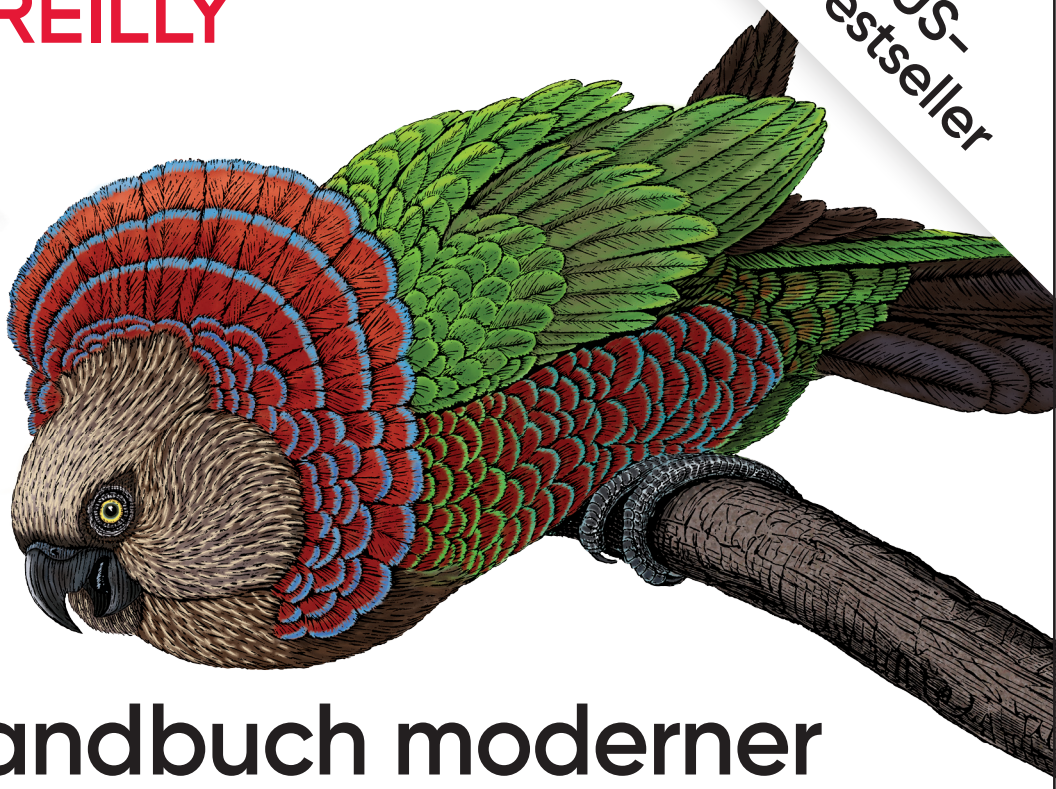
O'REILLY®

US-Bestseller



Handbuch moderner Softwarearchitektur

Architekturstile, Patterns und Best Practices

Mark Richards & Neal Ford
Übersetzung von Jørgen W. Lang

Lob für *»Handbuch moderner Softwarearchitektur«*

Neal und Mark sind nicht nur herausragende Softwarearchitekten, sondern auch außergewöhnliche Lehrer. Mit »Handbuch moderner Softwarearchitektur« haben sie es geschafft, das weite Thema der Architektur zu einem umfassenden Werk zu kondensieren, das eine jahrzehntelange Erfahrung widerspiegelt. Unabhängig davon, ob Sie neu in der Rolle sind oder bereits viele Jahre als Architekt arbeiten – dieses Buch wird Ihnen helfen, Ihren Job noch besser zu machen. Ich hätte nur gehofft, dieses Buch wäre früher in meiner Laufbahn geschrieben worden.

– Nathaniel Schutta, Architect as a Service, ntschutta.io

Mark und Neal haben sich aufgemacht, ein großes Ziel zu erreichen – die vielschichtigen Grundlagen zu erklären, die nötig sind, um überragende Ergebnisse in der Softwarearchitektur zu erreichen –, und sie haben ihre Aufgabe gemeistert. Das Feld der Softwarearchitektur entwickelt sich ständig weiter, und die Rolle erfordert eine Unmenge an Wissen und Fähigkeiten. Dieses Buch wird über Jahre vielen, die auf dem Weg sind, die Softwarearchitektur zu meistern, als Richtschnur dienen.

– Rebecca J. Parsons, CTO, ThoughtWorks

Mark und Neal ist es gelungen, Ratschläge aus der echten Welt zusammenzutragen, die Technologen zu Spitzenleistungen in der Architektur anspornen. Das gelingt ihnen, indem sie die wichtigsten Vor- und Nachteile von Architekturen identifizieren, die für den Erfolg nötig sind.

– Cassie Shum, Technical Director, ThoughtWorks

Handbuch moderner Softwarearchitektur

Architekturstile, Patterns und Best Practices

Mark Richards und Neal Ford

Deutsche Übersetzung von
Jørgen W. Lang

O'REILLY®

Mark Richards und Neal Ford

Lektorat: Ariane Hesse
Übersetzung: Jørgen W. Lang
Fachliche Unterstützung: Benjamin Schmid, Oliver Starke
Korrektorat: Claudia Lötschert, *www.richtiger-text.de*
Satz: III-satz, *www.drei-satz.de*
Herstellung: Stefanie Weidner
Umschlaggestaltung: Michael Oréal, *www.oreal.de*
Druck und Bindung: mediaprint solutions GmbH, 33100 Paderborn

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.

