mehr zum thema:
http://blog.codecentric.de/
www.meettheexperts.de/

WER BRAUCHT EINEN ARCHITEKTEN? ÜBER ZIELE UND AUFGABEN VON ARCHITEKTUR UND ARCHITEKTEN

Agile Methoden wie Scrum und XP kennen keinen Architekten – das Team ist der Architekt. Aber funktioniert das auch immer so? Kann ein Entwicklungsteam immer die notwendigen Architekturaufgaben übernehmen oder braucht man unter bestimmten Voraussetzungen doch einen dedizierten Architekten? Der Artikel beschreibt die Ziele und Aufgaben der Architektur und versucht, Antworten auf diese Fragen jenseits der üblichen Argumentationspfade zu finden.

der autor

Uwe Friedrichsen
(E-Mail: uwe.friedrichsen@codecentric.de)
hat langjährige Erfahrungen als Architekt, Projektleiter und Berater. Seine aktuellen Schwerpunkte sind Softwarearchitekturen und Unternehmensarchitekturen im Kontext agiler Konzepte.

Sind Architekten obsolet?

Schon 2003 stellte Martin Fowler in „Who needs an Architect" die Notwendigkeit eines Architekten in der Softwareentwicklung in Frage (vgl. [Fow03]). Dazu passend kennen die heute dominierenden agilen Methoden Scrum und XP keine expliziten Architekten. Stattdessen wird die Architekturentwicklung auf das (Entwickler-)Team verteilt (vgl. z. B. [Lip09]) und Architekturen sind „emergent".

Das bedeutet – vereinfacht ausgedrückt –, dass die Architektur nicht mehr vor Beginn der Entwicklung festgelegt wird, sondern sich im Laufe der Entwicklung unter Berücksichtigung einiger mehr oder minder einfacher Designprinzipien aus dem erstellten Code „herausschält". Wir sprechen hier also eher von einer impliziten als von einer expliziten Architekturentwicklung. Folgt man dieser Argumentation, wird in einem solchen Umfeld kein dedizierter Architekt mehr benötigt.

Aber stimmt das tatsächlich? War die über viele Jahre hinweg etablierte Rolle des Architekten nur schwergewichtiger Selbstzweck ohne tieferen Nutzen oder hat diese Rolle vielleicht doch einen Sinn? Um diese Frage vernünftig beantworten zu können, möchte ich im Folgenden ein paar grundlegende Gedanken zum Thema Architektur anstellen. Die elementaren Fragen sind, wofür man Architektur eigentlich benötigt und was die Ziele von Architektur sind.

Architekturziele

Damit Architektur kein Selbstzweck ist, muss sie einen Nutzen bringen, z. B. wirtschaftlich oder durch den Abbau von Risiken. Bei der Suche nach dem Nutzen von Architektur gelangt man sehr schnell zu den so genannten Qualitätsmerkmalen von Software (vgl. z. B. [ISO]): Architektur soll die Erfüllung der Qualitätsmerkmale sicherstellen. Wenn man sich jetzt überlegt, welchen Nutzen die Erfüllung dieser Merkmale letztlich bringt, kommt man zu zwei Zielen:

- Zum einen sollen die verschiedenen beteiligten Parteien (*Stakeholder*) möglichst gut zufrieden gestellt werden. Hier ist also eine möglichst effiziente (schnelle und kostengünstige) und effektive (vollständige und korrekte) Umsetzung der verschiedenen funktionalen und nicht-funktionalen Anforderungen gefragt.
- Da Geld in allen IT-Projekten eine große Rolle spielt, ist das zweite Ziel, die entstehenden Kosten so gering wie möglich zu halten.

Hierbei ist es essenziell, dass es sich nicht um eine einmalige Aufgabe handelt, sondern dass Architektur die Aufgabe hat, die Ziele über den gesamten Lebenszyklus einer Anwendung möglichst gut zu unterstützen – und nicht nur während der Ersterstellung.

Zahlreiche Untersuchungen belegen, dass die Erstentwicklung einer Anwendung in Relation zu ihrem gesamten Lebenszyklus im Schnitt nur 10 bis 20 % bezogen auf Zeit und Aufwand beträgt – Tendenz fallend (vgl. z. B. [Kos03]). Offensichtlich muss Architektur also zur Maximierung der genannten Ziele stets den gesamten Lebenszyklus einer Anwendung im Fokus haben. Damit kann man die Ziele von Architektur als eine Art „Optimierung von Summen über die Zeit" darstellen:

- Architektur hat die Aufgabe, die Interessen aller Stakeholder am System über den gesamten Lebenszyklus zu maximieren.
- Architektur hat die Aufgabe, die Gesamtkosten für das System über den gesamten Lebenszyklus zu minimieren.

Wichtig ist hierbei, dass die Interessen *aller* beteiligten Parteien maximiert werden, also nicht nur versucht wird, eine Partei zu befriedigen, und dass der gesamte Lebenszyklus und nicht nur ein bestimmter Zeitabschnitt im Fokus ist. Analog ist zu beachten, dass die *Gesamtkosten* betrachtet werden, also nicht nur die reinen Entwicklungskosten, sondern auch Einführungs-, Migrations-, Betriebs- sowie Wartungs- und Weiterentwicklungskosten – und das ebenfalls wieder über den gesamten Lebenszyklus. Was nützt eine möglichst billige Entwicklung, wenn Betrieb und Wartung der Anwendung kostentechnisch ein Alptraum sind?[1])

[1]) *Das ist eine Stelle, an der viele der heute üblichen Business-Case-Kalkulationsverfahren noch zu kurz greifen, weil sie die Folgekosten für Architekturentscheidungen nicht korrekt berücksichtigen. So mag ein „schneller Hack", also eine einfache Lösung an den bestehenden Strukturen vorbei, auf dem Papier attraktiv erscheinen, wenn man nur die Entwicklungskosten berücksichtigt. Berücksichtigt man in der Kalkulation aber, dass durch solche „Hacks" die Wartung des Systems deutlich aufwändiger wird und die Systemstabilität leidet, d. h. die Fehlerquote steigt, dann sieht der Business-Case schnell ganz anders aus.*

