Kapitel 1. Einführung

Du interessierst dich also für Softwarearchitektur. Vielleicht bist du ein Entwickler, der den nächsten Karriereschritt machen will, oder du bist ein Projektmanager, der verstehen will, was passiert, wenn Softwarearchitekturen funktionieren. Vielleicht bist du auch ein "zufälliger Architekt": jemand, der Architekturentscheidungen trifft (siehe unten), aber nicht den Titel "Softwarearchitekt" trägt... noch nicht.

Warum solltest du dich mit der Softwarearchitektur beschäftigen? Vielleicht hast du schon Erfahrung mit vielen Projekten und möchtest besser verstehen, wie die größeren Teile eines Systems zusammenpassen und welche Kompromisse es gibt. Dann ist die Softwarearchitektur ein naheliegender nächster Karriereschritt.

Dieses Buch ist für euch alle gedacht. Es gibt einen Überblick über den extrem vielseitigen Beruf des "Softwarearchitekten".

Softwarearchitekten müssen Softwaresysteme in ihrer ganzen Komplexität verstehen und analysieren und wichtige Entscheidungen über Kompromisse treffen, manchmal mit unvollständigen Informationen. Viele Softwareentwicklerinnen und -entwickler, die befürchten, dass generative KI sie allmählich ersetzen könnte, erwägen,

in die Softwarearchitektur zu wechseln, eine Aufgabe, die viel schwerer zu ersetzen ist. Softwarearchitekten treffen genau die Art von Entscheidungen, die KI nicht treffen kann, indem sie Kompromisse in komplexen, sich verändernden Kontexten abwägen.

Architektur kann, wie vieles in der Kunst, nur im Kontext verstanden werden. Architekten orientieren sich bei ihren Entscheidungen an den Gegebenheiten ihres Umfelds. Ein Beispiel: Eines der Hauptziele der Software-Architektur des späten 20. Jahrhunderts war es, die gemeinsam genutzte Infrastruktur und die Ressourcen so effizient wie möglich zu nutzen, denn Betriebssysteme, Anwendungsserver, Datenbankserver und so weiter waren allesamt kommerziell und sehr teuer.

Im Jahr 2002 wäre der Versuch, eine Architektur wie Microservices aufzubauen, unvorstellbar teuer gewesen. Stell dir vor, du gehst in ein Rechenzentrum im Jahr 2002 und sagst dem Betriebsleiter: "Hey, ich habe eine tolle Idee für eine revolutionäre Architektur, bei der jeder Dienst auf einer eigenen isolierten Maschine mit einer eigenen Datenbank läuft. Ich brauche 50 Windows-Lizenzen, weitere 30 Lizenzen für Anwendungsserver und mindestens 50 Lizenzen für Datenbankserver." Solche Architekturen können wir heute nur dank der Einführung von Open Source und den aktualisierten technischen Verfahren der DevOps-Revolution aufbauen. Alle Architekturen sind das Produkt ihres Kontextes - vergiss das nicht, wenn du dieses Buch liest.

Definition der Software-Architektur

Was also ist Software-Architektur? Abbildung 1-1 zeigt, wie wir über Softwarearchitektur denken. Diese Definition hat vier Dimensionen. Die Softwarearchitektur eines Systems besteht aus einem *Architekturstil* als Ausgangspunkt, kombiniert mit den *Architekturmerkmalen*, die sie unterstützen muss, den *logischen Komponenten* zur Implementierung des Verhaltens und schließlich den *Architekturentscheidungen*, die das Ganze rechtfertigen. Die Struktur des Systems wird durch die dicken schwarzen Linien dargestellt, die die Architektur unterstützen. Wir gehen diese Dimensionen kurz in der Reihenfolge durch, in der Architekten sie analysieren.

