Kapitel 7. Der Umfang der architektonischen Merkmale

Die Denkweise von Softwarearchitekten muss sich mit unserem Ökosystem weiterentwickeln, und nirgendwo wird das deutlicher als bei der Festlegung von Architektureigenschaften. Viele der veralteten Rahmenwerke zur Bestimmung von Architekturmerkmalen hatten einen fatalen Fehler: Sie gingen von einem Satz von Architekturmerkmalen für das gesamte System aus. Das trifft zwar manchmal noch zu, aber viele moderne Architekturen, wie z. B. Microservices, enthalten unterschiedliche Architekturmerkmale auf Service- und Systemebene.

Der Umfang der architektonischen Merkmale ist ein nützliches Maß für Architekten, insbesondere wenn es darum geht, den am besten geeigneten Architekturstil als Ausgangspunkt für die Implementierung zu bestimmen. Als wir unser Buch *Building Evolutionary Architectures* geschrieben haben, brauchten wir eine Methode, um die strukturelle Evolvierbarkeit bestimmter Architekturstile zu messen. Keines der vorhandenen Verfahren bot den richtigen Detaillierungsgrad. Im Abschnitt "Strukturelle Messgrößen" wird eine Vielzahl von Messgrößen auf Code-Ebene erörtert, mit denen Architekten die strukturellen

Aspekte einer Architektur analysieren können - aber keine dieser Messgrößen spiegelt den Umfang wider. Sie geben Aufschluss über Details auf niedriger Ebene des Codes, können aber keine abhängigen Komponenten außerhalb der Codebasis (wie z. B. Datenbanken) bewerten, die sich auf viele Architekturmerkmale auswirken, vor allem auf betriebliche Merkmale. Egal, wie viel Mühe sich ein Architekt gibt, eine Codebasis so zu gestalten, dass sie performant oder elastisch ist, wenn die Datenbank des Systems nicht zu diesen Eigenschaften passt, werden seine Bemühungen nicht erfolgreich sein.

Da uns ein gutes Maß für den Umfang fehlschlug, haben wir eines entwickelt. Wir nennen es *Architektur-Quantum*.

Architektonische Quanten und Granularität

Die Kopplung auf Komponentenebene ist nicht das Einzige, was die Software zusammenhält. Viele Geschäftskonzepte verbinden Teile des Systems semantisch miteinander und schaffen so einen *funktionalen Zusammenhalt*. Um Software erfolgreich zu entwerfen, zu analysieren und weiterzuentwickeln, müssen Architekten und Entwickler alle Kopplungspunkte berücksichtigen, die brechen könnten.

PLURALE FÜR LATEINISCHE FACHBEGRIFFE

Quantum stammt aus dem Lateinischen, was bedeutet, dass der Plural mit einem *a* endet. Wenn du mehr als ein Quantum hast, hast du *Quanten - wie die*gebräuchlicheren *Daten*, die ebenfalls aus dem Lateinischen stammen. Architekten sprechen selten über ein einzelnes Datum. Das *Chicago Manual of Style* merkt dazu an: "Obwohl dieses Wort ursprünglich ein Plural von *datum* war, wird es heute üblicherweise als Massensubstantiv behandelt und mit einem Singularverb verbunden" (18. Auflage, 2024, S. 336).

Viele naturwissenschaftlich bewanderte Architekten kennen den Begriff "Quantum" aus der Physik, wo er sich auf die kleinstmögliche Menge von etwas bezieht - in der Regel Energie. Das Wort *Quantum* stammt aus dem Lateinischen und bedeutet "wie groß" oder "wie viel". Im allgemeinen Sprachgebrauch bedeutet es in der Regel "kleines, unzerstörbares Ding", so wie wir ein *Architektur-Quant* definieren. Eine andere gebräuchliche informelle Definition von *Architekturquantum* ist "der kleinste Teil des Systems, der unabhängig läuft". Zum Beispiel bilden Microservices oft Architekturquanten, was diese Definition veranschaulicht: Ein Service kann innerhalb der Architektur unabhängig laufen, einschließlich seiner eigenen Daten und andererAbhängigkeiten.

Ein *Architekturquantum* legt den Rahmen für eine Reihe von Architekturmerkmalen fest. Es weist folgende Merkmale auf:

- Unabhängiger Einsatz von anderen Teilen der Architektur

- Hoher funktionaler Zusammenhalt
- Geringe statische Kopplung der externen Implementierung
- Synchrone Kommunikation mit anderen Quanten

Diese Definition besteht aus mehreren Teilen. Lasst sie uns aufschlüsseln:

Legt den Rahmen für eine Reihe von Architekturmerkmalen fest
Architekten verwenden das Architektur-Quantum als Grenze, die eine Reihe von Architekturmerkmalen abgrenzt, insbesondere operative Merkmale. Da sie unabhängig einsetzbar ist und eine hohe funktionale Kohäsion aufweist (siehe unten), ist das Architektur-Quantum ein nützliches Maß für die Modularität der Architektur.

Unabhängig einsetzbar
Ein Architektur-Quantum umfasst alle notwendigen Komponenten, die unabhängig von anderen Teilen der Architektur funktionieren. Wenn eine Anwendung zum Beispiel eine Datenbank verwendet, ist diese Datenbank Teil des Quants, weil das System ohne sie nicht funktioniert. Diese Anforderung bedeutet, dass praktisch alle Legacy-Systeme, die mit einer einzigen Datenbank arbeiten, per Definition ein Quantum Eins bilden. In der Microservices-Architektur enthält jedoch jeder Dienst seine eigene Datenbank (Teil der Philosophie der *begrenzten Kontextsteuerung*, die in Kapitel 18 ausführlich beschrieben wird). Dadurch entstehen mehrere Quanten innerhalb dieser Architektur, denn jeder Dienst hat seinen eigenen Architekturbereich.