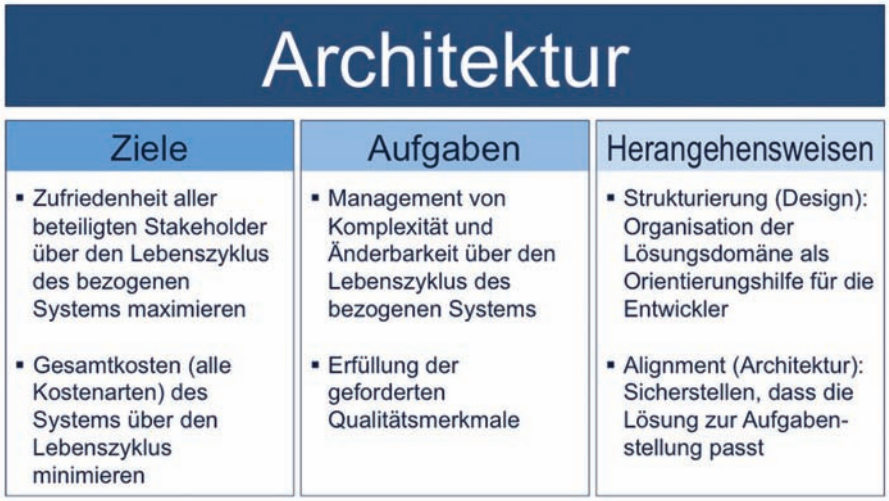

Abb. 1: Ziele, Aufgaben und Herangehensweisen von Architektur im Überblick.

Aufgaben von Architektur

Welche Aufgaben muss ein Architekt wahrnehmen, um diese Ziele zu erreichen? Diese Frage hat John Zachman, einer der wichtigsten Unternehmensarchitekten unserer Zeit, einmal sehr prägnant beantwortet: „In short, the reasons why you need Architecture: Complexity and change" (vgl. [Zac08]). Zachman hat dabei über Architektur im Allgemeinen gesprochen. Durch Ergänzen der im Kontext von IT-Systemen sehr wichtigen Qualitätsattribute lassen sich die Aufgaben von Architektur folgendermaßen formulieren:

Architektur ist das Management von Komplexität und Änderbarkeit bei Erfüllung der geforderten Qualitätsmerkmale.

Hierzu sind noch ein paar ergänzende Erläuterungen notwendig. Zunächst einmal könnte man – berechtigterweise – anmerken, dass Änderbarkeit ein Qualitätsmerkmal ist und dass Komplexitätsmanagement eine Voraussetzung für Änderbarkeit ist. Das ist prinzipiell korrekt, nur ist das Management von Komplexität und Änderbarkeit heute ein so bedeutendes Thema in der IT-Architektur, dass es wichtig ist, diese beiden Punkte gesondert hervorzuheben.

Gerade vor dem Hintergrund der immer komplexer werdenden IT-Systeme, der immer höheren Änderungsrate bei immer kürzeren Zeitstrecken und der immer längeren Betriebs- und Weiterentwicklungsphasen kommt dem Management von Komplexität und Änderbarkeit eine Schlüsselrolle in der Architektur zu. Diese beiden Aspekte einfach nur als ein Qualitätsmerkmal unter vielen zu positionieren, wird dem nicht gerecht. Sie sind *der Schlüssel* zum Erreichen der zuvor beschriebenen Architekturziele und müssen entsprechend gegenüber den anderen Qualitätsmerkmalen – wie etwa Ergonomie und Portierbarkeit – deutlich hervorgehoben werden.

Ein zweiter Punkt ist die Erfüllung der geforderten Qualitätsmerkmale an sich. Hier ist nicht nur von nicht-funktionalen Anforderungen die Rede, sondern von allen Qualitätsmerkmalen, also auch den funktionalen Anforderungen. Man könnte jetzt einwerfen, dass sich Architektur auf die Erfüllung der nicht-funktionalen Anforderungen beschränkt. Diese verbreitete Sichtweise greift jedoch zu kurz und hat schon in vielen Projekten zu unzureichenden Architekturen geführt. Es ist korrekt, dass Architektur die eine Stelle ist, die sich um die Umsetzung der nicht-funktionalen Anforderungen kümmert. Das bedeutet aber nicht, dass sich ein Architekt darauf beschränken kann. Um die Komplexität und Änderbarkeit eines Systems über seinen Lebenszyklus beherrschen zu können, muss er insbesondere auch die funktionalen Anforderungen berücksichtigen. Mindestens 90 % aller Anforderungen im Lebenszyklus eines typischen Anwendungssystems sind fachlich motiviert. Wie soll ein Architekt eine gute Architektur erstellen, wenn er die fachlichen Anforderungen nicht verstanden und angemessen in seiner Architektur berücksichtigt hat?

Wie lassen sich diese vielfältigen Aufgaben in der Praxis umsetzen? Hier finden sich sehr viele Methoden mit vielen Varianten, wobei sich zwei Herangehensweisen unterscheiden lassen: *Strukturierung* und *Alignment*. Diese beiden Herangehensweisen müssen unterschieden werden, um die Frage beantworten zu können, ob ein agiles Team einen Architekten braucht. Abbildung 1 fasst die Ziele, Aufgaben und Herangehensweisen noch einmal zusammen.

Strukturierung

Warum ist Struktur so wichtig? Strukturierung ist eine Reaktion auf eine Beschränkung des menschlichen Gehirns. Unser Gehirn kann nur eine gewisse Menge nicht weiter strukturierter Informationen auf einmal erfassen und damit arbeiten. Haben wir z. B. eine größere Menge an Dokumenten, fangen wir automatisch an, diese zu sortieren, in Ordnern zu organisieren, die Ordner in Bereichen von Schränken usw. Das müssen wir tun, da wir sonst den Überblick verlieren würden. Vielleicht würden wir in einer überschaubaren Anzahl an Dokumenten (weniger als 500) auch ohne eine große Strukturierung am nächsten Tag noch so ziemlich alles wiederfinden, aber spätestens, wenn wir nach einer Unterbrechung von mehreren Monaten wieder an unserem unsortierten und unstrukturierten Dokumentenstapel weiterarbeiten, müssten wir komplett von vorne anfangen.

