

# Einleitung

---

Dieses Buch wird Ihnen helfen, ein besserer Programmierer zu werden.

Egal ob Sie alleine programmieren, in einem großen Projektteam oder als Berater mit mehreren Kunden gleichzeitig arbeiten, dieses Buch wird Ihnen helfen, Ihre Arbeit besser zu machen. Dies ist kein Buch über Theorien – wir konzentrieren uns auf pragmatische Themen und darauf, die eigenen Erfahrungen zu nutzen, um fundierte Entscheidungen zu treffen. Das Wort *pragmatisch* kommt vom lateinischen *pragmaticus* – „geschäftskundig“ –, was wiederum vom griechischen *πραττειν* abgeleitet ist und „tun“ bedeutet. In diesem Buch geht es ums Tun.

Programmieren ist ein Handwerk. Vereinfacht gesagt geht es darum, einen Computer dazu zu bringen, das zu tun, was man von ihm erwartet (oder was die Anwender von ihm erwarten). Als Programmierer sind Sie teils Zuhörer, teils Ratgeber, teils Dolmetscher und teils Diktator. Sie versuchen, schwer fassbare Anforderungen festzuhalten und sie so auszudrücken, dass eine bloße Maschine ihnen gerecht werden kann. Sie versuchen, Ihre Arbeit zu dokumentieren, so dass andere sie verstehen können und Sie versuchen, Ihre Programme so zu konstruieren, dass andere darauf aufbauen können. Nicht zuletzt tun Sie das alles im Wettlauf gegen die Zeit. Sie arbeiten jeden Tag an kleinen Wundern.

Das ist eine schwierige Aufgabe.

Von vielen Seiten wird Hilfe angeboten. Werkzeughersteller werben mit den Wundern, die ihre Produkte vollbringen können. Methoden-Gurus versprechen garantierten Erfolg mit ihrer Lehre. Jeder behauptet, seine Programmiersprache sei die beste, und jedes Betriebssystem ist angeblich die Antwort auf alle erdenklichen Übel.

Natürlich ist nichts davon wahr. Es gibt keine einfachen Antworten und nicht *die beste* Lösung, sei es ein Werkzeug, eine Programmiersprache oder ein Betriebssystem. Ob ein System besser als ein anderes ist, hängt immer von den gegebenen Umständen ab.

An dieser Stelle ist Pragmatismus notwendig. Sie sollten nicht mit einer bestimmten Technologie verheiratet sein, sondern einen ausreichend weiten Horizont und breite Erfahrung haben, um in konkreten Situationen gute Lösungen auszuwählen. Ihr Horizont beruht auf dem Verständnis der EDV-Grundlagen und Ihre Erfahrung entspringt einer breiten Basis praktischer Projekte. Gemeinsam machen Theorie und Praxis stark.

## Einleitung

Sie passen Ihr Vorgehen den aktuellen Gegebenheiten und der Umgebung an. Sie wichten die Bedeutung aller Faktoren, die auf ein Projekt wirken. Sie bieten angemessene Lösungen, die auf Erfahrung beruhen. Das ist für Sie ein kontinuierlicher Prozess, der jedes Projekt begleitet. Pragmatische Programmierer erledigen ihre Arbeit, und sie machen es gut.

### Wer sollte dieses Buch lesen?

Dieses Buch ist an Leser gerichtet, die erfolgreichere und produktivere Programmierer werden wollen. Vielleicht sind Sie frustriert, weil Sie glauben, Ihr wahres Potenzial nicht zu erreichen, oder Sie sehen, dass Kollegen mit Hilfe von Werkzeugen produktiver sind als Sie. Möglicherweise setzen Sie ältere Technologien ein und wollen erfahren, wie neue Ideen bei Ihrer Arbeit eingesetzt werden können.

