HANSER

Leseprobe

zu

„Vorgehensmuster für Softwarearchitektur“

von Stefan Toth

Print-ISBN: 978-3-446-46004-1
E-Book-ISBN: 978-3-446-46009-6
E-Pub-ISBN: 978-3-446-46282-3

Weitere Informationen und Bestellungen unter

http://www.hanser-fachbuch.de/978-3-446-46004-1

sowie im Buchhandel

Inhalt

Geleitwort

Das Märchen vom agilen Architekten

„Heißt du etwa Rumpelstilzchen?“ –
„Das hat dir der Teufel gesagt, das hat dir der Teufel gesagt!“

(Kinder- und Hausmärchen der Brüder Grimm, 7. Auflage 1857)

Die schöne Müllerstochter, die aus Stroh Gold spinnen sollte, hat den Namen von Rumpelstilzchen nicht etwa geraten. Dazu hätte sie mehr als die drei bei den Gebrüdern Grimm beschriebenen Iterationen gebraucht. Sie hatte Wissen (nicht vom Teufel). Und als Ali Baba „Sesam, öffne Dich“ sprach, um in die Höhle mit unermesslichen Schätzen zu gelangen, hat er sich das auch nicht selbst ausgedacht. Er hat es sich abgeguckt von 40 Leuten, die schon mal drin waren in der Höhle. Er konnte auf deren Erfahrung zurückgreifen.

Der Schatz, um den es in diesem Buch von Stefan Toth geht, lässt sich verkürzt als Antwort auf folgende Frage beschreiben: Wie passt Softwarearchitekturmethodik zu einem zeitgemäßen Vorgehen? Oder besser noch: Wie können sie gemeinsam größeren Nutzen bringen?

Dass diese Frage viele bewegt, erlebe ich selbst regelmäßig in Workshops zu meinem Lieblingsthema Architekturdokumentation. Dort geht es darum, wie man Softwarearchitektur nachvollziehbar festhält und kommuniziert; in den Veranstaltungen drehen sich Fragen und Diskussionen regelmäßig darum, ob und wenn ja wie die gezeigten Zutaten zu einem agilen Vorgehen wie beispielsweise Scrum passen. Ganz allgemein können Sie das Interesse aber auch an den zahlreichen Blog- und Konferenzbeiträgen der letzten Jahre ablesen. Diese verknüpfen die Begriffe „agil“ (als griffigstes Wort für zeitgemäßes Vorgehen) und „Architektur“ mal mehr mal weniger pfiffig im Titel, etwa: „Jenseits des Elfenbeinturms – der agile Architekt“ oder „Architektur und agiles Vorgehen – ein Widerspruch?“. Und mehr noch ist es abzulesen an den vollen Sälen, wenn solche Vorträge stattfinden. Die Frage weckt Interesse. Gibt es gute Antworten?

Konferenzbeiträge – zumindest die, die ich gesehen habe – folgten in ihrem Ablauf häufig einem Schema: Zunächst werden die Begriffe „Agilität“ und „Architektur“ ausführlich definiert oder zumindest geklärt. Bei Agilität ist es dabei Folklore, das agile Manifest mit seinen berühmten vier Wertpaaren („Individuen und Interaktionen vor Prozessen und Werkzeugen“ etc.) auf eine Folie zu bannen. Dann wird der angebliche Widerspruch herausgearbeitet, der umso dramatischer ausfällt, je schwergewichtiger und klassischer das Verständnis von Softwarearchitektur, der zugrunde liegende Entwicklungsprozess und die damit verbundenen Artefakte in Notationen der 1990er-Jahre geschildert werden. Schließlich wird der Widerspruch durch sogenannte Best Practices aufgelöst („funktioniert doch super zusammen“).

Wolkige Tipps wie zum Beispiel: kein „Big Upfront Design“, auf die SOLID-Prinzipien achten, die Architektur iterativ und inkrementell entwickeln wie „den Rest“ auch ...

Die Zuhörer verlassen den Saal etwas enttäuscht. Alles richtig, gut und schön, aber wie genau machen wir das jetzt in unserem Projekt? Wo fangen wir an? Wenn schon kein Big Upfront Design, wie klein ist dann das richtige Small? Es liegt wohl auch, aber nicht nur am Format des Frontalvortrags und der oft kurzen Vortragsdauer (beliebt: 45 Minuten), dass die wirklich spannenden Fragen auf Konferenzen oft unbeantwortet bleiben. Mitunter fehlt es auch schlicht an ausreichenden praktischen Projekterfahrungen. Märchenstunde?

Für mich steht außer Zweifel, dass Stefan Toth die nötige Erfahrung besitzt. Er hat sehr unterschiedliche Projekte über einen längeren Zeitraum begleitet und zahlreiche einschlägige Workshops durchgeführt. Bei den Kunden wurde mal klassisch, mal agil, mal irgendwie dazwischen vorgegangen und auch die Branchen könnten unterschiedlicher kaum sein. Vom Finanzsektor bis zur Gaming-Plattform war alles dabei. Das Themenspektrum umfasste die methodische Softwarearchitektur vom Entwurf bis zur Bewertung von konkreten Architekturentscheidungen. So hat Stefan beispielsweise ein agiles Team begleitet und befähigt, regelmäßige Architekturbewertungen in ihren Entwicklungsprozess zu integrieren und eigenverantwortlich durchzuführen. Während viele bei Architekturbewertung sofort an schwergewichtige Methoden wie ATAM denken, wirkt hier nun ein schlankes, aber wirkungsvolles Set an Elementen, bei großer Akzeptanz im Team.

Das ist vielleicht auch schon die Grundidee des Buchs: Es gibt nicht den einen Weg für alle Projekte. Aber es gibt bewährte und schlanke Praktiken in Form von Puzzleteilen, die Nutzen stiften.

In einigen Projekten und Workshop-Situationen, eigentlich in zu wenigen, hatte ich als Kollege das Vergnügen, mit Stefan Toth zusammenzuarbeiten, und konnte wie die Mitarbeiter der Kunden an seinem Wissen und seinen Erfahrungen teilhaben. Und so freut es mich, dass Sie nun als Leser dieses Buchs ebenfalls davon profitieren können.

Denn Stefan Toth hat ein passendes Format zur Vermittlung seines Wissens und Könnens gewählt. Anders als es in einem knappen Vortrag möglich wäre, stellt er hier im Buch seine Ideen ausführlich dar und illustriert sie mit Beispielen. Gleichzeitig ist das Buch lebendig und kein langweiliger Schmöker. Stefan hat viel von seinem Witz in die Zitate und Antipatterns einfließen lassen, ohne dabei albern oder unsachlich zu werden. Die Idee, die einzelnen Zutaten als kombinierbare Muster darzustellen, macht die Inhalte nicht nur leichter erlernbar, sondern vor allem auch einzeln anwendbar. Das erleichtert den Start in Ihrem Projekt ungemein. Die einzelnen Zutaten sind trotzdem kein loses Schüttgut, sondern gut aufeinander abgestimmt und in ihrer Gesamtheit schlüssig. Ausdrucksstarke Visualisierungen – eine besondere Spezialität von Stefan – vermitteln komplizierte Inhalte gut erinnerbar und verknüpfen die einzelnen Muster.

Aus eigener Erfahrung kann ich sagen, dass die Erarbeitung und Aufbereitung von Inhalten in Form eines Buchs große Vorteile bietet (die hier auch ausgeschöpft wurden), aber auch einen nicht zu unterschätzenden Nachteil, zumindest verglichen mit Vorträgen oder einem Workshop. Es besteht die Gefahr, dass man als Autor weniger Feedback bekommt. Ich möchte Sie daher ermutigen, Erfahrungen, die Sie mit den dargestellten Praktiken machen konnten, zu teilen. Tauschen Sie sich aus, mit dem Autor und auch mit anderen Lesern.