Genauso geht es uns bei der Softwareentwicklung, bei der wir in einem normalen Anwendungssystem mit tausenden und mehr verschiedenen Information und Details konfrontiert sind, die wir alle irgendwie im Blickfeld behalten müssen. Ohne eine Form der Strukturierung ist da selbst der genialste Entwickler irgendwann rettungslos verloren. Daher strukturiert er sich sein System – implizit oder explizit – in irgendeiner Form: etwa in Methoden, Klassen, Modulen, Komponenten, in bestimmten Verfahren, wie er auf Datenbanken zugreift, wie er Schnittstellen bedient, wie er die Benutzungsoberfläche steuert usw. Wie gut die gewählte Struktur war, kann man erkennen, wenn der Entwickler nach drei oder sechs Monaten Arbeit an einem anderen System in dem alten System eine Änderung umsetzen soll. Wenn er dann erst einmal mehrere Tage benötigt, um seinen Code zu verstehen, war die Struktur wahrscheinlich nicht so gut gewählt.

Den Fall, dass nur ein Entwickler an einem Anwendungssystem arbeitet, gibt es

heutzutage nur noch selten. Meistens sind mehrere, manchmal sogar hunderte Entwickler an einem Anwendungssystem beteiligt. In diesem Fall müssen sich *alle* im Quelltext orientieren und darin Änderungen und Erweiterungen umsetzen können. Es muss also eine Struktur geben, an der sich alle Entwickler orientieren können, und die ihnen hilft, ihre Aufgaben effizient und effektiv umsetzen zu können. Sie müssen schnell die Stellen im Code finden können, an denen sie ihre Modifikationen vornehmen, und sie benötigen einen Rahmen, wie sie ihre Modifikationen vornehmen, damit diese die Struktur nicht verletzen und damit das resultierende System immer noch für alle beteiligten Entwickler verständlich bleibt. Architektur im Sinne von Strukturierung ist daher in jedem nicht-trivialen System immer vorhanden, egal ob sie implizit oder explizit, vor oder während der Entwicklung erstellt worden ist.

In der Praxis bedeutet Strukturierung das Bilden und Verfeinern von Strukturen, das Auflösen von Problemen, die bei der Verfeinerung eventuell sichtbar werden, sowie – im Idealfall – das regelmäßige Vereinfachen der Strukturierung z. B. durch Refaktorisierungen. Dies findet praktisch vollständig in der Lösungsdomäne statt. So sind auch die üblicherweise in diesem Kontext genannten Hilfsmittel (vgl. z. B. [Mar]) Anleitungen zur (verfeinerten oder verbesserten) Strukturierung der Lösung aus Sicht der Lösungsdomäne. Dieser Teilaspekt der Architektur wird typischerweise auch nur betrachtet, wenn Architektur und Design als die gleiche Tätigkeit auf unterschiedlichen Detailebenen dargestellt werden.

Alignment

Für viele beschränkt sich Architektur auf den Bereich „Strukturierung" – sie denken nur an den strukturellen Aspekt, manchmal noch unter expliziter Berücksichtigung einiger nicht-funktionaler Anforderungen wie Skalierbarkeit oder Portabilität. Diese Herangehensweise ist jedoch nur „notwendig, aber nicht hinreichend". Architektur umfasst natürlich die Strukturierung von Systemen auf einer bestimmten Ebene, nur wissen wir bei der beschriebenen Herangehensweise letztlich nicht, ob die gewählte Architektur im Sinne der Architekturziele „gut" oder „schlecht" ist. Die Beantwortung dieser Frage ist nur bei einer weiter greifenden Herangehensweise an Architektur möglich, die ich als *Alignment* bezeichne.

Architektur	Design
Die wesentlichen von der Aufgabenstellung vorgegebenen Elemente.	Entscheidungen, die kompatibel mit der Architektur sind.
Eine andere Architektur impliziert eine andere Aufgabenstellung.	Unterschiedliche Designs können die gleiche Aufgabenstellung adressieren.
Definiert eine Klasse zulässiger Lösungen.	Definiert eine einzelne spezifische Lösung.
Betrifft Angemessenheit oder Eignung gemäß Definition durch die Aufgabenstellung.	Betrifft ingenieursmäßige Optimierung innerhalb der durch die Architektur vorgegebenen Grenzen.
Primäre Aufgabe des Architekten ist es, die korrekten Folgerungen (Inferenzen) zu ziehen.	Primäre Aufgabe des Designers ist es, die richtigen Entscheidungen zu treffen.
Architektur wird von Architekten gemacht.	Design wird von Entwicklern gemacht.
Primäre Zielgruppe sind die Stakeholder der Aufgabenstellung und der Lösung. Diese können Designer und Entwickler beinhalten.	Primäre Zielgruppe sind die Entwickler.
Betrifft das ganzheitliche System in seinem umgebenden Kontext.	Betrifft Komponenten und Subsysteme des Systems.

Tabelle 1: Architektur (Alignment) und Design (Strukturierung) im Vergleich (nach [Feh08]).

Alignment – also Ausrichtung, Justierung oder Einklang – bedeutet, einfach ausgedrückt, dass Architektur sicherstellen muss, dass die Lösung zur Aufgabenstellung passt. Wie geschieht das? Bei einem normalen Softwareentwicklungsprojekt gibt es eine ganze Reihe unterschiedlicher beteiligter Parteien (*Stakeholder*), die verschiedene Interessen und Anforderungen bezüglich der zu entwickelnden Anwendung haben:

- Die Anwender wollen ein System, das sie optimal bei ihrem Tagesgeschäft unterstützt.
- Die Fachbereiche möchten neue Geschäftsideen und -anforderungen möglichst schnell umsetzen können.
- Der Sponsor möchte möglichst viel Geschäftswert für sein Geld.
- Der Projektmanager möchte ein möglichst risikoarmes Projekt.
- Die Systemplaner wollen möglichst keine unbekannten Infrastrukturkomponenten und klar definierte Lastprofile für die Systemplanung.
- Die Administratoren wollen ein möglichst einfach betreibbares System.
- Die Entwickler möchten neue Technologien einsetzen können und möglichst abwechslungsreiche Aufgaben.