Hoher funktionaler Zusammenhalt

Kohäsion im Komponentendesign bezieht sich darauf, wie einheitlich der enthaltene Code in seinem Zweck ist. Eine Komponente `Customer` mit Eigenschaften und Methoden, die sich alle auf einen *Kunden* beziehen, weist zum Beispiel eine hohe Kohäsion auf. Eine `Utility` Komponente mit einer wahllosen Sammlung verschiedener Methoden würde dies nicht tun. Eine hohe funktionale Kohäsion bedeutet, dass ein Architekturquantum etwas Sinnvolles tut. Diese Unterscheidung spielt bei traditionellen monolithischen Anwendungen mit einer einzigen Datenbank kaum eine Rolle, da hier der Zusammenhalt im Wesentlichen das gesamte System betrifft. Bei verteilten Architekturen wie ereignisgesteuerten oder Microservices ist es jedoch wahrscheinlicher, dass die Architekten jeden Dienst so gestalten, dass er zu einem einzigen Arbeitsablauf passt (ein *begrenzter Kontext*, wie in "Domain-Driven Design's Bounded Context" beschrieben), so dass dieser Dienst eine hohe funktionale Kohäsion aufweisen würde.

DOMAIN-DRIVEN DESIGN IM BEGRENZTEN KONTEXT

Eric Evans' Buch *Domain-Driven Design* (Addison-Wesley Professional, 2003) hat das moderne Architekturdenken nachhaltig beeinflusst. *Domain-Driven Design* (DDD) ist eine Modellierungstechnik, die es Architekten ermöglicht, komplexe Problemdomänen auf organisierte Weise aufzuschlüsseln. DDD definiert einen *begrenzten Kontext*, in dem alles, was mit einem Teil der Domäne zu tun hat, intern sichtbar, aber für andere begrenzte Kontexte undurchsichtig ist.

Vor DDD versuchten Architekten, Code ganzheitlich über gemeinsame Einheiten innerhalb der Organisation wiederzuverwenden. Sie stellten jedoch fest, dass die Erstellung gemeinsamer Artefakte eine Reihe von Problemen mit sich bringt, z. B. eine enge Kopplung, eine schwierigere Koordination und eine höhere Komplexität. Das Konzept des begrenzten Kontexts erkennt an, dass jede Einheit am besten in einem lokal begrenzten Kontext funktioniert. Anstatt also eine einheitliche `Customer` Klasse für die gesamte Organisation zu schaffen, kann jede Problemdomäne ihre eigene `Customer` Klasse erstellen und die Unterschiede an den Kommunikationspunkten mit anderen Domänen ausgleichen.

Um die nächsten Teile der Definition zu verdeutlichen, müssen wir einige genauere Unterscheidungen über die Arten der Kopplung treffen:

Semantische Kopplung

Die*semantische Kopplung* beschreibt die natürliche Kopplung des Problems, für das ein Architekt eine Lösung entwickelt. Die Kopplung einer Anwendung zur Auftragsabwicklung umfasst zum Beispiel Dinge wie Inventar, Kataloge, Warenkörbe, Kunden und Verkäufe. Die Art des Problems, das zur Entwicklung einer Softwarelösung führt, definiert diese Kopplung. Architekten haben nur wenige Techniken, die verhindern, dass sich Änderungen an der Domäne auf das System auswirken: Eine Änderung der Domäne (und damit der Semantik) bedeutet, dass wir die Anforderungen an das System ändern. Architekten können sich zwar darauf einstellen, aber kein magisches Architekturmuster verhindert, dass eine Änderung des Kernproblems Auswirkungen auf die Architektur hat.

Kopplung der Implementierung

Die Implementierungskopplung beschreibt, wie ein Architekt und ein Team entscheiden, bestimmte Abhängigkeiten zu implementieren. Bei einer Anwendung zur Auftragsabwicklung muss das Team bei der Festlegung der Domänengrenzen eine Reihe von Einschränkungen berücksichtigen. Sollen z. B. alle Daten in einer einzigen Datenbank gespeichert werden, oder sollen einige davon zur besseren Skalierbarkeit oder Verfügbarkeit aufgeteilt werden? Sollen wir eine monolithische oder eine verteilte Architektur aufbauen? Die Antworten auf diese Fragen haben wenig Einfluss auf die semantische Kopplung des Systems, beeinflussen aber die Architekturentscheidungen erheblich.

Statische Kopplung

Die statische Kopplung bezieht sich auf die "Verdrahtung" einer Architektur, also darauf, wie die Dienste voneinander abhängen. Zwei Dienste sind Teil desselben Architekturquantums, wenn sie beide von demselben Kopplungspunkt abhängen. Nehmen wir zum Beispiel an, dass zwei Microservices, `Catalog` und `Shipping`, Adressinformationen austauschen müssen und deshalb eine Abhängigkeit zu einer gemeinsamen Komponente herstellen. Da beide Dienste an diese Abhängigkeit gekoppelt sind, sind sie Teil desselben Architekturquantums.

Es gibt einen einfachen Weg, um über Kopplung in der Softwarearchitektur nachzudenken: Zwei Dinge sind gekoppelt, wenn die Änderung des einen das andere zerstören könnte. Die statische Kopplung definiert die Abhängigkeiten im Rahmen einer Architektur. Wenn zum Beispiel mehrere Dienste dieselbe relationale Datenbank nutzen, sind sie Teil desselben Quantums.

Dynamische Kopplung
Dynamische Kopplung beschreibt die Kräfte, die auftreten, wenn Architekturquanten miteinander kommunizieren müssen. Wenn zum Beispiel zwei Dienste laufen, müssen sie miteinander kommunizieren, um Workflows zu bilden und Aufgaben innerhalb des Systems zu erfüllen. Architekten müssen die Kompromisse bei der Kommunikation zwischen Diensten in einer verteilten Architektur berücksichtigen, was in Kapitel 15 ausführlich behandelt wird.

Mit diesen Definitionen können wir unsere *Architektur-Quantum-Definition* vervollständigen:

Geringe statische Kopplung der externen Implementierung
Der Grad der Kopplung zwischen den Architekturquanten sollte gering sein. Dieses Merkmal leitet sich auch aus der DDD-Philosophie der geringen Kopplung zwischen begrenzten Kontexten ab. Quanten bilden die operativen Bausteine der Architektur, die eine Abstraktionsebene über den Komponenten liegen. Sie überschneiden sich oft mit den Grenzen von Diensten. Dieses Ziel spiegelt die allgemeine Vorliebe der Architekten für eine lose Kopplung zwischen verschiedenen Teilen der Architektur wider.