Wir behaupten weder, wir hätten Antworten auf alle (oder auch nur die meisten) Fragen, noch dass unsere Ideen in allen Situationen anwendbar wären. Aber wenn Sie unserer Herangehensweise folgen, werden Sie schnell Erfahrung sammeln, Ihre Produktivität wird steigen und Sie werden ein besseres Verständnis für Software-Entwicklung im Ganzen bekommen. Nicht zuletzt werden Sie bessere Software schreiben.

### Was macht einen Pragmatischen Programmierer aus?

Jeder Software-Entwickler hat seine individuellen Stärken und Schwächen, Vorlieben und Abneigungen. Im Laufe der Zeit baut sich jeder seine persönliche Arbeitsumgebung auf, die seine Individualität genauso widerspiegelt wie seine Hobbys, Kleidung oder Haarschnitt. Pragmatische Programmierer haben aber viele der folgenden Eigenschaften gemeinsam.

- **Neues früh aufgreifen und anwenden (early adopter/fast adapter).** Sie haben ein untrügliches Gespür für Technologien und Techniken und Sie probieren gerne Dinge aus. Sie erfassen Neues schnell und bauen es in Ihr übriges Wissen ein. Ihr Selbstvertrauen wächst mit Ihrer Erfahrung.
- **Neugierde.** Sie stellen Fragen. „Das ist gut – Wie haben Sie das gemacht? Hatten Sie Probleme mit dieser Bibliothek? Ich habe von BeOS gehört, was ist da dran? Wie sind symbolische Links implementiert?“ Sie sammeln wie ein Eichhörnchen kleine Wissensbrocken, von denen jeder auch Entscheidungen beeinflussen kann, die Sie erst Jahre später treffen.
- **Kritisches Denken.** Sie nehmen nichts als gegeben hin, ohne die Hintergründe zu kennen. Wenn ein Kollege behauptet: „Das machen wir immer so“ oder ein Verkäufer die Lösung aller Probleme verspricht, werden Sie misstrauisch.

- **Realismus.** Sie gehen Problemen ganz auf den Grund, um sie vollständig zu verstehen. Dadurch bekommen Sie ein gutes Gefühl dafür, wie kompliziert das Problem ist und wie viel Zeit die Lösung erfordert. Das gibt Ihnen das nötige Durchhaltevermögen für Ihre Arbeit.
- **Hans Dampf in allen Gassen.** Sie sind ständig bemüht, sich in eine Vielzahl von Technologien und Umgebungen einzuarbeiten. Auch wenn Sie im Moment hochspezialisiert arbeiten, sind Sie immer bereit für neue Aufgaben und Herausforderungen.

Die grundlegendsten Eigenschaften haben wir uns bis zum Schluss aufgehoben. Sie gelten für alle Pragmatischen Programmierer und daher formulieren wir sie als Tipp:

### *Tipp 1: Kümmern Sie sich um Ihr Können* ◀

---

Wir sind der Meinung, dass Software-Entwicklung sinnlos ist, wenn man sich nicht darum kümmert, es gut zu machen.

### *Tipp 2: Denken Sie über Ihre Arbeit nach* ◀

---

Wenn Sie ein Pragmatischer Programmierer sein wollen, denken Sie bei allem was Sie tun, über Ihr Tun nach. Es geht nicht um ein einmaliges Reflektieren Ihrer gegenwärtigen Arbeit, sondern um ein ständiges, kritisches Bewerten jeder einzelnen Entscheidung. Schalten Sie nie auf Autopilot. Überdenken und kritisieren Sie Ihre eigene Arbeit immer in Echtzeit. Das alte Firmenmotto von IBM, *THINK!*, ist das Mantra des Pragmatischen Programmierers.