Um zum Schluss noch mal auf Rumpelstilzchen zurückzukommen: In diesem Buch lernen Sie nicht, wie Sie aus Stroh Gold spinnen. Dafür viele andere Dinge, die Sie jetzt vermutlich

auch noch nicht können. Und es ist kein Märchen. Alles ist wahr. Wenn Sie mögen, schließen Sie das Buch nun kurz, sprechen mir nach: „Sesam, öffne Dich“, und schlagen es wieder auf. Und es tut sich tatsächlich ein reicher Schatz an Erfahrungswissen auf, der nur darauf wartet, Stück für Stück heraus in Ihr Projekt getragen zu werden. Mir bleibt nur noch, Ihnen viel Freude damit zu wünschen.

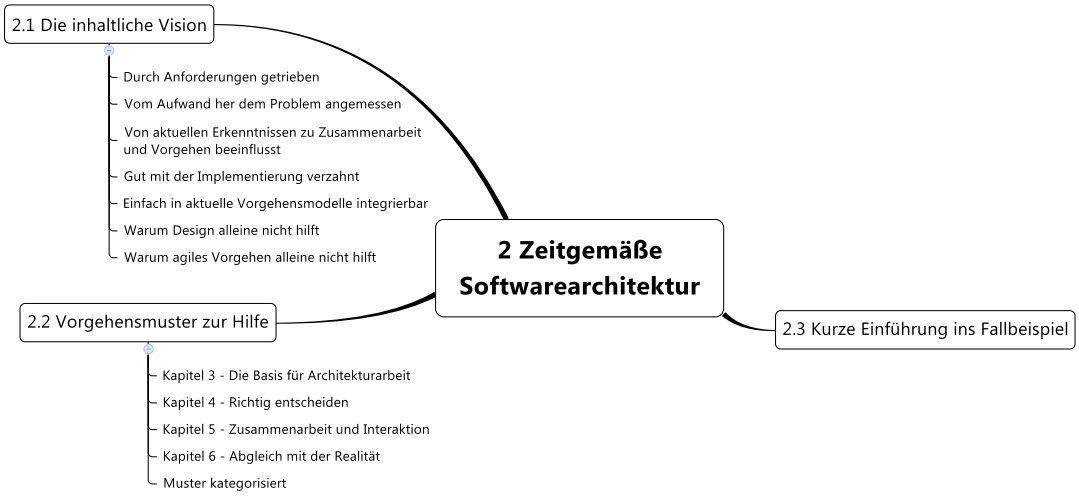
Stefan Zörner

2 Zeitgemäße Softwarearchitektur

Auf den nächsten Seiten tauchen Sie in die inhaltliche Vision des Buchs ein. Die übergreifende Idee hinter den 30 Vorgehensmustern für Softwarearchitektur ist in Abschnitt 2.1 detailliert dargestellt und mit den Konzepten der übrigen Kapitel verbunden.

Nach diesem vielleicht wichtigsten Abschnitt des gesamten Buchs zeige ich, welche Vorgehensmuster die beschriebene zeitgemäße Architekturarbeit ermöglichen. Abschnitt 2.2 gibt einen kompakten Überblick aller Muster des Buchs, inklusive Problem und Kurzbeschreibung. Außerdem werden die Kapitel, in welche die Muster eingegliedert sind, kurz vorgestellt.

Zum Abschluss stelle ich kurz das Fallbeispiel vor, das Sie durch alle Vorgehensmuster begleiten wird (Abschnitt 2.3).



2.1 Die inhaltliche Vision

Hinter den Vorgehensmustern dieses Buchs steht eine konsistente Vision zeitgemäßer Softwarearchitekturarbeit. Bereits die in Abschnitt 1.3 genannten Definitionen von Softwarearchitektur scheren nicht alle Softwareentwicklungsvorhaben über einen Kamm. Menge und Ausprägung von grundlegenden, risikoreichen Fragestellungen sind von System zu System unterschiedlich. Zeitgemäße Softwarearchitektur erkennt diese Individualität auf vielen Ebenen an und greift aktuelle Strömungen der Softwareentwicklung auf. Zeitgemäße Softwarearchitektur ist:

1. **Durch Anforderungen getrieben**
2. **Vom Aufwand her dem Problem angemessen**
   - In dynamischen Umfeldern nicht behindernd
   - In architektonisch risikoreichen Kontexten ausreichend fundiert
3. **Von aktuellen Erkenntnissen zu Zusammenarbeit und Vorgehen beeinflusst**
4. **Gut mit der Entwicklung verzahnt (Feedback!)**
5. **Einfach in aktuelle Vorgehensmodelle integrierbar**
   - Iterativ leistbar
   - In aktuellen Konzepten des Vorgehens verankert
   - Frei von behindernden oder umständlichen Ergänzungen

Ich greife diese Punkte im Folgenden auf, beschreibe sie etwas detaillierter und referenziere auf wichtige Vorgehensmuster.

2.1.1 Durch Anforderungen getrieben

Wenn Sie eine fachliche Methode ausimplementieren oder ein neues Feld im UI vorsehen, orientieren Sie sich an Wünschen und Anforderungen des Kunden. Dasselbe sollten Sie tun, wenn Sie Technologien auswählen oder Fremdsysteme anbinden. Was auch immer die grundlegenden Fragestellungen in Ihrem Fall sind: Lassen Sie sich von Anforderungen leiten.

Qualitätsanforderungen kommt dabei eine besondere Bedeutung zu. Sie beschreiben die nichtfunktionalen Aspekte der zu erstellenden Lösung, also *wie* eine Funktionalität bereitgestellt werden soll.[1] Soll die Funktionalität ohne Unterbrechung zur Verfügung stehen, sind Zuverlässigkeit und Verfügbarkeit wichtig. Wollen wir in Zukunft mehr Benutzer mit unserer Funktionalität beglücken, ist Skalierbarkeit spannend. Wollen wir verhindern, dass Unbefugte heikle Funktionalität nutzen, ist Sicherheit ein Thema. Diese Qualitätsmerkmale beziehen sich oft auf weite Systemteile oder sogar das Gesamtsystem. Zuverlässigkeit lässt sich nicht durch eine neue Klasse oder Komponente sicherstellen, die gesamte Anwendung und deren Basis müssen entsprechenden Prinzipien gehorchen.

[1] Der Begriff „Nichtfunktionale Anforderung“ erfährt immer größere Ablehnung in der Fachwelt. Ich werde in diesem Buch deshalb von „Qualitätsanforderungen“ oder „Qualitäten“ sprechen.

Qualität ist somit meist *querschnittlich* und betrifft viele bis alle Entwickler. Wir erreichen Qualitätsmerkmale durch den Einsatz der richtigen Technologien, Plattformen, Frameworks, Muster oder die breite Adaptierung von Arbeitsweisen. Das ist grundlegende Arbeit am Fundament. Entsprechende Entscheidungen sind weitreichend und oft aufwendig in der Umsetzung. Wir sind damit mitten in der Architekturdomäne und es ist wenig überraschend, dass Qualitätsanforderungen als *die* Architekturanforderungen gesehen werden.

Wie dieses Buch hilft

Jedes Entwicklungsvorhaben, egal wie leichtgewichtig oder agil, muss seine qualitativen Anforderungen kennen. In diesem Buch stelle ich einen leichtgewichtigen Ansatz zur Verankerung und gemeinsamen Bearbeitung dieser Anforderungen vor. Den Start macht **Kapitel 3** – „Die Grundlage von Architekturarbeit".

Die wichtigsten Muster für diesen Teil der Vision:

- 3.1 – Initialer Anforderungs-Workshop
- 3.3 – Szenarien als Architekturanforderungen
- 3.6 – Architekturarbeit im Backlog
- 4.4 – Architekturentscheidungen treffen

2.1.2 Vom Aufwand her dem Problem angemessen

Stellen Sie sich ein neu zu entwickelndes Produkt vor, das auf einem bekannten Technologiestack aufsetzt. Es gibt ein passendes, unternehmensspezifisches Applikationsframework, das einzige Umsetzungsteam hat bereits ähnliche Produkte gebaut und kennt die Domäne. Die zeitliche Planung ist realistisch und der Aufwand ist überschaubar. Dieses Vorhaben kommt wohl mit weniger Architekturaufwänden aus als ein Großprojekt, das sich um die Umsetzung einer neuartigen Flugsicherungssoftware kümmern soll. Im ersten Kontext ergeben sich wahrscheinlich weniger risikoreiche Fragestellungen. Das Umfeld ist weniger komplex, das zu lösende Problem und der Lösungsweg sind recht gut verstanden. Im Großprojekt hingegen sind einige Komplexitätstreiber zu finden – Architekturarbeit wird spannender. Bild 2.1 zeigt, wie sich Architekturaufwände und Komplexitätstreiber die Waage halten sollten.