Im Allgemeinen ist eine enge Kopplung wünschenswert, wenn eine hohe Kohäsion erforderlich ist, z. B. innerhalb eines Dienstes oder Subsystems. Wir nennen eine Architektur "spröde", wenn eine einzige Implementierungsänderung unerwartete Nebeneffekte haben kann, die viele andere (scheinbar nicht miteinander verbundene) Dinge zerstören. Je breiter die Kopplung des Systems ist, desto mehr trägt die lose Kopplung dazu bei, die Architektur weniger brüchig zu machen. Ein Architekt könnte zum Beispiel ein Feld in einem Dienstaufruf von `State` in `StateCode` umbenennen, in der Annahme, dass dies nur einen Aufrufer betrifft, um dann festzustellen, dass viele andere Abhängigkeiten unerwartet unterbrochen wurden.

TIPP

Eine höhere Kopplung ist bei engeren Geltungsbereichen zulässig; je breiter der Geltungsbereich, desto lockerer sollte die Kopplung sein.

Synchrone Kommunikation

Kommunikation bezieht sich auf dynamische Kopplung - wenn Architekturquanten sich gegenseitig aufrufen, was in verteilten Architekturen häufig vorkommt. Dies betrifft vor allem die Familie der operativen Architektureigenschaften, die unter "Operative Architektureigenschaften" beschrieben werden , denn sie bestimmen oft wichtige Zeitabläufe und Blockierungen in verteilten Architekturen.

Nehmen wir zum Beispiel eine Microservices-Architektur mit einem `Payment` Service und einem `Auction` Service. Wenn eine Auktion endet, sendet der Dienst `Auction` die Zahlungsinformationen *synchron* an den Dienst `Payment`. Nehmen wir jedoch an, dass der `Payment` Dienst nur eine Zahlung alle 500 ms verarbeiten kann. Was passiert, wenn eine große Anzahl von Auktionen auf einmal endet? Die beiden Dienste haben unterschiedliche Architekturmerkmale. Der erste Aufruf wird funktionieren, aber dann werden die Aufrufe fehlschlagen, weil der `Payment` Dienst die Anzahl der Anfragen nicht bewältigen kann - er ist nicht so skalierbar wie der `Auction` Dienst.

Wir sprechen hier von synchroner Kommunikation, weil asynchrone Kommunikation weniger Auswirkungen haben kann. Wenn z. B. der Dienst `Auction` den Dienst `Payment` asynchron über eine Warteschlange aufruft, kann die Warteschlange als Puffer dienen, damit die beiden Systeme arbeiten können. Wenn der Dienst `Auction` ständig mehr Nachrichten sendet, als `Payment` verarbeiten kann, wird die Warteschlange natürlich irgendwann überlaufen. Wenn die

Nachrichtenflut jedoch in Schüben kommt, kann die Warteschlange die ausstehenden Nachrichten aufbewahren, bis der Empfänger bereit ist. Synchrone Kommunikation ist in verteilten Architekturen unnachgiebig, vor allem wenn Teile der Architektur unterschiedliche Eigenschaften haben. Kapitel 15 befasst sich ausführlich mit den Arten der Kommunikation in ereignisgesteuerten Architekturen.

Das Konzept des Architektur-Quantums bietet eine neue Möglichkeit, über den Umfang nachzudenken. In modernen Systemen definieren die Architekten die architektonischen Merkmale auf der Quantenebene und nicht auf der Systemebene. Dies liefert wichtige Informationen für die Analyse eines neuen Problembereichs.

Die Auswirkungen des Scoping

Architekten können den Umfang der architektonischen Merkmale nutzen, um geeignete Dienstgrenzen zu bestimmen, die im Entscheidungsbaum in Abbildung 7-1 dargestellt und in den Schritten weiter ausgeführt werden.

Architecture characteristics and domain analysis
Choose architecture characteristics
1 set of architecture characteristics...
>1 Set of architecture characteristics and/or domain constraints
1. Monolithic architecture
1. Distributed architecture
2. Choose quanta
2. Determine persistence
3. Determine persistence
Database
Database
Database
Database
Database
Database
4. Determine communication styles between quanta

Abbildung 7-1. Ein Entscheidungsbaum, der den Umfang der architektonischen Merkmale nutzt, um den Architekturstil zu bestimmen

Rahmen und architektonischer Stil

Die Bestimmung der Quantengrenzen der Problemdomäne hilft bei der Wahl eines Architekturstils: Ist eine monolithische oder eine verteilte Architektur am besten geeignet? Betrachte den ersten Teil des in Abbildung 7-2 dargestellten Entscheidungsdiagramms.