Die Hauptaufgabe eines Architekten ist es, alle diese unterschiedlichen Interessen und Anforderungen gegeneinander abzuwägen und auszubalancieren sowie Widersprüche aufzulösen und geeignete Kompromisse zu finden. Das ist in erster Linie keine technische oder konzeptionelle Aufgabe, sondern hat sehr viel mit dem Verstehen der verschiedenen Anforderungsdomänen sowie mit Kommunikation, Moderation und Konfliktmanagement zu tun, also mit dem Umgang mit Menschen und ihren Bedürfnissen.

Diese abgestimmten und ausbalancierten Interessen und Anforderungen muss ein Architekt verwenden, um daraus eine Lösungsstruktur zu entwerfen, d. h. er muss die Transformation der Anforderungen in die Lösungsdomäne vornehmen. Beim Erstellen des Konzepts und dem Vermitteln an die beteiligten Entwickler können neue Probleme und Widersprüche entstehen, die den Erfordernissen der Lösungsdomäne und der Entwickler als eine wichtige beteiligte Partei geschuldet sind. Diese Probleme und Widersprüche muss der Architekt abwägen, ausbalancieren, auflösen und geeignete Kompromisse finden. Bei all diesen Tätigkeiten muss er im Auge behalten, dass die Komplexität der Lösung nicht überhandnimmt und die Änderbarkeit gewährleistet bleibt.

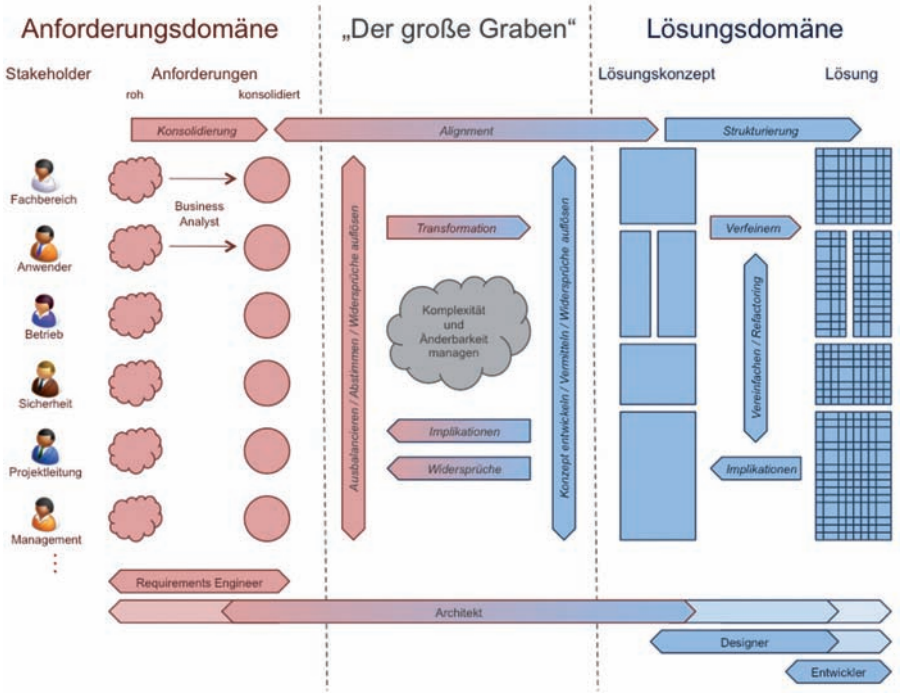

Abb. 2: Alignment und Strukturierung auf dem Weg von der Anforderung zur Lösung.

Alignment als Herangehensweise liefert im Unternehmenssinne einen Mehrwert, also einen Geschäftswert. Im Gegensatz zur reinen Strukturierung kann *Alignment* aber nicht implizit erfolgen, sondern immer explizit. „Zufälliges” *Alignment* gibt es nicht bzw. es ist zumindest sehr unwahrscheinlich.

Risiken und Nebenwirkungen

Der große Unterschied zwischen *Alignment* und Strukturierung ist, dass *Alignment* sich primär damit beschäftigt, die verschiedenen Problemdomänen zu verstehen und eine geeignete Transformation in die Lösungsdomäne zu finden, während Strukturierung sich primär damit befasst, die Lösungsdomäne zu organisieren.

Diese Unterscheidung wird auch an verschiedenen anderen Stellen aufgegriffen, so z. B. von Leonard Fehskens in [Feh08], der zwischen Architektur (*Alignment*) und Design (*Strukturierung*) unterscheidet. Interessant sind die ergänzenden Merkmale, anhand derer er die beiden Tätigkeiten unterscheidet (siehe Tabelle 1).

So ist es das Ziel von Architektur, eine Lösungsklasse zu finden, die den Anforderungen gerecht wird, während Design eine spezielle Lösung innerhalb der durch die Architektur vorgegebenen Grenzen definiert. Während eine andere Architektur immer eine andere Aufgabenstellung impliziert, können verschiedene Designs die gleiche Aufgabenstellung abbilden. Diese Unterscheidung macht die – zugegebenermaßen fließenden – Grenzen zwischen Architektur und Design ziemlich gut greifbar.

Abbildung 2 zeigt die verschiedenen Herangehensweisen und die damit verbundenen Rollen im Überblick. Die eventuell an einem Projekt beteiligten Business-Analysten bzw. Requirements-Ingenieure befassen sich mit der Organisation und Konsolidierung der Anforderungen, d. h. sie bewegen sich komplett in der Anforderungsdomäne (bzw. im Fall des Business-Analysten nur in der Fachdomäne). Sie lösen in der Regel auch keine Widersprüche zwischen den verschiedenen Problemdomänen auf der Anforderungsseite auf, sondern konsolidieren nur die Anforderungen der einzelnen Domänen. Die Strukturierung findet vollständig auf der Lösungsseite statt und hat keinen oder wenig Bezug zur Anforderungsseite. Das *Alignment* hat die Aufgabe, den „großen Graben” zwischen Anforderungs- und Lösungsseite zu überbrücken und die auf dem Weg auftretenden Widersprüche und Probleme aufzulösen – sollte es keinen Requirements-Ingenieur oder Business-Analysten im Projekt geben, muss ein *Alignment*-Architekt sich in der Regel auch noch um die Konsolidierung der Anforderungen kümmern.

