david THOMAS
andrew HUNT

Jubiläums-ausgabe

2. Auflage

DER PRAGMATISCHE PROGRAMMIERER

Ihr Weg zur Meisterschaft

Listings zum Buch unter:
plus.hanser-fachbuch.de

HANSER

Thomas/Hunt

Der Pragmatische Programmierer

Ihr Plus – digitale Zusatzinhalte!

Auf unserem Download-Portal finden Sie zu diesem Titel kostenloses Zusatzmaterial.

Geben Sie auf **plus.hanser-fachbuch.de** einfach diesen Code ein:

plus-4h16t-r2ag5

Bleiben Sie auf dem Laufenden!

Unser **Computerbuch-Newsletter** informiert Sie monatlich über neue Bücher und Termine. Profitieren Sie auch von Gewinnspielen und exklusiven Leseproben. Gleich anmelden unter:

www.hanser-fachbuch.de/newsletter

David Thomas
Andrew Hunt

Der Pragmatische Programmierer

Ihr Weg zur Meisterschaft

Jubiläumsausgabe

übersetzt von Jürgen Dubau

HANSER

Titel der Originalausgabe: „The Pragmatic Programmer: Your Journey To Mastery“, 20th Anniversary Edition

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.

Lektorat: Brigitte Bauer-Schiewek
Copy editing: Petra Kienle, Fürstenfeldbruck
Fachlektorat: Steffen Gemkow, Dresden
Layout: Manuela Treindl, Fürth
Umschlagdesign: Marc Müller-Bremer, München, *www.rebranding.de*
Umschlagrealisation: Max Kostopoulos
Titelmotiv: © Max Kostopoulos
Datenbelichtung, Druck und Bindung: Eberl & Kösel, Krugzell
Ausstattung patentrechtlich geschützt. Kösel FD 351, Patent-Nr. 0748702
Printed in Germany

Print-ISBN: 978-3-446-46384-4
E-Book-ISBN: 978-3-446-46632-6
E-Pub-ISBN: 978-3-446-46633-3

Für Juliet und Elli,
Zachary und Elizabeth,
Henry und Stuart

Inhalt

Geleitwort

Ich weiß noch, als Dave und Andy das erste Mal über die Neuauflage dieses Buchs twitterten. Das war eine große Meldung. Ich sah zu, wie die Entwicklerwelt mit Begeisterung reagierte. Mein Feed kam gar nicht zur Ruhe. Nach zwanzig Jahren ist *Der Pragmatische Programmierer* heute genauso relevant wie damals.

Es sagt viel aus, dass ein Buch mit einer solchen Geschichte solch eine Reaktion bewirkt. Ich hatte das Privileg, für dieses Vorwort ein unveröffentlichtes Exemplar zu lesen, und ich verstand, warum es solches Aufsehen erregte. Es ist zwar ein technisches Buch, aber es als solches zu bezeichnen, erweist ihm einen schlechten Dienst. Technische Bücher schüchtern oft ein. Sie sind vollgestopft mit großen Worten, obskuren Begriffen, überfrachteten Beispielen, bei denen man sich unabsichtlich dumm fühlt. Je erfahrener ein Autor ist, desto leichter vergisst er, wie es ist, neue Konzepte zu lernen oder ein Anfänger zu sein.

Trotz ihrer jahrzehntelangen Programmiererfahrung haben Dave und Andy die schwierige Herausforderung des Schreibens mit der gleichen Begeisterung gemeistert wie Menschen, die diese Lektionen gerade erst gelernt haben. Sie reden nicht von oben herab mit Ihnen. Sie setzen nicht voraus, dass Sie Experte sind. Sie gehen nicht einmal davon aus, dass Sie die erste Ausgabe gelesen haben. Sie nehmen Sie so, wie Sie sind: als Programmierer, die einfach nur besser werden wollen. Dazu haben sie dieses Buch verfasst, um Ihnen dabei zu helfen, dorthin zu gelangen, jeweils einen machbaren Schritt nach dem nächsten.

Fairerweise muss man sagen, dass sie dies bereits vorher gemacht hatten. Die ursprüngliche Veröffentlichung war voller konkreter Beispiele, neuer Ideen und praktischer Tipps zum Aufbau Ihrer Entwickler-Muskeln und zur Entwicklung Ihres Entwickler-Hirns, die auch heute noch gelten. Aber in dieser aktualisierten Ausgabe werden zwei Sachen verbessert.

Die erste ist offensichtlich: Einige ältere Verweise oder veraltete Beispiele entfallen und werden durch neue, moderne Inhalte ersetzt. Sie werden keine Beispiele für Schleifeninvarianten oder Build Machines finden. Dave und Andy sorgen mit ihren eindringlichen Inhalten dafür, dass die Lektionen immer noch wirksam sind und es keine Ablenkung durch veraltete Beispiele gibt. Hier werden alte Ideen wie DRY (*Don't Repeat Yourself*, dt. „Wiederholen Sie sich nicht") entstaubt und bekommen einen frischen Anstrich, der sie wirklich zum Glänzen bringt.

Aber die zweite Sache macht diese Veröffentlichung wirklich spannend. Nach der ersten Ausgabe hatten beide Gelegenheit, darüber nachzudenken, was sie zu sagen versuchten, was sie ihren Lesern mitgeben wollten und wie das aufgenommen wurde. Sie erhielten Rückmeldungen zu diesen Lektionen. So konnten sie erkennen, wo es knirschte, was verfeinert werden musste, was missverstanden wurde. In den zwanzig Jahren, in denen dieses Buch seinen Weg durch Hände und Herzen von Programmierern auf der ganzen Welt gefunden hat, studierten Dave und Andy diese Reaktionen und formulierten neue Ideen und neue Konzepte.

Sie haben die Bedeutung der Handlungsfähigkeit kennengelernt und erkannt, dass Entwickler vermutlich mehr davon besitzen als die meisten anderen Fachleute. Sie beginnen dieses Buch mit der einfachen und doch tiefgründigen Botschaft: „Es ist Ihr Leben." Das erinnert uns an unsere Macht bei unserer Code-Basis, in unseren Jobs, in unseren Karrieren. Es gibt den Ton für alles andere im Buch an – dass es mehr ist als nur ein weiteres technisches Buch voller Code-Beispiele.