Wenn Ihnen das anstrengend erscheint, zeigt das Ihren Realismus. Es wird einiges Ihrer kostbaren Zeit in Anspruch nehmen – Zeit, die wahrscheinlich sowieso schon viel zu knapp ist. Die Belohnung dafür ist eine engere Beziehung zu der Arbeit, die Sie gerne tun, das Gefühl, immer mehr Fachgebiete zu beherrschen und die Freude an kontinuierlicher, persönlicher Entwicklung. Auf lange Sicht wird sich Ihre Investition auszahlen. Sie und Ihr Team arbeiten effektiver, schreiben leichter wartbaren Quelltext und Sie verbringen weniger Zeit in Besprechungen.

## Einzelne Pragmatiker, große Teams

Manche Leute glauben, dass es in großen Teams und komplexen Projekten keinen Platz für Individualität gibt. Sie behaupten: „Software-Entwicklung ist eine Ingenieursdisziplin, die zusammenbricht, wenn einzelne Team-Mitglieder selbst Entscheidungen treffen.“

Wir widersprechen.

## Einleitung

Software-Entwicklung *sollte* eine Ingenieursdisziplin sein. Das schließt aber individuelles, handwerkliches Können nicht aus. Denken Sie an die großen Kathedralen des Mittelalters. Jede davon verschlang, verteilt über viele Jahrzehnte, Tausende von Personen-Jahren Arbeit. Gesammelte Erfahrungen wurden an die nächste Generation von Baumeistern weitergegeben, die mit dem Erreichten die Baukunst vorantrieben. Die Zimmermänner, Steinmetze, Bildhauer und Glasbläser waren alles Handwerker, die bauliche Anforderungen interpretierten, um ein Ganzes zu schaffen, das den rein mechanischen Aspekt der Konstruktion übertrifft. Der Glaube an ihren individuellen Beitrag stützte das Bauvorhaben.

*Wir, die bloß Steine behauen, müssen uns immer Kathedralen vorstellen.*

*Credo eines Steinmetzes*

In einem Projekt gibt es immer Raum für Individualität und handwerkliches Geschick. Das gilt insbesondere für den gegenwärtigen Stand der Softwaretechnologie. In hundert Jahren mag unsere jetzige Form der Software-Entwicklung so archaisch aussehen wie die Techniken der Dombaumeister heute für Bauingenieure. Aber unser handwerkliches Können wird trotzdem geschätzt werden.

## Es ist ein kontinuierlicher Prozess

*Ein Tourist fragte beim Besuch des Eton College den Gärtner, wie er den Rasen so perfekt hinbekomme. „Das ist einfach“, antwortete er, „Sie wischen jeden Morgen den Tau ab, mähen ihn jeden zweiten Tag und walzen ihn einmal pro Woche.“ „Ist das alles?“, fragte der Tourist. „Ja.“, antwortete der Gärtner, „Machen Sie das 500 Jahre lang und auch Sie haben einen perfekten Rasen.“*

Großartiger Rasen braucht täglich ein wenig Pflege, genauso wie großartige Programmierer. Managementberater lassen in Unterhaltungen gerne das Wort „Kaizen“ fallen. „Kaizen“ ist der japanische Begriff für das Konzept kleiner, kontinuierlicher Verbesserungsschritte. Es gilt als einer der Hauptgründe für die dramatischen Produktivitäts- und Qualitätssteigerungen in der japanischen Produktion und wurde in aller Welt kopiert. Kaizen lässt sich auch auf die persönliche Entwicklung anwenden. Verbessern Sie Ihre Fähigkeiten täglich und erweitern Sie Ihr Repertoire um neue Werkzeuge. Im Laufe der Zeit werden Sie erstaunt sein, wie Ihr Erfahrungsschatz aufblüht und Ihre Fähigkeiten wachsen.

## Aufbau des Buches

Dieses Buch ist als Sammlung kurzer Abschnitte geschrieben, die jeweils ein besonderes Thema behandeln und in sich abgeschlossen sind. Zahlreiche Querverweise sollen helfen, jedes Thema in Zusammenhänge einzuordnen. Sie können das Buch in beliebiger Reihenfolge lesen.

