Kapitel 21. Architektonische Entscheidungen

Eine der wichtigsten Erwartungen an einen Architekten ist es, architektonische Entscheidungen zu treffen. Architekturentscheidungen betreffen in der Regel die Struktur der Anwendung oder des Systems, aber sie können auch Technologieentscheidungen beinhalten, insbesondere wenn diese Technologieentscheidungen architektonische Merkmale beeinflussen. Unabhängig vom Kontext ist eine gute architektonische Entscheidung eine, die den Entwicklungsteams hilft, die richtigen technischen Entscheidungen zu treffen. Um architektonische Entscheidungen zu treffen, muss man genügend relevante Informationen sammeln, die Entscheidung begründen, sie dokumentieren und sie den richtigen Stakeholdern effektiv mitteilen.

Antipatterns für architektonische Entscheidungen

Der Programmierer Andrew Koenig definiert ein *Antipattern* als etwas, das am Anfang wie eine gute Idee aussieht, dich aber in Schwierigkeiten bringt. Eine andere Definition eines Antipatterns ist ein wiederholbarer

Prozess, der zu negativen Ergebnissen führt. Die drei häufigsten Antipattern bei Architekturentscheidungen, die entstehen können (und in der Regel auch entstehen), wenn ein Architekt Entscheidungen trifft, sind das Covering Your Assets Antipattern, das Groundhog Day Antipattern und das Email-Driven Architecture Antipattern. Diese drei Muster folgen in der Regel einem progressiven Fluss: Die Überwindung des "Covering Your Assets"-Musters führt zum "Groundhog Day"-Muster und die Überwindung dieses Musters führt zum "Email-Driven Architecture"-Muster. Um effektive und genaue Architekturentscheidungen zu treffen, müssen alle drei überwunden werden.

Das Antipattern "Covering Your Assets

Das Antipattern "Covering Your Assets" tritt auf, wenn ein Architekt eine architektonische Entscheidung vermeidet oder aufschiebt, weil er Angst hat, die falsche Entscheidung zu treffen. Es gibt zwei Möglichkeiten, dies zu überwinden. Die erste besteht darin, mit einer wichtigen architektonischen Entscheidung bis zum *letzten verantwortlichen Moment* zu warten: Das heißt, wenn genügend Informationen vorliegen, um die Entscheidung zu rechtfertigen und zu validieren, aber nicht so lange, dass es die Entwicklungsteams aufhält oder den Architekten in die *Analyse-Lähmung* treibt, wo er für immer in der Analyse der Entscheidung stecken bleibt. Ein guter Weg, um den letzten verantwortlichen Moment zu bestimmen, ist die Frage, wann die Kosten für das Aufschieben der Entscheidung das mit der Entscheidung verbundene Risiko übersteigen. In Abbildung 21-1 siehst du, dass die

Kosten (dargestellt durch die durchgezogene Linie) in den frühen Phasen der Entscheidungsskala niedrig sind, weil weniger Zeit für die Entscheidung aufgewendet wird, aber das Risiko (dargestellt durch die gestrichelte Linie) hoch ist, weil weniger über das Problem oder die Lösung bekannt ist. Ein längerer Aufschub der Entscheidung erhöht die Kosten, verringert aber auch das Risiko, weil der Architekt eine vollständigere Analyse des Problems und der möglichen Alternativen vornehmen kann. Die Zeit bis zur Entscheidung ist der Punkt, an dem sich diese beiden Faktoren überschneiden und die Kostensteigerung das geringere Risiko übersteigt.

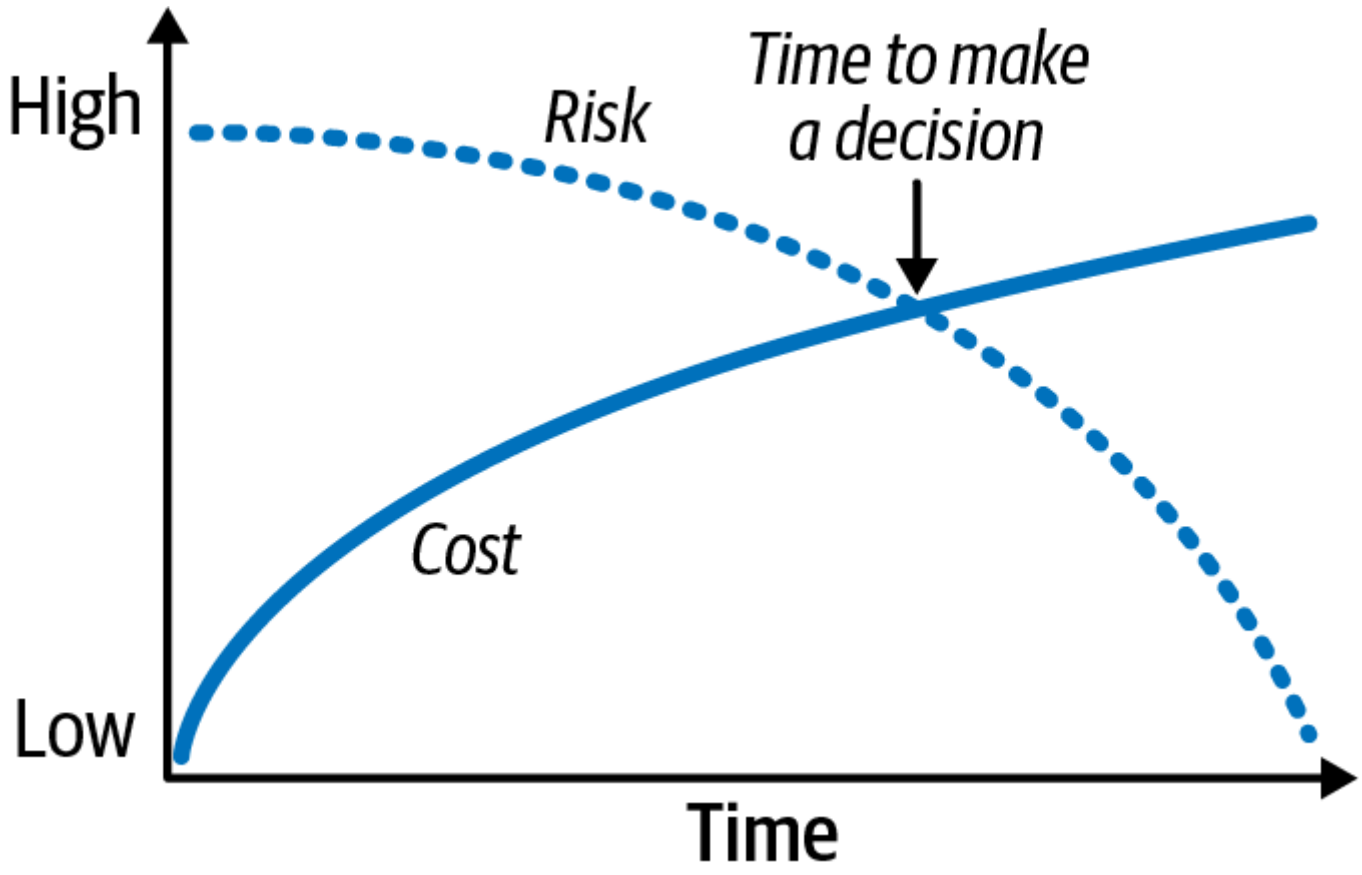

Abbildung 21-1. Letzter verantwortlicher Moment

Eine weitere Möglichkeit, dieses Muster zu vermeiden, besteht darin, mit den Entwicklungsteams zusammenzuarbeiten, um sicherzustellen, dass die Entscheidung wie erwartet umgesetzt werden kann. Das ist von entscheidender Bedeutung, denn kein Architekt kann jedes Detail über ein Problem im Zusammenhang mit einer bestimmten Technologie kennen. Durch die enge Zusammenarbeit mit den Entwicklungsteams kann der Architekt schnell reagieren, mehr Erkenntnisse gewinnen und das Risiko einer falschen Entscheidung verringern.