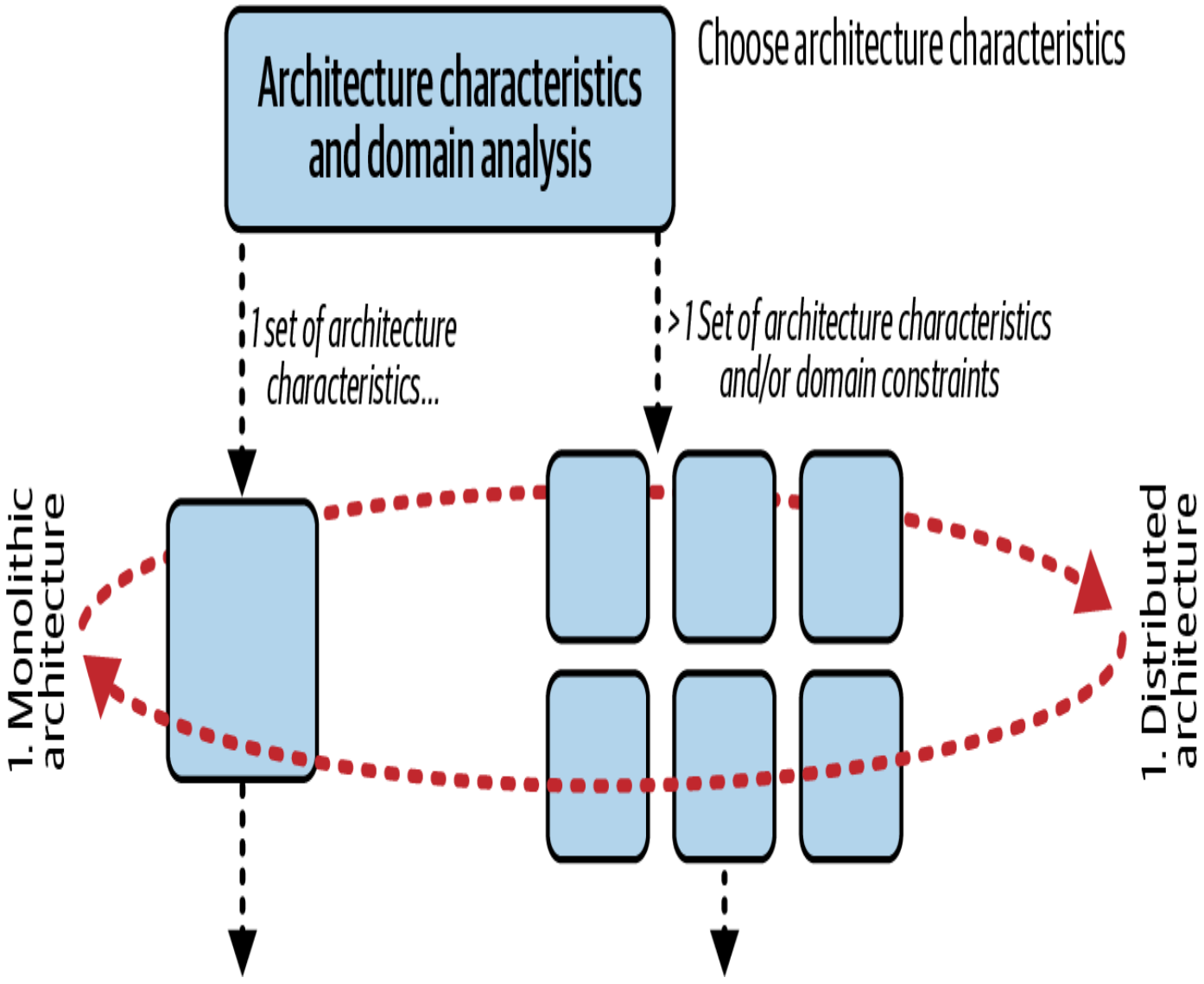

Abbildung 7-2. Auswahl eines geeigneten Baustils anhand der architektonischenMerkmale

Wie wir in Kapitel 4 besprochen haben, müssen Architekten die architektonischen Merkmale und die Domäne analysieren, um den am besten geeigneten Architekturstil zu bestimmen. Ein Teil dieser Analyse betrifft die Frage, ob die Lösung mehrere Gruppen von Architektureigenschaften benötigt. In Schritt eins in Abbildung 7-2 bestimmt der Architekt, ob das System mit einer einzigen Gruppe von Architekturmerkmalen erfolgreich sein kann oder ob mehr als eine Gruppe erforderlich ist (siehe "Kata: Going Green" für ein Beispiel).

Wenn der Architekt feststellt, dass ein einziger Satz von Architekturmerkmalen ausreicht, kann er sich für eine monolithische Architektur entscheiden, wodurch sich die Anzahl der nachfolgenden Auswahlmöglichkeiten verringert. Wenn er sich für eine verteilte Architektur entscheidet, muss er im nächsten Schritt die Quantengrenzen festlegen, wie in Abbildung 7-3 dargestellt.