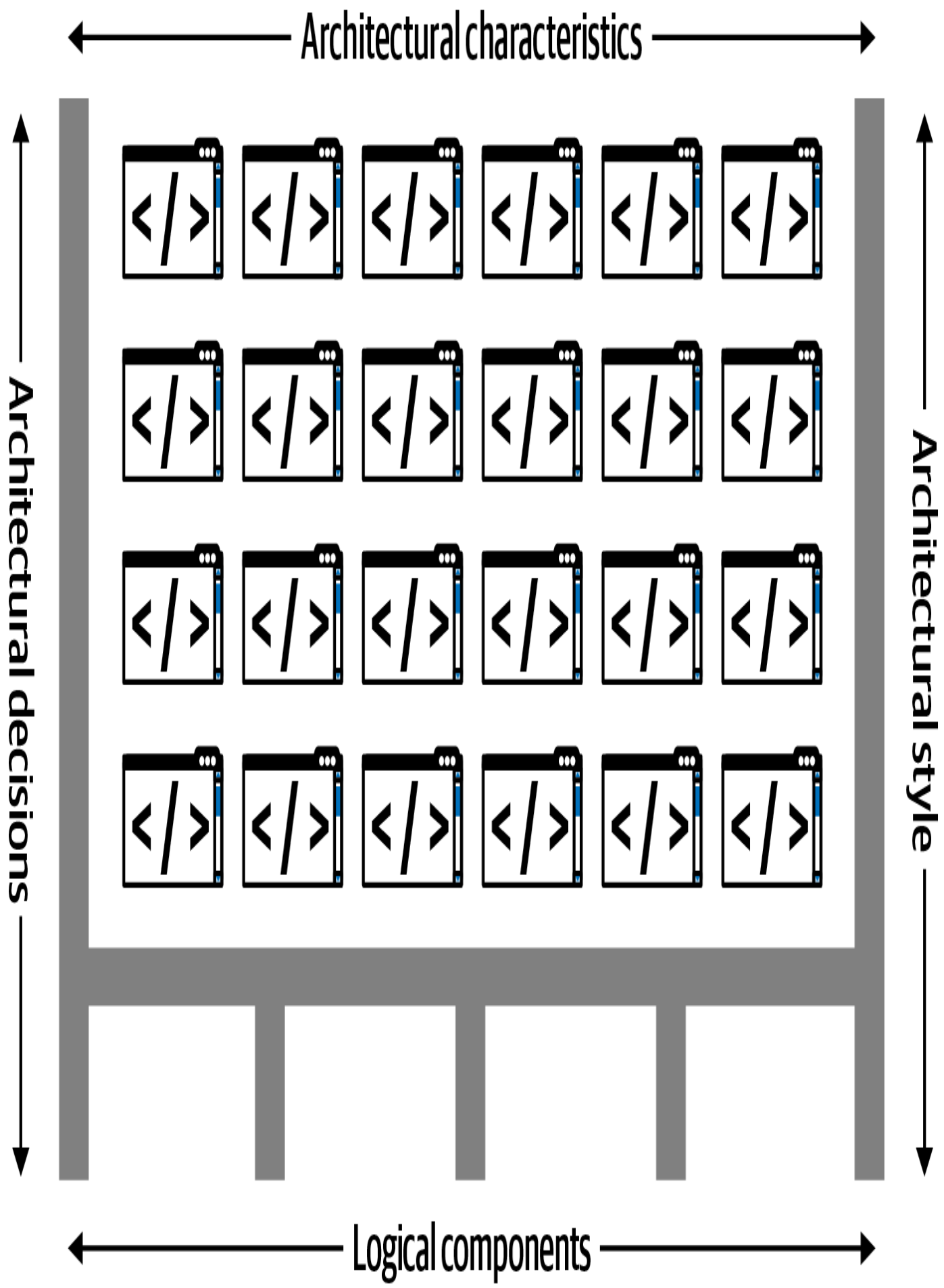
Architectural characteristics
Architectural decisions
Architectural style
Logical components

Abbildung 1-1. Die Architektur besteht aus der Struktur des Systems, kombiniert mit Architekturmerkmalen ("-fähigkeiten"), logischen Komponenten, Architekturstilen und Entscheidungen

Architekturmerkmale (siehe Abbildung 1-2) definieren die *Fähigkeiten* eines Systems (üblicherweise als "-ilities" abgekürzt) und die Kriterien für seinen Erfolg: kurz gesagt, was das System *tun* soll. Architekturmerkmale sind so wichtig, dass wir dem Verständnis und der Definition dieser Merkmale mehrere Kapitel in diesem Buch gewidmet haben.

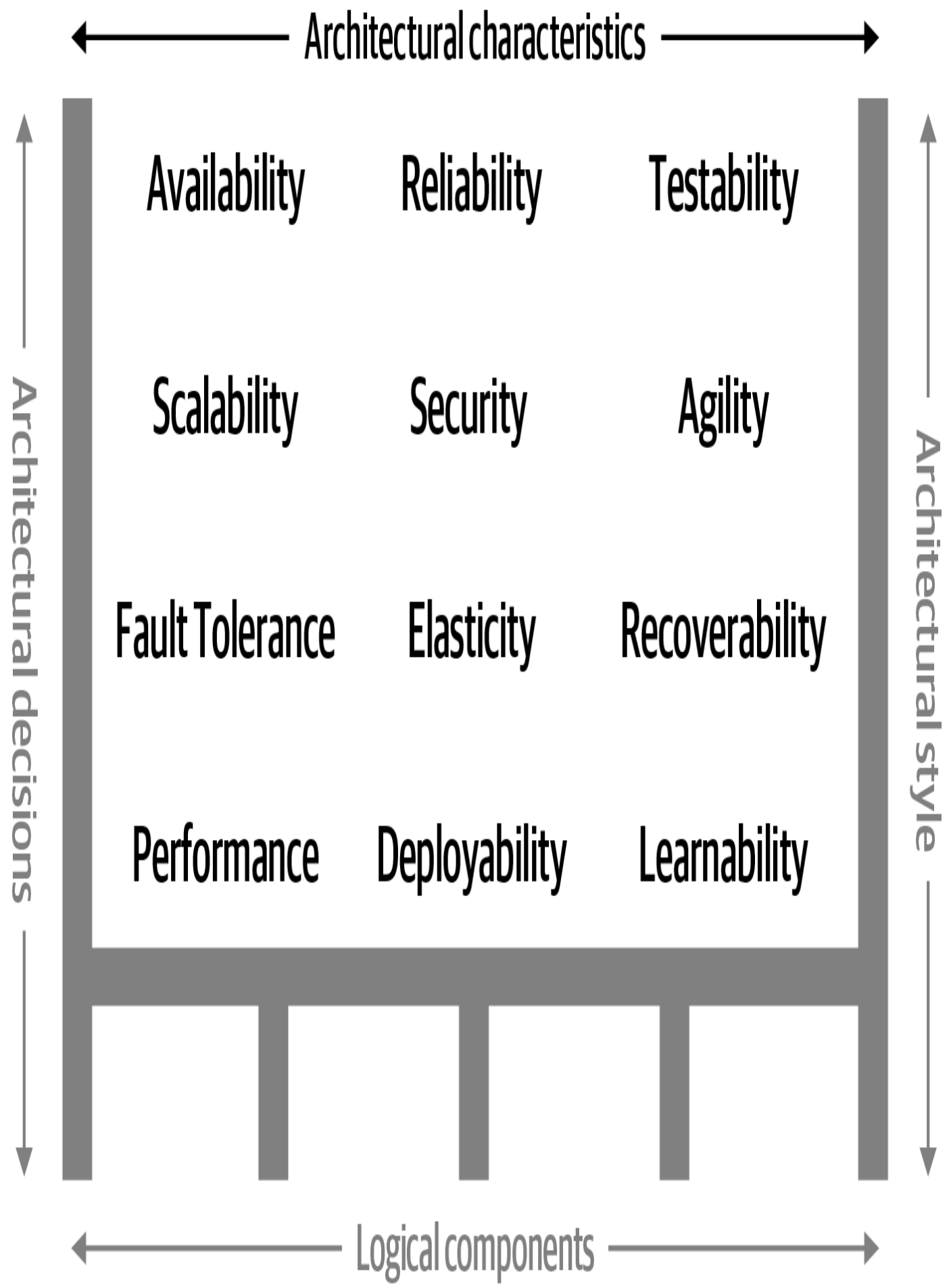
Architectural characteristics
Availability
Reliability
Testability
Scalability
Security
Agility
Fault Tolerance
Elasticity
Recoverability
Performance
Deployability
Learnability
Architectural decisions
Architectural style
Logical components