Arbeit an der Softwarearchitektur hat das Ziel, gute Entscheidungen zum richtigen Zeitpunkt zu treffen und das Risiko teurer Irrwege zu minimieren. Zu hohe Aufwände für Architekturarbeit machen die Entwicklung schwerfällig, langsam und aufwendiger als nötig. Erstellen Sie etwa einen Prototypen für eine einfach umzusetzende Anforderung, verzögern Sie die Umsetzung und die damit verbundene Rückmeldung. Ihr Aufwand hat zudem wenig bis keinen Nutzen. Eine solche „Verschwendung" behindert vor allem in weniger komplexen, dynamischen Projekten und macht Sie starrer als nötig.

Auf der anderen Seite führt zu wenig Arbeit an der Softwarearchitektur zu zufälliger Architektur und potenziell zur Verfehlung wichtiger Ziele. In architektonisch risikoreichen Umfeldern muss folglich ausreichend fundierte Architekturarbeit geleistet werden.

Wichtig ist die richtige Balance, die sich für jedes Vorhaben anders gestaltet.

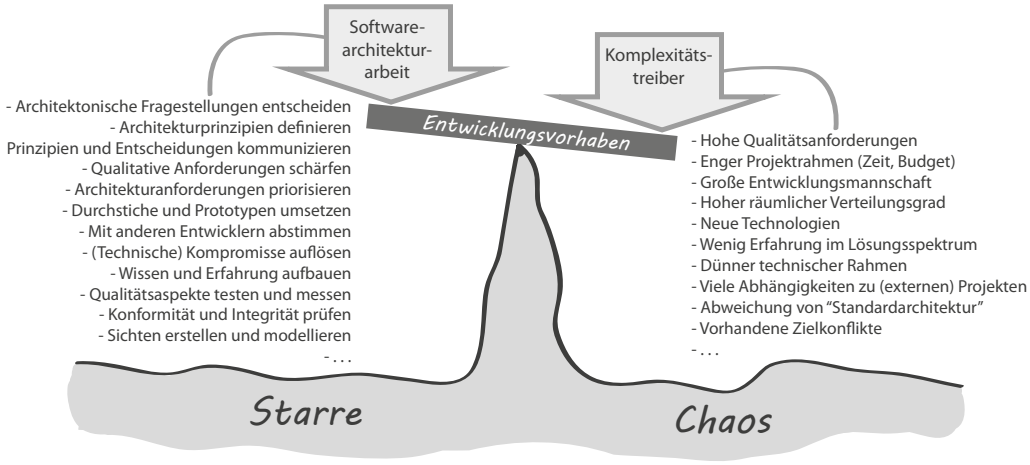


Bild 2.1 Das richtige Maß für Softwarearchitekturarbeit

Wie dieses Buch hilft

Das richtige Maß an Softwarearchitekturarbeit ist in jeder Entwicklungsphase interessant. In diesem Buch bespreche ich einerseits die Menge an vorab zu leistender Architekturarbeit, andererseits zeige ich, wie Sie bei konkreten Fragestellungen entscheiden, ob Architekturarbeit notwendig ist und wann diese Arbeit erfolgen sollte.

Die wichtigsten Muster für diesen Teil der Vision:

- 4.1 – Architekturarbeit vom Rest trennen
- 4.2 – Der letzte vernünftige Moment
- 4.3 – Gerade genug Architektur vorweg

2.1.3 Von aktuellen Erkenntnissen zu Zusammenarbeit und Vorgehen beeinflusst

Auch wenn die Wurzeln der Disziplin noch weiter zurückreichen, Softwarearchitektur ist ein Kind der 1990er-Jahre. Im universitären Umfeld und mit großer finanzieller Unterstützung des amerikanischen Verteidigungsministeriums wurden Muster, Sprachen und Methoden erarbeitet[2]. Weil Rollen- und Prozessmodelle ihre Blütezeit erlebten, konnte man die Disziplin relativ leicht einem „Architekten" zuschlagen.

Die Softwareentwicklung hat seit den 1990er-Jahren viel gelernt. Agile Softwareentwicklung, Lean Development oder auch die Organisationstheorie beinhalten viele Erkenntnisse zu Zusammenarbeit, Komplexität und Dynamik. Auch Softwarearchitektur kann als Disziplin von diesen Erkenntnissen profitieren.

[2] Eine zentrale Rolle spielte die Carnegie Mellon Universität mit ihren Veröffentlichungen - etwa [Sha96].

Wie wäre es mit Praktiken, die es ermöglichen, Architekturaufgaben effektiv auf mehrere Schultern zu verteilen? Praktiken, die dynamisches Vorgehen nicht bremsen? Was halten Sie von zeitgemäßen Methoden zur Minimierung von Unsicherheiten und Risiken? Und was wäre, wenn Softwarearchitektur so transparent wird, dass Sie stetig und gewinnbringend mit großen Entwicklungsgruppen oder Stakeholdern zusammenarbeiten können?

Wie dieses Buch hilft

Die herausragendsten Errungenschaften moderner Vorgehensmodelle betreffen gesteigerte Dynamik und Flexibilität. In diesem Buch zeige ich, wie Sie Architekturarbeit daran teilhaben lassen. Zentral ist dabei **Kapitel 5** – „Zusammenarbeit und Interaktion". Praktiken der anderen Musterkapitel unterstützen Sie bei der Anwendung dieser Konzepte.

Die wichtigsten Muster für diesen Teil der Vision:

- 4.6 – Risiken aktiv behandeln
- 5.1 – Informativer Arbeitsplatz
- 5.2 – Gemeinsam entscheiden
- 5.5 – Wiederkehrende Reflexion

2.1.4 Gut mit der Implementierung verzahnt (Feedback)

Bild 2.2 zeigt ein vereinfachtes Bild des generischen Entwicklungsprozesses, den ich in Abschnitt 7.2 genauer beschreiben werde. Er zeigt, wie Anforderungen die iterative Entwicklung speisen (Mitte) und der Umsetzungszyklus auslieferbare Software erstellt (rechts). Fundamentale Fragestellungen wandern vor der Implementierung durch den Architekturzyklus (links) bzw. liefern Erkenntnisse und Probleme aus der Umsetzung (rechts) die Grundlage

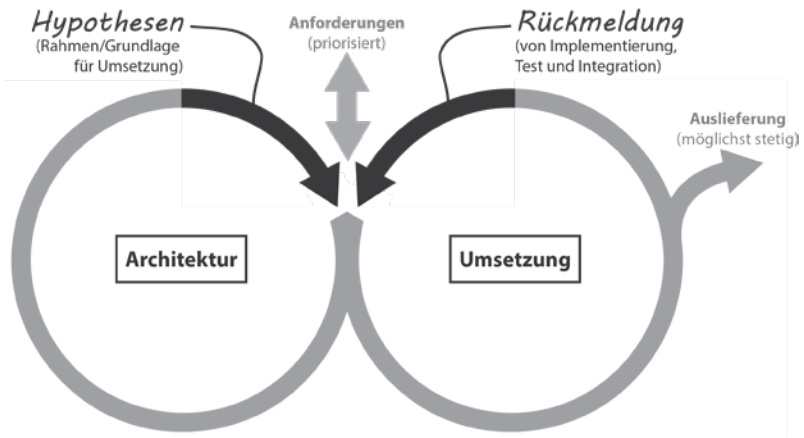


Bild 2.2 Iterative Architekturarbeit mit Umsetzung verzahnt

für gezieltere architektonische Betrachtungen (links). Ich durchwandere das Bild mit Hilfe eines vereinfachten Beispiels, um die Verzahnung von Architektur und Implementierung zu illustrieren.