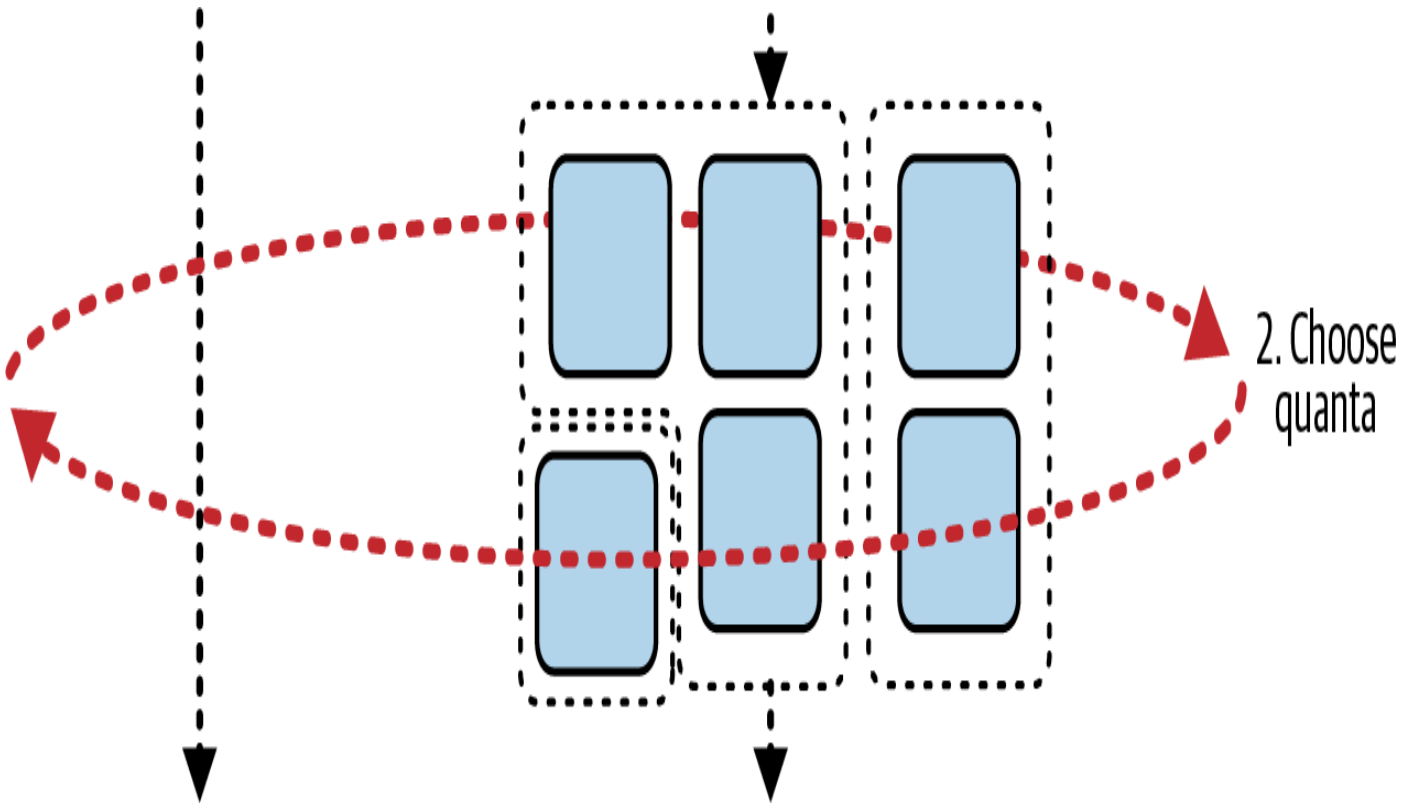

Abbildung 7-3. In verteilten Architekturen müssen die Architekten die geeigneten Quantengrenzen wählen

In Kapitel 18 finden Sie einige Richtlinien zur Bestimmung der Granularität. Der nächste Schritt ist die Wahl eines Persistenzmechanismus, der beide Architekturfamilien betrifft, wie in Abbildung 7-4 dargestellt.

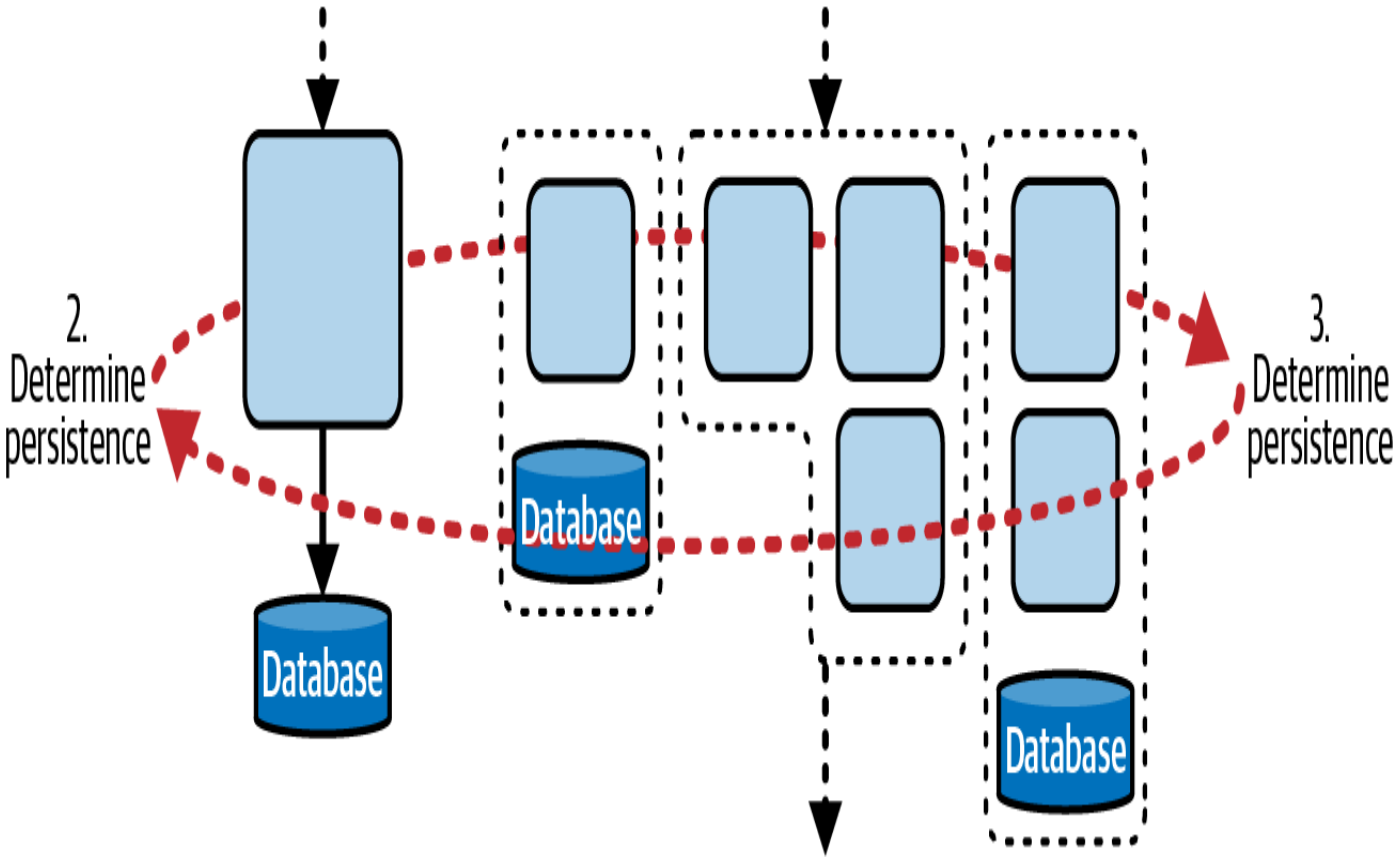

Abbildung 7-4. Beide Architekturstile erfordern im Allgemeinen eine Art von Persistenz

Für monolithische Architekturen ist in der Regel eine einzige monolithische Datenbank geeignet. Die Architektur und die Datenbank werden gemeinsam entwickelt und eingesetzt, im Gleichschritt. Damit ist der Prozess abgeschlossen - der Architekt kann sich nun für den am besten geeigneten monolithischen Stil entscheiden.