Dieses Buch sticht wirklich aus den Regalen der Fachbücher heraus, weil hier Leute verstehen, was es bedeutet, ein Programmierer zu sein. Bei der Programmierung geht es darum, zu versuchen, die Zukunft weniger schmerzhaft zu gestalten. Es geht darum, Dinge für unsere Teamkollegen einfacher zu machen. Es geht darum, Fehler zu machen und wieder auf die Beine zu kommen. Es geht darum, gute Gewohnheiten zu etablieren. Es geht darum, die eigenen Instrumente zu verstehen. Coding ist nur ein Teil der Welt des Programmiererdaseins und dieses Buch untersucht diese Welt.

Ich verbringe viel Zeit damit, über diese Reise als Entwickler nachzudenken. Mit Coding bin ich nicht aufgewachsen, habe es nicht am College studiert. Ich habe meine Teenagerjahre nicht damit verbracht, an der Technik herumzubasteln. Ich bin mit Mitte zwanzig in die Welt der Programmierung eingetreten und musste lernen, was es heißt, Programmierer zu sein. Diese Gemeinschaft unterscheidet sich sehr von anderen, denen ich angehörte. Hier engagiert man sich auf einzigartige Weise für das Lernen und die praktische Anwendung, die sowohl erfrischend als auch einschüchternd ist.

Für mich war es, als würde ich in eine neue Stadt ziehen. Ich musste die Nachbarn kennenlernen, Geschäfte suchen, die besten Cafés finden. Es dauerte, bis mir die Gegend vertraut war und ich die effizientesten Routen kannte, Straßen mit hohem Verkehrsaufkommen vermeiden konnte und wusste, wann die Rush Hour ist. Für das andere Wetter brauchte ich auch neue Klamotten.

Die erste Zeit in einer neuen Stadt kann beängstigend sein. Wie schön, freundliche, sachkundige Nachbarn zu haben, die dort schon eine Weile leben? Wer zeigt einem die Nachbarschaft mit den Cafés und so? Jemand, der Kultur und Puls der Stadt versteht, sodass man sich nicht nur zu Hause fühlt, sondern zur Gemeinschaft beiträgt? Dave und Andy sind solche Nachbarn.

Als relativer Neuling wird man leicht überwältigt – nicht vom Akt des Programmierens, sondern von dem Prozess, ein Programmierer zu werden. Hier muss ein kompletter Mentalitätswandel stattfinden – Gewohnheiten, Verhaltensweisen und Erwartungen müssen sich ändern. Der Prozess, ein besserer Programmierer zu werden, geschieht nicht nur, weil man gelernt hat, wie man programmiert; er muss mit Absicht und bewusster Praxis durchgeführt werden. Dieses Buch ist eine Handreichung, damit Sie ein besserer Programmierer, eine bessere Programmiererin werden.

Aber täuschen Sie sich nicht – es sagt Ihnen nicht, wie Entwickeln geht. Es ist nicht in dieser Weise philosophisch oder wertend. Hier erfahren Sie schlicht und einfach, was Pragmatische Programmierer sind, wie sie arbeiten und wie sie an Code herangehen. Die Autoren überlassen es Ihnen zu entscheiden, ob Sie einer werden wollen. Wenn Sie das Gefühl haben, dass das nichts für Sie ist, werden sie es Ihnen nicht übelnehmen. Aber wenn Sie entscheiden, dass dies Ihr Weg ist, sind das freundliche Nachbarn, die Ihnen den Weg zeigen.

Saron Jitbarek

Gründer und CEO von CodeNewbie
Host von Command Line Heroes

Vorwort zur zweiten Ausgabe

In den 1990er-Jahren arbeiteten wir mit Unternehmen, deren Projekte Probleme bereiteten. Wir ertappten uns dabei, wie wir jedem das Gleiche sagten: Vielleicht sollten Sie das testen, bevor Sie es ausliefern? Warum geht der Build für diesen Code nur auf dem Rechner von Maria? Warum hat niemand die Anwender gefragt?

Um bei neuen Kunden Zeit zu sparen, begannen wir mit unseren Notizen. Und aus diesen Notizen wurde *Der Pragmatische Programmierer*. Zu unserer Überraschung schien das Buch den Nerv der Zeit zu treffen und es ist in den letzten 20 Jahren nach wie vor beliebt.

Aber 20 Jahre sind in Bezug auf Software viele Lebensalter. Nehmen Sie einen Entwickler von 1999 und setzen Sie ihn heute in ein Team und er würde sich sehr abkämpfen, in dieser fremden neuen Welt klarzukommen. Aber die Welt der 1990er-Jahre ist dem heutigen Entwickler ebenso fremd. Die Verweise des Buchs auf Dinge wie CORBA, CASE-Tools und indexbasierte Schleifen waren bestenfalls kurios und wahrscheinlich eher verwirrend.

Gleichzeitig haben 20 Jahre keinerlei Auswirkungen auf den gesunden Menschenverstand gehabt. Die Technologie mag sich geändert haben, aber die Menschen eben nicht. Praktiken und Ansätze, die damals eine gute Idee waren, sind auch heute noch gut. Diese Aspekte des Buchs sind gut gealtert.

Als es also an der Zeit war, zum 20. Jahrestag diese Ausgabe zu erstellen, mussten wir eine Entscheidung treffen. Wir könnten die Technologien, auf die wir verweisen, durchgehen, aktualisieren und dann Feierabend machen. Oder wir könnten die Annahmen, die den von uns empfohlenen Praktiken zugrunde liegen, im Lichte weiterer zwei Jahrzehnte Erfahrung noch einmal überprüfen.

Am Ende haben wir beides getan.

Daher ist dieses Buch so etwas wie ein Schiff des Theseus.[1] Etwa ein Drittel der Themen in diesem Buch sind brandneu. Von den übrigen wurden die meisten teilweise oder vollständig umgeschrieben. Unsere Absicht war es, die Dinge klarer, relevanter und hoffentlich etwas zeitloser zu machen.

Wir mussten einige schwierige Entscheidungen treffen. Wir haben den „Ressourcen"-Anhang weggelassen, sowohl weil es unmöglich wäre, auf dem neuesten Stand zu bleiben, als auch weil es einfacher ist, das Gewünschte zu finden. Angesichts des derzeitigen Überflusses an parallel arbeitender Hardware und des Mangels an guten Möglichkeiten, damit umzugehen, haben wir Themen im Zusammenhang mit Concurrency neu organisiert und verfasst. Ergänzt wird das durch Inhalte, die die sich verändernden Einstellungen und Umgebungen reflektieren: von der agilen Bewegung, die wir mit ins Leben gerufen haben, über eine steigende

[1] Wenn im Laufe der Jahre jede Komponente eines Schiffs ersetzt wird, wenn sie ausfällt, ist das daraus resultierende Schiff dann immer noch das gleiche Schiff?