Zur Veranschaulichung: Angenommen, du entscheidest als Architekt, dass alle produktbezogenen Referenzdaten (wie Produktbeschreibung, Gewicht und Abmessungen) in allen Service-Instanzen, die diese Informationen benötigen, zwischengespeichert werden sollen. Du entscheidest dich für einen replizierten Cache mit Lesezugriff, wobei der primäre Cache dem Dienst `Catalog` gehören soll. (Ein *replizierter* oder *speicherinterner* Cache bedeutet, dass der Dienst `Catalog` seinen Cache aktualisiert, wenn Änderungen an den Produktinformationen vorgenommen oder neue Produkte hinzugefügt werden, die dann über ein repliziertes Cache-Produkt an alle anderen Dienste, die diese Daten benötigen, weitergegeben werden). Du begründest diese Entscheidung damit, dass die Kopplung zwischen den Diensten reduziert werden soll und die Daten effektiv gemeinsam genutzt werden können, ohne dass ein Interservice-Aufruf erforderlich ist. Die Entwicklungsteams, die diese architektonische Entscheidung umsetzen, stellen jedoch fest, dass diese Entscheidung aufgrund der Skalierbarkeitsanforderungen einiger Dienste mehr prozessinternen Speicher erfordern würde, als zur Verfügung steht. Da du eng mit diesen Teams zusammenarbeitest, wirst

du schnell auf das Problem aufmerksam und passt deine Architekturentscheidung entsprechend an.

Murmeltiertag Antipattern

Das Murmeltiertag-Antipattern tritt auf, wenn Menschen nicht wissen, warum ein Architekt eine bestimmte Entscheidung getroffen hat, also diskutieren sie immer und immer wieder darüber und kommen nie zu einer endgültigen Lösung oder Einigung. Der Name stammt aus dem Film *"Groundhog Day"* von 1993, in dem Bill Murrays Figur den 2. Februar jeden Tag aufs Neue erleben muss.

Dieses Muster tritt auf, weil Architekten es versäumen, ihre Entscheidung zu begründen (oder sie nicht vollständig zu begründen). Es ist wichtig, eine architektonische Entscheidung sowohl technisch als auch geschäftlich zu rechtfertigen.

Nehmen wir an, du entscheidest dich, eine monolithische Anwendung in einzelne Dienste aufzuteilen. Du begründest dies damit, dass die funktionalen Aspekte der Anwendung entkoppelt werden sollen, damit jeder Teil der Anwendung weniger Ressourcen für virtuelle Maschinen benötigt und separat gewartet und bereitgestellt werden kann. Das ist zwar eine gute technische Begründung, aber was fehlt, ist die *geschäftliche* Begründung - mit anderen Worten, warum sollte das Unternehmen für diese architektonische Umgestaltung bezahlen? Eine gute geschäftliche Begründung für diese Entscheidung könnte darin bestehen, neue Geschäftsfunktionen schneller bereitzustellen und so die

Markteinführung zu beschleunigen. Ein anderer Grund könnte sein, die Kosten für die Entwicklung und Freigabe neuer Funktionen zu senken.

Die Angabe des Geschäftswerts ist von entscheidender Bedeutung, wenn es darum geht, eine Architekturentscheidung zu rechtfertigen. Es ist auch ein guter Lackmustest, um festzustellen, ob die Architekturentscheidung überhaupt getroffen werden sollte. Wenn sie keinen geschäftlichen Nutzen bringt, sollte der Architekt seine Entscheidung vielleicht noch einmal überdenken.

Vier der häufigsten geschäftlichen Rechtfertigungen sind Kosten, Markteinführungszeit, Nutzerzufriedenheit und strategische Positionierung. Überlege, was für die Interessengruppen wichtig ist. Eine bestimmte Entscheidung allein mit Kosteneinsparungen zu rechtfertigen, ist vielleicht nicht die richtige Entscheidung, wenn die Interessengruppen mehr Wert auf die Markteinführungszeit legen.

Antipattern der E-Mail-gesteuerten Architektur

Sobald ein Architekt seine Entscheidungen getroffen und begründet hat, taucht oft ein weiteres Architektur-Antimuster auf: E-Mail-gesteuerte Architektur. Dieses Antipattern tritt auf, wenn Menschen eine architektonische Entscheidung verlieren oder vergessen oder gar nicht wissen, dass sie getroffen wurde und sie deshalb unmöglich umsetzen können. Um dieses Verhaltensmuster zu überwinden, muss man Architekturentscheidungen effektiv kommunizieren. E-Mails sind ein

großartiges Kommunikationsmittel, aber ein schlechtes Dokumentenspeichersystem.

Glücklicherweise kann ein Architekt das Antipattern der E-Mail-getriebenen Architektur leicht vermeiden, indem er lernt, wie er architektonische Entscheidungen effektiv kommuniziert. Erstens solltest du darauf achten, dass du die Entscheidung nicht in den Text einer E-Mail schreibst. Dadurch wird die Entscheidung in mehreren Systemen gespeichert, weil jede E-Mail eine Kopie der Entscheidung enthält, anstatt sie nur an einer Stelle zu haben. In vielen dieser E-Mails werden wichtige Details der Entscheidung (einschließlich der Begründung) ausgelassen, wodurch das Murmeltiertag-Muster erneut auftritt. Wenn die architektonische Entscheidung jemals geändert oder ersetzt wird, ist es außerdem schwierig zu wissen, ob alle betroffenen Personen die überarbeitete Entscheidung erhalten haben.

Ein besserer Ansatz ist es, nur die Art und den Kontext der Entscheidung im Text der E-Mail zu erwähnen und einen Link zu dem System anzugeben, in dem die architektonische Entscheidung und die entsprechenden Details gespeichert sind (sei es ein Link zu einer Wiki-Seite oder ein Verweis auf ein Dokument in einem Dateisystem).

Betrachte die folgende E-Mail über eine architektonische Entscheidung:

> *"Hallo Sandra, ich habe eine wichtige Entscheidung bezüglich der Kommunikation zwischen den Dienststellen getroffen, die sich direkt auf dich auswirkt. Du kannst die Entscheidung unter folgendem Link einsehen: ...."*

Beachte, dass in der Formulierung am Anfang des ersten Satzes der Kontext erwähnt wird (Kommunikation zwischen Diensten), aber nicht die eigentliche Entscheidung selbst. Auch der zweite Teil des ersten Satzes ist wichtig: Wenn sich eine architektonische Entscheidung nicht direkt auf die Person auswirkt, warum sollte man diese Person dann damit belästigen? Dies ist ein guter Lackmustest, um zu bestimmen, welche Stakeholder (einschließlich der Entwickler) direkt über eine Architekturentscheidung informiert werden sollten. Der zweite Satz in diesem Beispiel enthält einen Link zu dem Ort, an dem die architektonische Entscheidung getroffen wurde, so dass die Entscheidungen in einem einzigen System erfasst werden können.

Architektonische Bedeutung

Viele Architekten glauben, dass eine Entscheidung, die eine bestimmte Technologie beinhaltet, keine architektonische Entscheidung ist, sondern eine technische. Das ist nicht immer richtig. Wenn sich ein Architekt für eine bestimmte Technologie entscheidet, weil sie ein bestimmtes Merkmal der Architektur direkt unterstützt (z. B. Leistung oder Skalierbarkeit), dann ist es immer noch eine architektonische Entscheidung.

Michael Nygard, ein bekannter Softwarearchitekt und Autor der zweiten Ausgabe von *Release It!* (Pragmatic Bookshelf, 2018), befasst sich mit dem Problem, für welche Entscheidungen ein Architekt verantwortlich sein sollte (und was somit eine architektonische Entscheidung ist), indem er den Begriff " *architektonisch bedeutsam*" prägt. Laut Nygard

sind architektonisch bedeutsame Entscheidungen solche, die die Struktur eines Systems, nichtfunktionale Merkmale, Abhängigkeiten, Schnittstellen oder Konstruktionstechniken betreffen.