Abbildung 1-2. Die "Architekturmerkmale" beziehen sich auf die "-fähigkeiten", die das System unterstützen muss

Während architektonische Merkmale die Fähigkeiten eines Systems definieren, legen *logische Komponenten* sein *Verhalten* fest. Das Entwerfen logischer Komponenten ist eine der wichtigsten strukturellen Aktivitäten für Architekten. In Abbildung 1-3 bilden die logischen Komponenten die Domänen, Entitäten und Arbeitsabläufe der Anwendung.

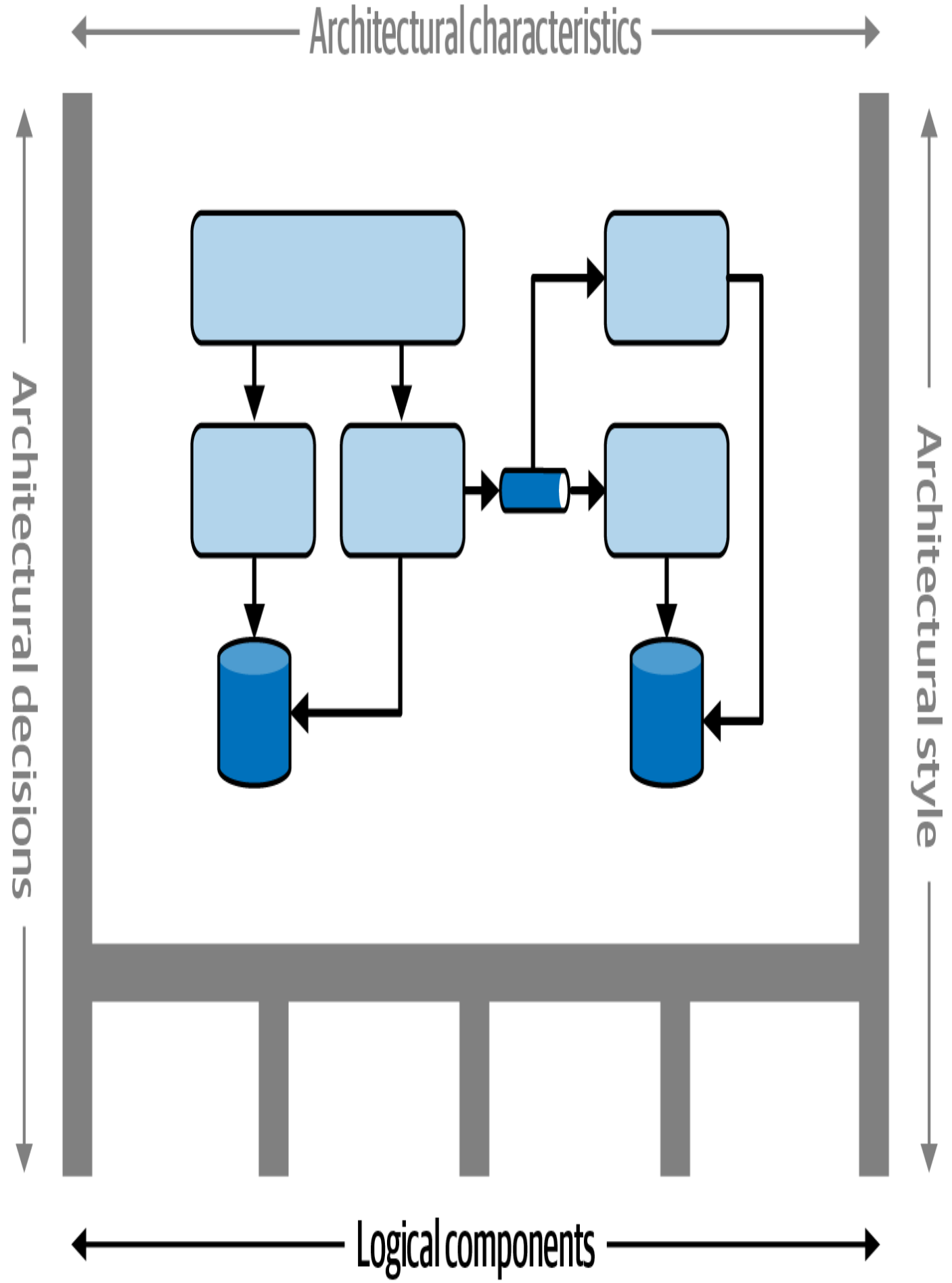
Architectural characteristics
Architectural decisions
Architectural style
Logical components

Abbildung 1-3. Logische Komponenten strukturieren das Verhalten des Systems

Sobald ein Architekt die architektonischen Merkmale und die logischen Komponenten, die das System benötigt, analysiert hat (beides wird später ausführlich beschrieben), weiß er genug, um einen geeigneten Architekturstil als Ausgangspunkt für die Umsetzung seiner Lösung zu wählen Abbildung 1-4.