Akzeptanz funktionaler Programmiersprachen bis hin zum wachsenden Bedürfnis nach Berücksichtigung von Privatsphäre und Sicherheit.

Interessanterweise gab es zwischen uns jedoch wesentlich weniger Diskussionen über den Inhalt dieser Ausgabe als beim Verfassen der ersten. Wir waren beide der Meinung, dass die wichtigen Sachen leichter zu erkennen sind.

Das Buch, das Sie in Händen halten, ist jedenfalls das Ergebnis. Bitte genießen Sie es. Vielleicht eignen Sie sich ein paar neue Praktiken an. Vielleicht beschließen Sie, dass einige der von uns vorgeschlagenen Dinge falsch sind. Lassen Sie sich auf Ihr Handwerk ein. Sagen Sie uns Ihre Meinung.

Denken Sie aber vor allem daran, dass es Spaß machen soll.

Aufbau des Buchs

Dieses Buch ist mehr als eine Sammlung von Tipps. Jedes Thema ist in sich abgeschlossen und behandelt einen bestimmten Aspekt. Zahlreiche Querverweise sollen helfen, jedes Thema in Zusammenhänge einzuordnen. Sie können das Buch mit seinen *Topics* 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 betonen einerseits entscheidende Stellen im Text, andererseits sind sie auch für sich alleine gültig – wir selbst wenden sie täglich an. Eine Zusammenfassung aller Tipps finden Sie auf dem Beihefter im Buch.

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 unsere Lösungen zu den Übungen im Anhang zusammengetragen. Nur wenige Übungen haben eine einzige richtige Lösung. Die Aufgaben sind eher als Grundlage für Diskussionen oder Ausarbeitungen in Programmierkursen für Fortgeschrittene gedacht.

Am Ende finden Sie auch eine kurze Bibliografie, in der die Bücher und Artikel aufgeführt sind, auf die wir ausdrücklich verweisen.

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 Fachjargonschnipsel – 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 Informatiklehrbuch nachzuschlagen, und schicken Sie uns eine E-Mail, damit wir in der nächsten Auflage eine Definition angeben können.

Wir haben uns entschieden, Rache an den Informatikern zu nehmen. Manchmal gibt es durchaus gute Fachwörter für Konzepte, doch wir haben beschlossen, diese Wörter zu ignorieren. Warum? Weil der vorhandene Fachjargon in der Regel auf einen bestimmten Problembereich oder auf eine bestimmte Entwicklungsphase beschränkt ist. Eine der Grundphilosophien dieses Buchs ist jedoch, dass die meisten von uns vorgeschlagenen Techniken universell sind: Modularität zum Beispiel betrifft Quelltext, Entwurf und Teamorganisation. 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 Sie von der Plus.Hanser-Webseite herunterladen können:

Die Quelltexte zu diesem Buch

Geben Sie auf *plus.hanser-fachbuch.de* einfach diesen Code ein:

```
plus-4h16t-r2ag5
```

Dann steht Ihnen der Download zur Verfügung.

Hinweis: In diesem Buch stehen die Namen der entsprechenden Dateien jeweils vor den Codes, die diesen Quelltexten entnommen wurden.

Außerdem haben wir eine eigene Website zum Pragmatischen Programmierer:

Die Website zum Pragmatischen Programmierer

Auf unserer Website

https://pragprog.com/titles/tpp20

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

Danksagungen für die zweite Ausgabe

Wir haben in den letzten 20 Jahren buchstäblich Tausende von interessanten Gesprächen über die Programmierung geführt, Menschen auf Konferenzen, bei Kursen und manchmal sogar im Flugzeug getroffen. Jedes einzelne hat unser Verständnis des Entwicklungsprozesses erweitert und zu den Aktualisierungen in dieser Ausgabe beigetragen. Wir danken Ihnen allen (und sagen Sie uns immer wieder, wenn wir uns irren).

Dank geht auch an die Teilnehmer am Beta-Prozess des Buchs. Ihre Fragen und Kommentare haben uns geholfen, die Dinge besser zu erklären.

Bevor wir zur Beta-Version übergingen, baten wir einige Leute um Kommentare zu diesem Buch. Dank an VM (Vicky) Brasseur, Jeff Langr und Kim Shrier für eure detaillierten Kommentare und an José Valim und Nick Cuthbert für eure technischen Überprüfungen.

Wir bedanken uns bei Ron Jeffries, dass wir das Sudoku-Beispiel verwenden durften.

Vielen Dank an die Leute von Pearson, die bereit waren, dass wir dieses Buch auf unsere Weise verwirklichen durften.

Ein besonderer Dank gilt der unentbehrlichen Janet Furlow, die alles meistert, was sie angeht, und uns bei der Stange hält.

Und schließlich Applaus für all die Pragmatischen Programmierer da draußen, die in den letzten zwanzig Jahren die Programmierung für alle besser gemacht haben! Wir freuen uns auf zwanzig weitere Jahre.

Aus der Einleitung zur ersten Ausgabe

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 πραγματικός abgeleitet ist und etwa „tüchtig" 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, sodass 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.

Sie passen Ihr Vorgehen den aktuellen Gegebenheiten und der Umgebung an. Sie gewichten 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 richtet sich an Leserinnen und Leser, die erfolgreichere und produktivere Programmiererinnen und Programmierer werden wollen. Vielleicht sind Sie frustriert, weil Sie glauben, Ihr wahres Potenzial nicht zu erreichen, oder Sie sehen, dass Kollegen mithilfe 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 die Individualität genauso widerspiegelt wie 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 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 Quantencomputern gehört, was ist da dran? Wie sind symbolische Links implementiert?“* Wie Eichhörnchen sammeln sie 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. Ein eingehendes Verständnis davon, dass ein Prozess schwierig sein oder länger dauern könnte, 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 machen, 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.

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 bloße Steine behauen, müssen uns immer Kathedralen vorstellen.*

In einem Projekt gibt es immer Raum für Individualität und handwerkliches Geschick. Das gilt insbesondere für den gegenwärtigen Stand der Software-Technologie. 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, dann haben auch Sie 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. Anders als beim Rasen in Eton werden Sie schon nach wenigen Tagen Ergebnisse sehen. Im Laufe der Zeit werden Sie erstaunt sein, wie Ihr Erfahrungsschatz aufblüht und Ihre Fähigkeiten wachsen.