Struktur bezieht sich in diesem Zusammenhang auf Entscheidungen, die sich auf die verwendeten Architekturmuster oder -stile auswirken. Die Entscheidung eines Architekten, Code zwischen einer Reihe von Microservices zu teilen, wirkt sich zum Beispiel auf den begrenzten Kontext des Microservices und damit auf die Struktur des Systems aus.

Die *nicht-funktionalen Merkmale* des Systems sind die architektonischen Merkmale, die für das zu entwickelnde oder zu wartende System wichtig sind. Wenn sich zum Beispiel die Wahl der Technologie auf die Leistung auswirkt und die Leistung ein wichtiger Aspekt der Anwendung ist, wird diese Wahl zu einer architektonischen Entscheidung, auch wenn sie ein bestimmtes Produkt, Framework oder eine bestimmte Technologie vorgibt.

Abhängigkeiten beziehen sich auf die Kopplungspunkte zwischen Komponenten und/oder Diensten innerhalb des Systems. Abhängigkeiten können sich auf Architekturmerkmale wie Skalierbarkeit, Modularität, Agilität, Testbarkeit, Zuverlässigkeit usw. auswirken, so dass Entscheidungen über Abhängigkeiten zu Architekturentscheidungen werden.

Schnittstellen beziehen sich darauf, wie auf Dienste und Komponenten zugegriffen und diese orchestriert werden: in der Regel über ein Gateway, einen Integration Hub, einen Service Bus, einen Adapter oder

einen API-Proxy. Um Entscheidungen über Schnittstellen zu treffen, müssen in der Regel Verträge definiert werden, die auch Strategien für die Versionierung und das Auslaufen von Schnittstellen beinhalten. Schnittstellen haben Auswirkungen auf andere Nutzer des Systems und sind daher von architektonischer Bedeutung.

Die *Konstruktionstechniken* schließlich beziehen sich auf Entscheidungen über Plattformen, Frameworks, Werkzeuge und sogar Prozesse, die zwar technischer Natur sind, sich aber auf einen Aspekt der Architektur auswirken können.

Aufzeichnungen über architektonische Entscheidungen

Eine der effektivsten Möglichkeiten, architektonische Entscheidungen zu dokumentieren, sind die *Architectural Decision Records*(ADRs). Michael Nygard propagierte ADRs erstmals 2011 in einem Blogbeitrag und 2017 empfahl das Thoughtworks Technology Radar die Technik für eine breite Anwendung.

Ein ADR besteht aus einer kurzen Textdatei (normalerweise ein bis zwei Seiten lang), die eine bestimmte architektonische Entscheidung beschreibt. ADRs können als reiner Text oder in einer Wiki-Seitenvorlage verfasst werden, aber normalerweise werden sie in einem Textdokumentenformat wie AsciiDoc oder Markdown geschrieben.

Auch für die Verwaltung von ADRs gibt es Hilfsmittel. Nat Pryce, Mitautor von *Growing Object-Oriented Software, Guided by Tests* (Addison-Wesley, 2009), hat ein Open-Source-Tool namens ADR Tools geschrieben, das eine Befehlszeilenschnittstelle für die Verwaltung von ADRs bietet, einschließlich Nummerierungsschemata, Speicherorten und überholter Logik. Micha Kops, ein Softwareentwickler aus Deutschland, liefert einige großartige Beispiele für die Verwendung von ADR-Tools zur Verwaltung von Architekturentscheidungsdatensätzen.

Grundlegende Struktur

Die Grundstruktur eines ADRs besteht aus fünf Hauptabschnitten: *Titel*, *Status*, *Kontext*, *Entscheidung* und *Konsequenzen*. In der Regel fügen wir der Grundstruktur zwei weitere Abschnitte hinzu: *Einhaltung* und *Anmerkungen*. Im Abschnitt "Einhaltung" kannst du dir Gedanken darüber machen und dokumentieren, wie die architektonische Entscheidung geregelt und durchgesetzt werden soll (manuell oder durch automatische Fitnessfunktionen). Im Abschnitt Notizen kannst du Metadaten über die Entscheidung angeben, z. B. den Autor, wer sie genehmigt hat, wann sie erstellt wurde usw.

Es ist kein Problem, diese Grundstruktur (wie in Abbildung 21-2 dargestellt) zu erweitern, um andere benötigte Abschnitte hinzuzufügen. Achte nur darauf, dass die Vorlage einheitlich und übersichtlich ist. Ein gutes Beispiel dafür wäre ein Abschnitt über *Alternativen*, in dem alle anderen möglichenLösungen analysiert werden.

ADR Format

TITLE

Short description stating the architecture decision

STATUS

Proposed, Accepted, Superseded

CONTEXT

What is forcing me to make this decision?

DECISION

The decision and corresponding justification

CONSEQUENCES

What is the impact of this decision?

COMPLIANCE

How will I ensure compliance with this decision?

NOTES

Metadata for this decision (author, etc.)

Abbildung 21-2. Grundlegende ADR-Struktur

Titel

ADR-Titel sind in der Regel fortlaufend nummeriert und enthalten einen kurzen Satz, der die architektonische Entscheidung beschreibt. Ein ADR-Titel, der die Entscheidung zur Verwendung asynchroner Nachrichten zwischen dem Dienst `Order` und dem Dienst `Payment` beschreibt, könnte zum Beispiel lauten: "42. Verwendung von asynchronen Nachrichten zwischen Bestell- und Zahlungsdiensten". Der Titel sollte kurz und prägnant, aber aussagekräftig genug sein, um jede Unklarheit über die Art und den Kontext der Entscheidung zu beseitigen.

Status

Jedes ADR hat einen von drei Status: *Vorgeschlagen*, *Angenommen* oder *Aufgehoben*. Der Status "Vorgeschlagen" bedeutet, dass die Entscheidung von einem übergeordneten Entscheidungsträger oder einem architektonischen Führungsgremium (z. B. einem Architekturprüfungsausschuss) genehmigt werden muss. Angenommen bedeutet, dass die Entscheidung genehmigt wurde und zur Umsetzung bereit ist. Ersetzt bedeutet, dass die Entscheidung geändert und durch ein anderes ADR ersetzt wurde. Der Status "Ersetzt" setzt immer voraus, dass der vorherige ADR-Status "Angenommen" war. Mit anderen Worten: Ein ADR-Vorschlag würde nie durch einen anderen ADR ersetzt werden, sondern bis zur Annahme geändert werden.

Der Status "Ersetzt" ist eine gute Möglichkeit, historische Aufzeichnungen darüber zu führen, welche Entscheidungen getroffen wurden, warum sie damals getroffen wurden, wie die neue Entscheidung lautet und warum sie geändert wurde. Wenn eine Entscheidung ersetzt wurde, wird sie normalerweise mit der Nummer der Entscheidung gekennzeichnet, die sie ersetzt hat. Ebenso wird die Entscheidung, die ein anderes ADR ersetzt, mit der Nummer des ADR gekennzeichnet, das sie ersetzt hat.

Nehmen wir zum Beispiel an, ADR 42 ("Verwendung asynchroner Nachrichten zwischen Bestell- und Zahlungsdiensten") hat den Status "Genehmigt". Aufgrund späterer Änderungen an der Implementierung und dem Standort des Dienstes `Payment` entscheidest du, dass nun REST zwischen den beiden Diensten verwendet werden soll. Du erstellst daher ein neues ADR (Nummer 68), um diese geänderte Entscheidung zu dokumentieren. Die entsprechenden Status sehen wie folgt aus:

> *ADR 42. Verwendung von asynchronen Nachrichten zwischen Auftrags- und Zahlungsdiensten*
>
> *Status: Ersetzt durch 68*
>
> *ADR 68. Verwendung von REST zwischen Bestell- und Zahlungsdiensten*
>
> *Status: Angenommen, ersetzt 42*

Durch die Verknüpfung und den Verlauf zwischen ADR 42 und 68 kannst du die unvermeidliche Frage "Was ist mit der Verwendung von Nachrichten?" bei ADR 68 vermeiden.