Sie haben immer wieder wichtige Entscheidungen in der Entwicklung zu treffen. Nehmen wir zum Beispiel an, ein Teil Ihrer Applikation nimmt komplizierte Berechnungen vor. Sie haben den Applikationsteil bereits in Bausteine zerlegt und sehen sich nun mit Anforderungen konfrontiert, die hohe Flexibilität im Berechnungsablauf fordern. Da die Fragestellung nicht isoliert betrachtet werden kann und viele Bausteine betrifft, wandern Sie in den Architekturzyklus aus Bild 2.2.

Um möglichst lose Kopplung zu erreichen, entwerfen Sie einen einfachen Eventmechanismus. Sie sehen vor, dass Komponenten einen eigenen Berechnungszustand halten und bei Änderungen an diesem Zustand entsprechende Events feuern. Andere Bausteine können auf diese Events reagieren. Sie erstellen eine kleine Implementierung, die die Möglichkeiten Ihrer Plattform nutzt, um diese Idee umzusetzen. Es funktioniert.

An dieser Stelle definieren Sie die Idee als brauchbare Möglichkeit und entscheiden sich für eine breitere Umsetzung. Sie schaffen damit die Grundlage für Implementierungstätigkeiten, Sie stellen eine kommunizierbare Hypothese auf (siehe Bild 2.2, oben links). Es handelt sich um den ersten wichtigen Berührungspunkt zwischen Architektur- und Umsetzungsarbeit.

In der Umsetzung wenden Sie das Konzept auf Ihre Bausteine an (vielleicht nicht sofort auf alle). Sie versuchen, Zustandsübergänge zu definieren, eine produktivtaugliche Implementierung für den Zustand selbst zu kreieren und entwerfen fachliche Events. Erst hier haben Sie das Problem annähernd vollständig vor Augen: Sie erkennen, wie kompliziert sich Zustände teilweise zusammensetzen, welche Daten mit den Events übertragen werden müssen und wie diese Lösung mit anderen Konzepten Ihrer Bausteine zusammenwirkt. Haben Sie wichtige Teile umgesetzt, können Sie mit Tests eine Idee vom Laufzeitverhalten bekommen.

Hier ist der zweite wichtige Berührungspunkt zwischen Architektur und Implementierung: die Rückmeldung aus der Implementierung, samt den Erkenntnissen aus Integration und Test (siehe Bild 2.2, oben rechts). Sie sollten diese Rückmeldung *häufig* und *zeitnah* suchen. So prüfen Sie architektonische Hypothesen und minimieren den Raum für Annahmen und Spekulationen[3]. Technische oder konzeptionelle Probleme, die auf Implementierungsebene auftreten, stellen einen sekundären Architekturtreiber dar (neben den weiter oben besprochenen Anforderungen). Insgesamt entsteht eine gelebte Softwarearchitektur, die durch die Implementierung nicht verwässert, sondern bereichert wird. Hypothesen erhärten sich über das Feedback aus der Umsetzung und werden nach und nach zu breit akzeptierten Entscheidungen.

Zeitgemäße Softwarearchitektur zeichnet sich durch häufige und schlanke Durchläufe des Architekturzyklus aus. Die Übergänge an beiden Berührungspunkten zur Umsetzung sind gut verstanden und mit geringen Aufwänden verbunden.

[3] Es wird oft versucht, viel Architekturaufwand VOR der Entwicklung zu treiben, um bessere Vorhersagen zu erreichen. Die Erreichung von Qualitätsmerkmalen ist allerdings schwer vorhersagbar. Versuchen Sie es, verzögern Sie wahrscheinlich nur den Weg zur Wahrheit: der laufenden Applikation.

Wie dieses Buch hilft

Der schlanke, häufige Durchlauf des Architekturzyklus wird durch die Anforderungskonzepte aus Kapitel 3 ermöglicht. In Kapitel 4 – „Richtig entscheiden" finden Sie Hinweise zur Erarbeitung von „Hypothesen" und „Kandidaten-Architekturen". Passende Rückmeldungen aus der Umsetzung, die möglichst häufig Architekturideen prüfen, sind das Thema von **Kapitel 6** – „Abgleich mit der Realität". Dort beschreibe ich den Kern der Verzahnung von Implementierung und Architektur.

Die wichtigsten Muster für diesen Teil der Vision:

- 3.5 Technische Schulden als Architekturanforderungen
- 6.3 Qualitative Eigenschaften testen
- 6.5 Code und Architektur verbinden
- 6.6 Kontinuierlich integrieren und ausliefern

2.1.5 Einfach in aktuelle Vorgehensmodelle integrierbar

Immer mehr Projekte adoptieren ein Vorgehen, das mit so wenig Verzögerung wie möglich Richtung Auslieferung von Software drängt. Das Stichwort „agil" ist so omnipräsent, dass sich viele bereits genervt abwenden, wenn das Thema zur Sprache kommt. Ich verweigere mich jedem religiösen Fanatismus an dieser Stelle und möchte hier auch nicht dogmatisch werden. Nüchtern betrachtet setzen immer mehr Unternehmen auf agile Praktiken – und es funktioniert. Viele Studien und Umfragen zeigen Erfolge von agilen Projekten [Ric07], [Bar06], [Vig09], [Wol08]. Eine jährlich durchgeführte Umfrage von VersionOne [Ver18] befragte über 5.000 IT-Mitarbeiter aus Europa und den USA zum „State of Agile Development". 97 % der Organisationen setzen demnach agile Methoden ein, nur 4 % der Unternehmen geben an, keine agilen Initiativen durchzuführen oder zu planen. Scrum ist, wenig überraschend, am weitesten verbreitet und kommt auf 72 % Marktanteil unter den agilen Methoden (Varianten mit eingerechnet).

Was bedeutet das für die Disziplin der Softwarearchitektur? Zeitgemäße Softwarearchitektur muss *auch* in agile Entwicklungsvorhaben passen und sollte die Konzepte, Praktiken und Rollen dieser Ansätze nutzen und annehmen. Sie muss zumindest iterativ leistbar sein und sollte eher erklären, wie Architekturpraktiken in moderne Vorgehensmodelle passen, als diese Vorgehensmodelle mit behindernden oder umständlichen Ergänzungen zu versehen. Wenn 80 % der Projekte Iterationsplanungstreffen abhalten, 53 % kontinuierlich integrieren und 61 % Kanban nutzen (nach [Ver18]), sollte Softwarearchitektur zumindest ihren Platz in diesen Praktiken kennen.

Wie dieses Buch hilft

Auf dem Weg von Anforderungen über die Umsetzung bis zur Auslieferung darf Architektur nicht im Weg sein. Die Muster dieses Buchs nutzen deshalb agile Konzepte oder erweitern sie, ohne den Zweck zu verwässern. Andockpunkte für Scrum und Kanban finden sich über den gesamten beschriebenen Entwicklungszyklus. Trotzdem sind die Muster auch in klassischeren Umfeldern brauchbar (iterative Entwicklung vorausgesetzt).

Die wichtigsten Muster für diesen Teil der Vision:

- 3.6 – Architekturarbeit im Backlog
- 3.7 – Architekturarbeit auf Kanban
- 4.5 – Release-Planung mit Architekturfragen
- 5.4 – Stakeholder involvieren

2.1.6 Warum Design alleine nicht hilft

Es gibt wichtige Fähigkeiten, die ein guter Entwickler haben sollte. Dazu zählen zweifellos Praktiken und Prinzipien rund um den Entwurf und das Design von Software. Bild 2.3 gibt einen Überblick zu einem Teil der entsprechenden Fähigkeiten und Denkweisen. Sie gehen über das stumpfe „Runterprogrammieren“ von Anforderungen hinaus.