1 Eine Pragmatische Philosophie

In diesem Buch geht es um Sie.

Vertun Sie sich nicht: Dies ist *Ihre* Karriere und wichtiger noch, siehe *Topic 1*, *Es ist Ihr Leben*. Es gehört *Ihnen*. Sie sind hier, weil Sie wissen, dass Sie ein besserer Entwickler werden und anderen ebenfalls dabei helfen können. Sie können ein Pragmatischer Programmierer werden.

Was zeichnet Pragmatische Programmierer aus? Wir denken, es ist die Einstellung, der Stil und die Philosophie, mit der sie Probleme in Angriff nehmen und lösen. Sie denken über das unmittelbare Problem hinaus, versuchen immer, es in einem größeren Zusammenhang zu sehen, und sind sich stets des großen Ganzen bewusst. Kann man überhaupt pragmatisch sein, ohne die Zusammenhänge zu sehen? Wie kann man sonst kluge Kompromisse schließen und überlegte Entscheidungen treffen?

Ein weiterer Schlüssel zu ihrem Erfolg ist die Bereitschaft, für alles, was sie tun, Verantwortung zu übernehmen. Das werden wir in *Topic 2*, *Der Hund hat meinen Quelltext gefressen*, diskutieren. Verantwortungsbewusst wie sie sind, werden Pragmatische Programmierer nicht untätig zusehen, wenn ihre Projekte durch Nachlässigkeit den Bach hinuntergehen. In *Topic 3*, *Software-Entropie*, beschreiben wir, wie man Projekte in tadellosem Zustand hält.

Die meisten Leute sträuben sich gegen Veränderungen. Manchmal aus gutem Grund, manchmal aber einfach nur aus Trägheit. In *Topic 4*, *Steinsuppe und gekochte Frösche*, beleuchten wir eine Strategie, mit der man Veränderungen einleiten kann. Im Interesse der Ausgewogenheit erzählen wir aber auch die abschreckende Geschichte einer Amphibie, welche die Gefahren schleichender Veränderung ignorierte.

Wenn Ihnen die Zusammenhänge Ihres Projekts bewusst sind, ist es einfacher zu erkennen, wie gut Ihre Software sein muss. Manchmal muss man Perfektion anstreben, aber im Allgemeinen geht es darum, Kompromisse zu schließen. Das betrachten wir in *Topic 5*, *Gut ist gut genug*.

Natürlich braucht man einen breiten Hintergrund an Wissen und Erfahrung, um all das zustande zu bringen. Lernen ist ein ständiger und kontinuierlicher Prozess. In *Topic 6*, *Ihr Wissensportfolio*, diskutieren wir einige Strategien, um dabei mitzuhalten.

Niemand von uns arbeitet in einem luftleeren Raum. Wir interagieren die meiste Zeit mit anderen. *Topic 7*, *Kommuniziere!*, zeigt Wege auf, wie wir das besser machen können.

Pragmatisches Programmieren erwächst aus einer Philosophie des pragmatischen Denkens. Dieses Kapitel soll das Fundament dafür legen.

1 Topic 1: Es ist Ihr Leben

Ich bin nicht auf dieser Welt, um deine Erwartungen zu erfüllen,
und du bist nicht auf dieser Welt, um meine zu erfüllen.

Bruce Lee

Es ist *Ihr* Leben. Es gehört Ihnen. Sie führen es. Sie gestalten es.

Viele Entwickler, mit denen wir sprechen, sind frustriert. Sie haben verschiedene Sorgen. Manche haben das Gefühl, dass sie in ihrem Job stagnieren, anderen meinen, dass sie bei der Technologie nicht mehr mithalten können. Einige finden, dass man mit ihnen nicht wertschätzend umgeht, oder sie fühlen sich unter Wert bezahlt oder stecken in einem toxischen Team. Manche wünschen sich, nach Asien oder Europa zu gehen oder im Homeoffice zu arbeiten.

Und allen stellen wir die gleiche Frage:

„Warum kannst du das nicht ändern?“

Software-Entwicklung muss auf jeder Liste von Berufen, bei denen Sie die Kontrolle haben, ganz oben stehen. Unsere Fähigkeiten sind gefragt, unser Wissen geht über geografische Grenzen hinaus, wir können aus der Ferne arbeiten. Wir werden gut bezahlt. Wir können wirklich so ziemlich alles tun, was wir wollen.

Aber aus irgendeinem Grund scheinen sich die Entwickler gegen Veränderungen zu sträuben. Sie gehen in Deckung und hoffen, dass die Dinge besser werden. Sie sehen passiv zu, wie ihre Fähigkeiten langsam veralten, und beklagen sich, dass ihre Unternehmen sie nicht fortbilden. Sie betrachten im Bus Anzeigen für exotische Reiseziele, steigen dann hinaus in den kalten Regen und trotten zur Arbeit.

Hier ist also der wichtigste Tipp dieses Buchs.

Tipp 3

Sie haben Einfluss.

Lässt Ihr Arbeitsumfeld zu wünschen übrig? Ist Ihr Job langweilig? Versuchen Sie, das zu beheben. Aber versuchen Sie es nicht ewig. Wie Martin Fowler sagt: „Sie können Ihre Organisation ändern oder Ihre Organisation ändern.“[1]

Wenn Technologie scheinbar an Ihnen vorüberzieht, nehmen Sie sich die Zeit (in Ihrer Freizeit), um neue Dinge zu lernen, die interessant wirken. Sie investieren in sich selbst, daher ist es nur vernünftig, dies auch außerhalb der Arbeitszeit zu tun.

Möchten Sie remote arbeiten? Haben Sie danach gefragt? Wenn Sie ein Nein bekommen, dann finden Sie jemanden, der Ja sagt.

Diese Branche bietet Ihnen eine bemerkenswerte Reihe von Möglichkeiten. Seien Sie proaktiv und ergreifen Sie diese Möglichkeiten.

[1] *http://wiki.c2.com/?ChangeYourOrganization*

Verwandte Topics

- Topic 4, *Steinsuppe und gekochte Frösche*
- Topic 6, *Ihr Wissensportfolio*

Topic 2: Der Hund hat meinen Quelltext gefressen

2

Die größte aller Schwächen ist die Angst, schwach zu erscheinen.