Architectural characteristics

Architectural decisions

Architecture styles used

Architectural style

Logical components

Abbildung 1-4. Bei der Wahl eines Architekturstils geht es darum, den einfachsten Implementierungspfad für einen bestimmten Satz von Anforderungen zu finden

Die vierte Dimension, die die Softwarearchitektur definiert, sind *Architekturentscheidungen*, die die Regeln für den Aufbau eines Systems festlegen. Ein Architekt könnte zum Beispiel entscheiden, dass nur die Business- und die Serviceschicht innerhalb einer Schichtenarchitektur auf die Datenbank zugreifen können (siehe Abbildung 1-5) und die Präsentationsschicht keine direkten Datenbankaufrufe tätigen darf. Architekturentscheidungen bilden die Grenzen des Systems und geben den Entwicklungsteams vor, was erlaubt ist und was nicht.

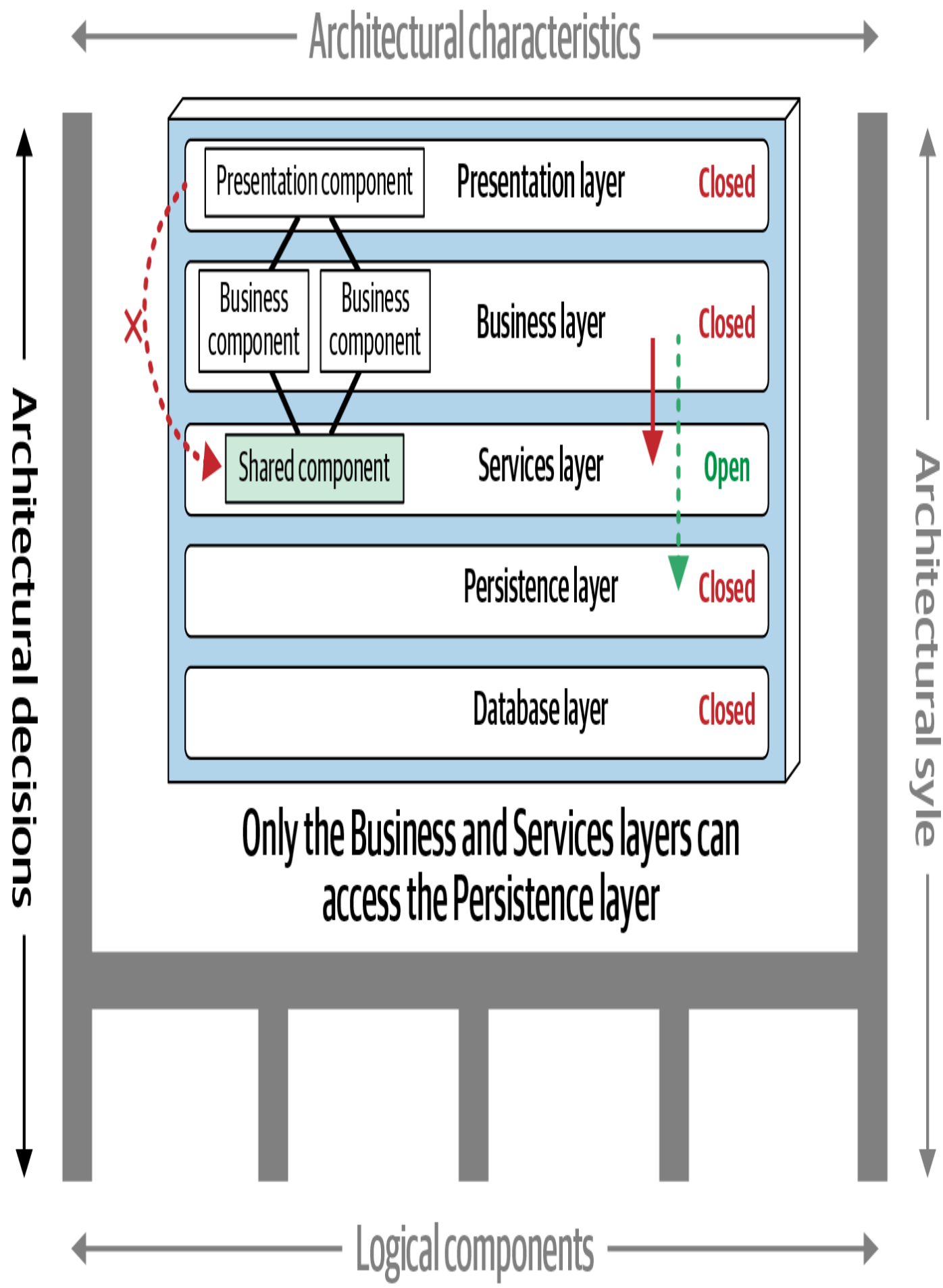
Architectural characteristics
Presentation component
Presentation layer
Closed
Business component
Business component
Business layer
Closed
Shared component
Services layer
Open
Persistence layer
Closed
Database layer
Closed
Only the Business and Services layers can access the Persistence layer
Architectural decisions
Architectural syle
Logical components

Abbildung 1-5. Architekturentscheidungen sind Regeln für den Aufbau von Systemen

In Kapitel 21 besprechen wir Architekturentscheidungen und wie man sie prägnant dokumentiert.