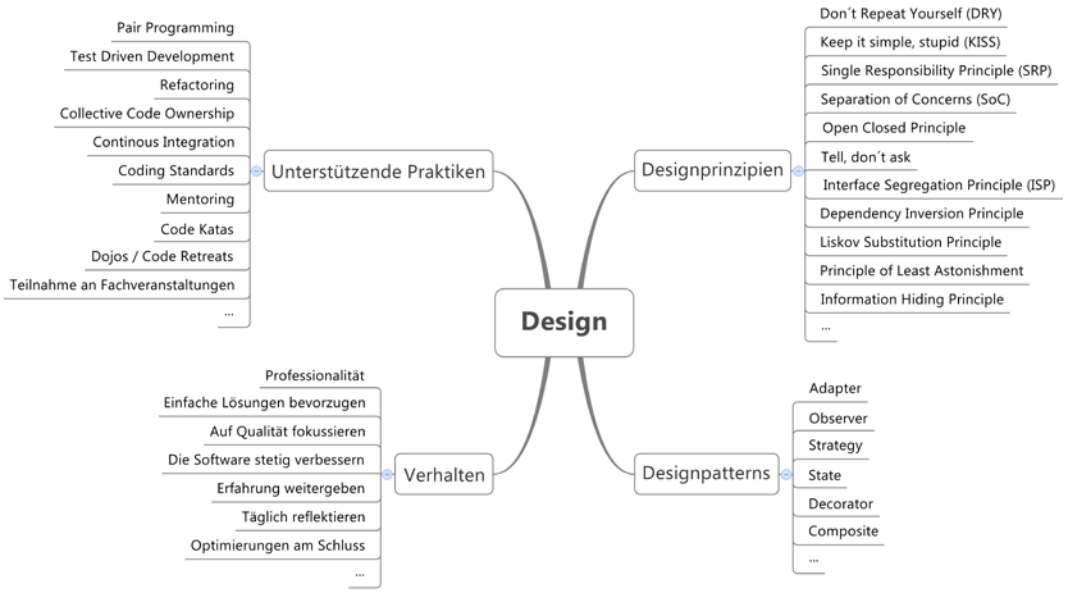


Bild 2.3 Praktiken, Prinzipien und Haltung für das Design von Software

Es ist durchaus sinnvoll, die Ideen aus Bild 2.3 als eigene Disziplin zu betrachten und entsprechendes Wissen breit zu streuen. Ich nenne diese Disziplin wenig überraschend „Design“. Auch wenn es Überschneidungen mit Softwarearchitektur gibt, sind Design und Architektur nicht deckungsgleich. Betrachten wir die enthaltenen Praktiken und Prinzipien genauer, sind zwei Dinge spannend:

- Mit dem Fokus auf einfache, bewegliche, gut verständliche Lösungen unterstützt Design vor allem ein Qualitätsmerkmal: Wartbarkeit. Architektur hat einen breiteren Fokus auf alle Qualitätsmerkmale und gleichzeitig das Ziel, eventuelle Kompromisse aufzulösen. Der Einsatz von Designpraktiken ist damit selbst eine Architekturentscheidung. Architektur bildet den Rahmen für die Designdisziplin.
- Der Einsatz von Designpraktiken beeinflusst die Struktur der Softwarelösung und hält sie flexibel. Entscheidungen rund um die Funktionalität, die Klassenstruktur und den Interfaceschnitt werden leichter änderbar – und damit weniger architekturrelevant. Gutes Design reduziert die nötige Architekturarbeit. Die Struktur kann potenziell durch Implementierungs- und Refactoring-Zyklen entstehen und wächst über die Zeit, statt zu Beginn vollständig geplant zu werden (modisches Stichwort: „emergentes Design“).

Ich werde den Blick in weiterer Folge auf Architektur fokussieren. Nicht weil Design, wie ich es in diesem Abschnitt abgegrenzt habe, nicht wichtig ist! Design ist essenziell und macht Architekturarbeit sicher einfacher. Gleichzeitig ist die Designdisziplin aber gut verstanden und aktuell viel besprochen. Die Verzahnung mit der Implementierung ist relativ geradlinig und in aktuellen Bewegungen wie „Software Craftsmenship“ ausreichend behandelt.

2.1.7 Warum agiles Vorgehen alleine nicht hilft

Es ist risikoreich, architekturelle Fragestellungen leichtfertig zu entscheiden oder sie zu ignorieren. Kümmern Sie sich in Ihrer Entwicklung nicht explizit um Softwarearchitektur, entsteht eine „zufällige Architektur“ (engl. accidential architecture [Boo06]), die nur eventuell die qualitativen Anforderungen und Vorstellungen Ihrer Stakeholder erfüllt. Mit dieser Art von Architekturarbeit kommen Sie nur bei Standardproblemen oder „einfachen“ Vorhaben davon. Sobald das Umfeld komplexer wird, haben Sie ein Problem oder brauchen viel Glück.

Scrum ist das am weitesten verbreitete Vorgehensmodell für agile Projekte [Ver18]. Es wurde allerdings nicht für große oder komplexe Projekte erdacht und liefert wenig Hilfestellung für architekturelle Probleme (vgl. [Lef10]). Ein Symptom, das immer wieder zu beobachten ist, ist das Stocken des Entwicklungsflusses in puristischen Scrum-Kontexten. Nach Phasen, in denen Features mit stetiger Geschwindigkeit ausgeliefert werden, gibt es Rückschläge und die Produktivität sinkt drastisch. Durch die Integration leichtgewichtiger Architekturpraktiken können Unterbrechungen des Entwicklungsflusses effektiv bekämpft werden [Bel13].

Agiler Fokus auf Design

In der agilen Diskussion um Softwarearchitektur nehmen Designpraktiken einen hohen Stellenwert ein. Testgetriebene Entwicklung, Clean Code, Refactoring, Pair Programming oder die Befolgung von Designprinzipien werden breit gepredigt und gelebt. Wie in Abschnitt 2.1.6 besprochen, kann die Struktur von Software damit flexibler gehalten werden, Architekturarbeit wird aber nur teilweise ersetzt. Qualitätsanforderungen wie Sicherheit oder Zuverlässigkeit werden durch Designarbeit nicht adressiert und Entscheidungen zu Plattformen, Frameworks, Programmierstilen oder Protokollen sind meist grundlegender Natur. Lösungen zu diesen Themen wachsen nicht aus gutem Design, sondern aus Architekturarbeit.

Fehlende Qualitätsanforderungen

Wartbarkeit und Erweiterbarkeit sind häufig die prominentesten Qualitätsmerkmale in agilen Entwicklungsmannschaften. Selbst diese Qualitätsmerkmale werden aber meist nicht in Anforderungen gegossen. Backlogs von Scrum-Teams sind oft *nur mit funktionalen Stories* gefüllt – mit Qualitätsanforderungen fehlt die wichtigste Grundlage für Architekturarbeit. Das hat mehrere Konsequenzen:

- **Kompromisse** zwischen konkurrierenden Qualitäten werden **erst spät erkannt** und müssen mühevoll aufgelöst werden, wenn bereits viel Programmcode entwickelt wurde. Entsprechende Anpassungen können projektgefährdend sein.
- **Architekturanforderungen** sind **nicht sichtbar**. Entwickler können folglich schwerer einschätzen, hinter welchen Backlog-Einträgen sich Architekturaufgaben verbergen. Die Aufwandsschätzung wird schwieriger und es ist manchmal nur zu raten, ob man das nötige Know-how hat, um sich bestimmte Aufgaben zu nehmen und zu bearbeiten.
- Die **Kommunikation** ist **unfokussierter**. Qualitätsanforderungen sind querschnittlich, Architekturentscheidungen betreffen viele Projektmitglieder. Sie müssen getroffene Entscheidungen breit kommunizieren und Feedback am Weg zur Entscheidung wäre nicht verkehrt. Wenn Sie diese Fragestellungen nicht erkennen, können Sie nicht gezielt zusammenarbeiten, die Kommunikation enthält viel Rauschen. Größere Projekte oder Produktentwicklungen zerbrechen dann unter zu hohem Kommunikationsdruck oder treffen verteilte, integritätsbedrohende Entscheidungen, die schwer zurückzunehmen sind.

Die Reaktion auf diese Phänomene ist häufig alles andere als agil. Ich habe gesehen, wie „agile“ Projekte Architektur großspurig wiedereinführen, eigene Architekturabteilungen wiederbeleben und dazu übergehen, wieder früh zu planen bzw. harte Governance auf Architekturvorgaben zu setzen. Die Muster in diesem Buch zeigen, wie es anders geht.