Alle genannten Rollen werden für die erfolgreiche Umsetzung von Anforderungen in eine tragfähige und nachhaltige Lösung benötigt. In der Praxis erlebt man es allerdings häufig, dass Architektur mit Strukturierung, d. h. mit Design, gleichgesetzt wird und dass der *Alignment*-Aspekt vernachlässigt wird. Damit fehlt ein wichtiges Glied in der Kette von den Anforderungen zur Lösung. In der Folge führt das häufig zu einer nicht optimalen Strukturierung der Lösung im Sinne der Anforderungen, was wiederum dazu führt, dass die Architekturziele nicht erreicht werden. Das bedeutet in der Regel höhere Kosten, eine höhere Fehleranfälligkeit, schlechtere Änderbarkeit und eine geringere Zufriedenheit der Stakeholder.

Was heißt es nun, einen Architekten mit an Bord zu haben, der das Alignment im Fokus hat? Die Chance, die Architekturziele zu erreichen, ist damit deutlich höher. Die Frage ist allerdings, ob man diese Rolle mit einer dedizierten Person besetzen muss oder ob sie vom Entwicklerteam übernommen werden kann. Zur Beantwortung dieser Frage muss man einen Blick auf die für diese Rolle erforderlichen Fähigkeiten werfen: Neben sehr guten Kommunikationsfähigkeiten und einer guten Portion Fingerspitzengefühl sollte ein *Alignment*-Architekt ein gutes Verständnis für die Aufgaben und inhärenten Treiber der verschiedenen Parteien haben, um zum einen die Bedürfnisse verstehen und bewerten zu können, und zum anderen auf Augenhöhe mit den Parteien kommunizieren zu können. Optimalerweise hat ein solcher Architekt in der Praxis schon in möglichst vielen dieser Rollen gearbeitet oder zumindest eng mit diesen Rollen zusammengearbeitet. Das ist notwendig, damit er die notwendige Glaubwürdigkeit bei der Gegenseite hat und das Verständnis für die Probleme und Bedürfnisse der Gesprächspartner besitzt, um die erforderlichen Abstimmungen durchführen und Alternativen diskutieren zu können.

Man kann dies auch gut anhand von Dana Bredemeyers „Architect Competency Framework” (siehe Tabelle 2) nachvollziehen. Die Grundidee hinter dem Framework ist, dass ein Architekt im Laufe seiner Entwicklung Kompetenz in einer Reihe von Disziplinen aufbauen sollte. Typischerweise beginnt er in der obersten Disziplin „Technologie”. Hat er hier eine profunde Kompetenz aufgebaut, beginnt er, zusätz-

	Was Du kennst	Was Du tust	Was Du bist
Technologie	• Tiefes Verständnis der IT-Domäne und der einschlägigen Technologien • Verständnis, welche technischen Themen erfolgskritisch sind • Entwicklungsmethoden und Modellierungstechniken	• Modellieren • Tradeoffs analysieren • Prototypen/Experimente/Simulationen • Architekturdokumente und Präsentationen vorbereiten • Technologietrend-Analysen und Roadmaps • System-Sichtweise einnehmen	• Kreativ • Investigativ • Praktisch/pragmatisch • Einsichtig • Tolerant bzgl. Mehrdeutigkeiten, bereit zurückzugehen, mehrere Lösungsmöglichkeiten suchen • Gut im Arbeiten auf einer abstrakten Ebene
Beratung	• Erhebungstechniken • Beratungs-Frameworks und Methoden	• „Trusted Advisor"-Beziehungen aufbauen • Verständnis, was die Entwickler von der Architektur wollen und benötigen • Entwicklern helfen, den Wert von Architektur zu verstehen und wie sie sie erfolgreich nutzen können • Junior-Architekten coachen	• Dem Erfolg der anderen verpflichtet • Einfühlsam, zugänglich • Ein effektiver Berater für Veränderungen, prozesserfahren • Ein guter Mentor und Lehrer
Strategie	• Die Geschäftsstrategie des Unternehmens inklusive Begründung • Den Wettbewerb (Produkte, Strategien, Prozesse) • Die Geschäftsmethoden des Unternehmens	• Geschäftsstrategie beeinflussen • Geschäftsstrategie in IT-Vision und Strategie übersetzen • Kunden- und Markttrends verstehen • Kunden-, Organisations- und Geschäftsanforderungen an die Architektur erfassen	• Visionär • Unternehmerisch denkend
Organisationspolitik	• Wer die Schlüsselpersonen in der Organisation sind • Was die Personen wollen, sowohl geschäftlich als auch persönlich	• Kommunizieren, kommunizieren, kommunizieren • Zuhören, Netzwerke pflegen, beeinflussen • Die Vision verkaufen, die Vision am Leben erhalten • Immer wieder die Meinung aller wichtigen Beeinflusser der Architektur abholen	• Fähig, von verschiedenen Standpunkten aus zu sehen und an diese zu verkaufen • Souverän und gut verständlich • Ambitioniert und engagiert • Geduldig und ungeduldig • Belastbar • Sensibel, wo die Macht im Unternehmen ist und wie sie fließt
Führung	• Dich selbst	• Team-Kontext definieren (Vision) • Entscheidungen treffen • Teams aufbauen • Motivieren	• Du und andere sehen Dich als eine Führungskraft • Charismatisch und glaubwürdig • Du glaubst, es kann und sollte gemacht werden und Du kannst die notwendigen Aktivitäten führen • Du bist engagiert, mit Leib und Seele dabei • Du siehst den gesamten Aufwand in einem breiteren geschäftlichen und persönlichen Kontext

Tabelle 2: *Architekten-Kompetenz-Framework (nach [Bre02]).*

lich Kompetenz in der folgenden Disziplin „Beratung" aufzubauen, gefolgt von „Strategie", „Organisationspolitik" und „Führung". Design bedarf der Disziplin „Technologie" und vielleicht noch ein wenig „Consulting". Architektur im Sinne von *Alignment* bedarf aber zumindest der ersten vier Disziplinen, optimalerweise auch der letzten Disziplin „Führung" für eine erfolgreiche Umsetzung.