ISBN:
Print 978-3-96009-149-3
PDF 978-3-96010-429-2
ePub 978-3-96010-430-8
mobi 978-3-96010-431-5

1. Auflage 2021

Wieblinger Weg 17
69123 Heidelberg

Authorized German translation of the English edition of *Fundamentals of Software Architecture*
ISBN 9781492043454 © 2020 Mark Richards, Neal Ford. This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

Dieses Buch erscheint in Kooperation mit O'Reilly Media, Inc. unter dem Imprint »O'REILLY«.
O'REILLY ist ein Markenzeichen und eine eingetragene Marke von O'Reilly Media, Inc. und wird mit Einwilligung des Eigentümers verwendet.

Hinweis:
Dieses Buch wurde auf PEFC-zertifiziertem Papier aus nachhaltiger Waldwirtschaft gedruckt. Der Umwelt zuliebe verzichten wir zusätzlich auf die Einschweißfolie.

PEFC zertifiziert
Das Papier für dieses Buch stammt aus nachhaltig bewirtschafteten Wäldern und kontrollierten Quellen.
PEFC™
PEFC/04-31-0810 www.pefc.de

Schreiben Sie uns:
Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: kommentar@oreilly.de.

5 4 3 2 1 0

Inhalt

Teil II Architekturstile

Teil III Techniken und Soft Skills

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.

KAPITEL 1
Einleitung

Die Berufsbezeichnung »Softwarearchitekt« steht auf vielen Listen der besten Jobs auf der ganzen Welt ganz oben. Für die anderen Berufe auf der Liste (zum Beispiel Krankenpfleger oder Finanzmanager) gibt es einen klaren Karrierepfad. Warum gibt es keinen Karrierepfad für Softwarearchitekten?

Zunächst einmal hat die Branche selbst keine gute Definition von Softwarearchitektur. Wenn wir Grundlagenkurse unterrichten, fragen die Studenten oft nach einer klaren Definition für das, was ein Softwarearchitekt tut. Bisher haben wir ihnen diese Antwort hartnäckig verweigert. Und damit sind wir nicht alleine. In seinem berühmten Whitepaper »Who Needs an Architect?« (*https://oreil.ly/-Dbzs*) weigerte sich Martin Fowler bekanntlich, auch nur zu versuchen, den Begriff »Softwarearchitekt« zu definieren. Stattdessen wich er auf das folgende berühmte Zitat aus:

> Bei der Softwarearchitektur geht es um wichtige Dinge ... welche auch immer das sind.
>
> *– Ralph Johnson*

Erst unter Druck haben wir die in Abbildung 1-1 gezeigte Mindmap erstellt. Sie ist hoffnungslos unvollständig, zeigt aber, wie groß das Feld der Softwarearchitektur tatsächlich ist. In Kürze werden wir Ihnen unsere eigene Definition der Softwarearchitektur vorstellen.

Außerdem zeigt die Mindmap, dass die Rolle des Softwarearchitekten sehr viele Verantwortungsbereiche umfasst, die immer weiter wachsen. Noch vor zehn Jahren haben sich Softwarearchitekten nur mit den rein technischen Aspekten der Architektur befasst, wie Modularität, Komponenten und Patterns. Durch neue Architekturstile, die zusätzliche Möglichkeiten (zum Beispiel Microservices) nutzen, hat sich auch die Rolle des Softwarearchitekten erweitert. Die vielen Überschneidungen zwischen der Architektur und dem Rest des Unternehmens betrachten wir in »Überschneidungen von Architektur und ...« auf Seite 13.

Durch die fortschreitende Evolution der Softwareentwicklung ist auch die Softwarearchitektur ständig in Bewegung. Jede heute gültige Definition wird schon in ein paar Jahren hoffnungslos veraltet sein. Die Wikipedia-Definition der Softwarearchitektur (*https://de.wikipedia.org/wiki/Softwarearchitektur*) gibt hier einen guten Überblick.

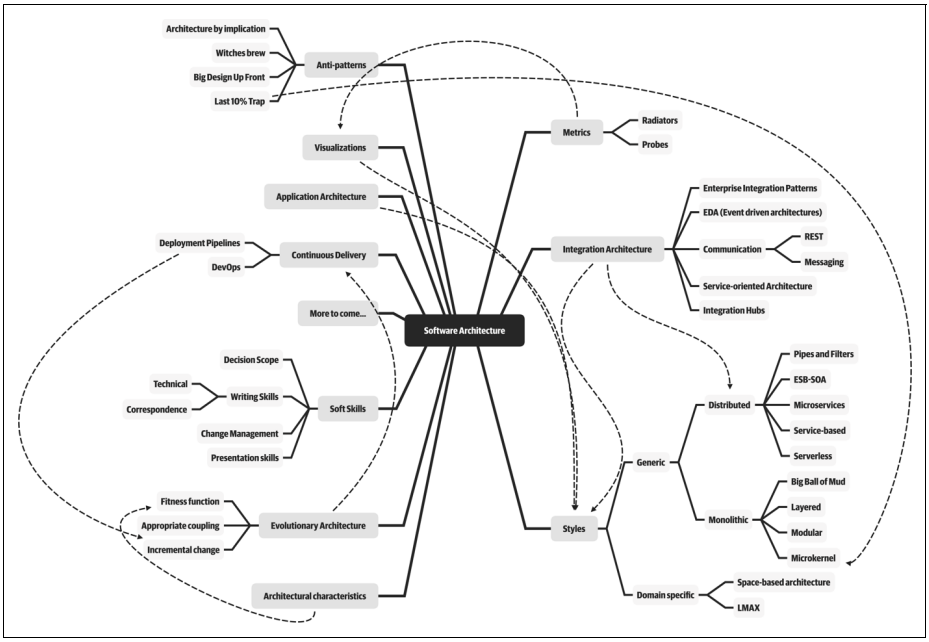


Abbildung 1-1: Die Verantwortlichkeit eines Softwarearchitekten umfasst technische Fähigkeiten, Soft Skills, Unternehmensbewusstsein und eine Reihe weiterer Aspekte.