2.2 Vorgehensmuster zur Hilfe

Die im letzten Abschnitt skizzierte Vision einer zeitgemäßen Architekturdisziplin ist nicht einfach umsetzbar und schon gar nicht mit einem Big-Bang-Ansatz über eine Organisation oder ein Projekt zu stülpen. Die Muster dieses Buchs stückeln die wichtigsten Ideen deshalb in handhabbare Größe und machen eine iterative Verbesserung in Ihrer täglichen Architektur- und Entwicklungsarbeit möglich. In diesem Unterabschnitt fasse ich die vier Musterkapitel (3 bis 6) zusammen und zeige alle enthaltenen Muster mit Problemstellung und Kurzbeschreibung. Anschließend teile ich die Muster in drei Kategorien ein: zentrale Muster für häufig auftretende und grundlegende Probleme, unterstützende Muster, die bei der Musteranwendung oder dem Verständnis helfen, und weiterführende Muster mit ergänzenden Praktiken.

2.2.1 Kapitel 3 – die Basis für Architekturarbeit

Architekturarbeit ist dort sinnvoll, wo Entscheidungen risikoreich sind. Idealerweise erkennen Sie dieses Risiko, *bevor* die Entscheidung getroffen und die Lösung dafür umgesetzt wurde – also auf Anforderungsebene. In diesem Kapitel werden Muster besprochen, die Ihnen dabei helfen, die richtigen Anforderungen abzuholen, sie zu strukturieren, zu priorisieren und laufend zu verfeinern. Auch die iterative Abarbeitung von Architekturanforderungen in Backlogs oder die Verarbeitung mit Kanban sind Themen. Die Muster schaffen damit die Grundlage für Architekturentscheidungen (Kapitel 4) und die Überprüfung von Architektur im Code (Kapitel 6). Tabelle 2.1 zeigt die Muster von Kapitel 3 inklusive Problem und Kurzbeschreibung.

2.2.2 Kapitel 4 – richtig entscheiden

Die Architekturdisziplin beinhaltet viele Praktiken, Techniken und Mittel, die bei genauerer Betrachtung alle um ein Thema kreisen: Entscheidungen. Ganze Bibliotheken sind mit Büchern gefüllt, die Ihnen die konzeptionelle und technische Basis für Entscheidungen vermitteln wollen. Irgendwann müssen Sie aber auch zum Herzstück selbst vordringen, müssen entscheiden. In diesem Kapitel werden Muster besprochen, die Ihnen dabei helfen, Architekturentscheidungen von unwichtigeren Entscheidungen zu trennen, sie zu planen und bei Bedarf über mehrere Iterationen hinweg zu bearbeiten, sie zum richtigen Zeitpunkt in der richtigen Granularität zu treffen und dabei auftauchende Risiken aktiv zu behandeln. Tabelle 2.2 zeigt die Muster von Kapitel 4 inklusive Problem und Kurzbeschreibung.

Tabelle 2.1 Muster zu Architekturanforderungen

Musternname	Problem	Kurzbeschreibung
Initialer Anforderungs-Workshop (→ Abschnitt 3.1)	Wie können Architekturanforderungen effektiv erhoben und kommuniziert werden?	Erheben Sie Architekturanforderungen in einem gemeinsamen Workshop während der Kick-off-Phase.
Anforderungspflege-Workshops (→ Abschnitt 3.2)	Wie kann auf Basis einer Anforderungsliste mit architekturrelevanten Inhalten ein stetiger Fluss iterativ verarbeitbarer Aufgaben gewährleistet werden?	Aktualisieren Sie Ihre Sicht auf die Anforderungen in jeder Iteration, indem Sie die wichtigsten Einträge der Anforderungsliste detaillieren.
Szenarien als Architekturanforderungen (→ Abschnitt 3.3)	Wie drückt man Qualitätsanforderungen aus, um (1) Architekturarbeit sinnvoll zu leiten und (2) Stakeholder-gerecht zu kommunizieren?	Erheben und beschreiben Sie qualitative Anforderungen in Form konkreter Beispiele.
Szenarien kategorisieren (→ Abschnitt 3.4)	Wie können Szenarien in iterativen und/oder agilen Prozessen abgearbeitet werden, ohne zu verzögern oder zu behindern?	Gliedern Sie Szenarien nach ihrer Abhängigkeit von einzelnen funktionalen Anforderungen.
Technische Schulden als Architekturanforderungen (→ Abschnitt 3.5)	Wie werden architektonische Probleme und Versäumnisse effizient, transparent und in die restliche Architekturentwicklung integriert behandelt?	Suchen Sie aktiv nach Architekturschwächen und sorgen Sie für deren fachliche Bewertbarkeit.
Architekturarbeit im Backlog (→ Abschnitt 3.6)	Wie kann Architekturarbeit (1) iterativ, (2) stetig priorisiert und (3) mit funktionalen Aufgaben verwoben erledigt werden?	Ordnen Sie Szenarien entweder einzelnen funktionalen Anforderungen zu oder erzeugen Sie eigene Backlog-Einträge.
Architekturarbeit auf Kanban (→ Abschnitt 3.7)	Wie kann die Architekturarbeit von der Idee bis zur Auslieferung optimiert werden, so dass ein mit Umsetzungsaufgaben verwobener, stetiger und sichtbarer Fluss von Aufgaben entsteht?	Lassen Sie wichtige Szenarien gemeinsam mit funktionalen Anforderungen über ein Kanban fließen und kommunizieren Sie die Ergebnisse der Architekturbemühungen.

Tabelle 2.2 Muster zu Architekturentscheidungen

Musternname	Problem	Kurzbeschreibung
Architekturarbeit vom Rest trennen (→ Abschnitt 4.1)	Wie lassen sich jene Aufgaben identifizieren, die (1) Umsicht bei der Entscheidung, (2) evtl. tiefes Architektur- oder Technologieverständnis und (3) breite Kommunikation und Transparenz benötigen?	Anhand einiger leitender Fragen lassen sich schwer änderbare, teure oder anderweitig risikoreiche Anforderungen identifizieren.
Der letzte vernünftige Moment (→ Abschnitt 4.2)	Wann sollte eine architekturelle Fragestellung idealerweise entschieden werden, um (1) unter größtmöglicher Sicherheit zu entscheiden und (2) das Risiko einer Fehlentscheidung zu minimieren	Entscheidungen sollten so spät wie sinnvoll möglich fallen, um unter dem bestmöglichen Wissen zu entscheiden. Lernfenster sollten aktiv genutzt werden.
Gerade genug Architektur vorweg (→ Abschnitt 4.3)	Wie viel Architekturarbeit muss vor dem Einstieg in die Realisierung geleistet sein?	Frühe Architekturarbeit sollte mindestens die Systemziele und Rahmenbedingungen architekturell aufbereiten und die Voraussetzungen für eine gemeinsame Entwicklungsarbeit schaffen.
Architekturentscheidungen treffen (→ Abschnitt 4.4)	Welche Tätigkeiten sind nötig, um Architekturentscheidungen effektiv zu treffen, und wie werden sie zeitlich eingeordnet und wahrgenommen?	Architekturentscheidungen beinhalten die Analyse der Fragestellung, die Generierung von Entscheidungsalternativen, Feedback und die informierte Entscheidung selbst.
Release-Planung mit Architekturfragen (→ Abschnitt 4.5)	Wie werden Abhängigkeiten, Risiken und die Dringlichkeit von architekturellen Fragestellungen geplant berücksichtigt, ohne den Prozess der „normalen“ Umsetzung von Funktionalität unnötig aufzublähen?	Architekturfragen werden durch Szenarien und technische Schulden repräsentiert und unter Berücksichtigung von Abhängigkeiten und Dringlichkeiten in Releases geplant.
Risiken aktiv behandeln (→ Abschnitt 4.6)	Wie sollten Unsicherheiten, die architekturrelevante Auswirkungen haben, gefunden und behandelt werden?	Risiken werden mit unterschiedlichen Techniken aktiv gesucht und, je nach Risikokategorie und Risikobewertung, mitigiert oder beobachtet.
Im Prinzip entscheiden (→ Abschnitt 4.7)	Wie können mehrere Entwickler oder Architekten (Architektur-)entscheidungen bearbeiten, ohne die Konsistenz und Integrität der Software entscheidend zu verringern?	Statt Einzelentscheidungen für spezielle Probleme zu treffen, werden Richtlinien definiert, die bei einer Reihe von Entwurfs- und Architekturproblemen helfen.
Ad-hoc Architekturtreffen (→ Abschnitt 4.8)	Wie können architektonische Herausforderungen oder Unsicherheiten, die während der Umsetzung auftauchen, schnell und trotzdem solide behandelt werden?	Kurzfristig einberufene Treffen zu Architekturproblemen nutzen große Visualisierungsflächen und direkte Kommunikation, um schnell zu Ergebnissen zu kommen.