Gesetze der Softwarearchitektur

Als eure beiden Autoren sich daran machten, die erste Ausgabe dieses Buches zu schreiben, hatten wir ein ehrgeiziges Ziel: Wir hofften, Dinge zu finden, die in Bezug auf die Software-Architektur allgemeingültig zu sein schienen, und sie als "Gesetze" der Software-Architektur festzuhalten. Während wir schrieben, hielten wir die Augen nach Dingen offen, die wir festhalten konnten; wir hatten gehofft, vielleicht 10 oder 15 zu finden. Zu unserer Überraschung fanden wir in der ersten Ausgabe nur zwei Gesetze und entdeckten beim Schreiben der zweiten Ausgabe ein weiteres. Getreu unserer ursprünglichen Absicht scheinen diese drei Gesetze ziemlich allgemeingültig zu sein und bieten viele wichtige Perspektiven für die Arbeit von Softwarearchitekten.

Wir haben das erste Gesetz der Softwarearchitektur gelernt, indem wir immer wieder darüber gestolpert sind, und wir glauben, dass es den Kern dessen trifft, warum diese universellen Wahrheiten so schwer zu fassen sind:

> *Alles in der Softwarearchitektur ist ein Kompromiss.*
>
> —Erstes Gesetz der Softwarearchitektur

Nichts existiert auf einem schönen, klaren Spektrum. Jede Entscheidung, die ein Softwarearchitekt trifft, muss eine große Anzahl von Variablen berücksichtigen, die je nach den Umständen unterschiedliche Werte annehmen. Kompromisse sind das A und O von Softwarearchitekturentscheidungen.

> *Wenn du denkst, dass du etwas entdeckt hast, das kein Kompromiss ist, hast du wahrscheinlich nur den Kompromiss nicht erkannt... noch nicht.*
>
> —Korollar 1

> *Du kannst die Analyse der Kompromisse nicht nur einmal machen und damit fertig sein.*
>
> —Korollar 2

Teams lieben Standards, und es wäre schön, wenn Architekten einfach ein großes Kompromiss-Jamboree veranstalten könnten, um zu entscheiden, welche Architekturstile verwendet werden sollen, wie Teile der Architektur kommunizieren sollen, wie gemeinsame Funktionen verwaltet werden sollen und eine Menge anderer schwieriger Entscheidungen. Das können wir aber nicht, denn in jeder Situation müssen wir all diese Kompromisse neu abwägen. (Wir haben gesehen, wie Teams dies versucht haben - zum Beispiel, indem sie versuchten, in verteilten Workflows standardmäßig nur Choreografie zu verwenden, nur um dann festzustellen, dass dies manchmal funktioniert und

manchmal ein spektakuläres Desaster ist. Siehe "Choreografie und Orchestrierung" für unsere Diskussion).

Architektur ist mehr als nur die Kombination von Strukturelementen. Deshalb umfasst unsere dimensionale Definition auch Prinzipien, Merkmale usw. Dies spiegelt sich in unserem zweiten Gesetz der Softwarearchitektur wider:

> *DasWarum ist wichtiger als das Wie.*
>
> —Zweites Gesetz der Softwarearchitektur

Wenn mir als erfahrener Architekt jemand eine Architektur zeigt, die ich noch nie gesehen habe, kann ich zwar verstehen , *wie* sie funktioniert, aber ich kann mir nicht erklären, *warum* der vorherige Architekt oder das Team bestimmte Entscheidungen getroffen hat. Architekten treffen Entscheidungen in einem sehr spezifischen Kontext, was pauschale und generische Entscheidungen schwierig macht. *Warum* ein Architekt eine bestimmte Entscheidung getroffen hat, beinhaltet auch die Kompromisse, die er in Betracht gezogen hat, und trägt dazu bei, warum diese Entscheidung besser ist als eine andere. Es zeigt sich, dass eines der wichtigsten Merkmale von Softwarearchitekturentscheidungen darin besteht, dass sie selten binär sind. Das bringt uns zum dritten Gesetz der Softwarearchitektur:

> *Die meisten Architekturentscheidungen sind nicht binär, sondern bewegen sich in einem Spektrum zwischen den Extremen.*
>
> —Drittes Gesetz der Softwarearchitektur

Im gesamten Buch zeigen wir auf, *warum* Architekten bestimmte Entscheidungen treffen und welche Kompromisse sie eingehen müssen. In Kapitel 21 zeigen wir außerdem gute Techniken, um wichtige Entscheidungen festzuhalten.

Die Leser werden diese Gesetze im gesamten Buch wiedererkennen und wir empfehlen, sie bei der Bewertung von Entscheidungen über die Softwarearchitektur im Hinterkopf zu behalten. Wir kommen auf diese Gesetze in Kapitel 27 mit einigen zusätzlichen Beispielen zurück.

Nachdem wir nun eine Arbeitsdefinition der Software-Architektur haben, können wir uns der eigentlichen Aufgabe zuwenden.

