Architekturarbeit richtig verankern

Hisst die Segel!

Softwarearchitektur ermöglicht es, die Vogelperspektive einzunehmen. Damit alle Projektbeteiligten den Überblick behalten, muss sie in den Unternehmenskontext eingebettet werden und abteilungsübergreifend angelegt sein. Dabei gilt es, strategische Entscheidungen zu treffen, die in hohem Maße von konkreten Anforderungen der spezifischen Branche abhängen. Dieser Artikel zeigt anhand praktischer Beispiele, wie sich die Architektur für moderne Lösungen von Anfang an in den Entwicklungsprojekten verankern lässt.

von Pascal Stieber

„Wir können den Wind nicht ändern, aber wir können die Segel richtig setzen." Dieser 2 300 Jahre alte Satz wird häufig zitiert und soll von Aristoteles stammen. Es gibt jedoch Indizien dafür, dass Aristoteles diesen Satz niemals gesagt oder geschrieben haben soll.

Nichtsdestotrotz kann man sagen, dass sich dieser Aphorismus sehr zutreffend auf Projekte in der Softwareentwicklung gerade in mittelständigen und großen Unternehmen abstrahieren lässt. Als Softwarearchitekt erlebe ich jeden Tag, dass es von unzähligen Faktoren abhängt, ob eine Lösung erfolgreich ist. Neben dem klassischen magischen Dreieck des Projektmanagements – Budget, Zeit und Qualität – entscheiden zum Beispiel auch das Skillset der Teammitglieder oder die Unternehmenskultur über den Erfolg oder Misserfolg eines digitalen Produkts. Großen Einfluss kann aber auch die zugrunde liegende Softwarearchitektur nehmen. Diese wird in vielen Projekten eher aus einer technischen Perspektive betrachtet. Dabei sind hier – wenn wir genauer hinsehen – weitere Fragestellungen interessant und für die Softwareentwicklung relevant:

Wo fängt Architekturarbeit an, wo hört sie auf? Wer ist für die Architektur verantwortlich? Zu welchem Projektstatus muss die Architektur finalisiert sein und der Softwareentwicklung vorliegen? Oder kann gar die Softwareentwicklung Einfluss auf die bestehende Architektur haben? Wie soll mit diesen Rückkopplungen verfahren werden? Welche zusätzlichen Herausforderungen entstehen bei der Architekturarbeit in verteilten Systemen?

Wir zeigen, wie Sie die Segel bei der Softwareentwicklung auf Erfolgskurs stellen, indem Sie Ihre Softwarearchitektur besser ausrichten.

Agile Architektur

Die Gesellschaft der Informatik definiert Softwarearchitektur als „grundlegende Organisation eines Systems, dargestellt durch dessen Komponenten, deren Beziehungen zueinander und zur Umgebung sowie den Prinzipien, die den Entwurf und die Evolution des Systems bestimmen" [1]. Architekturentscheidungen haben also einen hohen Einfluss auf das Softwareprodukt und sind später nur schwer veränderbar. Diese Beschreibung könnte uns zu dem Irrglauben verleiten, dass Architektur top-down, also zu Projektbeginn einmalig erstellt wird. Das Gegenteil ist jedoch der Fall!

In agilen Projektkontexten entsteht auch die Architektur iterativ inkrementell. Ähnlich wie mit Scrum oder Kanban in der agilen Softwareentwicklung üblich, kann hier auch die Architektur in Inkrementen aufgebaut, gefestigt und näher spezifiziert werden. Architekturentscheidungen werden dann über Iterationen bewertet und gegebenenfalls angepasst. Architekturarbeit und Softwareentwicklung bleiben dabei eng beieinander und stimmen sich gut ab. Letztendlich muss bei jeder Anforderung entschieden werden, wer zuständig ist: Architektur oder Entwicklung. Die folgende Grafik – im Volksmund auch „Architekturbrezel" genannt – visualisiert diesen Vorgang (**Abb. 1**).

