Gene Kim • Steve Yegge

Deutsche Ausgabe des Bestsellers

Vibe Coding

Prinzipien der Softwareentwicklung mit GenAI, Chat und Agenten

dpunkt.verlag

4 Die dunkle Seite: Wenn Vibe Coding furchtbar schiefgeht

Wir haben die positiven FAAFO-Seiten des Vibe Codings beleuchtet. Aber wie jede neue Technologie besitzt auch das KI-unterstützte Coding seine dunkle Seite. Ihre KI-Souschefin wird vielleicht Ihre hilfreichste Mitarbeiterin sein, aber wenn Sie nicht aufpassen, kann sie auch ein atemberaubend destruktives Potenzial entwickeln.

Ein ähnliches Muster zeigte sich bei der Einführung der Elektrizität in der Fertigungsindustrie. Das außerordentliche Potenzial der Elektrizität war offensichtlich, aber erst 20 Jahre nach ihrer Entdeckung haben Fabrikbesitzer gelernt, ihre linearen, bandgetriebenen Anlagen aufzugeben und sich Ausgestaltungen zuzuwenden, die die Flexibilität der Elektrizität zu nutzen wussten.

Die heutige KI-Programmierungsrevolution folgt einem vergleichbaren Muster – wir können das gewaltige Potenzial sehen, lernen aber immer noch, es zu beherrschen, ohne Fehler auszulösen, die Monate der Arbeit in Minuten zerstören, Codebasen auslöschen oder physische Hardware beschädigen können.

Schauen wir uns die Geschichte der Software an, gibt es viele Gründe dafür, Hoffnung zu haben. So wie Sir Tony Hoare[1] es erlaubte, Zeiger im Speicher auf null zeigen zu lassen – sein berühmter »Milliarden-Dollar-Fehler« – oder die manuelle Speicherverwaltung in C Jahrzehnte lang Pufferüberläufen und Sicherheitslücken zuließ, haben wir es doch irgendwann geschafft, Technologien zu erschaffen, die die schlimmsten dieser Probleme abmildern.

KI-Coding kann für systematische Risiken sorgen, die sich über Entwicklungsökosysteme verteilen können. Die Einsätze können höher und die Fehler spektakulärer sein als alles, was wir in der klassischen Softwareentwicklung bisher erlebt haben. Aber wir glauben, dass die Prinzipien und Praktiken, die unsere Softwareprozesse in den letzten Jahrzehnten verbessert

1. Auch bekannt als C. A. R. Hoarse hat Sir Hoarse Quicksort und ALGOL entwickelt (der Vorläufer so gut wie jeder Programmiersprache, einschließlich C, Smalltalk, Java und so weiter). Außerdem hat er CSP erschaffen (Communication Sequential Processes), nach dem das Parallelverarbeitungsmodell von Go modelliert ist.

haben, so angepasst werden können, dass potenzielle Fallstricke vermieden werden. Im Folgenden lesen Sie reale Geschichten vom Vibe Coding, das furchtbar schiefging. Lassen Sie unsere mühsam errungenen Lektionen ein Ticket für Ihren Erfolg sein.

Fünf abschreckende Beispiele aus der Küche

Die verschwindenden Tests: Wo ist mein Code?

Steve hat zwei Wochen nach Beginn des Einsatzes von Coding-Agenten eine beängstigende Erfahrung gemacht. Er begann damit, die automatisierte Testsuite für Wyvern mit einem Agenten zu konvertieren und musste von seinem Kollegen lernen, dass der Coding-Agent still und heimlich die Tests deaktiviert oder gehackt hat, damit sie funktionieren, und ganze 80 % der Testfälle in einer großen Suite gelöscht hat.

Schlimmer noch – bis Steve das herausfand, waren schon eine Reihe von Commits vergangen. Seitdem waren viele produktive Änderungen am Branch eingebaut worden, sodass ein Rollback nicht zielführend war. Steve steckte in einem Dilemma. In der Nacht schrieb er an Gene: »Ich habe Claude Code gebeten, sich um meine Tests zu kümmern – und das hat es auch getan, so wie Godzilla sich um Tokio gekümmert hat.«

Steves KI-Assistent hatte weder erwähnt, dass er diese Tests gelöscht hat, noch hatte er um Erlaubnis gefragt – sie wurden einfach entfernt. Wir beschreiben in Teil 2, was passieren kann und warum dies geschieht. In Teil 3 geht es dann darum, was Sie dagegen tun können.