Gelegentlich werden Sie auf einen Kasten mit der Beschriftung *Tipp n* stoßen (so wie Tipp 1, *Kümmern Sie sich um Ihr Können*). Tipps stellen einerseits entscheidende Stellen im Text heraus, andererseits stehen sie auch für sich alleine – wir selbst wenden sie täglich an. Eine Zusammenfassung aller Tipps finden Sie auf der Karte im Buch.

Anhang A enthält eine Fülle von Quellen: Das Literaturverzeichnis, URLs zu Quellen im Internet und eine Liste empfohlener Zeitschriften, Bücher und Berufsvereinigungen. Im ganzen Buch finden Sie Verweise auf das Literaturverzeichnis und die URLs – wie beispielsweise [KP99] und [URL 16].

An geeigneten Stellen haben wir Übungen und weiterführende Aufgaben eingefügt. Während die Übungen relativ einfache Lösungen haben, sind die Aufgaben ziemlich offen gestellt. Um Ihnen eine Vorstellung von unserer Denkweise zu geben, haben wir in Anhang B unsere Lösungen zu den Übungen zusammengetragen. Nur wenige Übungen haben eine *einzig* richtige Lösung. Die Aufgaben sind eher als Grundlage für Diskussionen oder Ausarbeitungen in Programmierkursen für Fortgeschrittene gedacht.

## Schall und Rauch

„Wenn ich ein Wort gebrauche“, sagte Humpty Dumpty in leicht verächtlichem Ton, „so bedeutet es das, was ich will, dass es bedeutet – nicht mehr und nicht weniger.“

Lewis Carroll, „Alice hinter den Spiegeln“

Quer über das Buch verteilt finden sich Fetzen von Fachjargon – entweder vollkommen vernünftige Wörter, die für eine technische Bedeutung missbraucht wurden, oder schreckliche Wortschöpfungen, denen von Informatikern ohne Gefühl für Sprache eine Bedeutung zugewiesen wurde. Wir versuchen, solche Fachbegriffe zu definieren oder zumindest ihre Bedeutung zu beschreiben, wenn wir sie zum ersten Mal verwenden. Dennoch sind uns sicher einige durch die Lappen gegangen und andere wie *Objekt* und *Relationale Datenbank* sind so gebräuchlich, dass eine Definition nur langweilen würde. Wenn Sie auf ein unbekanntes Wort stoßen, überspringen Sie es bitte nicht. Nehmen Sie sich die Zeit, es im Internet oder einem Informatik-Lehrbuch nachzuschlagen, und schicken Sie uns eine Email, damit wir in der nächsten Auflage eine Definition angeben können.

## Einleitung

Wir haben uns entschieden, Rache an den Informatikern zu nehmen. Für einige Fachbegriffe gibt es perfekte Wörter, die wir bewusst ignorieren, weil Fachjargon im Allgemeinen auf ein bestimmtes Problemgebiet oder eine bestimmte Phase der Software-Entwicklung beschränkt ist. Eine der Grundphilosophien dieses Buches ist jedoch, dass die meisten von uns vorgeschlagenen Techniken universell sind: Modularität zum Beispiel betrifft Quelltext, Entwurf und Team-Organisation. Wenn wir gebräuchliche Fachwörter in einem breiteren Kontext verwenden wollten, erschien es verwirrend und wir konnten den ganzen Ballast der ursprünglichen Bedeutung nicht loswerden. In diesen Fällen haben wir zum Verfall der Sprache beigetragen und unsere eigenen Wörter erfunden.

## Quelltexte und andere Quellen

Die meisten Beispiele in diesem Buch sind kompilierbaren Quelltexten entnommen, die auf unserer Webseite verfügbar sind.

[www.pragmaticprogrammer.com](http://www.pragmaticprogrammer.com)

Dort finden Sie auch Links zu nützlichen Quellen zusammen mit Aktualisierungen des Buches und Neuigkeiten zu weiteren Aktivitäten der Pragmatischen Programmierer.