J. B. Bossuet, „Die Politik nach den Worten der Heiligen Schrift“, 1709

Einer der Eckpfeiler der pragmatischen Philosophie ist die Idee, Verantwortung für sich selbst, seine Handlungen in Bezug auf den beruflichen Fortschritt, seine eigene Bildung, sein Projekt und seine tägliche Arbeit zu übernehmen. Pragmatische Programmierer nehmen ihre Karriere selbst in die Hand und schrecken nicht davor zurück, Unwissen oder Fehler zuzugeben. Es ist sicherlich nicht der angenehmste Teil des Programmierens, aber es kommt vor – selbst in den besten Projekten. Trotz gründlicher Tests, guter Dokumentation und solider Automatisierung gehen Dinge schief. Auslieferungen verspäten sich. Unvorhergesehene technische Probleme treten auf.

So etwas passiert einfach und wir versuchen, damit so professionell wie möglich umzugehen. Das bedeutet, offen und ehrlich zu sein. Wir können stolz auf unsere Fähigkeiten sein, aber wir müssen zu unseren Schwächen stehen – und ehrlich damit umgehen: mit Unwissen genauso wie mit Fehlern.

Teamvertrauen

Vor allem muss Ihr Team in der Lage sein, Ihnen zu vertrauen und sich auf Sie zu verlassen – und Sie müssen sich auch auf jeden einzelnen von ihnen verlassen können. Vertrauen in ein Team, so sagt es die Forschung, ist für Kreativität und Zusammenarbeit absolut unerlässlich.[2] In einer gesunden und vertrauensvollen Umgebung können Sie sicher Ihre Meinung sagen, Ihre Ideen präsentieren und sich auf Ihre Teammitglieder verlassen, die sich wiederum auf Sie verlassen können. Ist kein Vertrauen vorhanden, tja ...

Stellen Sie sich vor, Sie wollen mit einem High-Tech-Ninja-Team verdeckt das böse Versteck des Schurken infiltrieren. Nach monatelanger Planung und heikler Ausführung gelingt es, dort einzudringen. Jetzt sind Sie dabei, das Laserführungsgitter einzurichten, doch dann sagt einer: „Tut mir leid, Leute, ich habe den Laser nicht mit. Die Katze wollte zu Hause mit dem roten Punkt spielen, da habe ich ihn zu Hause gelassen.“

Diese Art von Vertrauensbruch dürfte schwer zu beheben sein.

---

[2] Siehe z. B. eine gute Meta-Analyse bei *Trust and team performance: A meta-analysis of main effects, moderators, and covariates*, *http://dx.doi.org/10.1037/apl0000110*

Verantwortung übernehmen

Auf Verantwortung muss man sich aktiv einigen. Sie verpflichten sich, dass etwas ordnungsgemäß erledigt wird, aber Sie haben nicht immer auch die direkte Kontrolle über alles, was damit zusammenhängt. Sie müssen nicht nur Ihr Bestes geben, sondern auch Risiken analysieren, die außerhalb Ihres Einflusses liegen. Sie haben das Recht, die Verantwortung *abzulehnen*, wenn die Sache aussichtslos oder das Risiko zu groß ist oder die ethischen Implikationen unklar sind. Diese Entscheidung müssen Sie selbst anhand Ihrer Werte treffen.

Wenn Sie Verantwortung *übernehmen*, müssen Sie davon ausgehen, zur Rechenschaft gezogen zu werden. Geben Sie es offen zu, wenn Sie (wie wir alle) Fehler gemacht haben, und bieten Sie mögliche Auswege an.

Schieben Sie die Schuld nicht auf irgendjemand anderen und erfinden Sie keine Ausreden. Machen Sie nicht den Hersteller, die Programmiersprache, das Management oder Ihre Kollegen verantwortlich. Jeder davon oder auch alle zusammen sind eventuell daran beteiligt, aber Ihre Aufgabe ist es, Lösungen zu präsentieren und nicht Ausreden.

Wenn das Risiko besteht, dass ein Hersteller nicht liefern kann, brauchen Sie einen Plan B. Wenn die Festplatte kaputt geht, dabei all Ihren Quelltext mitreißt und Sie keine Sicherheitskopie haben, ist das Ihr Fehler. Wenn Sie Ihrem Chef erzählen: „Der Hund hat meinen Quelltext gefressen“, hilft das auch nicht weiter.

Tipp 4

Bieten Sie Alternativen statt billiger Ausreden.

Halten Sie inne und denken Sie nach, bevor Sie irgendjemandem sagen, wieso etwas unmöglich sei, sich verzögert oder nicht funktioniert. Reden Sie mit der Gummi-Ente auf Ihrem Monitor oder Ihrem Hund. Klingt Ihre Ausrede vernünftig oder dumm? Wie wirkt sie auf Ihren Chef?

Lassen Sie das Gespräch in Ihrem Kopf ablaufen. Was wird Ihr Gegenüber sagen? Wird er fragen: „Haben Sie schon dies ausprobiert?“ oder „Haben Sie schon daran gedacht?“ Was werden Sie antworten? Gibt es noch etwas, das Sie versuchen können, bevor Sie die schlechten Neuigkeiten verkünden? Sie wissen meist genau, welche Antwort Sie bekommen werden, also ersparen Sie sich und anderen den Ärger.

Bieten Sie Alternativen statt Ausreden. Erzählen Sie nicht, dass etwas unmöglich sei, sondern wie die Situation gerettet werden kann. Falls Code gelöscht werden muss, machen Sie das allen deutlich und erklären Sie den Wert des Refactoring (siehe *Topic 40*, *Refactoring*).

Wenn Sie Zeit brauchen, um etwas auszuprobieren, dann machen Sie einen Plan, wie Sie vorgehen wollen (siehe *Topic 13*, *Prototypen und Post-it-Zettel*). Müssen Sie bessere Tests (siehe *Topic 41*, *Testen fürs Entwickeln*, und *Schonungsloses Testen*) oder Automatisierung einführen, um zu verhindern, dass sich das wiederholt?

Vielleicht benötigen Sie zusätzliche Ressourcen, um diese Aufgabe zu erfüllen. Oder vielleicht müssen Sie mehr Zeit mit den Benutzern verbringen? Vielleicht liegt es auch an Ihnen: Müssen Sie sich eine Technik oder Technologie grundsätzlich aneignen? Würde ein Buch oder Seminar helfen? Scheuen Sie sich nicht, danach zu fragen oder zuzugeben, dass Sie Hilfe brauchen.