2.2.3 Kapitel 5 – Zusammenarbeit und Interaktion

„Ich bin mehr und mehr davon überzeugt, dass es tatsächlich die alltägliche Kommunikation ist, die Softwareprojekte erfolgreich macht oder sie scheitern lässt. Programmierwerkzeuge, Praktiken und Methoden sind sicher wichtig, aber wenn die Kommunikation versagt, ist der Rest nur mehr bunte Bemalung für den Leichnam“[4].

Gojko Adzic [Adz09] hebt auf bestechende Art und Weise hervor, was in so vielen Realisierungsprojekten offensichtlich wird: Zusammenarbeit, Interaktion und Austausch sind zentral. Und für welche Disziplin der Softwareentwicklung sollte das mehr gelten als für Softwarearchitektur? Von Architekturarbeit sind schließlich viele bis alle Projektmitglieder betroffen. Die Muster dieses Kapitels zeigen, wie Sie effektiv mit Stakeholdern zusammenarbeiten können, wie Sie trotz der Arbeit mehrerer Entwickler oder Teams eine konsistente Architektur gewährleisten, wie Sie Wissensmonopole vermeiden und für Transparenz sorgen. Tabelle 2.3 zeigt die Muster von Kapitel 5 inklusive Problem und Kurzbeschreibung.

2.2.4 Kapitel 6 – Abgleich mit der Realität

Ihre Architektur ist nicht fertig, wenn Sie ein Konzept erstellt, ein Diagramm gezeichnet oder eine Idee formuliert haben. Architekturentscheidungen beeinflussen große Teile der Umsetzungsarbeit und erst durch die Rückmeldung aus der Umsetzung bzw. die Einhaltung der Architekturprinzipien in allen relevanten Systemteilen wird Architektur lebendig. Gute Architekturarbeit versucht, bearbeitete Fragestellungen möglichst schnell, mit möglichst objektivem Feedback zu versorgen. Die Muster dieses Kapitels verschreiben sich dieser Prüfung und zeigen, wie Sie frühe Rückmeldungen fördern können, wie Sie Architektureigenschaften im Code analysieren und prüfen können, wie Sie Architekturziele realistisch im Auge behalten und wie Sie mit gefundenen Problemen umgehen können. Tabelle 2.4 zeigt die Muster von Kapitel 6 inklusive Problem und Kurzbeschreibung.

Tabelle 2.3 Muster zu Zusammenarbeit und Interaktion

Musternname	Problem	Kurzbeschreibung
Informativer Arbeitsplatz (→ Abschnitt 5.1)	Wie können wichtige Informationen zur Architektur möglichst breit gestreut werden, um (1) Kontext für Entwurf und Entwicklung zu geben und (2) bei schwierigen Entscheidungen und Kompromissen für eine gemeinsame Basis zu sorgen?	Informationen zur Architektur werden aktuell und großflächig ausgestellt. Die „Architekturwand“ zeigt Ziele, Kontext, Szenarien, Big-Picture, Entscheidungen, Prinzipien, aktuelle Skizzen usw.

[4] Englisches Original: *„I am getting more and more convinced every day that communication is, in fact, what makes or breaks software projects. Programming tools, practices and methods are definitely important, but if the communication fails then the rest is just painting the corpse.“*

Tabelle 2.3 (*Fortsetzung*) Muster zu Zusammenarbeit und Interaktion

Musternname	Problem	Kurzbeschreibung
Gemeinsam entscheiden (→ Abschnitt 5.2)	Wie kann eine Entscheidung effektiv und konkret in einer Gruppe getroffen werden, wenn (1) ein Entscheider (Architekt) delegiert oder (2) die Gruppe selbst für die Entscheidung verantwortlich ist.	Entscheidungsverfahren, die Konsens herstellen, räumen vorrangig Widerstände aus und sind auch in größeren Gruppen noch effektiv. Es wird eine angemessene Entscheidung getroffen, die jeder mittragen kann.
Analog modellieren (→ Abschnitt 5.3)	Wie kann die Zusammenarbeit auf konzeptioneller Ebene unterstützt werden, um Kreativität, Spontanität und eine zielgerichtete, kollektive Problemlösung zu fördern?	Statt früh in Modellierungstools zu arbeiten, werden große Flächen und einfache Werkzeuge benutzt, um Entwürfe und Ideen möglichst flexibel zu beschreiben und Interaktivität optimal zu fördern.
Stakeholder involvieren (→ Abschnitt 5.4)	Wie können Anforderungen und Erwartungen an die Softwarearchitektur effektiv abgeholt und eingeordnet werden, um stetig informierte Architekturarbeit zu leisten?	Die Architekturarbeit erfolgt unter möglichst direkter Einbindung von wichtigen Stakeholdern, von Beginn an und regelmäßig.
Wiederkehrende Reflexion (→ Abschnitt 5.5)	Wie kann nach einer Serie von Entscheidungen mehrerer Entwickler (1) Konsistenz und Integrität sichergestellt werden, (2) das Big Picture im Auge behalten werden und (3) die Kommunikationslast dabei im Rahmen bleiben?	In regelmäßigen Abständen finden sich Entwickler und andere Stakeholder zusammen, um über erledigte Architekturarbeit und ihre Auswirkungen zu reflektieren.
Architecture Owner (→ Abschnitt 5.6)	Wie kann Architekturarbeit effektiv, koordiniert und gut erledigt werden, wenn Rahmenbedingungen keine völlig selbst organisierten Teams zulassen? Wie können dabei klassische Probleme eines alleinregierenden Architekten vermieden werden?	Die Rolle des „Architecture Owner“ wird als Teilzeitaufgabe erfahrenen Entwicklern zugeschlagen. Sie beinhaltet vor allem Unterstützungs-, Organisations-, und Mentoringtätigkeiten, jedoch keine alleinige Entscheidungsbefugnis.
Architekturcommunities (→ Abschnitt 5.7)	Wie können Mitarbeiter eines Vorhabens dabei unterstützt werden, (1) über Architekturthemen nachzudenken, (2) die eigenen Fähigkeiten dahingehend auszubauen und (3) ein gemeinsames Bild zu entwickeln, das konzeptionelle Integrität fördert?	In einer offenen Gruppe tauschen sich die Mitwirkenden zu Architekturarbeit und Architekturproblemen aus. Die Arbeit ist sichtbar, relevant und hat einen festen Rhythmus.
Architektur-Kata (→ Abschnitt 5.8)	Wie können Architekturfähigkeiten (1) wiederholt geübt und geschärft, (2) auf eine breitere Entwicklergemeinde übertragen und (3) stetig verbessert werden, ohne produktiv zu entwickelnde Systeme in Mitleidenschaft zu ziehen?	An der Architekturarbeit Beteiligte üben ihr Handwerk, indem sie wiederholt Beispiel-Systeme entwerfen und analysieren. Ohne Alltags-Druck verbessern sie methodische, fachliche und technische Kompetenzen.