Die Lovecraft'sche Horror Code Base: Wenn FAAFO stirbt

Um das Schreiben dieses Buchs zu unterstützen (und während des Schreibens dieses Buchs), hat Gene drei Generationen eines Autoren-Werkzeugkastens geschaffen, den wir in diesem Buch als Writer's Workbench bezeichen. Das Ziel war, die immense Menge an manuellem Handling von Prompts und Teilen des Manuskripts zu reduzieren, denn beides musste immer wieder in unterschiedliche Tools kopiert und von dort übernommen werden. Sein Writer's Workbench begann als Google Docs Add-on. Die dritte Iteration war schließlich eine Terminal-Anwendung, die eine Reihe von Weiterentwicklungen durchlief, während sie von ihm und Steve beim Entwurfs- und Bearbeitungsprozess intensiv zum Einsatz kam.

Alles lief gut. Gene hatte sie täglich von morgens bis abends verwendet und schließlich über 20 Millionen Token verarbeitet. Es war sehr einfach, den Writer's Workbench um zusätzliche Funktionalität zu ergänzen ... bis es

plötzlich nicht mehr einfach war. Die Codebasis wurde zu etwas, was Gene als »Eldritch Horror«, also gespenstischen Horror, beschrieb – eine riesige, 3.000 Zeilen lange Funktion ohne modulare Grenzen, die unmöglich zu verstehen war und sich auch nicht mehr anpassen ließ, ohne dass woanders etwas kaputt ging.

»Ich konnte die Funktion, die die KI geschrieben hatte, um die temporären Arbeitsdateien zu sichern, nicht verstehen«, erinnert sich Gene. »Ich brauchte 20 Minuten, um die drei Argumente zu verstehen, die von der Funktion genutzt wurden, und schon zehn Minuten später hatte ich sie wieder vergessen.« Gene verbrachte drei anstrengende Tage damit, den Code umzuschreiben und zu modularisieren (mithilfe der KI) und ihn durch Tests zu unterstützen, um die Korrektheit der Funktionalität sicherzustellen, auf die sie tagtäglich angewiesen waren.

Das brachte FAAFO schließlich wieder heraus aus dem kosmischen Abgrund, und dieses Tool half Gene und Steve dabei, den Lektorinnen und Lektoren 50 Millionen Token später den ersten Entwurf übergeben zu können. Wir werden die verwendeten Techniken in Teil 3 beschreiben, wo es darum geht, wie wir diese Art von Problemen verhindern, erkennen und korrigieren.

Das verschwundene Repository: Fast katastrophaler Datenverlust

Die vielleicht am meisten alarmierende Geschichte kommt von Steve, der eines Tages feststellte, dass sein TypeScript-Client-Code für Wyvern – ungefähr 10.000 Zeilen Code und Tausende von Dateien, die Wochen von Arbeit und etwa 1.000 Dollar in Claude-Code-Token entsprachen – verschwunden waren. Nicht nur aus seinem Projektverzeichnis – alle Dateien und ihre Backups waren weg. Auch im Remote-Bitbucket-Repository waren sie nicht mehr zu finden (Yay!). Steve erlebte »diesen Moment, wo dein Herz stehen bleibt und du die fünf Stufen der Verzweiflung in nur wenigen Millisekunden durchläufst« – so als wenn Sie unabsichtlich eine Produktivdatenbank löschen und wissen, dass es kein Backup gibt.

Durch pures Glück entdeckte Steve schließlich ein offenes Terminal-Fenster mit einem verwaisten Klon des Codes – es war die letzte verbleibende Kopie dieses Codes auf der ganzen Welt. Hätte er dieses Terminal geschlossen oder auch *nur das Verzeichnis gewechselt*,[2] wäre alles für immer verloren gewesen. Sein KI-Assistent hatte unzählige Git-Branches mit kryptischen Namen erstellt. Bei einer Aufräumaktion hatte Steve ihn angewiesen, »unnötige«

2. Dies ist das Deleted-Unix-File-System-Inode-Problem. Hätte er das Verzeichnis verlassen, wäre es, ohne Spuren zu hinterlassen, vom Garbage Collector abgeräumt worden.

Branches zu entfernen, dabei aber nicht gemerkt, dass diese Branches nicht eingecheckten Code enthielten, der unerwarteterweise auch nicht nach main gemergt worden war – mit einem Großteil des node-Clients. In Teil 3 beschreiben wir, wie Sie diese Art von Problemen verhindern, erkennen und korrigieren.