Versuchen Sie, billige Ausreden herunterzuschlucken, bevor Sie sie aussprechen. Wenn es trotzdem sein muss, erzählen Sie es erst Ihrem Hund. Praktisch, wenn das Hündchen dann auch die Schuld auf sich nimmt ...

Verwandtes Topic

- Topic 49, *Pragmatische Teams*

Aufgaben

- Wie reagieren Sie, wenn jemand – zum Beispiel ein Bankangestellter, ein Kfz-Mechaniker oder ein Verkäufer – eine billige Ausrede präsentiert? Was denken Sie dann über ihn und seine Firma?
- Wenn Sie sich dabei ertappen zu sagen: „Ich weiß es nicht", sollten Sie auf jeden Fall ein „... aber ich werde es herausfinden" nachschieben. So können Sie sehr gut zugeben, was Sie nicht wissen, aber dann auch wie ein Profi die Verantwortung übernehmen.

Topic 3: Software-Entropie

3

Für die Software-Entwicklung gelten fast keine physikalischen Gesetze, nur die unerbittliche Zunahme der *Entropie* trifft uns hart. *Entropie* ist ein Begriff aus der Physik, der das Maß der Unordnung in einem System beschreibt. Unglücklicherweise besagen die Gesetze der Thermodynamik, dass die Entropie im Universum das Maximum anstrebt. Wenn Unordnung in der Software zunimmt, sagen Programmierer: „Die Software vergammelt." Manche nehmen hier den optimistischeren Begriff „technische Schulden" und implizieren damit, dass diese Schulden eines Tages beglichen werden. Sie werden es wahrscheinlich nicht tun.

Doch egal wie man das nennt, sowohl Schulden als auch Schimmel können sich unkontrolliert ausbreiten.

Es gibt viele Faktoren, die zum „Vergammeln von Software" beitragen. Der wichtigste scheint die Psychologie beziehungsweise Arbeitskultur in einem Projekt zu sein. Selbst wenn man ein Ein-Mann-Team ist, kann die Projektkultur ein empfindliches Pflänzchen sein. Trotz bestens ausgearbeiteter Pläne und bester Leute kann ein Projekt im Laufe der Zeit dem Ruin und Verfall ausgeliefert sein. Im Gegensatz dazu gibt es andere Projekte, die trotz enormer Schwierigkeiten und ständiger Rückschläge erfolgreich gegen die natürliche Tendenz zur Unordnung ankämpfen und ziemlich gut dastehen.

Was macht den Unterschied aus?

In Innenstädten findet man oft schöne und saubere Gebäude, aber auch schäbige und heruntergekommene Betonklötze, häufig ganz nah beieinander. Warum? Kriminalitätsforscher und Städteplaner haben einen faszinierend einfachen Auslöser entdeckt, der saubere, intakte und bewohnte Gebäude in beschmierte, beschädigte und verlassene Bruchbuden verwandelt.[3]

[3] *The police and neighborhood safety* [WH82]

Eine zerbrochene Fensterscheibe.

Eine einzelne, zerbrochene Fensterscheibe, die nicht bald repariert wird, löst bei den Bewohnern des Gebäudes das Gefühl aus, *aufgegeben* worden zu sein, dass die Verantwortlichen sich also nicht um das Haus kümmern. So werden weitere Fensterscheiben zerbrochen und die Leute lassen ihren Müll überall liegen. Graffitis erscheinen und der Vandalismus nimmt seinen Lauf. In relativ kurzer Zeit wird das Gebäude so stark beschädigt, dass der Besitzer kein Interesse mehr an einer Renovierung hat. Aus dem Gefühl des Verlassenseins wird Realität.

Warum sollte das einen Unterschied machen? Psychologen haben Studien[4] durchgeführt, die zeigen, dass Hoffnungslosigkeit ansteckend sein kann. Denken Sie an das Grippevirus und wie es in beengten Verhältnissen wirkt. Das Ignorieren einer eindeutig kaputten Situation verstärkt die Vorstellung, dass vielleicht nichts repariert werden kann, dass sich niemand kümmert, dass alles dem Untergang geweiht ist; alles negative Gedanken, die sich unter den Teammitgliedern ausbreiten und eine bösartige Spirale in Gang setzen können.

Tipp 5

Akzeptieren Sie keine zerbrochenen Fensterscheiben.

Lassen Sie „zerbrochene Fensterscheiben" (schlechte Entwürfe, falsche Entscheidungen oder schlechter Quelltext) nicht so stehen. Reparieren Sie alles, sobald es entdeckt wird. Wenn nicht genügend Zeit ist, um es ordentlich zu beheben, müssen Sie den Schaden *provisorisch* ausbessern. Vielleicht können Sie den betroffenen Quelltext auskommentieren, durch Testdaten ersetzen oder eine Meldung ausgeben, dass diese Funktion noch nicht implementiert ist. Unternehmen Sie *irgendetwas*, um weiteren Schaden zu vermeiden und zu zeigen, dass Sie diesen Zustand beseitigen werden.

Wir haben deutlich gesehen, dass funktionierende Systeme sich ziemlich schnell verschlechtern, sobald Fensterscheiben zerbrochen werden. Es gibt noch andere Faktoren, die zum Vergammeln von Software beitragen, und wir werden an anderer Stelle darauf eingehen, aber die Vernachlässigung beschleunigt das Vergammeln von Software stärker als jeder andere Faktor.

Sie werden vielleicht denken, dass niemand Zeit dafür hat, all die Scherben eines Projekts aufzusammeln. Wenn Sie weiterhin so denken, überlegen Sie besser schon mal, sich eine Mülldeponie zu kaufen oder in einen anderen Stadtteil zu ziehen. Lassen Sie die Entropie nicht gewinnen.

Vor allem aber: Richten Sie keinen Schaden an

Andy hatte einmal einen Bekannten, der unverschämt reich war. Sein Haus war makellos, voll mit unbezahlbaren Antiquitäten, Kunstwerken und so weiter. Ein Wandteppich hing ein bisschen zu nahe am Kamin und fing Feuer. Die Feuerwehr eilte herbei, um zu retten, was zu retten war. Aber bevor sie die dicken, dreckigen Schläuche in sein Haus schleppten, hielten sie vor der Feuersbrunst inne, um eine Matte zwischen Haustür und Brandherd auszurollen.

Sie wollten den Teppich nicht ruinieren.