Tabelle 2.4 Muster zum Abgleich mit der Realität

Mustername	Problem	Kurzbeschreibung
Frühes Zeigen (→ Abschnitt 6.1)	Wie kann früh und in möglichst direkter Zusammenarbeit mit dem Kunden überprüft werden, ob sich die Softwarearchitektur entsprechend der Ziele und Wünsche entwickelt?	Frühe Rückmeldung von Fachseite wird durch frühe Integration und Auslieferung, Imitation des fertigen Systems, offene Architekturaktivitäten und Aufbereitung von technischen Erkenntnissen gefördert.
Realitätscheck für Architekturziele (→ Abschnitt 6.2)	Wie können Probleme bei der Erreichung von Architekturzielen frühzeitig erkannt werden?	In einem kurzen, regelmäßigen Treffen werden Risiken, Befürchtungen und Probleme abgeholt.
Qualitative Eigenschaften testen (→ Abschnitt 6.3)	Wie können Ziele, die die *äußere* Qualität des entwickelten Systems betreffen, objektiv geprüft werden und negative Seiteneffekte späterer Entwicklungstätigkeiten sichtbar gemacht werden?	Es werden automatisierte Tests für qualitative Aspekte des Systems bereitgestellt: Systemtests, Akzeptanztests und nichtfunktionale Tests (oder auch „Fitness Functions“).
Qualitätsindikatoren nutzen (→ Abschnitt 6.4)	Wie können Ziele, die die *innere* Qualität des entwickelten Systems betreffen, objektiv geprüft werden und negative Seiteneffekte späterer Entwicklungstätigkeiten aufgedeckt werden?	Qualitätsindikatoren (Metriken) werden ausgewählt, von Werkzeugen laufend gemessen und ausgewertet.
Code und Architektur verbinden (→ Abschnitt 6.5)	Wie können Architektur und Code am Auseinanderdriften gehindert werden, so dass (1) keine Verwässerung der Facharchitektur auftritt, (2) Architekturschwächen in puncto Umsetzbarkeit erkannt werden und (3) die Gültigkeit von Architekturprüfungen im Code gewährleistet bleibt?	Architekturvorgaben zu Programmstruktur und Qualitätsindikatoren werden in Werkzeugen hinterlegt, die Programmcode parsen und automatisiert dagegen prüfen.
Kontinuierlich integrieren und ausliefern (→ Abschnitt 6.6)	Wie können Ergebnisse von Metriken, Umsetzungsprüfungen oder Tests verschiedener Arten möglichst schnell zurück zum Entwickler fließen, um (1) direktes Feedback zu ermöglichen und (2) die Qualität des Systems stetig hoch zu halten?	Entwicklungsaufgaben sollten häufig in eine Versionsverwaltung übertragen werden, von wo aus der Build, Tests, Qualitäts- und Umsetzungsprüfungen automatisiert loslaufen, um schnelle Rückmeldung zu gewährleisten.
Problemen auf den Grund gehen (→ Abschnitt 6.7)	Wie können für die Architektur erkannte Probleme oder Risiken analysiert werden, um Verschwendung und Ineffektivität bei deren Beseitigung zu vermeiden?	Die Fehler-Ursachen-Analyse versucht, Ursachen von Problemen aufzuspüren und zu beseitigen. Einfaches Mittel dazu sind Ursache-Wirkungs-Diagramme.

2.2.5 Muster kategorisiert

Bild 2.4 zeigt die 30 Vorgehensmuster für Softwarearchitektur in drei Kategorien eingeteilt:

- **Zentrale Muster**:
  Die in diesen Mustern enthaltenen Praktiken behandeln häufig anzutreffende und grundlegende Probleme der Architekturarbeit. Die Praktiken sind direkt anwendbar und in vielen Kontexten wertvoll.
- **Unterstützende Muster**:
  Diese Muster unterstützen einige zentrale Muster direkt oder vermitteln grundlegende Ideen, die Ihnen bei der erfolgreichen Anwendung von zentralen Mustern helfen.
- **Weiterführende Muster**:
  Diese Muster beschreiben zusätzliche Praktiken, die je nach Umfeld spannend sein können. Die Muster sind nicht weniger wichtig als zentrale Muster, beeinflussen das grundsätzliche Architekturvorgehen aber weniger stark.

Sollten Sie in Eile sein, können Sie mit zentralen Mustern aus Ihrem Interessensgebiet starten und vor der Anwendung die relevanten unterstützenden Muster nacharbeiten. Weiterführende Muster enthalten danach eventuell noch einige gute Ideen für das ein oder andere Problem. Die Reihenfolge, in der die Muster in den jeweiligen Kapiteln geordnet sind, ist trotzdem sinnvoll – ich möchte Ihnen nur nicht vorschreiben, immer das gesamte Kapitel durchzuarbeiten ...

Bild 2.4 Zentrale, unterstützende und weiterführende Muster

2.3 Kurze Einführung ins Fallbeispiel

Durch die 30 Vorgehensmuster zieht sich ein Fallbeispiel. Jedes Muster wird durch ein Bruchstück dieses Fallbeispiels eingeleitet, in dem die Protagonisten das Muster entweder anwenden, es motivieren oder auf ein Problem stoßen, das mit dem Muster zu lösen wäre. Die Bruchstücke sind weitgehend unabhängig voneinander und sollten auch verständlich sein, wenn Sie nur ein einzelnes Muster betrachten. Trotzdem bauen einige Muster aufeinander auf oder ergänzen sich sehr gut. In diesen Fällen sind Verweise im Text zu finden. Ziel war, dass die einzelnen Bruchstücke in den Mustern selbsterklärend sind. Die Einführung hier ist möglichst knapp gehalten und soll einen leichten Einstieg ermöglichen.

Das Fallbeispiel ist größtenteils in Dialogform beschrieben. Entwickler, Projektleiter und Kunde tauschen sich über Softwarearchitektur und ihre Probleme in diesem Gebiet aus. Ich stelle hier nun kurz die Idee des Systems und die in den Projektbeispielen agierenden Mitarbeiter vor.

Das System – IT-Crunch

IT-Crunch ist ein Online-Magazin, in dem Redakteure und freie Autoren Artikel über wichtige Entwicklungen im IT-Sektor verfassen. Das Magazin soll als Plattform ausgebaut werden: Mitglieder können sich frei anmelden, in Foren diskutieren und eigene Artikel oder Videos einreichen. Redakteure prüfen diese Einreichungen, bevor sie im Community-Bereich live gehen. Besucher können Artikel und Forenbeiträge lesen.

Das Magazin existiert bereits online und hat eine große Leserschaft im deutschsprachigen Raum. Einige Inhalte sind kostenpflichtig (Premiuminhalte wie Gartner-Studien oder IEEE-Artikel) und können von Mitgliedern einzeln gekauft oder als Abonnement bezogen werden.

Das System gliedert sich in mehrere Systemteile. Nicht alle werden von dem im Beispiel agierenden Team umgesetzt (wie etwa das Archivsystem, das alte Artikel verwaltet und zum Kauf anbietet). Einige Systemteile werden zugekauft (Mailserver, Buchhaltungssystem, Ad-Server etc.).

Die Akteure des Fallbeispiels

- **Thorsten (Kunde):** Thorsten ist Vertreter von IT-Crunch und arbeitet mit dem Projekt zusammen, um Anforderungen zu erheben und zu detaillieren bzw. um Fragen zu Prioritäten zu beantworten und fachliche Probleme zu beheben. In Scrum wäre er der Product Owner.
- **Claudia (Projektleiter):** Claudia ist Projektleiterin und greift nicht aktiv in die Architekturentwicklung ein. Sie will das Projekt insgesamt zum Erfolg führen und setzt deshalb immer wieder Impulse.
- **Axel (Entwickler):** Axel ist ein erfahrener Entwickler und hat bereits mit Claudia zusammen Projekte realisiert. Er ist sehr gut mit vielen eingesetzten Technologien vertraut und hat in einigen agilen Projekten mitgewirkt.
- **Sarah, Tommy, Michael, Peter (alles Entwickler/innen):** Die vier Entwickler/innen haben unterschiedliche Stärken und Schwächen und helfen bei bestimmten Themen mit (oder stellen Fragen). Sarah ist die Erfahrenste unter ihnen, Peter hat eben erst sein Studium beendet.

Stichwortverzeichnis