Viele Aussagen sind aber auch jetzt schon veraltet – zum Beispiel: »Es ist Aufgabe der Softwarearchitektur, grundlegende strukturelle Entscheidungen zu treffen, deren spätere Änderung sehr kostspielig wären.« Mittlerweile gibt es moderne architektonische Stile, zum Beispiel Microservices, die die Idee der Inkrementierung bereits enthalten. Strukturelle Änderungen in Microservices sind nicht länger teuer. Natürlich hat eine solche Möglichkeit auch Nachteile, zum Beispiel bei der Kopplung. Viele Bücher über Softwarearchitektur behandeln das als statisches Problem. Ist es einmal gelöst, kann man es sicher ignorieren. In diesem Buch vertreten wir dagegen die Meinung, dass Softwarearchitektur grundsätzlich etwas Dynamisches ist – inklusive ihrer Definition.

Darüber hinaus hat ein Großteil der Materialien über Softwarearchitektur nur noch historische Bedeutung. Leser der Wikipediaseite werden die verwirrendende Ansammlung von Akronymen und Querverweisen zu einem ganzen Wissensuniversum bemerkt haben. Allerdings stehen viele dieser Akronyme für veraltete oder fehlgeschlagene Versuche. Selbst Lösungen, die vor einigen Jahren noch absolut gültig waren, können heute nicht mehr funktionieren, weil sich der Kontext verändert hat. Die Geschichte der Softwarearchitektur ist voll von gescheiterten Versuchen von Softwarearchitekten, die abgebrochen wurden, nachdem die schlechten Nebenwirkungen sichtbar wurden. Viele dieser Lehren werden wir in diesem Buch behandeln.

Warum haben wir ausgerechnet jetzt ein Buch über Grundlagen der Softwarearchitektur geschrieben? Die Softwarearchitektur ist schließlich nicht der einzige Bereich der Softwareentwicklung, der andauernden Änderungen unterworfen ist. Ständig gibt es neue Technologien, Techniken, Fähigkeiten ... Es ist tatsächlich leichter, die Dinge aufzulisten, die sich in den letzten zehn Jahren *nicht* verändert haben. Innerhalb dieses hochdynamischen Ökosystems müssen Softwarearchitekten in der Lage sein, Entscheidungen zu treffen. Da alles – inklusive der Grundlagen, auf deren Basis wir Entscheidungen treffen – ständig in Bewegung ist, sollten Architekten die grundlegenden Axiome früherer Publikationen immer wieder überprüfen und infrage stellen. DevOps spielten in früheren Büchern über Softwarearchitektur keine Rolle, weil es sie beim Schreiben dieser Bücher einfach noch nicht gab.

Beim Studium der Softwarearchitektur sollten die Leser sich darüber klar sein, dass sie – wie die Kunst – nur im richtigen Kontext verstanden werden kann. Viele der Entscheidungen von Softwarearchitekten wurden auf Basis der Realitäten getroffen, in denen sie sich gerade befanden. Eines der Hauptziele der Architektur des ausgehenden 20. Jahrhunderts war beispielsweise eine möglichst kosteneffiziente Nutzung verteilter Ressourcen. Damals war die gesamte Infrastruktur sehr teuer und kommerziell: Betriebssysteme, Application Server, Datenbankserver und so weiter. Stellen Sie sich vor, Sie betreten im Jahr 2002 ein Rechenzentrum und sagen dem Betriebsleiter: »Hey, ich habe eine tolle Idee für einen revolutionären Architekturstil. Dabei läuft jeder Dienst inklusive seiner eigenen Datenbank auf einem eigenen isolierten Rechner (was wir heute als Microservices kennen). Das würde bedeuten, wir bräuchten 50 Windows-Lizenzen, weitere 30 Lizenzen für Application Server und mindestens 50 Lizenzen für Datenbankserver.« Der Versuch, eine Architektur wie Microservices zu schaffen, wäre 2002 unermesslich teuer geworden. Durch das Aufkommen von Open-Source-Lösungen zusammen mit neuen Entwicklungspraktiken wie der DevOps-Revolution sind wir inzwischen jedoch in der Lage, eine Architektur wie die beschriebene zu erstellen. Die Leser sollten daher nicht vergessen, dass alle Architekturen ein Produkt ihres Kontexts sind.

Softwarearchitektur definieren

Die gesamte Branche hat sich bisher damit schwergetan, eine präzise Definition für den Begriff »Softwarearchitektur« zu finden. Einige Architekten verstehen darunter den *Bauplan* eines Systems, während andere sie als *Roadmap* für die Entwicklung eines Systems definieren. Das Problem mit diesen verbreiteten Definitionen besteht darin, dass man nicht weiß, was der Bauplan oder die Roadmap tatsächlich enthält. Was *genau* wird beispielsweise analysiert, wenn ein Architekt eine Architektur *analysiert*?