Erwartungen an einen Architekten

Die Rolle eines Softwarearchitekten kann von der Rolle eines erfahrenen Programmierers bis hin zur Festlegung der strategischen technischen Ausrichtung eines ganzen Unternehmens reichen. Das macht es schwierig, die Rolle zu definieren, aber wir können dir sagen, was von einem Softwarearchitekten *erwartet wird*. Wir haben acht zentrale Erwartungen an einen Softwarearchitekten/eine Softwarearchitektin

herausgearbeitet, unabhängig von ihrer Rolle, ihrem Titel oder ihrer Stellenbeschreibung:

- Entscheidungen zur Architektur treffen
- Kontinuierliche Analyse der Architektur
- Bleib auf dem Laufenden mit den neuesten Trends
- Sicherstellen der Einhaltung von Entscheidungen
- Verschiedene Technologien, Frameworks, Plattformen und Umgebungen zu verstehen
- Das Geschäftsfeld kennen
- Ein Team leiten und über zwischenmenschliche Fähigkeiten verfügen
- Organisationspolitik verstehen und steuern

Um als Softwarearchitekt erfolgreich zu sein, muss man jede dieser Erwartungen verstehen und erfüllen. In diesem Abschnitt werden alle acht nacheinander betrachtet.

Architekturentscheidungen treffen

Von einem Architekten wird erwartet, dass er die Architekturentscheidungen und Designprinzipien festlegt, die als Grundlage für Technologieentscheidungen im Team, in der Abteilung oder im gesamten Unternehmen dienen.

Leitfaden ist das Schlüsselwort in dieser ersten Erwartung: Architekten sollten die Wahl der Technologie *anleiten*, anstatt sie *vorzugeben*. Die Entscheidung, React.js für die Frontend-Entwicklung zu verwenden, ist zum Beispiel eine technische Entscheidung und keine architektonische

Entscheidung. Anstatt diese Entscheidung zu treffen, sollte der Architekt die Entwicklungsteams anweisen, ein reaktionsfähiges Framework für die Frontend-Webentwicklung zu verwenden, und ihnen die Wahl zwischen Angular, Elm, React.js, Vue oder einem anderen reaktionsfähigen Web-Framework empfehlen. Der Schlüssel ist die Frage, ob die Architekturentscheidung die Teams bei der Wahl der richtigen technischen Lösung *anleitet* oder ihnen die Wahl abnimmt. Es gibt jedoch auch Fälle, in denen Architekten bestimmte technische Entscheidungen treffen müssen, um ein bestimmtes architektonisches Merkmal wie Skalierbarkeit, Leistung oder Verfügbarkeit zu erhalten. Für Architekten ist es oft schwierig, dieses Gleichgewicht zu finden, deshalb geht es in Kapitel 21 ausschließlich um Architekturentscheidungen.

Analysiere die Architektur fortlaufend

Von einem Architekten wird erwartet, dass er die Architektur und die aktuelle Technologieumgebung kontinuierlich analysiert und Lösungen zur Verbesserung vorschlägt.

Die Vitalität der Architektur bewertet, wie lebensfähig eine Architektur, die vor drei oder mehr Jahren definiert wurde, angesichts der Veränderungen in der Wirtschaft und der Technologie *heute* ist. Unserer Erfahrung nach konzentrieren nicht genug Architekten ihre Energie darauf, bestehende Architekturen kontinuierlich zu analysieren. Infolgedessen unterliegen die meisten Architekturen einem gewissen strukturellen Verfall, der eintritt, wenn Entwickler Änderungen am Code

oder Design vornehmen, die sich auf die erforderlichen Architektureigenschaften wie Leistung, Verfügbarkeit undSkalierbarkeit auswirken.

Andere häufig vergessene Aspekte dieser Erwartung sind Test- und Freigabeumgebungen. Die Möglichkeit, Code schnell zu ändern, ist eine Art von Agilität mit offensichtlichen Vorteilen, aber wenn es Wochen dauert, die Änderungen zu testen, und Monate bis zur Freigabe, kann die Gesamtarchitektur keine Agilität erreichen.

Architekten müssen Veränderungen in der Technologie und der Problemdomäne ganzheitlich analysieren, um festzustellen, ob die Architektur weiterhin tragfähig ist. Diese Art von Überlegung sorgt dafür, dass Bewerbungen relevant bleiben, auch wenn sie selten in Stellenausschreibungen auftauchen.

Mit den neuesten Trends auf dem Laufenden bleiben

Von einem Architekten/einer Architektin wird erwartet, dass er/sie immer auf dem neuesten Stand der Technik und der Branchentrends ist.

Entwickler/innen müssen sich über die neuesten Technologien auf dem Laufenden halten, auch über die, die sie täglich nutzen, um relevant zu bleiben (und um ihren Job zu behalten!). Für Architekten ist es sogar noch wichtiger, sich über die neuesten technischen und branchenspezifischen Trends auf dem Laufenden zu halten. Entscheidungen, die Architekten treffen, sind in der Regel von langer

Dauer und lassen sich nur schwer ändern. Das Verständnis von Trends hilft Architekten, Entscheidungen zu treffen, die auch in Zukunft relevant bleiben. In den letzten Jahren mussten sich Architekten zum Beispiel mit cloudbasierter Speicherung und Bereitstellung befassen, und während wir diese zweite Ausgabe schreiben, hat generative KI einen massiven Einfluss auf viele Bereiche des Entwicklungsökosystems.

Trends zu verfolgen und auf dem Laufenden zu bleiben, ist schwer, besonders für einen Softwarearchitekten. In Kapitel 2 besprechen wir einige Techniken und Ressourcen, mit denen du das schaffen kannst.

Sicherstellen der Einhaltung von Beschlüssen

Von einem Architekten wird erwartet, dass er die Einhaltung von Architekturentscheidungen und Gestaltungsprinzipien sicherstellt.