ADRS UND REQUEST FOR COMMENTS (RFC)

Das Versenden eines ADR-Entwurfs oder von Kommentaren kann einem Architekten dabei helfen, seine Annahmen und Behauptungen bei einem größeren Kreis von Interessengruppen zu überprüfen. Ein effektiver Weg, um Entwickler einzubinden und eine Zusammenarbeit zu initiieren, ist die Erstellung eines neuen Status, der *Request for Comments* (RFC) genannt wird, und die Festlegung einer Frist, innerhalb derer die Prüfer ihr Feedback abgeben können. Sobald dieses Datum erreicht ist, kann der Architekt die Kommentare analysieren, alle notwendigen Anpassungen an der Entscheidung vornehmen, die endgültige Entscheidung treffen und den Status auf Vorgeschlagen (oder Angenommen, wenn der Architekt die Befugnis hat, die Entscheidung zu genehmigen) setzen.

Ein ADR mit RFC-Status würde folgendermaßen aussehen:

STATUS
Aufforderung zur Stellungnahme, Frist 09 JAN 2026

Ein weiterer wichtiger Aspekt des Abschnitts "Status" eines ADR ist, dass er den Architekten und seinen Chef oder leitenden Architekten dazu zwingt, die Kriterien für die Genehmigung einer architektonischen

Entscheidung zu besprechen und festzulegen, ob der Architekt diese Entscheidung allein treffen kann oder ob sie von einem übergeordneten Architekten, einem Prüfungsausschuss für Architektur oder einem anderen Gremium genehmigt werden muss.

Drei gute Ausgangspunkte für diese Gespräche sind Kosten, teamübergreifende Auswirkungen und Sicherheit. Die Kosten sollten den Kauf von Software oder Lizenzen, zusätzliche Hardwarekosten und den Gesamtaufwand für die Umsetzung der Architekturentscheidung umfassen. Um diesen zu schätzen, multiplizierst du die geschätzte Anzahl der Stunden für die Umsetzung der Architekturentscheidung mit dem *Standard-Vollzeitäquivalenzsatz* (VZÄ) deines Unternehmens. Der Projektverantwortliche oder Projektmanager hat in der Regel den VZÄ-Wert. Bei diesem Gespräch können sich alle darauf einigen, dass z. B. die Kosten für die architektonische Entscheidung, wenn sie einen bestimmten Betrag überschreiten, in den Status "Vorgeschlagen" gesetzt und von jemand anderem genehmigt werden müssen. Wenn die architektonische Entscheidung Auswirkungen auf andere Teams oder Systeme hat oder in irgendeiner Weise sicherheitsrelevant ist, muss sie von einem übergeordneten Gremium oder dem leitenden Architekten genehmigt werden.

Sobald sich das Team auf die Kriterien und die entsprechenden Grenzen geeinigt hat (z. B. "Kosten, die 5.000 USD übersteigen, müssen vom Prüfungsausschuss für Architektur genehmigt werden"), solltest du dies gut dokumentieren, damit alle Architekten, die ADRs erstellen, wissen,

wann sie ihre eigenen architektonischen Entscheidungen genehmigen können und wann nicht.

Kontext

Der Abschnitt "Kontext" eines ADR legt fest, welche Kräfte im Spiel sind. Mit anderen Worten: "Welche Situation zwingt mich dazu, diese Entscheidung zu treffen?" Dieser Abschnitt des ADR ermöglicht es dem Architekten, die besonderen Umstände zu beschreiben und die möglichen Alternativen kurz und bündig zu erläutern. Wenn der Architekt die Analyse jeder Alternative im Detail dokumentieren muss, füge einen Abschnitt über die Alternativen hinzu, anstatt diese Analyse in den Abschnitt "Kontext" aufzunehmen.

Der Abschnitt "Kontext" bietet auch einen Platz, um einen bestimmten Bereich der Architektur selbst zu dokumentieren. Indem der Architekt den Kontext beschreibt, beschreibt er auch die Architektur. Anhand des Beispiels aus dem vorherigen Abschnitt könnte der Abschnitt Kontext wie folgt lauten: "Der Bestelldienst muss Informationen an den Bezahldienst weitergeben, um eine Bestellung zu bezahlen, die gerade aufgegeben wird. Dies kann über REST oder asynchrones Messaging geschehen." Beachte, dass diese knappe Aussage nicht nur das Szenario, sondern auch die in Betracht gezogenen Alternativen beschreibt.

Entscheidung

Der Abschnitt Entscheidung des ADR enthält eine Beschreibung der architektonischen Entscheidung und eine ausführliche Begründung.

Nygard empfiehlt, architektonische Entscheidungen nicht im Passiv zu formulieren, sondern in einem affirmativen, befehlenden Ton. Die Entscheidung, asynchrone Nachrichtenübermittlung zwischen den Diensten zu verwenden, würde zum Beispiel lauten: "*Wir werden* asynchrone Nachrichtenübermittlung zwischen den Dienste*nverwenden* ". Das ist viel besser als "*Ich denke, dass* asynchrone Nachrichtenübermittlung zwischen den Diensten die beste Wahl wäre", aus der nicht hervorgeht, was die Entscheidung ist oder ob überhaupt eine Entscheidung getroffen wurde - nur die Meinung des Architekten.

Einer der wichtigsten Aspekte des Abschnitts "Entscheidung" in ADRs ist, dass der Architekt die Begründung für die Entscheidung hervorheben kann. Zu verstehen, *warum* eine Entscheidung getroffen wurde, ist viel wichtiger als zu verstehen , *wie* etwas funktioniert. Dies hilft Entwicklern und anderen Interessengruppen, die Gründe für eine Entscheidung besser zu verstehen, und macht es daher wahrscheinlicher, dass sie ihr zustimmen.

Zur Veranschaulichung: Nehmen wir an, du entscheidest dich, den Remote Procedure Call(gRPC) von Google für die Kommunikation zwischen zwei bestimmten Diensten zu verwenden, um die Netzwerklatenz zu verringern, weil du eine sehr hohe Reaktionsfähigkeit brauchst. Einige Jahre später beschließt ein neuer Architekt im Team, REST anstelle von gRPC zu verwenden, um die Kommunikation zwischen den Diensten einheitlicher zu gestalten. Da der neue Architekt nicht versteht , *warum* er sich für gRPC entschieden hat, wirkt sich seine Entscheidung erheblich auf die Latenz aus und führt

zu Timeouts in vorgelagerten Systemen. Hätte der neue Architekt Zugang zu einem ADR gehabt, hätte er verstanden, dass die ursprüngliche Entscheidung für gRPC dazu diente, die Latenzzeit zu verringern (auf Kosten von eng gekoppelten Diensten) und hätte dieses Problem verhindern können.

Konsequenzen

Jede Entscheidung, die ein Architekt trifft, hat irgendeine Auswirkung, ob gut oder schlecht. Der Abschnitt "Konsequenzen" eines ADR zwingt einen Architekten dazu, die Gesamtauswirkungen einer architektonischen Entscheidung zu beschreiben, damit er darüber nachdenken kann, ob die negativen Auswirkungen die Vorteile überwiegen.

Dieser Abschnitt ist auch ein guter Ort, um die Analyse der Kompromisse zu dokumentieren, die während des Entscheidungsprozesses durchgeführt wurden. Nehmen wir zum Beispiel an, du entscheidest dich, asynchrone Nachrichten (fire-and-forget) für die Veröffentlichung von Bewertungen auf einer Website zu verwenden. Du begründest diese Entscheidung damit, dass du die Reaktionszeit verbessern willst (von 3.100 Millisekunden auf 25 Millisekunden), weil die Nutzer/innen dann nicht mehr darauf warten müssen, dass die Rezension tatsächlich veröffentlicht wird, sondern nur noch darauf, dass die Nachricht an eine Warteschlange gesendet wird. Ein Mitglied deines Entwicklungsteams argumentiert, dass dies eine schlechte Idee ist, weil die Fehlerbehandlung bei einer asynchronen Anfrage so komplex ist: "Was

passiert, wenn jemand eine Rezension mit bösen Worten schreibt?" Was dieses Teammitglied nicht weiß, ist, dass ihr genau dieses Problem mit den Business Stakeholdern und anderen Architekten diskutiert habt, als ihr die Kompromisse dieser Entscheidung analysiert habt, und gemeinsam entschieden habt, dass es besser ist, die Reaktionsfähigkeit zu verbessern und mit der komplexen Fehlerbehandlung umzugehen, als die Wartezeit zu verlängern und eine Rückmeldung zu geben, ob die Bewertung erfolgreich war oder nicht. Wenn diese Entscheidung mit einem ADR dokumentiert worden wäre, hättest du diese Analyse der Kompromisse in den Abschnitt Konsequenzen aufnehmen können, um diese Art von Unstimmigkeiten zu vermeiden.