Abbildung 1-2 zeigt eine Möglichkeit, sich die Softwarearchitektur vorzustellen. In dieser Definition besteht sie aus der *Struktur* des Systems (dargestellt durch die dicken schwarzen Linien, die die Architektur stützen), kombiniert mit den *archi-*

tektonischen Eigenschaften (bzw. *Fähigkeiten, engl. »-ilities«*), die das System unterstützen muss, den *architektonischen Entscheidungen* und schließlich den *Designprinzipien*.

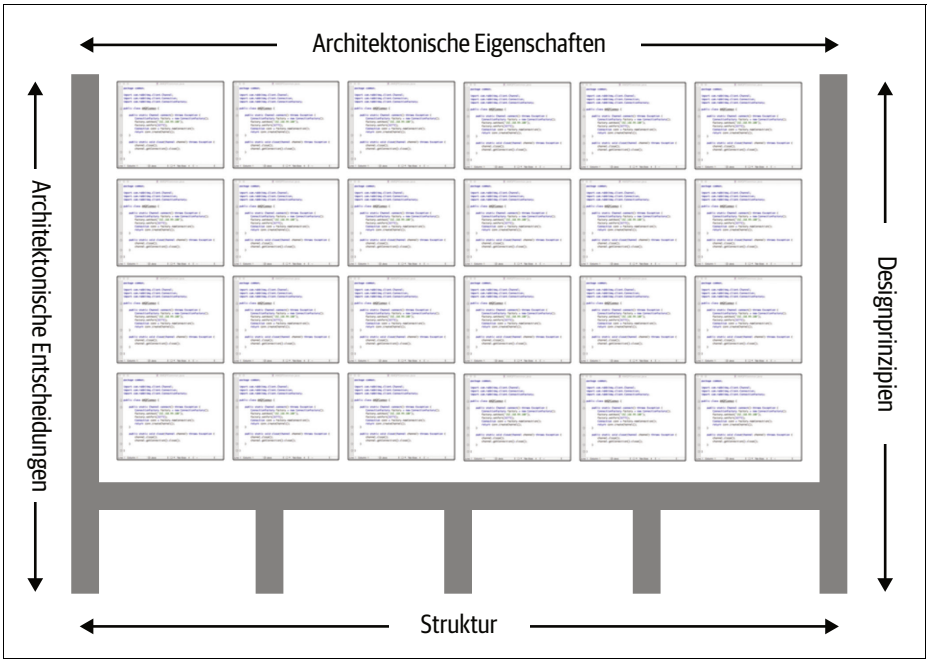


Abbildung 1-2: Architektur besteht aus Struktur, den architektonischen Eigenschaften (Fähigkeiten), architektonischen Entscheidungen und Designprinzipien

Die in Abbildung 1-3 gezeigte *Struktur* des Systems bezieht sich auf den Architekturstil (oder die Stile), in dem das System implementiert ist (zum Beispiel als Microservices, schichtbasiertes Modell oder Microkernel). Die Struktur allein reicht aber nicht, um eine Architektur zu beschreiben. Angenommen, ein Architekt soll eine Architektur beschreiben und seine Antwort lautet: »Es ist eine Microservices-Architektur.« In diesem Fall spricht der Architekt nur von der *Struktur* des Systems, aber nicht von seiner *Architektur*. Das Wissen um die architektonischen Eigenschaften, Entscheidungen und Designprinzipien ist genauso wichtig, um die Architektur eines Systems vollständig zu verstehen.

Die architektonischen Eigenschaften bilden eine weitere Dimension in der Definition einer Softwarearchitektur (siehe Abbildung 1-4). Die architektonischen Eigenschaften definieren die Erfolgskriterien eines Systems. Sie stehen üblicherweise senkrecht zur Systemfunktionalität. Beachten Sie, dass für sämtliche aufgelisteten Eigenschaften kein Wissen über die Systemfunktionalität notwendig ist. Dennoch werden sie gebraucht, damit das System korrekt funktioniert. Die architektonischen Eigenschaften sind so wichtig, dass wir ihrer Definition und ihrem Verständnis mehrere Kapitel dieses Buchs gewidmet haben.

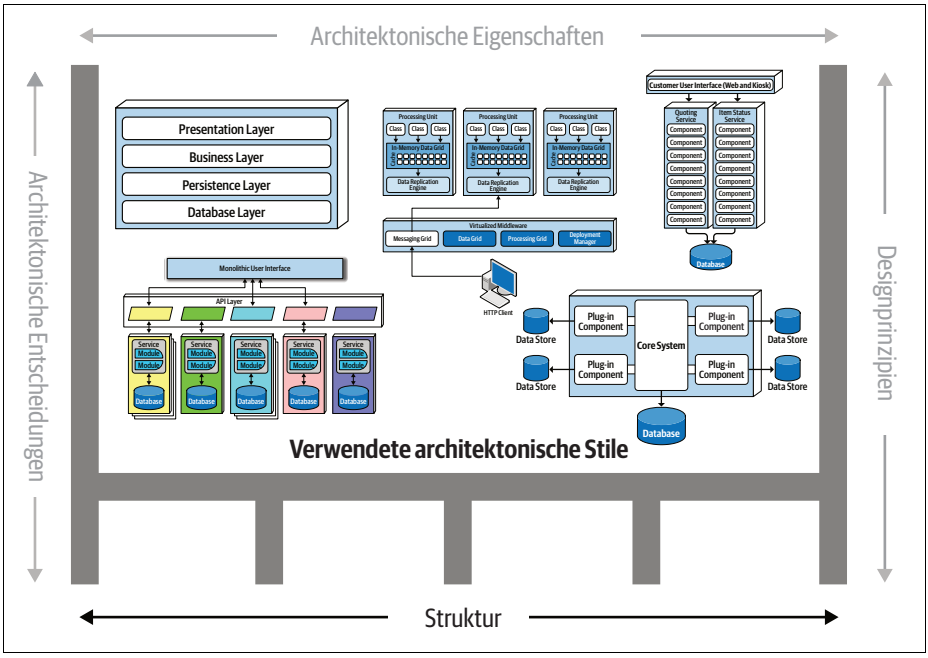


Abbildung 1-3: Die Struktur zeigt, welcher Typ des architektonischen Stils im System verwendet wird.