Um Kompetenz in diesen Disziplinen aufzubauen, benötigt man Zeit: Man muss das erforderliche Wissen erwerben und die notwendigen guten und schlechten Erfahrungen machen. Zusätzlich reichen für ein erfolgreiches Arbeiten als Architekt Erfahrungen in der Softwareentwicklung nicht aus – darüber hinaus benötigt man Erfahrungen in den Aufgabengebieten möglichst vieler der anderen Stakeholder. Das sind aber Fähigkeiten, die nicht jeder haben kann, wie man leicht nachvollziehen kann. Insbesondere sind die Fähigkeiten, die ein Architekt benötigt, sehr breit gefächert. Das sind ganz andere Fähigkeiten, als sie ein Softwareentwickler üblicherweise braucht. Ein Softwareentwickler benötigt für seine Arbeit relativ fokussierte, dafür aber sehr tiefe Fähigkeiten in seiner Domäne. Dieser Unterschied in den benö-

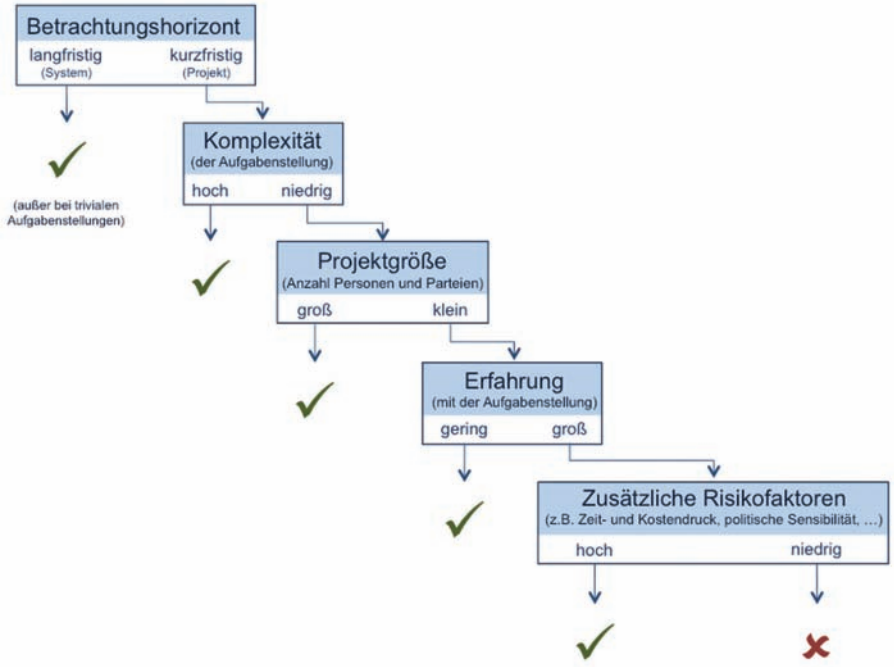

Abb. 3: Entscheidungsbaum für den Einsatz eines Architekten.

tigten Fähigkeiten verdeutlicht auch, dass die hier beschriebene Architektenrolle üblicherweise nicht – wie immer wieder behauptet – von einem (agilen) Entwicklerteam übernommen werden kann.

Wer braucht einen Architekten?

Einen „Strukturierungsarchitekten" – also einen Designer – benötigt man immer. Für ein nachhaltig wartbares und änderbares System ist es unabdingbar, ein gegebenes Lösungskonzept unter Einhaltung bestimmter Prinzipien weiter bis in den Code hinein zu verfeinern und sicherzustellen, dass sich das Design im Code wiederfindet. Allerdings muss diese Rolle nicht unbedingt von einer oder mehreren Personen dediziert wahrgenommen werden. Das Design kann häufig sehr gut von den erfahrenen Entwicklern eines Teams übernommen werden, also den Personen, die die Erfahrung und das Wissen um gute Strukturierung haben und die nah genug am Team dran sind, um dies in das gesamte Team hereinzutragen.

Die Frage nach einem Architekten (im Sinne von *Alignment*) ist schwieriger zu beantworten. Grundsätzlich ist es auch immer notwendig sicherzustellen, dass die Lösung (über ihren Lebenszyklus) zur Aufgabenstellung passt. Da der Architekt in der beschriebenen Form aber ganz spezielle, in der Regel nicht so häufig vorhandene Fähigkeiten benötigt, sollte man sich überlegen, wann man den Aufwand betreibt, diese Rolle explizit zu besetzen. Dazu muss man eine Reihe von Risikofaktoren betrachten:

- Zeithorizont: kurzfristig (Projekt-Scope) oder langfristig (System-Scope)
- Komplexität des zu erstellenden Systems: niedrig oder hoch
- Anzahl der beteiligten Parteien und Personen: klein oder groß
- Erfahrung aller beteiligten Parteien mit der Aufgabenstellung: gering oder groß
- Zusätzliche Risikofaktoren, z. B. Zeit- und Kostendruck oder politische Sensibilität des Themas: niedrig oder hoch

Es lassen sich sicherlich noch weitere Risikofaktoren identifizieren, aber die genannten Faktoren sind hinreichend, um die Notwendigkeit eines Architekten zu beschreiben: Je höher das Risiko eines kurz- oder auch langfristigen Fehlschlags aufgrund der Ausprägung eines der beschriebenen Faktoren wird, desto wichtiger ist es, einen Architekten einzusetzen. Anders ausgedrückt: Die Aufgabe eines Architekten ist es, das Risiko eines kurzfristigen Fehlschlags (Projektfehlschlag) oder eines langfristigen Fehlschlags (Systemfehlschlag) aufgrund der beschriebenen Risikofaktoren zu minimieren.