Fundamentale Fragestellungen und entsprechende Risiken liegen in den Händen der Architektur. Die Architektur schafft einen Rahmen für die Umsetzung. Zwischen Mikro- und Makroarchitektur zu unterscheiden, hilft, Anforderungen korrekt einzuordnen. Für die Makroarchitektur sind Anforderungen relevant, die von allen oder von mehreren Teams gleichzeitig getragen werden müssen. Die Makroarchitektur gibt einen Entscheidungsspielraum vor und setzt Leitplanken für weitere Entscheidungen, die dann – und hier befinden wir

uns auf der Ebene der Mikroarchitektur – in einzelnen Teams entschieden werden können. Jetzt befinden wir uns auf der Ebene der Mikroarchitektur. Üblicherweise werden Problemstellungen auf Makroarchitekturebene von Teamleads oder von interdisziplinären Teams in regelmäßigen „Communities of Practice“ besprochen und Lösungen ausgearbeitet.

Dennoch: Architekturen entstehen nicht hinter verschlossenen Türen, vielmehr werden sie in enger Absprache mit den Entwicklungsteams erstellt und kommuniziert. Das Feedback aus der Entwicklung ist essenziell und muss in zukünftige Iterationen der architekturalen Fragestellungen einfließen, sodass neu gewonnene Erkenntnisse zur Optimierung der erstellten Architektur beitragen können. Architekt:innen sollten sich dementsprechend nicht in der Kapitänsrolle, sondern als Teil der Crew wiederfinden.

Es gibt Fragestellungen, die ich niemals einzelnen Teams oder einer Architektur-Community-of-Practice überlassen würde. Dazu gehört die Auswahl einer Enterprise-Middleware oder eines Cloud-Providers. Hier kommt die Enterprise-Architektur ins Spiel oder etwas spezifischer: die Technologiearchitektur. Hier hat die Stabsstelle die Aufgabe, Technologien und Plattformen auf der Basis von Regulatorien und Rahmenbedingungen des Unternehmens auszuwählen. Beispielsweise wäre ein Finanzdienstleister an Vorgaben der Bundesanstalt für Finanzdienstleistungsaufsicht (BaFin) gebunden, was die Auswahl des Cloud-Providers einschränkt. Derartig kostenintensive Architekturentscheidungen sind strategisch und längerfristig zu treffen und gehören somit nicht in den Softwareentwicklungsprozess und auch nicht auf die Makroarchitekturebene. Feedback aus den Entwicklungsabteilungen kann hier dennoch helfen, nicht zuletzt, um vorhandene Skills des Entwicklungsteams bei der Providerauswahl zu berücksichtigen.

Architekturarbeit in verteilten Systemlandschaften

In Literatur, Magazinen, Blogposts und vielen anderen Medien der IT-Szene wurde in den letzten Jahren kräftig für verteilte Systeme geworben. Architekturstile wie Self-contained Systems und Microservices erfreuen sich hoher Beliebtheit. Ich stelle bei meinen Kunden immer wieder fest, dass die Erwartungshaltung gegenüber einer „renovierten“ Systemlandschaft hoch ist. Schließlich wurde viel Geld investiert. So mancher Auftraggeber wird daher nervös, wenn der Softwareentwicklungsprozess länger dauert, als er es erwartet hat. Bei Cloud-Native-Software werden die Kosten des Betriebs transparent, während im eigenen Rechenzentrum eine große Kostenstelle entsteht und sich einzelne Produkte und Ressourcen meist nicht mehr zuordnen lassen.