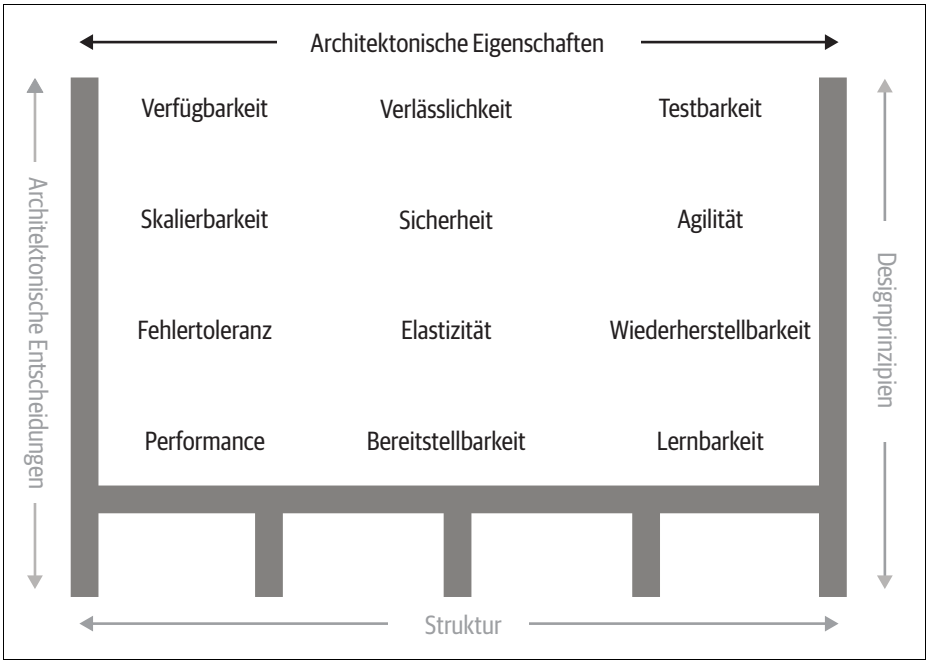


Abbildung 1-4: Architektonische Eigenschaften beziehen sich auf die Fähigkeiten, die das System unterstützen muss.

Der nächste Faktor, der eine Softwarearchitektur definiert, sind die *Architekturentscheidungen*. Architektonische Entscheidungen definieren die Regeln, nach denen ein System konstruiert wird. So könnte ein Architekt beispielsweise die Entscheidung treffen, dass nur die Business- und Service-Schichten innerhalb einer schichtbasierten Architektur auf die Datenbank zugreifen können (siehe Abbildung 1-5). Damit soll zum Beispiel verhindert werden, dass die Präsentationsschicht direkte Datenbankaufrufe durchführt. Architektonische Entscheidungen definieren die Beschränkungen eines Systems, dienen als Rahmen für die Entwicklungsteams und zeigen ihnen an, welche Dinge erlaubt sind und welche nicht.

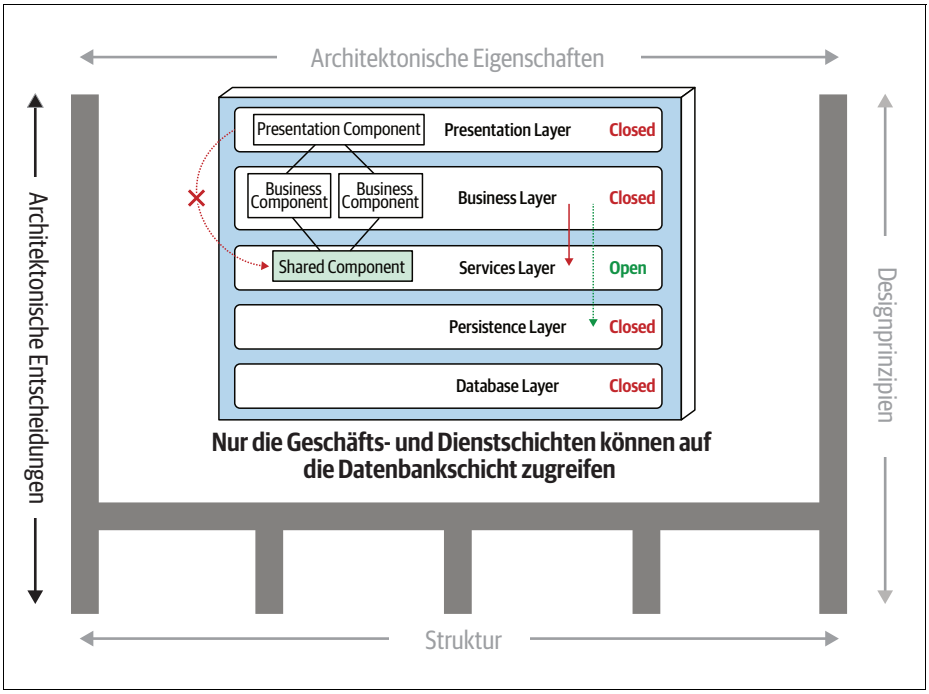


Abbildung 1-5: Architektonische Entscheidungen sind Regeln für die Konstruktion eines Systems.