[4] Siehe *Contagious depression: Existence, specificity to depressed symptoms, and the role of reassurance seeking* [Joi94]

Das klingt jetzt ziemlich extrem. Sicherlich ist es die erste Priorität der Feuerwehr, Feuer zu löschen, Kollateralschäden werden hingenommen. Aber sie hatten die Situation klar eingeschätzt, waren von ihrer Fähigkeit überzeugt, mit dem Feuer fertig zu werden, und achteten darauf, das Eigentum nicht unnötig zu beschädigen. So muss es auch bei Software sein: Verursachen Sie keine Kollateralschäden, nur weil es irgendeine Krise gibt. Ein einziges zerbrochenes Fenster ist eines zu viel.

Eine zerbrochene Fensterscheibe – ein schlecht entworfenes Stück Quelltext, eine unglückliche Managemententscheidung, mit der das Team das ganze Projekt lang leben muss – reicht aus, dass der Verfall ausgelöst wird. Wenn Sie selber in einem Projekt mit einigen zerbrochenen Fensterscheiben sind, ist es nur zu einfach, zu der Einstellung zu kommen: „Der ganze Rest des Quelltexts ist schon Schrott. Ich mach's einfach genauso." Es ist unerheblich, ob das Projekt bis zu diesem Zeitpunkt gut gelaufen ist. In dem Experiment, das zur „Theorie der zerbrochenen Fensterscheibe" führte, stand ein abgemeldetes Auto eine Woche lang unbeschädigt herum. Aber sobald ein einziges Fenster eingeschlagen war, wurde es binnen *Stunden* ausgeschlachtet und umgeworfen.

Wenn Sie selber in einem Projekt sind, in dem der Quelltext tadellos schön ist – sauber geschrieben, gut entworfen und konsequent gepflegt –, werden Sie aus diesem Grund besondere Sorgfalt walten lassen – wie die Feuerwehrmänner. Selbst wenn ein Brand wütet (ein Fertigstellungstermin, eine Demonstration auf einer Messe etc.), werden Sie nicht der Erste sein wollen, der Schaden anrichtet.

Sagen Sie sich einfach: „Keine zerbrochenen Fenster."

Verwandte Topics

- Topic 10, *Orthogonalität*
- Topic 40, *Refaktorisieren*
- Topic 44, *Dinge benennen*

Aufgaben

- Stärken Sie Ihr Team, indem Sie auf Ihre „Projekt-Nachbarschaft" aufpassen. Wählen Sie zwei oder drei „zerbrochene Fensterscheiben" aus und diskutieren Sie Probleme und mögliche Lösungen mit Ihren Kollegen.
- Merken Sie, wenn eine Fensterscheibe zerbricht? Wie reagieren Sie? Was können Sie tun, wenn es durch die Entscheidung eines anderen oder eine Managementvorgabe verursacht worden ist?

4 Topic 4: Steinsuppe und gekochte Frösche

Drei Soldaten kamen aus dem Krieg heim und waren hungrig. Als sie ein Dorf vor sich sahen, hellte sich ihre Stimmung auf. Sie waren sich sicher, dass die Dorfbewohner ihnen zu essen geben würden. Aber als sie das Dorf erreichten, waren die Türen verschlossen und die Fenster verriegelt. Nach vielen Jahren des Krieges mangelte es den Dorfbewohnern an Nahrung und sie hamsterten alles, was sie hatten.

Die Soldaten ließen sich davon nicht abschrecken, kochten in einem Topf Wasser und legten vorsichtig drei Steine hinein. Neugierig kamen die Dorfbewohner, um zu gaffen.

„Das ist Steinsuppe", erklärten die Soldaten. „Ist das alles, was ihr da reintut?", fragten die Dorfbewohner. „Genau – obwohl sie mit ein paar Karotten besser schmecken soll …" Ein Dorfbewohner lief weg und kehrte im Nullkommanichts mit einem Korb Karotten aus seinem Vorrat zurück.

Ein paar Minuten später fragte wieder einer der Dorfbewohner: „Ist das alles?"

„Na ja", sagten die Soldaten, „ein paar Kartoffeln würden ihr etwas Gehalt geben." Ein anderer Dorfbewohner lief los.

Eine Stunde lang zählten die Soldaten weitere Zutaten auf, die die Suppe verfeinern würden: Fleisch, Lauch, Salz und Kräuter. Jedes Mal lief ein anderer Dorfbewohner los und plünderte seine eigenen Vorräte.

Schließlich hatten sie einen großen Topf dampfende Suppe gekocht. Die Soldaten nahmen die Steine heraus und genossen mit den Dorfbewohnern die erste anständige Mahlzeit, die sie alle seit Monaten gegessen hatten.

In der Geschichte von der Steinsuppe stecken mehrere Lehren. Die Dorfbewohner werden von den Soldaten getäuscht, die deren Neugier ausnutzen, um an Essen zu kommen. Viel wichtiger ist aber, dass die Soldaten den Anstoß dazu geben, das Dorf zusammenzubringen und gemeinsam etwas zuzubereiten, was sie alleine nicht zustande gebracht hätten. Letztendlich gewinnt jeder dabei.

Hin und wieder möchten Sie es vielleicht genauso machen wie die Soldaten.

Sie sind vielleicht in einer Situation, in der Sie genau wissen, was und wie etwas getan werden muss. Sie haben das ganze System schon bildlich vor Augen und wissen, dass es funktionieren wird. Wenn Sie dann aber um Erlaubnis fragen, das Ganze in Angriff zu nehmen, ernten Sie bloß erstaunte Blicke und werden auf später vertröstet. Es werden Ausschüsse gebildet, Budgets müssen bewilligt werden und alles wird kompliziert. Jeder wird mit seinen Ressourcen geizen. Es ist manchmal nur die Angst vor dem ersten Schritt.

Dann ist es Zeit, die Steine herauszuholen. Überlegen Sie sich, um *was* Sie vernünftigerweise bitten können. Machen Sie etwas daraus, zeigen Sie es den Leuten, wenn es fertig ist, und lassen Sie sie staunen. Tun Sie so, als ob es nicht so wichtig sei, und warten Sie einfach darauf, dass die Leute Sie um die zusätzliche Funktionalität bitten, die Sie selbst ursprünglich haben wollten. Menschen finden es leichter einzusteigen, wenn etwas bereits erfolgreich ist. Geben Sie ihnen flüchtig einen Blick in die Zukunft und sie werden sich um alles bemühen.[5]

[5] Sie werden folgendes Zitat tröstlich finden, das Konteradmiral Dr. Grace Hopper zugeschrieben wird: „Es ist einfacher, um Vergebung zu bitten, als um Erlaubnis zu fragen."