Dazu kommen in verteilten Systemen eine Menge Artefakte, die zu warten sind und die die Lösung komplexer machen. Solche Dinge wirken sich bei der Entwicklung eines Softwareprodukts also nicht gerade vorteilhaft auf die Geschwindigkeit aus. Zwar werden die Applikationen selbst kleinteiliger, jedoch erhöht sich

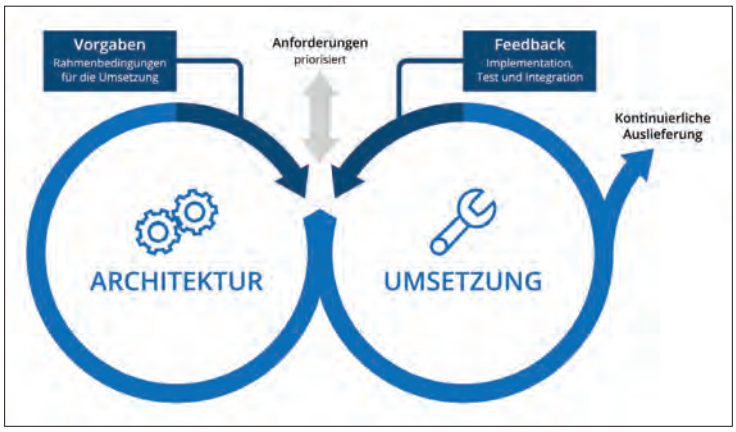

Abb. 1: Agile Architekturarbeit [2]

die Anzahl der beweglichen Teile und ihrer vielfältigen Abhängigkeiten und damit die Gesamtkomplexität.

Zur Klarstellung: Evolutionär bedingt sollen Architekturstile wie Microservices die fachliche Komplexität reduzieren. Die Gesamtkomplexität wird dadurch aber nicht geringer, sondern einige Dinge verschieben sich, wie zum Beispiel Aspekte aus Wartung und Betrieb rund um Deployment, Orchestrierung, Monitoring und die Integration der einzelnen Applikationen in bestehende Umsysteme.

Auch der Service-Schnitt von verteilten Service-orientierten Systemen ist in seiner Bedeutung für die Softwareentwicklung nicht zu unterschätzen. Mal eben nach Gutdünken entworfene oder einfach entlang der unternehmerischen Organisationsstruktur geschnittene Systemgrenzen können zu unsagbar schlechtem und unwartbarem Softwaredesign führen. Wie oft habe ich das schon erlebt! Folgen und Ausmaß werden häufig erst im späteren Projektverlauf sichtbar. Ähnlich wie bei erhöhten Kosten, die entstehen, je später ein Fehler im System gefunden wird, verhält es sich mit dem falsch designten Service-Schnitt in verteilten Systemen. Großangelegte Refactorings auf dieser Ebene sind kosten- und zeitintensiv.

Eine exorbitante Schnittstellenkommunikation kann ein erster Indikator dafür sein, dass der Service-Schnitt nicht korrekt gewählt wurde. Kollaborative Methoden wie Event Storming oder Domain Storytelling können bei der Ausarbeitung des Service-Schnitts helfen.

Ich arbeite in meinen Teams immer wieder iterativ und mit kollaborativen Methoden sowie in Workshops, um sicherzugehen, dass ich eine zukunftssichere und belastbare Detailtiefe erreiche. Diese verankern wir im agilen Softwareentwicklungsprozess. Das heißt, dass über Iterationen und Inkremente das Wissen über die Fachlichkeit an Detail gewinnt und näher spezifiziert wird. Neben dem verbesserten Service-Design tragen Event Storming und Domain Storytelling auch zu einem besseren Verständnis des zu implementierenden Business bei. Im Folgenden ein Beispiel:

Bei zwei konkurrierenden Konzernen aus dem E-Mobility-Umfeld brauchten wir einige Disziplin, bis wir ein Fingerspitzengefühl für den richtigen Service-Schnitt in

Bei der Architekturarbeit muss man viele Optionen abwägen. Technologiediversität und -heterogenität, die den Markt bestimmen und stetig zunehmen, machen die Wahl nicht leicht.

den verteilten Architekturen entwickelten. Belohnt wurde diese Disziplin am Ende mit einem zukunftsträchtigen Softwareprodukt, auf das alle Seiten stolz sein konnten.