Kann eine architektonische Entscheidung aufgrund bestimmter Bedingungen oder Beschränkungen in einem Systemsteil nicht umgesetzt werden, kann diese Entscheidung (oder Regel) durch die sogenannte *Varianz* gebrochen werden. In den meisten Unternehmen gibt es Varianzmodelle, die von einem Architektur-Prüfungsgremium (Architecture Review Board, ARB) oder einem Chefarchitekten eingesetzt werden können. Eine Ausnahme für eine bestimmte Architekturentscheidung wird vom ARB (oder dem Chefarchitekten, falls es kein ARB gibt) analysiert und basierend auf den Begründungen und möglichen Vor- und Nachteilen entweder genehmigt oder abgelehnt.

Der letzte Faktor bei der Definition einer Architektur sind die *Designprinzipien*. Der Unterschied zwischen einem Designprinzip und einer Architekturentscheidung besteht darin, dass ein Designprinzip eher eine *Richtlinie* als eine strenge und unverrückbare *Regel* darstellt. Das in Abbildung 1-6 gezeigte Designprinzip besagt beispielsweise, dass die Entwicklungsteams *möglichst* asynchrones Messaging für die Kommunikation zwischen den Diensten verwenden sollen, um die Performance zu steigern. Eine architektonische Entscheidung (Regel) könnte niemals alle Bedingungen und Optionen für die Kommunikation zwischen den Diensten abdecken. Hier kann ein Designprinzip helfen, um Richtlinien für die bevorzugte Methode (hier das asynchrone Messaging) aufzustellen. Auf diese Weise können Entwickler ggf. ein passenderes Kommunikationsprotokoll (zum Beispiel REST oder gRPC) wählen.

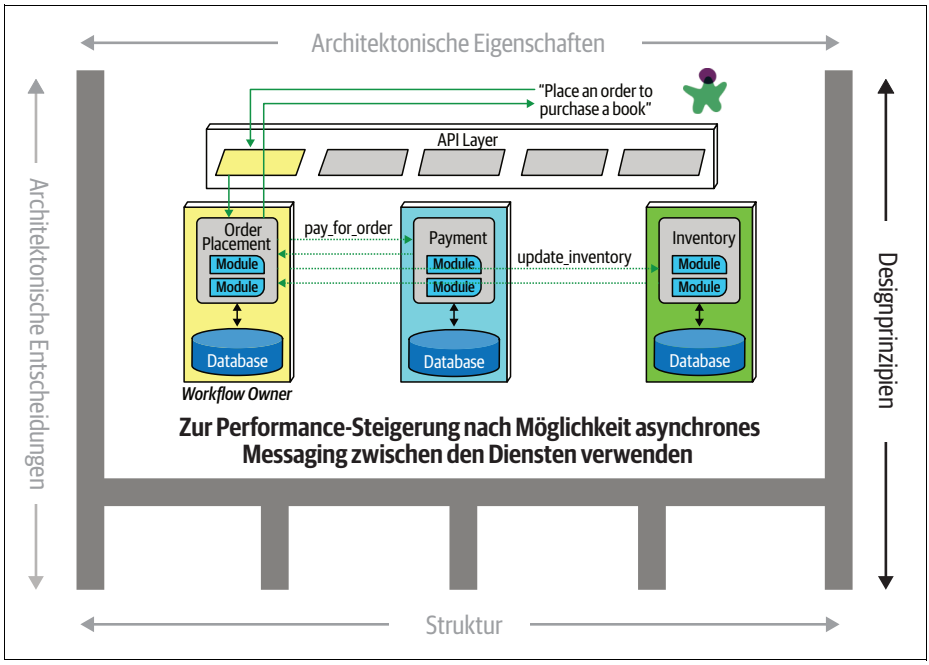


Abbildung 1-6: Designprinzipien sind Richtlinien für die Konstruktion von Systemen.

Erwartungen an Architekten

Die Definition der Rolle eines Softwarearchitekten gestaltet sich genauso schwierig wie die Definition der Softwarearchitektur selbst. Dabei kann es sich um einen erfahrenen Programmierer handeln oder um eine Person, die die strategisch-technische Richtung eines Unternehmens definiert. Anstatt mit der Definition der Rolle Zeit zu verschwenden, empfehlen wir Ihnen, sich auf die *Erwartungen* an einen Architekten zu konzentrieren.

Unabhängig von einer bestimmten Rolle, einem Titel oder einer Jobbeschreibung gibt es acht grundsätzliche Erwartungen an einen Softwarearchitekten:

- Architekturentscheidungen treffen
- Beständige Analyse der Architektur
- Bei den neuesten Trends auf dem Laufenden bleiben
- Sicherstellen, dass Entscheidungen eingehalten werden
- Vielfältige Kenntnisse und Erfahrungen besitzen
- Erfahrung im geschäftlichen Umfeld haben
- Fähigkeiten im zwischenmenschlichen Umgang besitzen
- Politik verstehen und sich in dieser Sqhäre bewegen können

Der Schlüssel zu Effektivität und Erfolg in der Rolle eines Softwarearchitekten hängt davon ab, alle diese Erwartungen zu verstehen und zu erfüllen.

Architekturentscheidungen treffen

Von einem Architekten wird erwartet, die architektonischen Entscheidungen und Designprinzipien festzulegen, die als Leitfaden (engl. »guide«) für die technologischen Entscheidungen innerhalb des Teams, der Abteilung oder im gesamten Unternehmen dienen.