Tipp 6

Geben Sie den Anstoß zu Veränderungen.

Aus Sicht der Dorfbewohner

Von der anderen Seite betrachtet geht es in der Geschichte um sanfte, allmähliche Täuschung. Sie handelt davon, sich zu sehr auf etwas zu konzentrieren. Die Dorfbewohner grübeln über die Steine und vergessen darüber alles andere. Wir fallen alle darauf herein, täglich. Die Dinge wachsen uns einfach über den Kopf.

Wir haben alle schon entsprechende Symptome gesehen. Projekte geraten langsam und unaufhaltsam außer Kontrolle. Die meisten Katastrophen bei der Software-Entwicklung fangen mit kaum zu bemerkenden Kleinigkeiten an, und einzelne Tage summieren sich zu großen Terminüberschreitungen. Systeme weichen Feature für Feature von ihrer Spezifikation ab, während Änderung über Änderung gemacht wird, bis nichts mehr vom ursprünglichen Quelltext übrig ist. Häufig sind es die kleinen Dinge, die Teams und Motivation zerstören.

Tipp 7

Denken Sie an das große Ganze.

Wir haben das natürlich nie ausprobiert, aber man sagt, wenn man einen Frosch in kochendes Wasser wirft, springt er sofort wieder heraus. Wenn man ihn jedoch in kaltes Wasser setzt und es langsam erhitzt, bemerkt der Frosch dies nicht und wird sitzen bleiben, bis er gar ist.

Beachten Sie, dass das Problem des Frosches ein anderes ist als das mit den zerbrochenen Fensterscheiben in *Topic 3*, *Software-Entropie*. In der „Theorie der zerbrochenen Fensterscheiben" verlieren die Leute den Willen, die Unordnung zu bekämpfen, weil sie denken, es kümmere sich niemand darum. Der Frosch nimmt nicht mal die Veränderung wahr.

Seien Sie nicht wie der sprichwörtliche Frosch. Betrachten Sie immer das große Ganze. Sie müssen stets beobachten, was um Sie herum passiert, nicht nur das, was Sie selbst tun.

Verwandte Topics

- Topic 1, *Es ist Ihr Leben*
- Topic 38, *Programmierung durch Zufall*

Aufgaben

- Bei der Begutachtung des Entwurfs der ersten Ausgabe dieses Buchs stellte John Lakos die folgende Frage: Die Soldaten täuschen die Dorfbewohner zunehmend, aber die Veränderung ist zu ihren Gunsten. Dem Frosch wird jedoch durch zunehmende Täuschung Schaden zugefügt. Wie kann man feststellen, ob man Steinsuppe oder Froschsuppe kocht, wenn man Veränderungen anstößt? Ist das eine objektive oder eine subjektive Entscheidung?
- Sagen Sie schnell, ohne jetzt hinzusehen, wie viele Lichter sind in der Decke über Ihnen? Wie viele Ausgänge hat der Raum? Wie viele Personen sind da? Gibt es etwas, das aus dem Zusammenhang gerissen ist, etwas, das aussieht, als gehöre es nicht dazu?

Hierbei handelt es sich um eine Übung des *Situationsbewusstseins*. Diese Technik wird von Pfadfindern bis hin zu Navy Seals praktiziert. Gewöhnen Sie sich an, Ihre Umgebung wirklich zu beobachten und wahrzunehmen. Dann machen Sie dasselbe bei Ihrem Projekt.

5 Topic 5: Gut ist gut genug

Wer bessern will, macht oft das Gute schlecht.

Shakespeare, „König Lear“, 1.4

Es gibt einen alten Witz über eine Firma, die bei einem japanischen Hersteller 100.000 integrierte Schaltkreise bestellt. Teil der Bedingungen war die Fehlerrate von 1 zu 10.000. Einige Wochen später kam die Lieferung: eine große Kiste mit Tausenden von Chips und eine kleine Kiste mit nur zehn Stück. Auf der kleinen Kiste stand: „Das sind die fehlerhaften.“

Wenn wir bloß dieses Ausmaß an Kontrolle über die Qualität hätten. Aber die Realität lässt uns nun mal kaum etwas wirklich perfekt herstellen, insbesondere keine fehlerfreie Software. Die Zeit, die Technologien und unser Naturell haben sich gegen uns verschworen.

Das muss uns trotzdem nicht frustrieren. Ed Yourdon beschreibt in einem Artikel in *IEEE Software* [You95], dass man sich disziplinieren kann, Software zu schreiben, die gut genug ist – gut genug für die Anwender, gut genug für jene, die sie zukünftig pflegen sollen, und gut genug für seine eigene innere Ruhe. Sie werden produktiver sein und Ihre Anwender zufriedener. Sie werden vielleicht entdecken, dass die Programme besser sind, wenn Sie nicht mehr so lange darüber brüten.

Bevor wir fortfahren, müssen wir klarstellen, was hier ausgesagt werden soll. Mit dem Ausdruck „gut genug“ meinen wir nicht schlampige oder schlechte Software. Alle Systeme müssen die Anforderungen ihrer Anwender erfüllen, um erfolgreich zu sein, und grundlegende Leistungs-, Datenschutz- und Sicherheitsstandards einhalten. Wir plädieren nur dafür, dass die Nutzer die Möglichkeit erhalten, sich an dem Prozess der Entscheidung zu beteiligen, wann das, was Sie produziert haben, gut genug für ihre Bedürfnisse ist.

Beteiligen Sie Ihre Anwender an dem Kompromiss

Im Allgemeinen schreiben Sie Software für andere Leute. Häufig können Sie sich auch noch daran erinnern, von ihnen Anforderungen bekommen zu haben.[6] Aber fragen Sie diese Leute jemals, *wie gut* ihre Software sein soll? Manchmal hat man keine Wahl. Wenn man an Herzschrittmachern, Autopiloten oder einer weit verbreiteten Basisbibliothek arbeitet, werden die Anforderungen strenger und Ihre Wahlmöglichkeiten eingeschränkter sein.

Wenn Sie jedoch an einem brandneuen Produkt arbeiten, sind Sie anderen Zwängen unterworfen. Die Leute vom Marketing müssen Versprechen erfüllen, die Anwender haben Pläne gemacht, die auf den Auslieferungsterminen basieren, und Ihre Firma muss regelmäßige

[6] Das war natürlich ein Witz!