Vorwort: Axiome infrage stellen

Axiom

> Eine Aussage oder Behauptung, die als etabliert, akzeptiert und allgemein zutreffend gilt.

Axiome sind die Grundlage für mathematische Theorien. Axiome sind Annahmen über Dinge, die unzweifelhaft wahr sind. Softwarearchitekten erstellen ihre Theorien ebenfalls anhand von Axiomen. Dabei ist die Welt der Software allerdings *weicher* als die Mathematik: Grundsätze ändern sich mit atemberaubender Geschwindigkeit, inklusive der Axiome, auf denen unsere Theorien basieren.

Das Ökosystem der Sofwareentwicklung befindet sich in einem beständigen dynamischen Gleichgewicht: Betrachtet man es zu einem bestimmten Zeitpunkt, befindet es sich in einem ausbalancierten Zustand; auf lange Sicht zeigt es jedoch ein *dynamisches* Verhalten. Ein gutes und aktuelles Beispiel für dieses Verhalten ist der Aufstieg der Containerisierung und die damit einhergehenden Veränderungen: Werkzeuge wie Kubernetes (*https://kubernetes.io*) gab es vor einem Jahrzehnt noch nicht, dennoch werden heute ganze Softwarekonferenze dazu abgehalten. Das Softwareökosystem verändert sich chaotisch: Eine kleine Veränderung bewirkt weitere kleinere Änderungen. Wiederholt man das einige Hundert Mal, entsteht ein neues Ökosystem.

Architekten haben die wichtige Verantwortung, die Annahmen und Axiome vergangener »Zeitalter« infrage zu stellen. Viele Bücher über die Softwarearchitektur wurden in einer Zeit geschrieben, die der heutigen Welt kaum noch ähnelt. Tatsächlich sind die Autoren der Meinung, dass grundsätzliche Axiome regelmäßig infrage gestellt werden müssen, um auf verbesserte Entwicklungspraktiken, betriebliche Ökosysteme und Softwareentwicklungsprosse – alles, was dieses unordentliche dynamische Gleichgewicht ausmacht, in dem Architekten und Entwickler arbeiten – einzugehen.

Betrachtet man die Softwarearchitektur im Laufe der Zeit, kann man eine Evolution der Eigenschaften feststellen. Innovationen wie das Extreme Programming (*http://www.extremeprogramming.org*), gefolgt von Continuous Delivery, der DevOps-Revolution, Microservices, Containerisierung und den aktuellen Cloud-basierten

Ressourcen, haben zu neuen Möglichkeiten, aber auch zu neuen Schwierigkeiten geführt. Durch die veränderten Möglichkeiten änderte sich auch die Sichtweise der Architekten auf die Branche. Für viele Jahre wurde Softwarearchitektur augenzwinkernd als »das Zeug, was später nur schwer zu ändern ist« bezeichnet. Später traten Microservices auf den Plan, bei denen *Veränderung* eine der wichtigsten Designentscheidungen ist.

Jedes neue Zeitalter erfordert neue Vorgehensweisen, Werkzeuge, Messgrößen, Muster und eine Vielzahl weiterer Änderungen. Dieses Buch betrachtet die Softwarearchitektur in einem modernen Licht und geht dabei auf all die Innovationen des vergangenen Jahrzehnts sowie einige neue Kennzahlen und Messgrößen ein, die zu den heutigen Strukturen und Perspektiven passen.

Entwickler wünschen sich schon lange, dass sich die Sofwareentwicklung von einem *künstlerischen Ansatz*, bei dem geschickte Künstler einmalige Werke schaffen, zu einer *Ingenieursdisziplin* wandelt, die von Wiederholbarkeit, Strenge und effektiver Analyse bestimmt wird. Gegenüber der Softwareentwicklung haben andere Ingenieursdisziplinen immer einen um mehrere Größenordnungen weiten Vorsprung (wobei man nicht vergessen sollte, dass die Softwareentwicklung gegenüber anderen Ingenieursdisziplinen noch sehr jung ist). Dennoch haben die Architekten sehr große Verbesserungen erreicht, über die wir hier sprechen werden. Insbesondere haben die agilen Entwicklungspraktiken einen großen Fortschritt ermöglicht, wenn es um die von Architekten erstellten Systemtypen geht.