Das Vorgeben einer Richtung spielt für die erste Erwartung die wichtigste Rolle. Ein Architekt sollte Technologieentscheidungen *anleiten*, anstatt sie zu *bestimmen*. Ein Architekt könnte beispielsweise entscheiden, React.js für die Frontend-Entwicklung einzusetzen. Statt einer Architekturentscheidung oder eines Designprinzips, das dem Entwicklungsteam für seine Auswahl eine Richtung vorgibt, trifft der Architekt hier eine technologische Entscheidung. In diesem Fall ist es besser, wenn der Architekt dem Entwicklungsteam vorgibt, ein *auf reaktiven Prinzipien basierendes Framework für die Entwicklung von Web-Frontends* zu verwenden. Dadurch gibt der Architekt dem Entwicklungsteam die Möglichkeit, selbst zu entscheiden, ob nun Angular, Elm, React, Vue oder ein anderes »reaktives« Web-Framework verwendet werden soll.

Die Vorgabe einer Richtung für technologische Wahlmöglichkeiten anhand architektonischer Entscheidungen und Designprinzipien ist schwierig. Damit das effektiv funktioniert, muss man sich fragen, ob die Architekturentscheidung dabei hilft, den Teams *eine Richtung vorzugeben*, die sie dabei unterstützt, die richtige technische Wahl zu treffen, oder ob die architektonische Entscheidung diese Wahl *trifft*. Manchmal ist es trotztdem nötig, dass ein Architekt klare technologische Vorgaben macht, um bestimmte architektonische Eigenschaften wie Skalierbarkeit, Performance oder Verfügbarkeit sicherzustellen. In diesem Fall würde man trotzdem von einer Architekturentscheidung sprechen, obwohl eine bestimmte Technologie vorgegeben wird. Es ist für Architekten nicht immer einfach, die richtige

Grenze zu finden, daher geht es in Kapitel 19 nicht ausschließlich um Architekturentscheidungen.

Kontinuierliche Analyse der Architektur

Von Architekten wird erwartet, dass sie die Architektur und die Architekturentscheidungen Umgebung laufend analysieren und Lösungen für ihre Verbesserung anbieten.

Diese Erwartung an Architekten bezieht sich auf die *Architekturvitalität*. Dabei wird auf Basis der geschäftlichen und technologischen Veränderungen überprüft, wie gültig eine vor drei oder mehr Jahren definierte Architektur *heute* noch ist. Unserer Erfahrung nach konzentrieren zuwenige Architekten ihre Energie auf die beständige Analyse bereits vorhandener Architekturen. Bei den meisten Architekturen führt das zu strukturellem Verfall. Dieser tritt auf, wenn Entwickler Änderungen an Code oder Design vornehmen, die sich auf die nötigen architektonischen Eigenschaften wie Performance, Verfügbarkeit und Skalierbarkeit auswirken.

Andere Aspekte dieser Erwartung, die Architekten oft vergessen, sind Testing und Release-Umgebungen. Agilität hat für Änderungen am Code offensichtliche Vorteile. Wenn Teams allerdings Wochen zum Testen und Monate für ein Release benötigen, dann können Architekten in der Gesamtarchitektur keine Agilität erreichen.

Architekten brauchen eine ganzheitliche Denkweise. Sie müssen Veränderungen in Technologie und Problembereichen analysieren, um die Stabilität der Architektur zu ermitteln. Diese Anforderungen sieht man in Stellenangeboten jedoch nur selten. Dennoch müssen Architekten diese Erwartung erfüllen, damit ihre Bewerbung relevant bleibt.

Bei aktuellen Trends auf dem Laufenden bleiben

Von Architekten wird erwartet, dass sie die aktuellsten technologischen und Branchentrends im Auge behalten.

Entwickler müssen bei den neuesten Technologien, die sie täglich benutzen, stets auf dem neuesten Stand sein, um relevant zu bleiben (und ihren Job zu behalten!) Für Architekten ist es noch wichtiger, die aktuellen technischen Fortschritte und Entwicklungen ihrer Branche im Auge zu behalten. Schließlich haben die Entscheidungen von Architekten meist langfristige Auswirkungen und sind nachträglich schwer zu ändern. Das Verständnis und Verfolgen von Schlüsseltrends hilft Architekten, sich auf die Zukunft vorzubereiten und die richtigen Entscheidungen zu treffen.

Dabei ist es nicht einfach, diese Trends immer im Auge zu behalten, besonders für Softwarearchitekten. Daher besprechen wir in Kapitel 24 verschiedene Techniken und Möglichkeiten, dies zu tun.

Sicherstellen, dass Entscheidungen eingehalten werden

Von Architekten wird erwartet, sicherzustellen, dass architektonische Entscheidungen und Designprinzipien eingehalten werden (Compliance).