Das hardwarenahe Desaster: Physische Konsequenzen

Digitale Fehler sind schlimm genug, aber die KI kann auch physisch Schaden anrichten. Unser Freund Luke Burton, ein Entwickler, der zwei Jahrzehnte bei Apple verbracht hat und nun bei Nvidia arbeitet, nutze einen Coding-Agenten, um ein Tool zum Automatisieren von Firmware-Updates auf eine CNC-Maschine zu erstellen. Allerdings hatte er während einer Vibe-Coding-Session fast schon die Eingabetaste gedrückt, bevor er bemerkte, dass sein KI-Assistent vorgeschlagen hatte, das CNC-Storage-Device zu eliminieren.

Luke schrieb uns alarmiert: »Das scrollte alles so schnell vorbei, dass ich es fast übersehen hätte. Ich war nur einen Alt-Tab entfernt davon, die Maschine in den Ausgangszustand zurückversetzen zu müssen. Dafür hätte ich an die Rückwand gemusst, und diese Maschine wiegt 50 kg.« Durch KI initiierte Coding-Fehler können über Software hinausgehen und physische Devices oder Systeme beschädigen. (Auch hierzu beschreiben wir Mittel zum Abmildern in Teil 3.)

Der ungehorsame Koch: Wenn die KI direkte Anweisungen ignoriert

Gene arbeitete mit der KI zusammen, um die Authentifizierung für die Trello-API einzurichten. Obwohl er sie explizit anwies: »Lies die Datei aus dem Java-Ressourcen-Verzeichnis – gehe dabei wie folgt vor ...«, ignorierte der Coding-Agent seine Anweisungen und schrieb weiterhin Code, der darauf direkt über das Dateisystem zugriff.

Der Code funktionierte weiterhin ..., wenn Gene ihn aus seinem Projektverzeichnis heraus ausführte. Aber hätte er diesen Fehler nicht erkannt, als er die Änderungen des Coding-Agenten untersuchte, würde der Code fehlschlagen, wenn er als Bibliothek in einem anderen Programm eingesetzt würde – eine subtile Zeitbombe, die vielleicht erst Wochen oder Monate später gezündet hätte. Wie wir in Teil 2 erläutern werden, kann die KI Probleme damit haben, Anweisungen zu folgen, was noch schlimmer werden kann, wenn die Kapazität des Kontextfensters ausgereizt wird. Wir werden Ihnen zeigen, wie Sie diese Problematik erkennen und was Sie dagegen tun können.

Genie – aber unvorhersagbar

Wie diese Geschichten zeigen, ist Vibe Coding wie das Arbeiten mit einer ausgesprochen talentierten, aber unfassbar inkonsistenten Souschefin. An guten Tagen kann sie Meisterwerke erzeugen, die Sie sich in Ihren wildesten Träumen nicht vorgestellt haben und wo einfache Zutaten in kulinarische Magie verwandelt werden. Aber an schlechten Tagen kann die gleiche Köchin Ihre Küche abfackeln, Ihre Gäste vergiften oder mitten im Service einfach verschwinden. Bei einer normalen Souschefin würden Sie vielleicht eine Mahlzeit verlieren oder ein paar Zutaten verschwenden. Aber bei der KI steht mehr auf dem Spiel – funktionierender Code, kritische Tests, ganze Repositories oder physische Hardware. (Und um das Ganze noch demütigender zu machen, wird der KI-Anbieter Sie für das Privileg zahlen lassen, Ihre Mahlzeit ungenießbar zu machen und die ruinierten Gerichte neu zuzubereiten.)

Diese abschreckenden Beispiele sollen Sie nicht vom Vibe Coding abhalten – wir sind aus vielen Gründen weiterhin enthusiastische Befürworter. Aber sie unterstreichen, warum die Techniken und Sicherheitsmaßnahmen aus dem Rest des Buchs so wichtig sind. Ohne ordentliche Überwachung, Verkosten/Testen und Küchenregeln kann sich Ihre KI-Souschefin von Ihrer größten Produktivitätshilfe in Ihren schlimmsten Albtraum verwandeln. Und wenn dieser Albtraum geschieht, werden Sie vielleicht der Grund dafür, warum die Geschäftsführung KI-Köchinnen und -Köche aus der Restaurant-Kette verbannt.