Hat man es beispielsweise mit einem einfachen Auskunftssystem zu tun, das eine Handvoll Entwickler erstellen, die auch schon die Thematik fachlich und technisch gut kennen, ist es wahrscheinlich nicht erforderlich, explizit einen Architekten einzusetzen. Ganz anders sieht es bei einem hoch komplexen Unterfangen mit vielen beteiligten Parteien aus, bei dem fachlich und technisch an diversen Stellen Neuland betreten wird und sich die Parteien untereinander nicht einig sind, wie die Lösung aussehen soll. Hier ist der Einsatz eines Architekten unabdingbar, um überhaupt die Chance auf einen Projekterfolg zu haben, vom langfristigen Erfolg (Stichwort „Lebenszyklus") ganz zu schweigen.

Abbildung 3 stellt das zuvor Beschriebene noch einmal als Entscheidungsbaum dar. Bei einer langfristigen Betrachtung des Systems über die Grenzen eines einzelnen Projekts hinaus, wie sie die Ziele der Architektur vorgeben, ist es außer bei sehr einfachen Unterfangen immer empfehlenswert, einen Architekten hinzuzuziehen. Hierbei gibt es das Problem, dass die wenigsten Unternehmen heutzutage eine Balance aus kurz- und längerfristigen Zielen anstreben, sondern meistens nur die kurzfristigen Ziele mit schnellem ROI – oftmals im unterjährigen Bereich – im Fokus haben. Ob diese kurzfristige Denkweise für Systeme, die häufig 15 und mehr Jahre in Produktion bleiben, uneingeschränkt sinnvoll ist, ist eine andere Diskussion.

Bei einer kurzfristigen Betrachtung, d. h. aus Projektsicht, hängt die Notwendigkeit eines Architekten von der Komplexität der Aufgabenstellung und der Organisation sowie der Erfahrung der Beteiligten mit der Aufgabenstellung ab. Weitere Risikofaktoren – wie hoher Zeit- und Kostendruck oder politische Sensibilität der Aufgabenstellung – setzen die Grenze herunter, ab der ein Architekt hinzugezogen werden sollte.

Braucht ein agiles Team einen Architekten?

Am einfachsten lässt sich diese Frage beantworten, wenn man betrachtet, welche Faktoren den Einsatz einer agilen Vorgehensweise treiben. Abbildung 4 zeigt diese Faktoren zusammen mit den zuvor betrachteten Faktoren für den Einsatz eines Architekten. Hier gibt es eine Reihe von Punkten auf beiden Seiten, die nur eines der beiden Themen treiben. So sind eine hohe

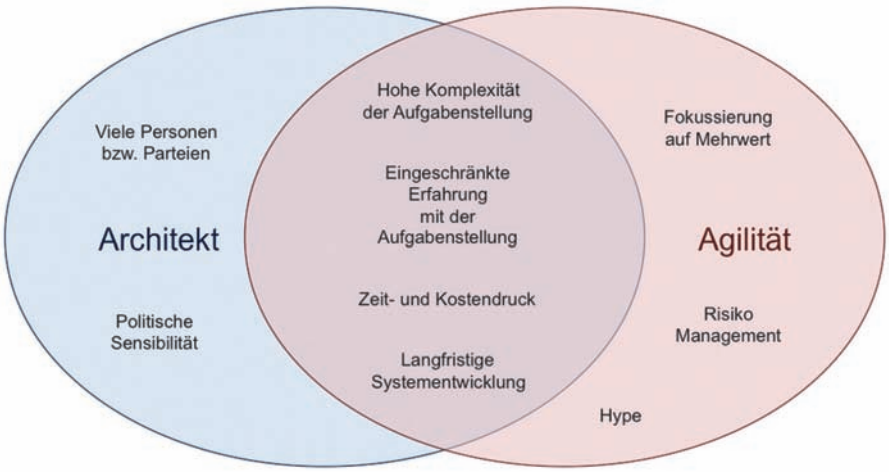

Abb. 4: Faktoren für den Einsatz eines Architekten bzw. von Agilität.

Anzahl an Personen bzw. Parteien oder die politische Sensibilität der Aufgabenstellung keine Treiber für den Einsatz einer agilen Vorgehensweise. Auf der anderen Seite sind eine Fokussierung auf das Produzieren von Mehrwert und aktives Risikomanagement (z. B. durch verkürzte Feedback-Zyklen) zwar Treiber für Agilität, aber nicht für Architektur. Auch den aktuellen Hype um das Thema Agilität darf man nicht aus dem Auge lassen. Es gibt derzeit sicherlich eine ganze Reihe von Projekten, deren primärer Treiber ist, dass es derzeit en vogue ist, agil zu sein.

Aber schaut man sich einmal die Schnittmenge in **Abbildung 4** an, gibt es eine Menge an Faktoren, die als Treiber sowohl für den Einsatz eines Architekten als auch für eine agile Vorgehensweise wirken. Am wichtigsten sind dabei die Komplexität der Aufgabenstellung und die Erfahrung der Beteiligten mit ihr. Je höher die Komplexität und je geringer die Erfahrung der Beteiligten mit der Aufgabenstellung sind, desto wichtiger ist der Einsatz entsprechender Maßnahmen zum aktiven Risikomanagement. Zwei zentrale Maßnahmen dabei sind der Einsatz agiler Vorgehensweisen und der eines Architekten. Ein weiterer gemeinsamer Treiber ist ein hoher Zeit- und Budgetdruck, die das Risiko eines Fehlschlags erhöhen.