Beide Konzerne nutzten eine ähnliche technische Umgebung. Konzern A hatte Event Storming als festen Bestandteil in seinen Softwareentwicklungsprozess etabliert, was die Abstimmung zwischen den Entwicklungsteams sehr vereinfachte. Die Vorteile liegen auf der Hand:

1. Aufwand für Kommunikation sinkt: Sowohl die verbale Kommunikation zwischen den Teams als auch technische Kommunikationen einzelner Services konnten reduziert werden.
2. Schnellere Time-to-Market: Wenn einzelne Services schneller entwickelt werden, beschleunigt das die Time-to-Market des digitalen Produkts.
3. Kosten werden gespart: Da weniger technische Kommunikation zwischen einzelnen Services nötig ist, fallen die Betriebskosten deutlich geringer aus.
4. Weniger Fehler: Je weniger technische Kommunikation nötig ist, umso kleiner wird das Fehlerpotenzial.

Als Architekt:in richtig entscheiden

Bei der Architekturarbeit müssen häufig sehr viele Optionen abgewogen werden. Bei der Technologiediversität und -heterogenität, die den Markt bestimmen und die stetig zunehmen, fällt es nicht immer leicht, die richtige Wahl zu treffen. Eine intuitive Entscheidung kann wie gesagt schnell ins Auge gehen, zumal wir sie nur schwer gegenüber Projektstakeholdern verantworten und begründen können. Wie also vorgehen, um die richtigen Entscheidungen zu treffen, fundierte Entscheidungen, zum Beispiel bei der Technologieauswahl, die wir gegenüber den Meinungen der anderen Projektbeteiligten sicher vertreten können?

Schauen wir uns dafür einmal die DNA einer Entscheidung an: Eine Entscheidung beruht auf Kriterien, Rahmenbedingungen, Handlungsoptionen und einer anschließenden Bewertung [3]:

- Kriterien sind Ziele, die wir mit der Entscheidung erreichen wollen.
- Rahmenbedingungen sind Dinge, die Einfluss auf eine Handlung haben und nicht veränderlich sind.
- Handlungsoptionen sind die Möglichkeiten, die uns zur Verfügung stehen.
- Die Bewertung der Handlungsoptionen findet im Hinblick auf die Kriterien statt.

So viel zur Theorie. Wie aber sieht so ein Entscheidungsprozess in der Praxis aus? Dazu möchte ich ein typisches Beispiel geben:

- Kriterium: Bei einem unserer Kunden aus dem E-Mobility-Umfeld sollten zwei Teams Daten über ihre Services austauschen. Der Fachabteilung war dabei ein zuverlässiger Austausch wichtiger als eine schnelle Antwortrate.
- Rahmenbedingungen: Wir bekamen die Vorgabe, dass sich die neue Lösung an bereits vorhandenen Technologien des Unternehmens bedienen sollte.
- Handlungsoptionen: Zwei Handlungsoptionen standen zur Diskussion. Variante A: Schnittstellen kommunizieren asynchron via Middleware (Kafka-Broker). Variante B: Schnittstellen kommunizieren synchron via HTTP (REST).
- Bewertung: Da eine hochverfügbar ausgelegte Middleware wie Kafka zuverlässiger ist als ein synchroner Aufruf, entschieden wir uns in diesem Projekt für die asynchrone Kommunikation mittels Kafka-Broker.