Compliance

Der Abschnitt über die Einhaltung der Vorschriften gehört nicht zu den Standardabschnitten eines ADR, aber wir empfehlen dringend, ihn hinzuzufügen. Im Abschnitt über die Einhaltung der Vorschriften wird festgelegt, wie die architektonische Entscheidung gemessen und geregelt wird. Wird die Prüfung der Einhaltung der Vorschriften für diese Entscheidung manuell durchgeführt oder kann sie mithilfe einer Fitnessfunktion automatisiert werden? Wenn sie automatisiert werden kann, kann der Architekt angeben, wie die Fitnessfunktion geschrieben werden soll und welche anderen Änderungen an der Codebasis erforderlich sind, um die Einhaltung dieser architektonischen Entscheidung zu messen.

Angenommen, du triffst in einer traditionellen n-schichtigen Architektur (wie in Abbildung 21-3 dargestellt) die Entscheidung, dass alle gemeinsam genutzten Objekte, die von Geschäftsobjekten in der Business-Schicht verwendet werden, in der Shared Services-Schicht liegen müssen, um die gemeinsam genutzten Funktionen zu isolieren und einzuschließen.

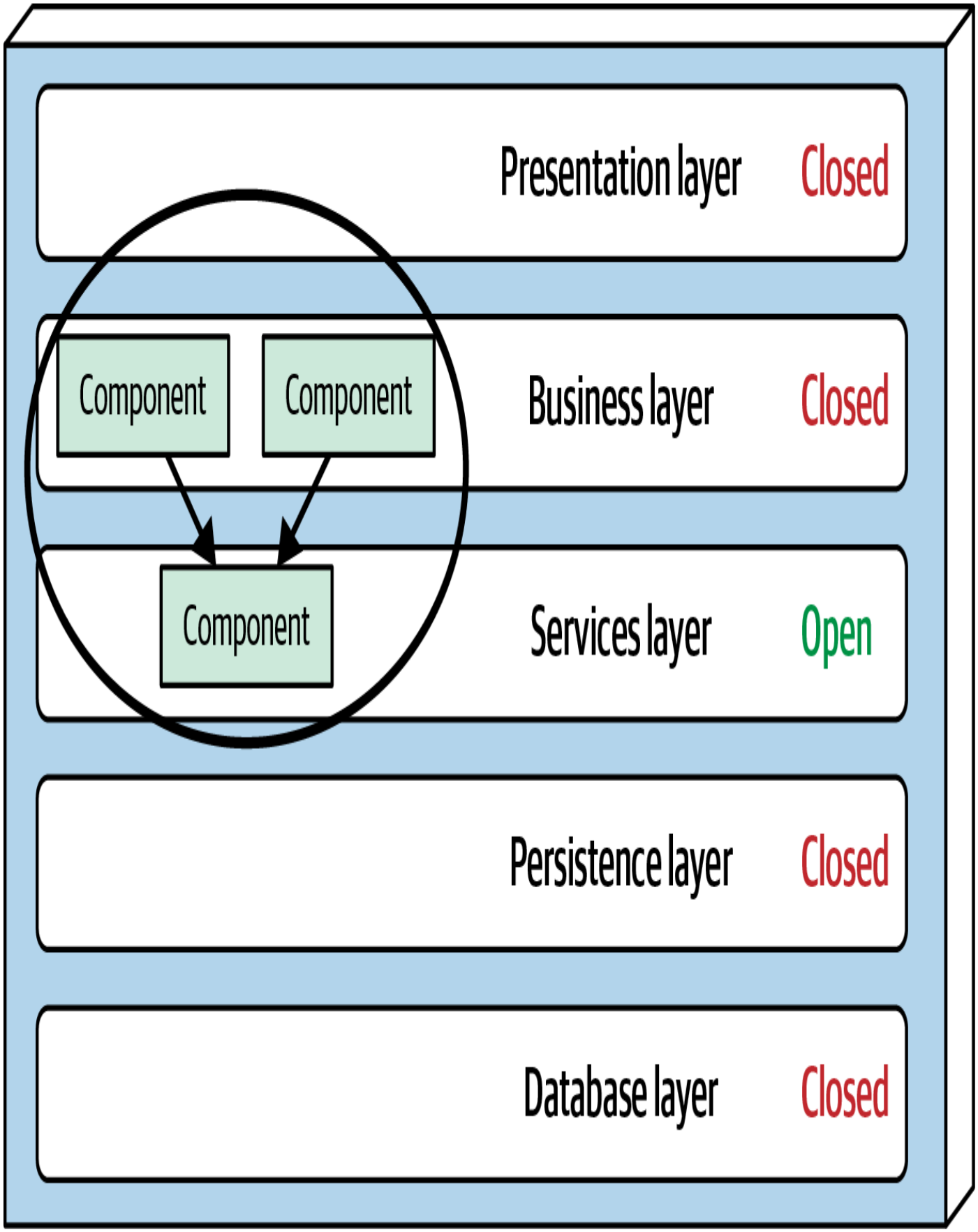

Abbildung 21-3. Ein Beispiel für eine architektonische Entscheidung

Diese architektonische Entscheidung kann mit einer Reihe von Automatisierungswerkzeugen gemessen und gesteuert werden, darunter ArchUnit in Java und NetArchTest in C#. Mit ArchUnit in Java könnte der automatisierte Fitness-Funktions-Test für diese Architekturentscheidung so aussehen:

```
@Test
public void shared_services_should_reside_in_services_layer()
    classes().that().areAnnotatedWith(SharedService.class)
        .should().resideInAPackage("..services..")
        .check(myClasses);
}
```

Für diese automatische Fitnessfunktion müssten neue Geschichten geschrieben werden, um eine Java-Annotation (`@SharedService`) zu erstellen und sie zu allen gemeinsam genutzten Klassen hinzuzufügen, um diese Methode der Steuerung zu unterstützen.

Anmerkungen

Ein weiterer Abschnitt, der nicht Teil eines Standard-ADR ist, den wir aber sehr empfehlen, ist der Abschnitt "Anmerkungen". Dieser Abschnitt enthält verschiedene Metadaten über das ADR:

- Original Autor
- Genehmigungsdatum
- Genehmigt von

- Ersetztes Datum
- Datum der letzten Änderung
- Geändert von
- Letzte Änderung

Auch wenn du ADRs in einem Versionskontrollsystem (wie Git) speicherst, ist es nützlich, zusätzliche Metadaten zu haben, die über das hinausgehen, was das Repository unterstützen kann. Wir empfehlen, diesen Abschnitt hinzuzufügen, unabhängig davon, wie und wo der Architekt ADRs speichert.

Beispiel

Unser Beispiel für das Going, Going, Gone (GGG) Auktionssystem umfasst Dutzende von Architekturentscheidungen. Die Aufteilung der Bieter- und Auktionator-Benutzeroberfläche, die Verwendung einer hybriden Architektur aus ereignisgesteuerten und Microservices, die Nutzung des Echtzeit-Transportprotokolls (RTP) für die Videoerfassung, die Verwendung eines einzigen API-Gateways und die Verwendung separater Warteschlangen für die Nachrichtenübermittlung sind nur einige der Architekturentscheidungen, die ein Architekt treffen würde. Jede architektonische Entscheidung, die ein Architekt trifft, egal wie offensichtlich sie ist, sollte dokumentiert und begründet werden.