Außerdem gehen wir auf den besonders wichtigen Punkt der *Kompromissanalyse* (»Trade-Off analysis«) ein. Als Softwareentwickler legt man sich leicht auf eine bestimmte Technologie oder einen Ansatz fest. Architekten müssen die Vor- und Nachteile jeder Wahl dagegen sehr nüchtern gegeneinander abwägen. In der Realität hat man fast nie die Wahl zwischen schwarz und weiß. Alles ist ein Kompromiss. Aufgrund dieser pragmatischen Sichtweise versuchen wir, Werturteile über Technologien auszuschalten und uns stattdessen auf die Analyse möglicher Kompromisse zu konzentrieren, um unseren Lesern einen analytischen Blick auf die möglichen Technologieentscheidungen zu bieten.

Dieses Buch macht Sie nicht über Nacht zu einem Sofwarearchitekten, denn dieses vielschichtige Spielfeld besitzt viele Facetten. Existierenden und angehenden Architekten wollen wir einen guten und modernen Überblick über die Softwarearchitektur und ihre vielen Aspekte von der Struktur bis hin zu den nötigen Soft Skills bieten. Auch wenn Sie in diesem Buch bekannte Muster finden, verwenden wir hier einen neuen Ansatz. Dieser basiert auf unseren eigenen Erfahrungen, Werkzeugen, Entwicklungspraktiken und weiteren Quellen. Wir betrachten die vielen existierenden Axiome der Softwarearchitektur und denken sie im Licht des aktuellen Ökosystems und der gegenwärtigen Design-Architekturen neu.

Hinweis des Übersetzers

In diesem Buch kommen oft Formulierungen wie »der Architekt« vor. Diese Schreibweise ist selbstverständlich unabhängig vom Geschlecht der jeweiligen Leser/innen, also grundsätzlich als »m/w/d« gemeint.

Konventionen in diesem Buch

Die folgenden typografischen Konventionen werden in diesem Buch genutzt:

Kursiv
: Für neue Begriffe, URLs, E-Mail-Adressen, Dateinamen und Dateierweiterungen.

`Nichtproportionalschrift`
: Für Programmlistings, aber auch für Codefragmente in Absätzen, wie zum Beispiel Variablen- oder Funktionsnamen, Datenbanken, Datentypen, Umgebungsvariablen, Anweisungen und Schlüsselwörter.

`Nichtproportionalschrift fett`
: Zeigt Befehle oder anderen Text an, der genauso vom Benutzer eingegeben werden muss.

`Nichtproportionalschrift kursiv`
: Zeigt Programmcode an, der durch Benutzereingaben oder durch kontextabhängige Werte ersetzt werden soll.

Dieses Zeichen steht für einen Tipp oder eine Empfehlung.

Codebeispiele verwenden

Ergänzungsmaterialien (Codebeispiele, Übungen usw.) stehen zum Download unter *https://fundamentalsofsoftwarearchitecture.com* bereit.

Dieses Buch soll Ihnen bei Ihrer täglichen Arbeit helfen. Falls Beispielcode zum Buch angeboten wird, dürfen Sie ihn im Allgemeinen in Ihren Programmen und für Dokumentationen verwenden. Sie müssen uns nicht um Erlaubnis bitten, es sei denn, Sie kopieren einen erheblichen Teil des Codes. Wenn Sie zum Beispiel ein Programm schreiben, das einige Codeblöcke aus diesem Buch verwendet, benötigen Sie keine Erlaubnis. Wenn Sie Beispiele aus O'Reilly-Büchern verkaufen oder vertreiben, benötigen Sie eine Erlaubnis. Wenn Sie eine Frage beantworten und dabei dieses Buch oder Beispielcode aus diesem Buch zitieren, brauchen Sie wiederum keine Erlaubnis. Möchten Sie allerdings erhebliche Teile des Beispielcodes aus

diesem Buch in die Dokumentation Ihres Produkts einfließen lassen, ist eine Erlaubnis einzuholen.