Das Sicherstellen der Compliance bedeutet, dass Architekten beständig überprüfen müssen, ob die Entwicklungsteams den von ihnen getroffenen, dokumentierten und kommunizierten architektonischen Entscheidungen und Designprinzipien folgen. Angenommen, ein Architekt entscheidet, in einer schichtbasierten Architektur den Datenbankzugriff nur für die Business- und Serviceschicht zu erlauben (aber nicht für die Präsentationsschicht). Das heißt, die Präsentationsschicht muss alle Schichten der Architektur durchlaufen, um selbst die einfachsten Datenbankaufrufe durchzuführen. Ein UI-Entwickler könnte diese Entscheidung missbilligen und aus Performance-Gründen versuchen, trotzdem direkt auf die Datenbank (oder die Persistenzschicht) zuzugreifen. Der Architekt hat die Entscheidung aber aus einem bestimmten Grund getroffen: um Änderungen kontrollieren zu können. Durch die Isolierung der einzelnen Schichten können Änderungern an der Datenbank vorgenommen werden, ohne dabei die Präsentationsschicht zu beeinflussen. Wird nicht auf die Einhaltung dieser Architekturentscheidungen geachtet, können Verstöße wie dieser auftreten. Das führt dazu, dass die Architektur nicht die erforderlichen architektonischen Eigenschaften (Fähigkeiten) erfüllt, wodurch die Architektur oder das System nicht wie erwartet funktioniert.

In Kapitel 6 sprechen wir darüber, wie die Compliance mithilfe automatischer Fitnessfunktionen und anderer Werkzeuge überprüft werden kann.

Vielfältige Kenntnisse und Erfahrungen

Von Architekten wird erwartet, dass sie Kenntnisse und Erfahrungen mit vielfältigen und verschiedenen Technologien, Frameworks, Plattformen und Umgebungen besitzen.

Diese Erwartung bedeutet nicht, dass Architekten Experten für jedes Framework, jede Plattform oder Sprache sein müssen. Dennoch sollten sie mit einer Reihe verschiedener Technologien vertraut sein. Die meisten Umgebungen sind heutzutage heterogen. Daher sollten Architekten zumindest wissen, wie auf verschiedene Systeme und Dienste unabhängig von Sprache, Plattform und Technologie zugegriffen werden kann.

Einer der besten Wege, diese Erwartung zu meistern, besteht darin, dass Architekten ihre Komfortzone erweitern. Die Konzentration auf nur eine bestimmte Technologie bietet nur vermeintliche Sicherheit. Effektive Softwarearchitekten sollten offensiv nach Gelegenheiten suchen, Erfahrungen mit verschiedenen Sprachen, Plattformen und Technologien zu sammeln. Dabei bietet es sich an, sich auf technische Breite (»technical breadth«) anstelle von technischer Tiefe (»technical depth«) zu konzentrieren. Zu technischer Breite gehören Dinge, zu denen sie etwas wissen, ohne jedes Detail zu kennen. Für Architekten ist es beispielsweise viel wert-

voller, die Vor- und Nachteile von 10 verschiedenen Caching-Lösungen zu kennen, als für jede dieser Lösungen ein Experte zu sein.

Wissen in der Fachdomäne des Problems

Von Architekten wird erwartet, dass sie über ein gewisses Maß an Fachwissen im geschäftlichen Umfeld verfügen.

Effektive Softwarearchitekten verstehen nicht nur die Technologie, sondern auch den geschäftlichen Teil eines Problemraums. Ohne geschäftliches Wissen ist es nicht einfach, die geschäftlichen Probleme, Ziele und Anforderungen zu verstehen. Das erschwert es, eine effektive Architektur zu entwerfen, die auch die Businessanforderungen erfüllt. Stellen Sie sich vor, Sie sind Architekt für eine Großbank, ohne zu wissen, was gängige Fachbegriffe wie der durchschnittliche Richtungsindex (»average directional index«, ADX), aleatorische Verträge, Kursrallye oder auch nicht vorrangige Schulden (»nonpriority debt«) bedeuten. Ohne dieses Wissen kann ein Architekt nicht mit Entscheidungsträgern und geschäftlichen Nutzern kommunizieren, und er verliert schnell seine Glaubwürdigkeit.

Die erfolgreichsten Architekten, die wir kennen, haben ein breites technisches Wissen und Erfahrungen aus erster Hand, gekoppelt mit einem tiefen Verständnis für eine bestimmte Domäne. Diese Softwarearchitekten können effektiv mit Führungskräften und geschäftlichen Nutzern kommunizieren, indem sie ihr geschäftliches Wissen einsetzen und die gleiche Sprache sprechen wie diese Stakeholder. Das schafft wiederum ein großes Vertrauen in die Fähigkeiten der Architekten und vermittelt den Eindruck, dass sie wissen, was sie tun, und kompetent genug sind, eine effektive und korrekte Architektur zu schaffen.

Fähigkeiten im zwischenmenschlichen Umgang

Von Architekten wird erwartet, dass sie über außergewöhnliche Fähigkeiten im Umgang mit Menschen verfügen, inklusive Teamwork, Moderation und Führung.

Für die meisten Entwickler und Architekten ist der Besitz außergewöhnlicher Führungs- und zwischenmenschlicher Fähigkeiten eine schwierige Erwartung. Schließlich wollen Technologen, Entwickler und Architekten technische Probleme lösen und keine menschlichen. Aber, wie Gerald Weinberg (*https://oreil.ly/wyDB8*) einmal gesagt hat: »Egal, was sie euch erzählen, es ist immer ein menschliches Problem.« Ein Architekt muss nicht nur das Team technisch anleiten, sondern auch die Entwicklungsteams durch die Implementierung der Architektur führen. Führungskompetenz macht mindestens die Hälfte dessen aus, was effektive Softwarearchitekten ausmacht – unabhängig von deren Rolle oder ihrer Berufsbezeichnung.

Die Branche ist überflutet mit Softwarearchitekten, die um eine begrenzte Anzahl von Architektur-Arbeitsplätzen konkurrieren. Gute Führungsqualitäten und zwischenmenschliche Fähigkeiten sind ein guter Weg, sich von anderen Architekten