Abbildung 21-4 veranschaulicht eine der architektonischen Entscheidungen innerhalb des GGG-Auktionssystems: die Verwendung von separaten Punkt-zu-Punkt-Warteschlangen zwischen den Diensten

Bid Capture, Bid Streamer und Bid Tracker anstelle eines einzigen Publish-and-Subscribe-Topics (oder sogar REST, um genau zu sein).

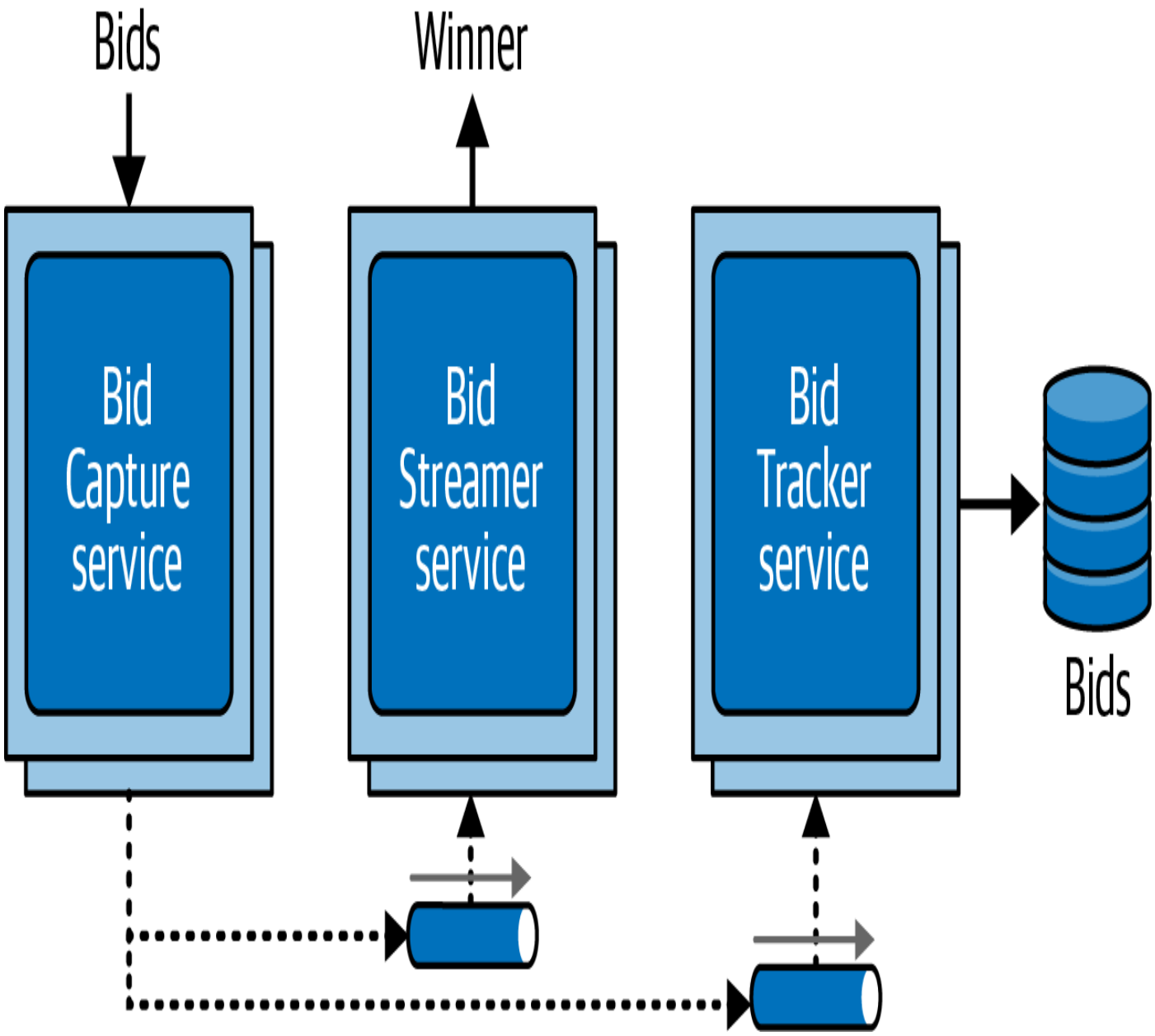

Abbildung 21-4. Verwendung von pub/sub zwischen Diensten

Ohne ein ADR, das diese Entscheidung rechtfertigt, könnten andere, die an der Gestaltung und Entwicklung dieses Systems beteiligt sind, anderer Meinung sein und sich für eine andere Umsetzung entscheiden.

Im Folgenden findest du ein Beispiel für ein ADR für diese architektonische Entscheidung:

ADR 76. Getrennte Warteschlangen für Bid Streamer und Bidder Tracker Services

STATUS

Angenommen

KONTEXT

Wenn der Dienst `Bid Capture` ein Gebot erhält, muss er dieses Gebot an die Dienste `Bid Streamer` und `Bidder Tracker` weiterleiten. Dies kann über ein einzelnes Topic (Pub/Sub), über separate Warteschlangen (Punkt-zu-Punkt) für jeden Dienst oder über REST über die Online-Auktions-API-Schicht erfolgen.

ENTSCHEIDUNG

Wir werden getrennte Warteschlangen für die Dienste `Bid Streamer` und `Bidder Tracker` verwenden.

Der Dienst `Bid Capture` benötigt keine Informationen von den Diensten `Bid Streamer` oder `Bidder Tracker` (die Kommunikation erfolgt nur in eine Richtung).

Der Dienst `Bid Streamer` muss die Gebote in genau der Reihenfolge erhalten, in der sie vom Dienst `Bid Capture` angenommen wurden. Durch die Verwendung von Nachrichten und Warteschlangen wird die Reihenfolge der Gebote für den Stream automatisch garantiert, indem die FIFO-Warteschlangen (First-in, First-out) genutzt werden.

Es gehen mehrere Gebote für denselben Betrag ein (z. B. "Höre ich einen Hunderter?"). Der Dienst `Bid Streamer` *benötigt nur das erste eingegangene Gebot für diesen Betrag, während der Dienst* `Bidder Tracker` *alle eingegangenen Gebote benötigt. Bei der Verwendung eines Themas (Pub/Sub) müsste* `Bid Streamer` *Gebote ignorieren, die mit dem vorherigen Betrag übereinstimmen, wodurch* `Bid Streamer` *gezwungen wäre, einen gemeinsamen Zustand zwischen den Instanzen zu speichern.*

Der Dienst `Bid Streamer` *speichert die Gebote für einen Artikel in einem In-Memory-Cache, während der Dienst* `Bidder Tracker` *die Gebote in einer Datenbank speichert. Der Dienst* `Bidder Tracker` *ist daher langsamer und benötigt möglicherweise Gegendruck. Eine eigene* `Bidder Tracker` *Warteschlange sorgt für diesen Gegendruck.*

KONSEQUENZEN
Wir benötigen ein Clustering und eine hohe Verfügbarkeit der Nachrichtenwarteschlangen.

Bei dieser Entscheidung muss der Dienst `Bid Capture` *die gleichen Informationen an mehrere Warteschlangen senden.*

Interne Gebotsereignisse umgehen die Sicherheitsprüfungen auf der API-Ebene.
UPDATE: Nach der Überprüfung auf der ARB-Sitzung am 14. Januar 2025 beschloss der ARB, dass dies ein akzeptabler Kompromiss ist und dass keine zusätzlichen Sicherheitsprüfungen für Gebotsereignisse zwischen diesen Diensten erforderlich sind.