## Sagen Sie uns Ihre Meinung

Wir würden gerne von Ihnen hören. Kommentare, Vorschläge, Hinweise auf Fehler im Text und Probleme mit den Beispielen sind uns willkommen. Schicken Sie eine Email an:

[ppbook@pragmaticprogrammer.com](mailto:ppbook@pragmaticprogrammer.com)<sup>1</sup>

## Danksagung

Als wir dieses Buch anfangen, hatten wir keine Ahnung, in was für einer Gemeinschaftsleistung das enden würde.

Addison-Wesley hat uns Hacker, die noch feucht hinter den Ohren waren, hervorragend durch den ganzen Prozess der Buchproduktion von der Idee bis zur Druckvorlage geführt. Großer Dank geht an John Wait und Meera Ravindiran für ihre Unterstützung am Beginn des Projektes, Mike Hendrickson, unseren begeisterten Redakteur (und genialen Cover-Designer!), Lorraine Ferrier und John Fuller für ihre Hilfe bei der Herstellung und die unermüdliche Julie DeBaggis, die uns alle zusammengehalten hat.

---

<sup>1</sup> Wenn es um die deutsche Übersetzung geht, schreiben Sie bitte an:  
[buch@derpragmatischeprogrammierer.de](mailto:buch@derpragmatischeprogrammierer.de)

## Einleitung

Dann sind da noch die Fachlektoren: Greg Andress, Mark Cheers, Chris Cleeland, Alistair Cockburn, Ward Cunningham, Martin Fowler, Thanh T. Giang, Robert L. Glass, Scott Henninger, Michael Hunter, Brian Kirby, John Lakos, Pete McBreen, Carey P. Morris, Jared Richardson, Kevin Ruland, Eric Starr, Eric Vought, Chris Van Wyk und Deborra Zukowski. Ohne ihre sorgfältigen Kommentare und wertvollen Einsichten wäre dieses Buch schlechter lesbar, ungenauer und doppelt so lang. Vielen Dank für Eure Zeit und Weisheit.

Die zweite Auflage dieses Buches profitierte erheblich von den Adлераugen unserer Leser. Vielen Dank an Brian Blank, Paul Boal, Tom Eckberg, Brent Fulgham, Louis Paul Hebert, Henk-Jan Olde Loohuis, Alan Lund, Gareth McCaughan, Yoshiki Shibata und Volker Wurst sowohl für das Finden der Fehler als auch für die höflichen Hinweise darauf.

Im Laufe der Zeit haben wir mit einer großen Zahl progressiver Kunden zusammengearbeitet, bei denen wir die Erfahrung, über die wir hier schreiben, gesammelt und vertieft haben. Kürzlich hatten wir das Glück, mit Peter Gehrke an einigen großen Projekten zu arbeiten. Seine Unterstützung und Begeisterung für unsere Techniken wissen wir sehr zu schätzen.

Dieses Buch wurde mit Hilfe von LaTeX, pic, Perl, dvips, ghostview, ispell, GNU make, CVS, Emacs, XEmacs, EGCS, GCC, Java, iContract und SmallEiffel mit den Bash- und zsh-Shells unter Linux erstellt. Das Umwerfende daran ist, dass diese tolle Software frei erhältlich ist. Wir schulden den tausenden Pragmatischen Programmierern in aller Welt, die zu diesen und anderen Werken beigetragen haben, großen Dank. Wir möchten uns besonders bei Reto Kramer für seine Hilfe mit iContract bedanken.

Keineswegs zuletzt stehen wir bei unseren Familien in tiefer Schuld. Sie haben sich nicht nur mit nächtlichem Tippen, riesigen Telefonrechnungen und unserer ständigen Abgelenktheit abgefunden, sondern hatten auch noch die Geduld, immer wieder zu lesen, was wir geschrieben haben. Danke, dass Ihr uns habt träumen lassen.

*Andrew Hunt  
David Thomas*