Um die Einhaltung der Vorschriften zu gewährleisten, muss ständig überprüft werden, ob die Entwicklungsteams die vom Architekten festgelegten, dokumentierten und kommunizierten Entscheidungen und Entwurfsprinzipien befolgen.

Stell dir vor, du als Architekt entscheidest dich, den Zugriff auf die Datenbank in einer Schichtenarchitektur nur auf die Geschäfts- und Serviceschichten zu beschränken (nicht auf die Präsentationsschicht). Das bedeutet (wie du in Kapitel 10 sehen wirst), dass die Präsentationsschicht alle Schichten der Architektur durchlaufen muss, um auch nur den einfachsten Datenbankaufruf zu tätigen. Ein Entwickler der Benutzeroberfläche (UI) ist jedoch mit dieser

Entscheidung nicht einverstanden und gibt der Präsentationsschicht aus Leistungsgründen direkten Zugriff auf die Datenbank. Du hast diese Architekturentscheidung aus einem bestimmten Grund getroffen: damit sich Änderungen an der Datenbank nicht auf die Präsentationsschicht auswirken. Wenn du nicht dafür sorgst, dass deine Architekturentscheidungen eingehalten werden, kann es zu Verstößen wie diesem kommen. Das kann dazu führen, dass die Architektur nicht die geforderten Eigenschaften aufweist und die Anwendung oder das System nicht wie erwartet funktioniert. In Kapitel 6 geht es um die Messung der Konformität mithilfe von automatischen Fitnessfunktionen und anderen Tools.

Verschiedene Technologien verstehen

Von einem Architekten wird erwartet, dass er mit mehreren und unterschiedlichen Technologien, Frameworks, Plattformen und Umgebungen vertraut ist.

Es wird nicht von jedem Architekten erwartet, dass er Experte für jedes Framework, jede Plattform und jede Sprache ist, aber er sollte zumindest mit einer Vielzahl von Technologien vertraut sein. Die meisten Umgebungen sind heutzutage heterogen, daher sollten Architekten zumindest wissen, wie sie mit mehreren Systemen und Diensten zusammenarbeiten können, unabhängig von Sprache, Plattform oder Technologie.

Eine der besten Möglichkeiten, diese Erwartung zu erfüllen, besteht darin, über den Tellerrand hinauszuschauen und offensiv nach Möglichkeiten zu suchen, Erfahrungen in verschiedenen Sprachen, Plattformen und Technologien zu sammeln. Architekten sollten sich eher auf die technische *Breite* als auf die technische Tiefe konzentrieren. Die technische Breite umfasst die Dinge, über die du zwar Bescheid weißt, aber nicht im Detail, kombiniert mit den Dingen, über die du viel weißt. Zum Beispiel ist es für einen Architekten viel wertvoller, mit den Vor- und Nachteilen von 10 verschiedenen Caching-Produkten vertraut zu sein, als nur ein Experte für eines davon zu sein.

Das Geschäftsfeld kennen

Von einem Architekten wird erwartet, dass er über ein gewisses Maß an Fachwissen in seinem Geschäftsbereich verfügt.

Effektive Softwarearchitekten verstehen das Geschäftsproblem, die Ziele und die Anforderungen, die mit der Architektur gelöst werden sollen, auch bekannt als die *Geschäftsdomäne* eines Problembereichs. Es ist schwierig, eine effektive Architektur zu entwerfen, wenn man die Anforderungen des Unternehmens nicht versteht. Stell dir vor, du wärst Architekt bei einer großen Bank und würdest gängige Finanzbegriffe wie *"durchschnittlicher Richtungsindex"*, " *aleatorische Verträge"*, " *Zinsrallye"* oder sogar *"nicht-vorrangige Schulden"* nicht verstehen. Du wärst nicht in der Lage, mit Stakeholdern und Geschäftsanwendern zu kommunizieren und würdest schnell an Glaubwürdigkeit verlieren.

Die erfolgreichsten Architekten, die wir kennen, verfügen über ein umfassendes, praxisorientiertes technisches Wissen *und* fundierte Kenntnisse in einem bestimmten Bereich. Sie können mit Führungskräften und Geschäftsanwendern in einer Sprache kommunizieren, die diese Beteiligten kennen und verstehen. Das schafft Vertrauen, dass sie wissen, was sie tun, und dass sie kompetent sind, eine effektive und korrekte Architektur zu erstellen.

Über zwischenmenschliche Fähigkeiten verfügen

Von einem Architekten wird erwartet, dass er außergewöhnliche zwischenmenschliche Fähigkeiten besitzt, einschließlich Teamarbeit, Moderation und Führung.

Als Technologen neigen Entwickler und Architekten dazu, lieber technische Probleme zu lösen als zwischenmenschliche Probleme, so dass außergewöhnliche Führungsqualitäten und zwischenmenschliche Fähigkeiten kaum erwartet werden. Aber wie Gerald Weinberg schon sagte: "Egal, was sie dir sagen, es ist immer ein menschliches Problem". Die Anleitung, die Architekten geben, ist nicht nur technischer Natur - sie umfasst auch die Führung der Entwicklungsteams bei der Umsetzung der Architektur. Führungsqualitäten sind *mindestens die Hälfte* dessen, was man braucht, um ein effektiver Softwarearchitekt zu werden, unabhängig von der Rolle oder dem Titel.