Verteilte Architekturen hingegen können eine einzige Datenbank verwenden (üblich in ereignisgesteuerten Architekturen) oder die Daten entsprechend der Granularität des Dienstes aufteilen (wie in Microservices-Architekturen). Es bleibt noch ein weiterer Schritt übrig: die Entscheidung, welche Art der Kommunikation zwischen den Quanten verwendet werden soll, synchron oder asynchron. (Wir besprechen diese Frage ausführlich in Kapitel 15.) Erinnere dich daran,

dass in manchen Systemen die Wahl der synchronen Kommunikation die durch statische Kopplung festgelegten Quantengrenzen verändern kann; die beiden Arten der Kopplung stehen häufig in Wechselwirkung.

Kata: Going Green

Versuchen wir, ein Problem zu analysieren, indem wir Architekturquantum als Rahmen für architektonische Merkmale verwenden. Das Problem mit dem Namen *Going Green* ist in Abbildung 7-5 dargestellt.

Du arbeitest mit Going Green (GG) zusammen, einem Unternehmen, das alte elektronische Geräte, wie z.B. Handys, recycelt und weiterverkauft. Das System besteht aus öffentlichen Kiosken und einer Website, die alle mit demselben System arbeiten. Ein Nutzer kann die Modellnummer und den Zustand seines Geräts hochladen, und GG gibt ein Gebot für das Gerät ab. Wenn der Nutzer das Gebot annimmt, kann er das Gerät im Kiosk abgeben, oder wenn er es über die Website kauft, schickt GG ihm einen Briefkasten, in den er es schicken kann. Nach Erhalt des Geräts bewertet GG es und schickt dem Nutzer seine Zahlung. GG schätzt dann den Wert des Geräts und recycelt oder verkauft es weiter. Sein System erstellt auch Berichte und andere Analysen.

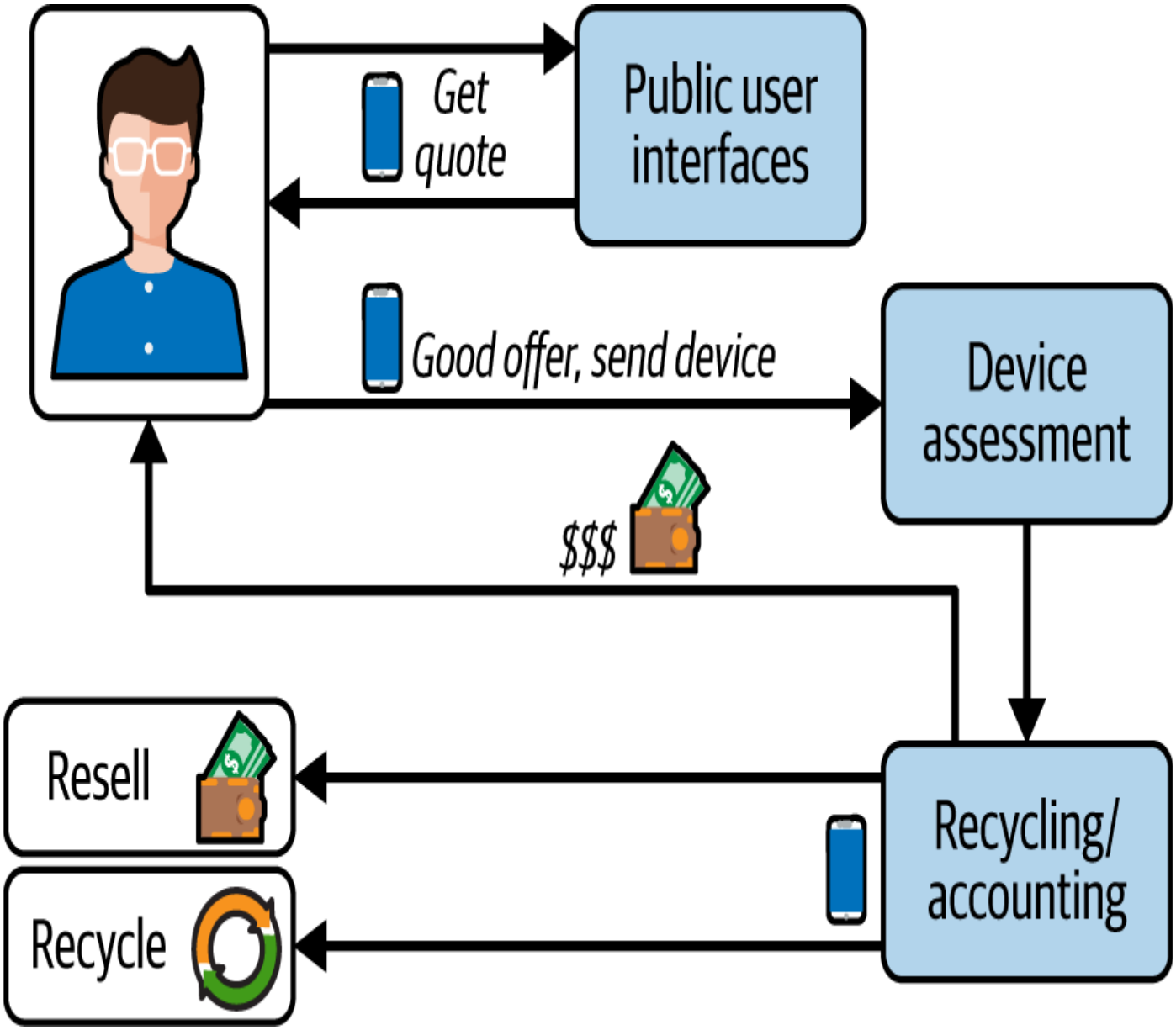

Abbildung 7-5. Übersicht der Anforderungen für Going Green

Während du deine Analyse der architektonischen Merkmale durchführst, stellst du fest, dass sich drei verschiedene Cluster bilden, wie in Abbildung 7-6 dargestellt.