KOMPLIZENZ
Wir werden regelmäßige manuelle Codeüberprüfungen durchführen, um sicherzustellen, dass asynchrone Pub/Sub-Nachrichten zwischen den Diensten `Bid Capture`, `Bid Streamer` *und* `Bidder Tracker` *verwendet werden.*

NOTEN
Autorin: Subashini Nadella

Genehmigt: ARB-Sitzung Mitglieder, 14 JAN 2025

Zuletzt aktualisiert: 14 JAN 2025

Speichern von ADRs

Wenn ein Architekt ein ADR erstellt, muss er es irgendwo speichern. Unabhängig davon, wo das ist, sollte jede architektonische Entscheidung eine eigene Datei oder Wikiseite haben. Manche Architekten legen ADRs gerne im selben Git-Repository ab wie den Quellcode, damit das Team ADRs genauso versionieren und verfolgen kann wie den Quellcode.

Bei größeren Organisationen raten wir jedoch aus mehreren Gründen von dieser Praxis ab. Erstens hat möglicherweise nicht jeder, der die architektonische Entscheidung sehen muss, Zugriff auf das Git-Repository mit den ADRs. Zweitens ist das Git-Repository der Anwendung kein guter Ort, um ADRs zu speichern, die in einem anderen Kontext stehen (z. B. Entscheidungen zur Integrationsarchitektur, Entscheidungen zur Unternehmensarchitektur oder Entscheidungen, die

für alle Anwendungen gelten). Aus diesen Gründen empfehlen wir, ADRs in einem speziellen ADR-Git-Repository zu speichern, auf das jeder Zugriff hat, in einem Wiki (mit einer Wiki-Vorlage) oder in einem gemeinsamen Verzeichnis auf einem gemeinsamen Dateiserver, auf das ein Wiki oder eine andere Software zur Darstellung von Dokumenten leicht zugreifen kann.

Abbildung 21-5 zeigt, wie diese Verzeichnisstruktur (oder die Navigationsstruktur einer Wiki-Seite) aussehen könnte.

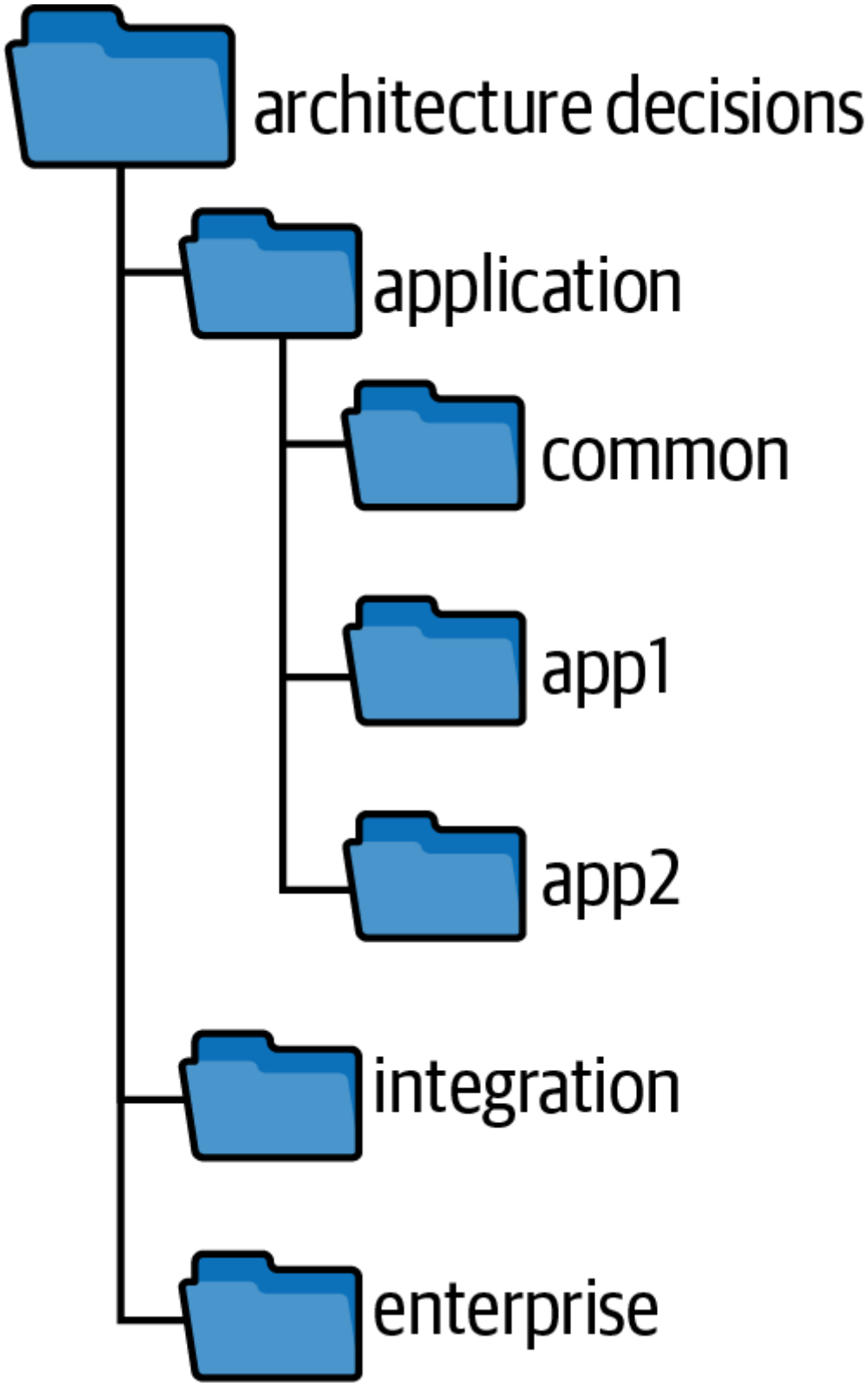

Abbildung 21-5. Beispielhafte Verzeichnisstruktur für die Speicherung von ADRs

Das *Anwendungsverzeichnis* enthält architektonische Entscheidungen, die für einen bestimmten Anwendungs- (oder Produkt-) Kontext

spezifisch sind. Dieses Verzeichnis ist in weitere Verzeichnisse unterteilt:

gemeinsame
Das *gemeinsame* Unterverzeichnis ist für architektonische Entscheidungen, die für alle Anwendungen gelten, wie z. B. "Alle Framework-bezogenen Klassen enthalten eine Annotation (@Framework in Java) oder ein Attribut ([Framework] in C#), das die Klasse als zum zugrunde liegenden Framework-Code gehörig kennzeichnet."

Bewerbung
Unterverzeichnisse unter dem *Anwendungsverzeichnis* entsprechen dem spezifischen Anwendungs- oder Systemkontext und enthalten architektonische Entscheidungen, die für diese Anwendung oder dieses System spezifisch sind (in diesem Beispiel die Anwendungen *app1* und *app2*).

Integration
Das *Integrationsverzeichnis* enthält ADRs, die die Kommunikation zwischen Anwendungen, Systemen oder Diensten betreffen.

Unternehmen
ADRs der Unternehmensarchitektur sind im *Unternehmensverzeichnis* enthalten, was bedeutet, dass es sich um globale Architekturentscheidungen handelt, die sich auf alle Systeme und Anwendungen auswirken. Ein Beispiel für ein ADR der Unternehmensarchitektur wäre: "Der Zugriff auf eine Systemdatenbank erfolgt nur vom eigenen System aus", wodurch

verhindert wird, dass Datenbanken von mehreren Systemen gemeinsam genutzt werden.

Wenn du ADRs in einem Wiki speicherst, gilt die gleiche Struktur, wobei jede Verzeichnisstruktur eine Navigationszielseite darstellt. Jedes ADR wird als eine einzelne Wikiseite innerhalb jeder Navigationszielseite (Anwendung, Integration oder Unternehmen) dargestellt.

Die in diesem Abschnitt angegebenen Namen für Verzeichnisse und Landingpages sind nur Empfehlungen und Beispiele. Wähle die Namen, die zur Situation deines Unternehmens passen, solange diese Namen in allen Teams einheitlich sind.