Die Branche ist überschwemmt mit Softwarearchitekten, die sich alle um eine begrenzte Anzahl von Stellen bewerben. Diejenigen mit starken Führungs- und Sozialkompetenzen heben sich von der Masse ab. Umgekehrt kennen wir viele Softwarearchitekten, die hervorragende Technologen sind, aber Schwierigkeiten haben, Teams zu leiten, Entwickler zu coachen und zu betreuen oder Ideen, Architekturentscheidungen und Prinzipien zu vermitteln. Natürlich ist es für diese Architekten schwierig, einen Job zu behalten.

Politik verstehen und navigieren

Von einem Architekten/einer Architektin wird erwartet, dass er/sie das politische Klima des Unternehmens versteht und in der Lage ist, sich in der Politik zurechtzufinden.

Es mag seltsam erscheinen, in einem Buch über Software-Architektur über Büropolitik zu sprechen, aber Verhandlungsgeschick ist für diese Rolle entscheidend. Um das zu veranschaulichen, betrachte zwei Szenarien.

Im ersten Szenario beschließt ein Entwickler, ein bestimmtes Entwurfsmuster zu verwenden, um die Komplexität eines bestimmten Teils des komplizierten Codes zu reduzieren. Das ist eine gute Entscheidung, für die der Entwickler keine Genehmigung einholen muss. Programmieraspekte wie die Codestruktur, das Klassendesign, die Auswahl von Entwurfsmustern und manchmal sogar die Wahl der Sprache sind alle Teil der Programmierkunst.

Im zweiten Szenario hat der Architekt, der für ein großes CRM-System (Customer Relationship Management) verantwortlich ist, Schwierigkeiten, den Datenbankzugriff von anderen Systemen zu kontrollieren, bestimmte Kundendaten zu sichern und Änderungen am Datenbankschema vorzunehmen. All diese Probleme rühren daher, dass zu viele andere Systeme die CRM-Datenbank nutzen. Der Architekt beschließt daher, *Anwendungssilos* zu schaffen, in denen jede Anwendungsdatenbank nur von der Anwendung aus zugänglich ist, der die Datenbank gehört. Mit dieser Entscheidung hat der Architekt eine bessere Kontrolle über die Kundendaten, die Sicherheit und die Änderungen.

Anders als bei der Entscheidung des Entwicklers im ersten Szenario kann der Architekt jedoch damit rechnen, dass fast jeder im Unternehmen seine Entscheidung anfechten wird (mit der möglichen Ausnahme des CRM-Anwendungsteams). Andere Anwendungen benötigen die CRM-Daten, und wenn sie nicht mehr direkt auf die Datenbank zugreifen können, müssen sie das CRM-System über Fernzugriffsaufrufe nach den Daten fragen. Produktverantwortliche, Projektmanager und Geschäftsinteressenten können sich gegen höhere Kosten oder einen höheren Aufwand wehren, während die Entwickler ihren Ansatz vielleicht für besser halten. *Fast jede Entscheidung, die ein Architekt trifft, wird in Frage gestellt werden.*

Wie auch immer die Einwände lauten, der Architekt muss sich durch die Organisationspolitik bewegen und Verhandlungsgeschick anwenden, um die meisten Entscheidungen durchzusetzen. Das kann sehr

frustrierend sein. Die meisten Softwarearchitekten haben als Entwickler angefangen und sich daran gewöhnt, Entscheidungen ohne Zustimmung oder gar Überprüfung zu treffen. Als Architekten sind sie nun endlich in der Lage, weitreichende und wichtige Entscheidungen zu treffen, aber sie müssen sich für fast jede Entscheidung rechtfertigen und dafür kämpfen. Verhandlungsgeschick ist ebenso wie Führungskompetenz so wichtig, dass wir ihm ein ganzes Kapitel gewidmet haben(Kapitel 25).

Straßenkarte

Dieses Buch besteht aus drei Teilen:

Teil I: Grundlagen
Teil I definiert die Schlüsselkomponenten der Softwarearchitektur und konzentriert sich dabei auf die beiden Schlüsselelemente der Architekturstruktur: architektonische Merkmale und logische Komponenten. Für die Analyse jedes dieser Elemente sind verschiedene Techniken erforderlich, die wir ausführlich behandeln. Die Ergebnisse dieser Aktivitäten liefern dem Softwarearchitekten genügend Informationen, um einen geeigneten Architekturstil zu wählen, der ein Gerüst für die allgemeine Philosophie der Implementierung darstellt.

Teil II: Architekturstile
Teil II enthält einen Katalog von *Architekturstilen*, den sogenannten Topologien der Softwarearchitektur. Wir zeigen die strukturellen und kommunikativen Unterschiede der einzelnen Stile auf und bieten eine

Grundlage für den Vergleich entlang eines breiten Spektrums, einschließlich Datentopologien, Teams und physischen Architekturen.

Teil III: Techniken und Soft Skills

Die Teile I und II konzentrieren sich auf die technischen Aspekte des Berufs, aber ein großer Teil der Rolle des Softwarearchitekten umfasst auch das, was traditionell als *Soft Skills* bezeichnet wird: Fähigkeiten, die mit anderen Menschen zu tun haben und nicht mit der Technik. Ironischerweise sind Soft Skills für angehende Softwarearchitekten oft am schwierigsten zu erlernen, weil die Hauptwege zur Rolle des Softwarearchitekten in der Regel mehr technische als zwischenmenschliche Brillanz erfordern. Wenn sie jedoch erst einmal im Job sind, stellen Architekten fest, dass diese Fähigkeiten von entscheidender Bedeutung sind. Teil III unseres Buches befasst sich daher mit einigen wichtigen Soft Skills, die dir helfen, in dieser Rolle erfolgreich zu sein.