Entscheidungswege dokumentieren

Um es schwarz auf weiß zu haben, können Architekturentscheidungen in sogenannten Decision Logs festgehalten und veröffentlicht werden. Hier wird der Entscheidungsweg vollständig dokumentiert und den Projektbeteiligten zugänglich gemacht. Damit verhindern wir, dass Entscheidungen zu sehr durch persönliche Erfahrungen und Wissen beeinflusst werden. Im Nachhinein hätte man sich vielleicht anders entschieden, doch auf dem damaligen Erkenntnisstand war die Entscheidung nachvollziehbar. Das müssen wir uns immer wieder klarmachen. Häufig treffen wir Annahmen über Unbekanntes, um schneller zu einem Ergebnis zu kommen. Damit wir hier nicht stehenbleiben, helfen Decision Logs, Entscheidungen zu reflektieren und von Zeit zu Zeit neu zu bewerten. Das erhöht unsere Flexibilität und wir können besser auf Veränderungen auf allen Ebenen eingehen. Hier gilt das Prinzip „Lösung vor Perfektion“. Das hilft, innovativ zu bleiben und schneller in die Umsetzung zu gehen.

Genauigkeit bei den Aussagen

In gleichem Maße war der Fachabteilung unseres Kundenunternehmens ein zuverlässiger Austausch wichtiger als eine schnelle Antwortrate. Solche und ähnliche Anforderungen aus der Fachabteilung sind zunächst wenig spezifiziert. Für Architekturentscheidungen und

Softwareentwicklung brauchen wir genauere Aussagen. Solche Qualitätsmerkmale (genormt durch ISO 25010) gemeinsam mit den Fachabteilungen auszuarbeiten, darin liegt die Handwerkskunst der Softwarearchitektur. In Workshops oder in Interviews lassen sich Werte ableiten, die Produktverantwortliche von einem Softwareprodukt erwarten. Solche Werte oder eher Qualitätsmerkmale wie Zuverlässigkeit, Effizienz, Sicherheit und Performanz – um nur die bekanntesten zu nennen – sind es, die am Ende über die Architektur entscheiden.

Fazit

Um Softwarelösungen zukunftswirksam aufzubauen, bedarf es Architekturarbeit auf verschiedenen Ebenen. Warum das so ist, und wie das in der Praxis aussehen kann, habe ich in diesem Beitrag zu erläutern versucht.

Die wichtigsten Punkte möchte ich an dieser Stelle noch einmal zusammenfassen, sozusagen als Erinnerungsstütze und Handlungsmaxime. An je mehr dieser Punkte ihr einen Haken machen könnt, umso besser wird eure Architekturarbeit:

- Wir sollten das Thema Architektur fest in unseren Softwareentwicklungsteams verankern, zum Beispiel durch interdisziplinäre Teams.
- Weitreichende und kostenintensive Entscheidungen wie die Auswahl eines Cloud-Providers sollten von Enterprise-Architekt:innen getroffen werden.
- Teamübergreifender Austausch in Form von Communities of Practice und kollaborativen Methoden ist zu fördern.
- Fachabteilungen sollten frühestmöglich in den Entstehungsprozess der Softwareprodukte einbezogen werden.
- Architekt:innen sollten bei der Spezifizierung von Anforderungen unterstützen, indem sie Qualitätsmerkmale gemeinsam mit den Produktverantwortlichen ausarbeiten.
- Architekturentscheidungen sollten nicht top-down an die Entwicklungsteams übergeben werden, sondern iterativ und mittels Feedbacks aus den Entwicklungsteams spezifiziert, verifiziert und in Decision Logs dokumentiert und allen Beteiligten zugänglich gemacht werden.

Pascal Stieber ist Solution Architect und Trainer bei der OPITZ CONSULTING Deutschland GmbH und beschäftigt sich mit technologischen Trendthemen in modernen Softwarearchitekturen. Seit über 15 Jahren unterstützt er Unternehmen im Rahmen von Design- und Entwicklungsprozessen und teilt sein Wissen als Trainer in Schulungen und Workshops.

Links & Literatur

[1] https://gi.de/informatiklexikon/software-architektur

[2] Toth, Stefan: „Vorgehensmuster für Softwarearchitektur. Kombinierbare Praktiken in Zeiten von Agile und Lean“; Hanser Verlag, 2019

[3] Rüping, Andreas: „Gute Entscheidung in IT-Projekten“; dpunkt Verlag, 2019