Wir schätzen eine Quellenangabe, verlangen sie aber nicht. Eine Quellenangabe umfasst in der Regel Titel, Autor, Verlag und ISBN, zum Beispiel: »Handbuch moderner Softwarearchitektur. Architekturstile, Patterns und Best Practices« von Mark Richards und Neal Ford (O'Reilly). 978-3-96009-149-3.«

Wenn Sie der Meinung sind, dass Sie die Codebeispiele in einer Weise verwenden, die über die oben erteilte Erlaubnis hinausgeht, kontaktieren Sie uns bitte unter kommentar@oreilly.de.

Danksagungen

Mark und Neal möchten sich bei allen Menschen bedanken, die unsere Kurse, Workshops, Konferenz-Sessions und Usergroup-Treffen besucht haben, sowie bei allen Leuten, die sich verschiedene Versionen dieses Materials angehört und Rückmeldungen von unschätzbarem Wert gegeben haben. Außerdem möchten wir uns beim gesamten O'Reilly-Team bedanken, das das Schreiben dieses Buchs so schmerzfrei wie nur möglich gestaltet hat. Wir möchten uns außerdem bei Jay Zimmerman, dem Direktor von No Stuff Just Fluff dafür bedanken, dass er eine Konferenzreihe gschaffen hat, die es guten technischen Inhalten ermöglicht, zu wachsen und sich zu verbreiten, sowie bei den anderen Sprechern, deren Feedback und tränendurchweichte Schultern wir sehr zu schätzen wissen. Außerdem bedanken wir uns bei verschiedenen Gruppen, die uns dabei halfen, unsere geistige Gesundheit zu bewahren und neue Ideen zu finden. Diese Zufallsoasen tragen Namen wie Pasty Geeks und das Hacker B&B.

Danksagungen von Mark Richards

Zusätzlich zu den oben genannten Danksagungen möchte ich mich bei meiner geliebten Frau Rebecca bedanken. Du hast zu Hause alles andere übernommen und dabei die Gelegenheit geopfert, Dein eigenes Buch zu schreiben, damit ich mehr Beratungstermine wahrnehmen, auf mehr Konferenzen und Trainings sprechen konnte, um so das Material für dieses Buch zu üben und zu verfeinern. Du bist die Beste.

Danksagungen von Neal Ford

Neal möchte sich bei seiner erweiterten Familie bedanken, ThoughtWorks als Kollektiv und Rebecca Parsonsn sowie Martin Fowler als einzelne Teile davon. ThoughtWorks ist eine außergewöhnliche Gruppe, die es schafft, Wert für ihre Kunden zu schaffen, ohne dabei aus den Augen zu verlieren, warum Dinge funktionieren und wie sie verbessert werden können. ThoughtWorks hat dieses Buch auf vielerlei Weise unterstützt und bildet auch weiterhin ThoughtWorker aus, die täg-

lich aufs Neue herausfordern und inspirieren. Neal möchte sich außerdem bei unserem Nachbarschafts-Cocktail-Club für regelmäßige Auszeiten von der Routine bedanken. Abschließend dankt Neal seiner Frau Candy, deren Toleranz für Dinge wie das Schreiben von Büchern und das Sprechen auf Konferenzen scheinbar grenzenlos ist. Seit Jahrzehnten hält sie mich auf dem Boden und gesund genug, um weiter zu funktionieren. Ich hoffe, dass sie auch für weitere Jahrzehnte die Liebe meines Lebens bleiben wird.

Danksagungen von Jørgen W. Lang

Ich möchte mich bei den Autoren bedanken, die mir in kürzester Zeit alle Fragen freundlich und professionell beantwortet und für Klarheit gesorgt haben. Mein größtes Dankschön gilt jedoch meiner Familie, Armelle und Mathis, ohne die ich jetzt vermutlich etwas ganz anderes machen würde.