Diese Sorgen bezüglich der potenziellen Nachteile der KI basieren nicht nur auf unseren persönlichen Erfahrungen – sie zeigen sich mittlerweile auch in den Daten. Die *State of DevOps Reports*, die Gene begleitet hat, werden von Googles DORA-Forschungsgruppe fortgeführt. Der Report von DORA aus dem Jahr 2024 enthielt eine überraschende Erkenntnis: Jeweils 25 % Wachstum beim Einsatz von GenAI korrelieren mit einer um 7 % schlechteren Stabilität (mehr Ausfälle und längere Wiederherstellungszeiten) und einem um 1,5 % langsameren Durchsatz (Deployment-Häufigkeit und Lead Times).[3]

Diese Ergebnisse passen sicherlich zu den zu Tränen rührenden Geschichten, die wir weiter oben erzählt haben. Aber wir nennen dieses Ergebnis die »DORA-Anomalie«, weil unserer allgemeinen Erfahrung nach die Chancen groß sind, dass Vibe Coding *auch* den Durchsatz erhöhen und die Stabilität steigern kann. Das führte uns dazu, Anfang 2025 ein gemeinsames Forschungsprojekt zu starten, und wir hoffen, weitere Ratschläge liefern zu

3. DeBellis et al., »The Impact of Generative AI in Software Development Report«

können, welche Faktoren erforderlich sind, damit Vibe Coding gut funktioniert. (Mehr dazu in Teil 4.)

Jede große, neue Technologie hat ihre Anfangsprobleme, die durch Unfälle und sogar Desaster sichtbar werden, bevor sich Sicherheitsmaßnahmen und gute Praktiken durchsetzen. Sie können das Risiko durch sorgfältiges Aufteilen von Aufgaben, strikte Verifikation, strategische Kontrollpunkte und mehr verringern, wie wir Ihnen in diesem Buch noch zeigen werden. Wir haben diese Fehler gemacht, sodass Sie sie nicht wiederholen müssen – und wir haben kampferprobte Vorgehensweise entwickelt, um sicherzustellen, dass Ihr Weg mit Vibe Coding all die FAAFO-Vorteile hervorbringt und die negativen Aspekte weglässt.

»Das scheinen ziemliche Anfängerfehler zu sein«

Viele Leute, die wir bewundern und deren Meinungen wir vertrauen, haben uns zu diesem Buch wundervolles Feedback gegeben. Aber einige Leute haben uns auch gesagt: Ihr zwei seid erfahrene Entwickler und habt entweder sehr große Systeme bei Amazon und Google aufgebaut oder seit Jahrzehnten umfassende Forschungen zu effektiven Software-Delivery-Praktiken betrieben. Und doch sieht es so aus, als ob ihr grundlegende Dinge wie Versionsverwaltung oder automatisiertes Testen vergessen habt. Das scheinen ziemliche Anfängerfehler zu sein, und ihr habt die KI sich austoben und Chaos in euren Code veranstalten lassen.

Vielleicht denken Sie das Gleiche – wir sind froh, dass das erwähnt wurde. Wir haben diese Fehler gemacht, obwohl wir unserer Meinung nach eine gesunde Portion Sorgfalt und Paranoia haben. Aber wir fühlten uns wie jemand, der seit Jahrzehnten ein Pferd reitet und nun die Schlüssel für ein modernes Auto bekommen hat. Oder – genauer gesagt – ein modernes Formel-1-Auto. Wir haben unser Auto geschrottet. Viele, viele Male.

Wie jeder auf diesem Planeten haben wir gelernt, diese neuen und neuartigen Tools zu verwenden, für die es wenige bis gar keine Vorbilder gibt. Jemandem, der es gewohnt war, Pferde zu reiten, werden das erforderliche mentale Modell, das Muskelgedächtnis und die Gewohnheiten fehlen, die zum Fahren eines Autos gebraucht werden. Die gute Nachricht ist, dass die gleichen zentralen Prinzipien und Praktiken, die es uns ermöglicht haben, Software schneller, sicherer und fröhlicher zu deployen, während wir von einem Deployment pro Jahr (was in den 2000ern typisch war) zu 136.000 Deployments pro Tag (was Amazon 2015 geschafft hat) gekommen sind, genauso nutzen lassen, wenn wir uns vom Erzeugen von einhundert Zeilen Code pro Tag zu mehreren Tausend und darüber hinaus bewegen.

Wir werden das genauer in Teil 3 betrachten, wo wir beschreiben, wie wir unsere inneren, mittleren und äußeren Entwicklungsschleifen anpassen sollten.