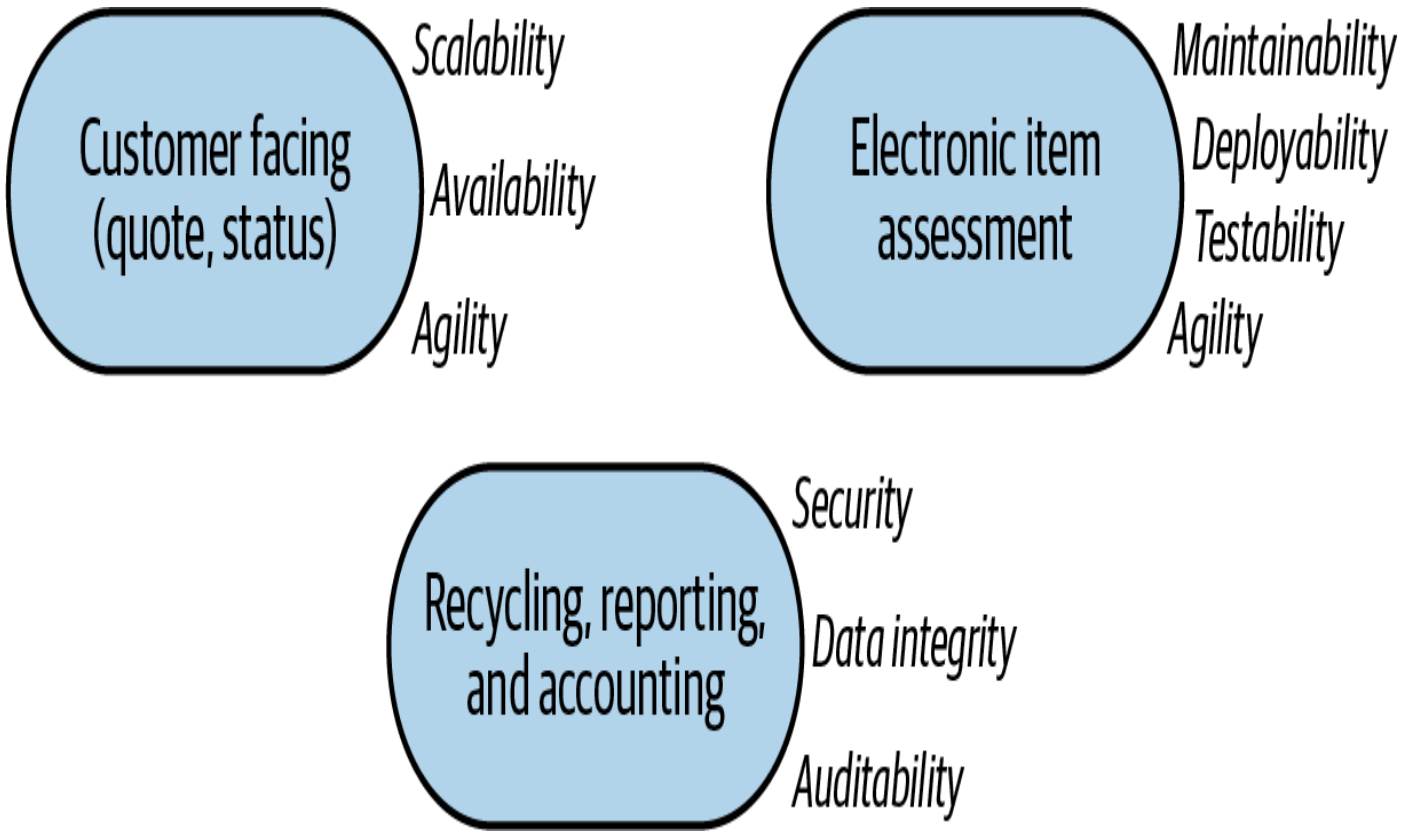

Abbildung 7-6. Die Analyse der GG-Architektureigenschaften ergibt drei Gruppen vonFähigkeiten

Die öffentlich zugänglichen Teile der Anwendung brauchen *Skalierbarkeit*, *Verfügbarkeit* und *Agilität*; die Back-Office-Funktionen brauchen *Sicherheit*, *Datenintegrität* und *Auditierbarkeit*; und der Bewertungsteil braucht *Wartbarkeit*, *Einsatzfähigkeit* und *Testbarkeit* (die das zusammengesetzte architektonische Merkmal der *Agilität* ausmachen). Warum braucht der Bewertungsteil einen eigenen Satz von Architekturmerkmalen? Dies ist ein gutes Beispiel dafür, wie sich geschäftliche Faktoren und architektonische Belange überschneiden. Das Geschäftsmodell von GG basiert auf dem Wiederverkauf der hochwertigsten gebrauchten Elektronikgeräte, und es kommen ständig neue Modelle auf den Markt. Je schneller sie ihre Gerätebewertungen

aktualisieren können, desto eher können sie neuere (und damit wertvollere) Geräte zum Wiederverkauf erhalten.

Könntest du ein System entwerfen, das all diese Kriterien erfüllt: Skalierbarkeit, Verfügbarkeit, Sicherheit, Datenintegrität, Auditierbarkeit, Wartbarkeit, Einsatzfähigkeit und Testbarkeit? Es ist möglich... aber es wäre schwierig. Diese architektonischen Merkmale stehen oft im Widerspruch zueinander: So ist es zum Beispiel schwieriger, eine schnelle Einsatzfähigkeit zu erreichen, wenn du gleichzeitig Back-Office-Belange wie die Auditierbarkeit in den Vordergrund stellst. Und die Benutzeroberfläche erfordert ein ganz anderes Maß an Skalierbarkeit als andere Teile des Systems.

Anstatt zu versuchen, alles zu tun, kannst du die Cluster der architektonischen Merkmale als Leitfaden für die Trennung der Quanten verwenden. In Abbildung 7-7 veranschaulichen die gestrichelten Linien jedes Architekturquantum. Die Verwendung des Merkmalsbereichs als Leitfaden für die Granularität der Dienste ist ein guter erster Schritt, um die vorteilhaftesten Kompromisse zu ermitteln. (Auf dieses Thema gehen wir in Kapitel 18 näher ein.)