ADRs als Dokumentation

Die Dokumentation von Softwarearchitektur war schon immer schwierig. Es gibt zwar einige Standards für die Darstellung von Architektur (z. B. das C4-Modell des Softwarearchitekten Simon Brown oder der ArchiMate-Standard der Open Group), aber es gibt keinen einheitlichen Standard für die Dokumentation von Softwarearchitektur. An dieser Stelle kommen ADRs ins Spiel.

ADRs können ein effektives Mittel sein, um eine Softwarearchitektur zu dokumentieren. Der Abschnitt "Kontext" bietet eine hervorragende Gelegenheit, den spezifischen Bereich des Systems zu beschreiben, der eine architektonische Entscheidung erfordert, und die Alternativen zu beschreiben. Noch wichtiger ist, dass der Abschnitt Entscheidung die Gründe beschreibt, warum eine bestimmte Entscheidung getroffen wird,

was bei weitem die beste Form der Architekturdokumentation ist. Der Abschnitt Konsequenzen fügt das letzte Puzzleteil hinzu, indem er die Analyse derKompromisse für die Entscheidung beschreibt - zum Beispiel die Gründe (und die Kompromisse), warum die Leistung der Skalierbarkeit vorgezogen wird.

ADRs für Standards verwenden

Nur sehr wenige Entwickler mögen Standards. Leider geht es bei Standards manchmal eher um Kontrolle als um einen nützlichen Zweck. Die Verwendung von ADRs für Standards kann diese schlechte Praxis ändern. Der Abschnitt "Kontext" eines ADR beschreibt zum Beispiel die Situation, die die Organisation dazu zwingt, eine bestimmte Norm einzuführen. Im Abschnitt "Entscheidung" eines ADR kann also nicht nur angegeben werden, was die Norm ist, sondern vor allem, warum es sie geben muss.

Dies ist eine gute Möglichkeit, um zu prüfen, ob ein bestimmter Standard überhaupt existieren sollte. Wenn ein Architekt ihn nicht begründen kann, dann ist es vielleicht kein guter Standard, den er festlegen und durchsetzen sollte. Und je mehr Bauherren verstehen, *warum* es einen bestimmten Standard gibt, desto eher werden sie ihn befolgen (und ihn dementsprechend nicht in Frage stellen). Der Abschnitt "Konsequenzen" in einem ADR ist ein weiterer guter Ort, um zu prüfen, ob ein Standard gültig ist: Hier muss der Architekt über die Auswirkungen und Konsequenzen des Standards nachdenken und dokumentieren, ob der Standard umgesetzt werden sollte oder nicht.

Verwendung von ADRs mit bestehenden Systemen

Viele Architekten stellen die Nützlichkeit von ADRs für bestehende Systeme in Frage. Schließlich wurden die architektonischen Entscheidungen bereits getroffen und das System ist in Produktion. Haben ADRs zu diesem Zeitpunkt überhaupt noch einen Sinn? In der Tat, das tun sie. Erinnere dich: ADRs sind mehr als nur eine Dokumentation - sie helfen Architekten und Entwicklern zu verstehen , *warum* eine Entscheidung getroffen wurde und ob sie die beste war.

Beginne damit, einige ADRs für die wichtigsten architektonischen Entscheidungen zu schreiben und zu hinterfragen, ob diese Entscheidungen richtig sind oder nicht. Ein Beispiel: Eine Gruppe von Diensten teilt sich eine einzige Datenbank. Aber warum? Gibt es dafür einen guten Grund? Sollten die Daten aufgeteilt werden?

Wenn du ADRs in ein bestehendes System einbauen willst, musst du ein wenig nachforschen, um diese Fragen *zu* klären. Leider hat die Person, die die ursprüngliche Entscheidung getroffen hat, das Unternehmen vielleicht schon vor langer Zeit verlassen, sodass niemand die Antwort kennt. In diesen Fällen muss der Architekt die Alternativen und die Kompromisse der einzelnen Optionen ermitteln und analysieren und versuchen, die bestehende Entscheidung zu bestätigen (oder zu entkräften). In jedem Fall beginnt das Schreiben von ADRs für diese Art von wichtigen Entscheidungen mit dem Aufbau von Begründungen und

Argumenten (und Brain Trust) für das System und kann helfen, architektonische Ineffizienzen und falsches Systemdesign zu erkennen.

Nutzung von generativer KI und LLMs für architektonische Entscheidungen

Einer der vielen faszinierenden Aspekte der generativen KI ist die Frage, ob Architekten sie nutzen können, um Entscheidungen zu treffen und zu validieren. Sollten Dienste Messaging, Streaming oder Event Sourcing verwenden, wenn sie Daten nachgelagert senden? Soll die Datenbank ein einziger Monolith bleiben oder soll sie in Domänen-Datenbanken aufgeteilt werden? Soll `Payment Processing` als ein einziger Dienst eingesetzt werden oder in mehrere Dienste aufgeteilt werden, einen für jede Zahlungsart?

Die meisten Architekten kennen die Antwort auf diese Fragen bereits - es kommt darauf an! Um auf unser erstes Gesetz der Softwarearchitektur zurückzukommen: *Alles in der Softwarearchitektur ist ein Kompromiss*. Entscheidungen wie diese hängen von vielen Faktoren ab, unter anderem von dem spezifischen Kontext, in dem die Entscheidung getroffen wird. Jede Situation und jede Umgebung ist anders, deshalb gibt es keine "bewährten Methoden" für diese Art von strukturellen Fragen.

Die meisten LLMs stützen sich bei ihren Ergebnissen weitgehend auf die Wahrscheinlichkeitsrechnung. Mit anderen Worten: Was ist die wahrscheinlichste Antwort im Kontext der Eingabeaufforderung, und

was ist die "bewährte Methode" für dieses Problem? Wahrscheinlichkeiten und "bewährte Methoden" haben jedoch keinen Platz bei Architekturentscheidungen. Die Beantwortung von Architekturfragen erfordert eine sorgfältige Analyse der damit verbundenen Kompromisse und die Anwendung eines spezifischen geschäftlichen und technischen Kontexts, um die am besten geeignete Wahl zu treffen. Wenn es dem Unternehmen zum Beispiel wichtig ist, Änderungen und neue Funktionen so schnell wie möglich auf den Markt zu bringen, ist *die Wartungsfreundlichkeit* viel wichtiger als die *Leistung* und wird den Großteil des Entscheidungsprozesses bestimmen.

Bei architektonischen Entscheidungen müssen geschäftliche Belange (wie die Zeit bis zur Markteinführung oder nachhaltiges Wachstum) in architektonische Merkmale (wie Wartbarkeit, Testbarkeit, Einsatzfähigkeit usw.) übersetzt werden. Diese Übersetzung ist nicht immer offensichtlich, und um sie richtig zu machen, braucht man jahrelange Erfahrung. Sobald sie abgeschlossen ist, dient sie als Grundlage für die Analyse von Kompromissen. Die Entscheidung, ob ein einzelner Dienst für die Zahlungsabwicklung oder ein Dienst pro Zahlungsart benötigt wird, ist ein Kompromiss zwischen Wartbarkeit und Leistung: Ein einzelner Dienst bietet eine bessere Leistung, aber mehrere Dienste bieten eine bessere Wartbarkeit. Wenn es dem Unternehmen in erster Linie um die Zeit bis zur Markteinführung geht, ist die Wartbarkeit viel wichtiger als die Leistung, so dass in diesem speziellen Kontext getrennte Dienste die richtige Wahl wären.

Da die Analyse von Kompromissen und der geschäftliche Kontext sehr spezifisch und individuell sind, ist es für die generative KI, wie sie derzeit existiert, schwierig, die beste architektonische Entscheidung zu treffen. Nach den jüngsten Experimenten, die deine Autoren durchgeführt haben, ist es im besten Fall möglich, dass ein generatives KI-Tool die möglichen Kompromisse einer Entscheidung aufzeigt und dabei hilft, übersehene Kompromisse zu identifizieren. Generative KI-Tools verfügen zwar über viel *Wissen*, aber es fehlt ihnen an der nötigen *Weisheit*, um die beste architektonische Entscheidung zu treffen.