Letztlich ist auch die langfristige Systementwicklung ein gemeinsamer Treiber. Agile Vorgehensweisen sind häufig eine gute Wahl für eine langfristige Systementwicklung. XING etwa verwendet ein agiles Vorgehen, um seine Plattform in vielen kurzen Release-Zyklen langfristig weiterzuentwickeln und zu optimieren (vgl. z. B. [Rep09]). Durch die konsequente Mehrwert-Fokussierung und die schnellen Feedback-Zyklen ist Agilität für solche Unterfangen meistens eine gute Wahl. Um allerdings langfristig die Beweglichkeit der Agilität sicherzustellen, muss sich ein Architekt darum kümmern, dass Komplexität und Änderbarkeit der Lösung gewährleistet bleiben. Als grobe Empfehlung möchte ich festhalten:

- In Vorhaben, in denen die Aufgabenstellung relativ zur Erfahrung der Beteiligten nicht sonderlich komplex ist, in denen Zeit- und Budgetdruck handhabbar sind und bei denen die langfristige Systementwicklung nicht im Vordergrund steht, kann man in der Regel auf einen expliziten Architekten verzichten. Die Aufgabe kann meistens von einem oder mehreren Senior-Entwicklern mit guten Design- und vernünftigen Beratungskenntnissen wahrgenommen werden.
- Werden die Rahmenbedingungen aber komplexer – sei es durch viele beteiligte Stakeholder, durch unklare Anforderungen oder durch den schieren Aufgabenumfang –, wird Architektur im Sinne von *Alignment* für eine erfolgreiche Umsetzung immer wichtiger. Da dies eine Aufgabe ist, die eine oder sogar mehrere Personen vollständig auslasten kann und die aufgrund der erforderlichen Fähigkeiten oftmals nicht sinnvoll von einem reinen Softwareentwickler wahrgenommen werden kann, sollte man hier einen oder mehrere explizite Architekten einzusetzen.

Das ist jetzt kein Ruf nach einem „Big Design Upfront" oder den berüchtigten Architektur-Elfenbeintürmen, ganz im Gegenteil! Elfenbeinturm-Architekten à la „Matrix Reloaded", die alles im stillen Kämmerlein von A bis Z durchdenken – möglichst bevor die erste Zeile Code geschrieben wird –, sind für *Alignment* nicht hilfreich. Die Hauptaufgabe des Architekten ist Kommunikation und Abstimmung. Das kann man als latent autistischer Elfenbeinturm-Bewohner nicht leisten. Sinnvollerweise ist ein Architekt Teil des Teams, allein schon deshalb, weil die Entwickler die Personen sind, die die eigentliche Anwendung erstellen. Für einen Architekten bedeutet das, dass das Team für ihn ein extrem wichtiger *Stakeholder* ist. Eine Entscheidung, die er nicht mit dem Team abgestimmt hat, kann und wird das Team in der Regel nicht umsetzen, womit die Entscheidung ohne Nutzen ist, weil sie nie den Weg in die fertige Anwendung finden wird.

Dennoch ist es zu einfach, pauschal zu sagen, dass Architektur „vom Team" übernommen wird. Aufgrund der sehr unterschiedlichen Fähigkeiten von Architekten und Softwareentwicklern ist der Architekt in der beschriebenen Form meistens eine dedizierte Person. Wenn die Architekturaufgaben keine Vollzeitbeschäftigung versprechen, kann er auch noch andere Aufgaben wahrnehmen, z. B. Design oder auch Implementierung (wie in **Abb. 2** angedeutet). Nur ist es häufig so, dass die Architekturaufgaben in Umfeldern, die einen Architekten erforderlich machen, so sehr zunehmen, dass der Architekt praktisch keine Zeit mehr für andere Aufgaben hat. Das ist in der Regel aber auch nicht schlimm, da das weiterführende Design, das der Architektur folgt, häufig auch noch von anderen Teammitgliedern übernommen werden kann. Wichtig ist dabei nur, dass keine Informationsbruchstelle zwischen dem Architekten und dem Rest des Teams entsteht.

Zusammenfassung

Die Aussage, dass ein agiles Team keinen Architekten benötigt, lässt sich nicht pau-

schal halten. Diese Aussage bezieht sich in der Regel auf Architektur im Sinne von Strukturierung, d. h. auf Design. Diese in praktisch jedem Kontext wichtige Aufgabe kann tatsächlich häufig von den erfahreneren Entwicklern eines Teams wahrgenommen werden.

Ganz anders sieht es mit Architektur im Sinne von *Alignment* aus. Ein Architekt, der diese Aufgabe wahrnimmt, benötigt ganz andere Fähigkeiten als ein Softwareentwickler, weshalb diese Aufgabe nicht einfach vom Entwicklerteam übernommen werden kann, sondern in der Regel von einer dedizierten Person mit den entsprechenden Fähigkeiten.

Ob man einen Architekten benötigt, hängt primär von der Komplexität der Aufgabe und der Erfahrung der beteiligten Personen – über alle *Stakeholder*-Gruppen hinweg – mit der Aufgabenstellung zusammen. Da agile Verfahren insbesondere für komplexe Umfelder geeignet sind, in denen die Beteiligten noch nicht ausreichend Erfahrung mit der Aufgabenstellung haben, benötigen agile Teams häufig auch einen Architekten. ■

Literatur & Links

[Bre02] D. Bredemeyer, Architect Competency Framework, 2002, siehe: www.bredemeyer. com/ pdf_files/ArchitectCompetencyFramework. PDF

[Feh08] L. Fehskens, Re-Thinking Architecture, The Open Group Conference Munich, 2008

[Fow03] M. Fowler, Who needs an Architect?, 2003, siehe: http://martinfowler.com/ ieeeSoftware/whoNeedsArchitect.pdf

[ISO] ISO/IEC 9126, siehe z. B.: http://de.wikipedia.org/wiki/ISO_9126

[Kos03] J. Koskinen, Software Maintenance Cost, 2003, siehe: www.cs.jyu.fi/~koskinen/ smcosts.htm

[Lip09] M. Lippert, S. Roock, Entkoppelte Systeme sind änderbare Systeme, in: OBJEKTspektrum 4/2009

[Mar] R.C. Martin, The Principles of OOD, siehe: http://butunclebob.com/ArticleS.UncleBob .PrinciplesOfOod

[Pop] Poppendieck.LLC, Lean Software Development, siehe: www.poppendieck.com/

[Rep09] S. Reppin, J. Mainusch, SCRUM in der Community Realität bei XING – ein Erfahrungsbericht, in: JAX 2009

[Rep10] S. Reppin, R. Wirdemann, Kanban bei XING – Wer am Ziel ist, irrt sich, in: OBJEKTspektrum 2/2010

[Zac08] J. Zachman, Introduction to the Zachman Framework, Enterprise Architecture Conference London, 2008