekennzeichnet.

Sie können diese Teile beim ersten Lesen gern überspringen und eventuell später genauer studieren.

Wie es weitergeht

Ganz gleich, aus welchen Gründen Sie sich für Clean Code interessieren. Fangen Sie an zu lesen – das muss nicht auf Seite 1 sein – und versuchen Sie Ihre Erkenntnisse sinnvoll anzuwenden.

Bevor Sie nun loslegen, möchte Ihnen aber noch einen Rat geben. Den können Sie jetzt gern überspringen, aber

lesen Sie ihn bitte auf jeden Fall noch, bevor Sie eventuell aufgeben:

Es gibt keinen Express auf dem Weg zum guten Clean-Code-Entwickler. Viel eher ist ein Fußmarsch vonnöten. Wie lang und wie beschwerlich dieser Marsch ist, hängt vom Ausgangspunkt und den spezifischen Schwierigkeiten ab, die Sie überwinden müssen.

Und sollte Ihnen unterwegs einmal alles zu lange dauern und der Mut sinken, dann kann Ihnen diese jüdische Weisheit vielleicht neue Energie und Zuversicht geben: »Wer langsam und besonnen geht, doch oft zuerst am Ziele steht.«

In diesem Sinne – auf denn!

Teil I

Das Clean-Code-Prinzip

IN DIESEM TEIL ...

- ✔ Erfahren Sie, warum Code wichtig ist
- ✔ Erfahren Sie, dass Clean Code mehr ist als eine Sammlung von Regeln
- ✔ Lernen Sie wichtige Voraussetzungen für das Schreiben sauberen Codes kennen
- ✔ Werden Sie vor leicht zu übersehenden Fallstricken gewarnt

Kapitel 1

Software ist Code

IN DIESEM KAPITEL

Was wird von Software erwartet?

Die Softwarekrise und ihre Ursachen

Code ist wichtig

In diesem Kapitel werden, ausgehend von der Bedeutung, die Software im Alltag spielt, Probleme erläutert, die bei der Softwareentwicklung immer wieder auftreten. Danach wird die zentrale Stellung des Programmcodes im Entwicklungsprozess begründet.

Erwartungen an Software

Software ist aus dem täglichen Leben nicht mehr wegzudenken. Sie findet sich nicht nur in Computern und Smartphones, sondern ersetzt zunehmend sogar einen Teil der mechanischen Bauteile, beispielsweise in Türschlössern und Nähmaschinen. Vor allem die aufwendig herzustellenden Steuerungskonstruktionen weichen elektronisch angesteuerten einfachen Schaltelementen.

Den meisten Benutzern ist es dabei völlig gleichgültig, was Software ist. Sie erwarten, dass das Gerät oder das Programm funktioniert.

Allerdings ist niemand total überrascht, wenn das einmal nicht der Fall ist. Überlegen Sie kurz, wie oft Sie selbst

schon schulterzuckend einen Softwarefehler hinnehmen mussten. Meist ist das zum Glück nur ärgerlich.

Es scheint unmöglich zu sein, weitgehend fehlerfreie Software zu produzieren.

Der Vergleich mit anderen komplexen technischen Apparaten wirft allerdings die Frage auf, wieso es bisher nicht gelungen ist, Verbesserungen in vergleichbarem Maße zu erreichen wie etwa bei der Sicherheit von Flugzeugen. Darauf gibt es keine einfache Antwort.

Die beschriebene Problematik ist fast so alt wie der Computer. Bereits 1972 erwuchs aus der unbefriedigenden Situation der Begriff *Softwarekrise*, und 1993 wurde auf einer Tagung gefragt: »Kann eine Krise 20 Jahre dauern?«

Trotz unleugbarer Fortschritte muss man eingestehen, dass die Krisensymptome heute noch genauso sichtbar sind wie damals. Geändert hat sich die Wahrnehmung. Das, was zunächst als Krise erschien, wird mittlerweile als Dauerzustand hingenommen.

Das heißt allerdings nicht, dass Sie als Entwickler diesen Zustand akzeptieren müssen. Ich hoffe, Sie lesen dieses Buch gerade deshalb, um daran etwas zu ändern. Denn Veränderung tut not, und zwar immer dringender. Je mehr Software in unser Leben eingreift, desto schwerwiegender werden ganz zwangsläufig die Auswirkungen von Fehlern sein.

Softwarekrise

In den Anfangsjahren der Computernutzung entfiel der Löwenanteil der Kosten auf die Hardware. Programmiert wurden vor allem gut aufbereitete und verstandene mathematische Algorithmen. Bezeichnenderweise war die erste Programmiersprache im heutigen Sinn das 1957 erschienene FORTRAN, kurz für FORmula TRANslator. Über Methoden zur Softwareentwicklung dachte niemand intensiver nach.

Der technische Fortschritt führte jedoch schnell zu sinkenden Hardwarekosten und in der Folge zu einer stärkeren Verbreitung von Computern. Die Aufgaben wurden anspruchsvoller und komplizierter. Dadurch stiegen die Kosten der Programmierung und begannen Mitte der 1960er-Jahre diejenigen der Hardware zu übersteigen.

Gleichzeitig kam es zu ersten spektakulären Folgen von Programmierfehlern wie 1962 dem Verlust der Venussonde Mariner-1. Es wurde unübersehbar, dass mangelhafte Software gewaltige Kosten verursachen kann. So wie bisher konnte es nicht weitergehen.

Die Entwicklung war in eine Krise, die Softwarekrise, geraten. Auf der Suche nach einem Ausweg entstand das *Softwareengineering*. Dieser Begriff tauchte zum ersten Mal 1968 im Namen einer NATO-Tagung auf.

Trotz entscheidender Verbesserungen kann die Softwarekrise bis heute nicht als tatsächlich überwunden gesehen werden.

Probleme haben Ursachen

Wenn man auf Probleme stößt, ist es erfahrungsgemäß nützlich, deren Ursachen aufzuklären. In den meisten Fällen gibt es dafür nicht nur einen Grund. Für die Softwarekrise lassen sich zwei wichtige Ursachengruppen finden.

Allgemeine Ursachen

Für die oft mangelhafte Qualität von Software gibt es ebenso wie für das ganze oder teilweise Scheitern von Entwicklungsprojekten verschiedene Gründe, unter anderem:

- ✔ Ungenügende Vorbereitung, insbesondere unvollständige Aufgabenbeschreibung und Abgrenzung
- ✔ Fehler in der Projektorganisation und der Projektleitung
- ✔ Unverständnis oder Unkenntnis im Management bezüglich der Besonderheiten der Softwareentwicklung
- ✔ Unzureichende Mittel (Zeit und Geld)