Die Versprechen von morgen versus die Realität von heute

Es wird der Tag kommen, an dem Sie sich an Ihre KI-Souschefin wenden können, sie bitten »Erstelle mir ein Fünf-Gänge-Menü für den wichtigen Kunden morgen« und dann weggehen. Der Souschefin, die mit Ihrer kulinarischen Philosophie, Ihren Geschmacksvorlieben und den Restaurantstandards wohlvertraut ist, können Sie vertrauen, sich ganz allein darum zu kümmern. Sie versteht Ihre expliziten Anweisungen, den nicht ausgesprochenen Kontext, die Geschichte Ihres Restaurants und Ihre langfristigen Visionen.

Kommen Sie am nächsten Tag wieder, ist das Menü geplant, die Zutaten sind vorbereitet, die Posten sind organisiert und alles ist für eine fehlerfreie Ausführung bereit – so wie Sie es gemacht hätten, oder sogar noch besser. Wir glauben, dass dieser Tag kommen wird. Aber Mitte 2025 haben wir noch einen weiten Weg vor uns, um so viel Vertrauen aufbringen zu können. Seit 2019 hat sich der Zeithorizont von Aufgaben, die die KI zuverlässig erledigen kann, alle sieben Monate verdoppelt[4] – von einer maximalen Aufgabenlänge, die sich 2019 noch in Sekunden messen ließ, bis heute, wo wir uns mehreren Stunden nähern.[5] Forschende sagen voraus, dass die KI noch in diesem Jahrzehnt Softwareaufgaben erledigen können wird, die Monate brauchen.

Aber Mitte 2025 müssen wir uns immer noch mit einer signifikanten Lücke bei den Fähigkeiten befassen. Ihre aktuelle KI-Souschefin ist ohne Zweifel klassisch im Umgang mit einem Messer ausgebildet und sie hat auch jedes Kochbuch gelesen. Aber wenn wir KI-Coding-Agenten unüberwacht an größeren Aufgaben arbeiten ließen, haben wir Folgendes erlebt:

- Codebasen auf eine Art und Weise umwandeln, die die Besitzerinnen und Besitzer schockiert
- in endlosen Forschungsschleifen gefangen sein und immer weiter Untersuchungen durchführen, ohne fertig zu werden
- sich zu immer komplexeren Lösungen versteigen, um Probleme in ihrem Code zu beheben

4. Kwa et al., »Measuring AI Ability to Complete Long Tasks«
5. Patel, »Is RL + LLMs Enough for AGI? – Sholto Douglas & Trenton Bricken«

- einfache Features zu aufwendig und mit unnötigen Abstraktionsschichten erledigen
- Dokumentation erstellen, die sich immer weiter von dem entfernt, was der Code tut
- wichtige Funktionalität teilweise abschalten oder umgehen, weil die ursprünglichen Anforderungen nicht mehr im Blick behalten werden

Wenn man diese Lücke versteht – die immer kleiner wird – und lernt, diese zu berücksichtigen, kann man auch effektives Vibe Coding betreiben. Statt von aktuellen Beschränkungen abgeschreckt zu werden, passen erfolgreiche Fachleute ihre Vorgehensweise an, um das Beste aus den aktuellen Fähigkeiten der KI herauszuholen und sich gleichzeitig auf ihre rapide Weiterentwicklung vorzubereiten:

1. **Mit Bedacht delegieren:** Wählen Sie wohldefinierte, kleinere Aufgaben, wo die Erfolgskriterien klar und überprüfbar sind.
2. **Ausreichend überwachen:** Beobachten Sie genauer, wenn die Aufgabe neu, komplex oder von hoher Auswirkung ist.
3. **Leitplanken einsetzen:** Legen Sie explizite Grenzen für das fest, was die KI anpassen sollte und was nicht.
4. **Die Arbeit regelmäßig überprüfen:** Kontrollieren Sie Ausgaben, um frühzeitig Probleme zu erkennen – insbesondere bei kritischen Systemkomponenten.
5. **Persistente Referenzen erstellen:** Erzeugen Sie Dokumentation, die Ihrem KI-Assistenten dabei hilft, Ihr Projekt und Ihre Vorlieben zu verstehen.

Es gibt eine Lücke, aber nur noch eine Zeit lang. Sie müssen lernen, diese effektiv zu überbrücken, um das massive Wachstum für sich nutzen zu können (wie Dr. Karpathy es formuliert). Was das genauer in der Praxis bedeutet, werden wir in den Teilen 2 und 3 beschreiben.