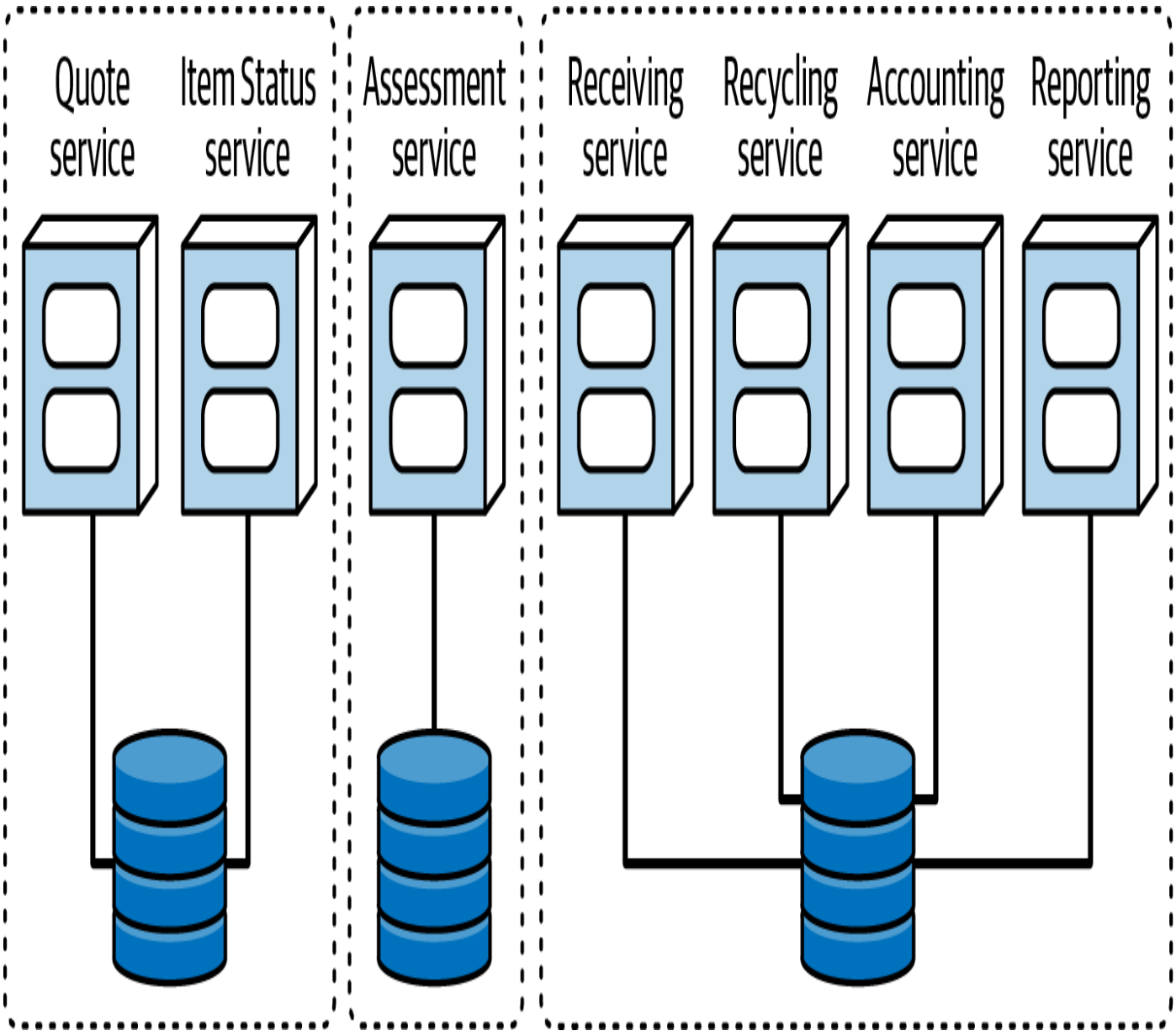

Abbildung 7-7. Diese GG-Architektur fasst jeden Satz von Architekturmerkmalen als Architekturgrenze zusammen

Scoping und die Cloud

Cloud-basierte Ressourcen verkomplizieren das Bild, weil sie so viele der operativen architektonischen Merkmale eines Systems einkapseln. Je nach Bereitstellungsmodell müssen Architekten mindestens zwei

Szenarien in Betracht ziehen, wenn sie Cloud-basierte Ressourcen für die gesamte oder einen Teil einer Anwendung nutzen:

Die Cloud als Host für Container nutzen
Viele Entwicklungsteams nutzen die Cloud als alternatives Betriebszentrum, in dem sie Container für Server betreiben (und orchestrieren). In solchen Situationen müssen die Architekten die architektonischen Merkmale der Container und die Einschränkungen durch das Orchestrierungstool (z. B. Kubernetes) berücksichtigen.

Nutzung von Cloud-Provider-Ressourcen als Systemkomponenten
Bei einer anderen Variante eines Cloud-basierten Systems werden die Anwendungen aus den Bausteinen des Cloud-Providers zusammengesetzt, z. B. aus ausgelösten Funktionen, Datenbanken und so weiter. In diesem Fall müssen die Architekten die Fähigkeiten, die der Anbieter anbietet (und hoffentlich auch pflegt), prüfen, um herauszufinden, wie sie für den jeweiligen Kontext geeignete Fähigkeiten entwickeln können.

Viele der Fähigkeiten, die wir heute als Konfigurationseinstellungen der Cloud-Provider kennen, wie z. B. Elastizität, wurden von der vorherigen Generation von Architekten, die mit physischen Systemen arbeiteten, hart erkämpft. Ihre Hauptanliegen sind einfacher geworden - aber wir haben unsere eigenen modernen Kompromisse, wie z. B. die Verfügbarkeit der Anbieter und erhöhte Sicherheitsbedenken. Die Details der Softwarearchitektur ändern sich ständig, aber die Aufgabe, Kompromisse zu analysieren, bleibt gleich.