Zusammenfassung

Das Gute: Trotz all dieser Beschränkungen können KI-Coding-Assistenten Ihren Entwicklungsprozess beschleunigen. Eine sorgfältig überwachte KI kann Ihnen dabei helfen, FAAFO-Vorteile zu erreichen – schneller arbeiten, ambitioniertere Projekte angehen, mehr autonom erreichen, mehr Spaß haben und mehr Optionen schaffen.

Die Lücke wird kleiner. Jeder Fortschritt beim Speicher der KI, beim Verwalten des Kontexts und beim Befolgen von Anweisungen bringt uns näher an das KI-Ideal, bei dem wir ihr vertrauen können, unüberwacht große Aufgaben in einem langen Zeitraum zu erledigen. Dr. Thomas Kwa et al. gehen in ihrem Artikel »Measuring AI Ability to Complete Long Tasks« davon aus, dass der Tag kommt, an dem KIs zuverlässig und unüberwacht Monate mit Aufgaben aus der Softwareentwicklung verbringen können. Die Techniken, die wir in diesem Buch vorstellen, helfen Ihnen nicht nur dabei, effektiv mit heutigen KI-Tools zu arbeiten, sondern sie positionieren Sie auch so, dass Sie von zukünftigen Verbesserungen profitieren können.

In Teil 2 werden wir uns genauere Strategien für die Arbeit innerhalb der aktuellen Grenzen anschauen, wozu auch Techniken der Supervision und der Qualitätskontrolle gehören. So viel sei gesagt: Wenden Sie sich Ihrer KI mit einem klaren Verständnis dafür zu, welches Potenzial, aber auch welche Grenzen sie besitzt. Das hilft Ihnen, die Vorteile zu maximieren und gleichzeitig die Fallstricke zu vermeiden, die eine Souschefin mit sich bringt, die sich manchmal nicht mehr erinnern kann, wo der Mülleimer ist, und dann improvisiert.

Inhaltsverzeichnis

Vibe Coding

- Das wegweisende Handbuch für die Softwareentwicklung mit GenAI
- Langlebige Kernprinzipien und erprobte Strategien
- Von den Branchenexperten Gene Kim und Steve Yegge

Vibe Coding verändert die professionelle Softwareentwicklung radikal: Entwickler*innen müssen sich nicht mehr mit Syntax und Boilerplate-Code abmühen – sie beschreiben, was sie erreichen wollen, und sehen dabei zu, wie die KI ihre Vorstellungen umsetzt. Vibe Coding steigert dabei die Produktivität, Kreativität und Freude enorm – und ermöglicht es, Projekte allein zu meistern, für die früher ganze Teams nötig waren.

Gene Kim und Steve Yegge zeigen in diesem Buch konkret, wie KI-Assistenten die Rolle von Entwickler*innen grundlegend verändern: Sie werden zu Supervisoren, während die KI als kompetenter Teampartner die Details erledigt. Die Autoren liefern langfristig gültige Leitlinien und praxisnahe Vorgehensweisen, die Entwickler*innen und Teams dabei unterstützen, GenAI effektiv einzusetzen, ambitionierte Projekte zu meistern und gleichzeitig technische Exzellenz sicherzustellen.

Ob erfahrener Programmierer, Teamleiterin oder Neuling: Dieses Buch ist ein spannender Leitfaden für alle, die schon jetzt die Zukunft der Softwareentwicklung erproben möchten.

»Ich bin überwältigt von der Tiefe und Breite der Erkenntnisse in diesem Buch. Machen Sie sich darauf gefasst, dass Sie diese Techniken umgehend selbst ausprobieren möchten.«

Jason Cox, The Walt Disney Company

Gene Kim ist preisgekrönter CTO, Researcher und Autor. Seit 1999 beschäftigt er sich mit hochleistungsfähigen Technologieunternehmen. Er war Gründer und CTO von Tripwire und ist Autor der Bestseller *Das DevOps-Handbuch*, *Projekt Phoenix* und *Accelerate*.

Steve Yegge ist Programmierer und Blogger mit 30 Jahren Branchenerfahrung, u. a. bei Google und Amazon. Er hat über eine Million Zeilen Code geschrieben und große Teams und Systeme verantwortet. Derzeit arbeitet er bei Sourcegraph an KI-Assistenten.

Gedruckt in Deutschland
Mineralölfreie Druckfarben
Zertifiziertes Papier

€ 39,90 (D)

ISBN 978-3-98889-067-2