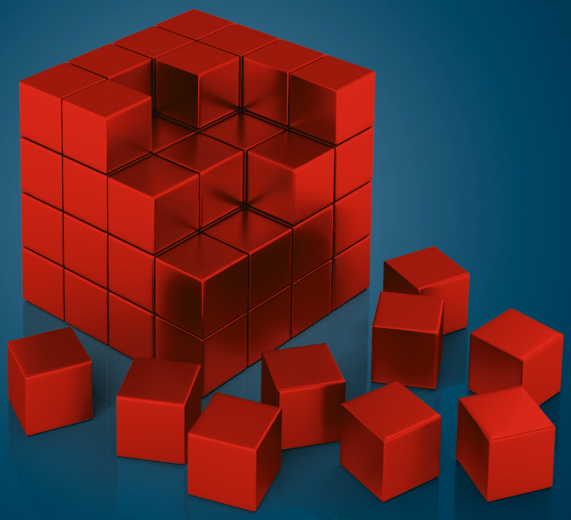
Ideal für das iSAQB® Advanced Modul AGILA

stefan TOTH

VORGEHENSMUSTER FÜR SOFTWARE-ARCHITEKTUR

3. Auflage



KOMBINIERBARE PRAKTIKEN IN ZEITEN VON AGILE UND LEAN

Im Internet: www.swamuster.de

HANSER

Toth

Vorgehensmuster für Softwarearchitektur

Stefan Toth

Vorgehensmuster für Softwarearchitektur

Kombinierbare Praktiken in Zeiten von Agile und Lean

3., aktualisierte und erweiterte Auflage

HANSER

Der Autor:
Stefan Toth, Hamburg

Bibliografische Information der Deutschen Nationalbibliothek:

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Lektorat: Brigitte Bauer-Schiewek
Copy editing: Petra Kienle, Fürstenfeldbruck
Layout: Manuela Treindl, Fürth
Umschlagdesign: Marc Müller-Bremer, www.rebranding.de, München
Umschlagrealisation: Max Kostopoulos
Druck und Bindung: Kösel, Krugzell
Ausstattung patentrechtlich geschützt. Kösel FD 351, Patent-Nr. 0748702
Printed in Germany

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

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

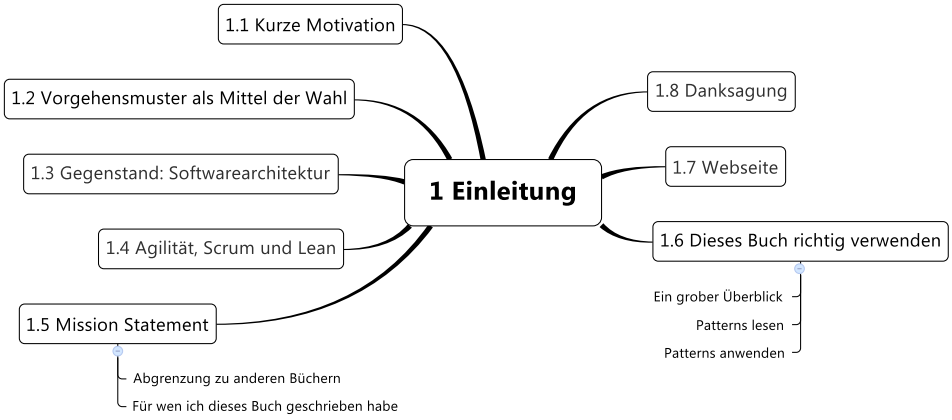
auch noch nicht können. Und es ist kein Märchen. Alles ist wahr. Wenn Sie mögen, schließen Sie das Buch nun kurz, sprechen mir nach: „Sesam, öffne Dich“, und schlagen es wieder auf. Und es tut sich tatsächlich ein reicher Schatz an Erfahrungswissen auf, der nur darauf wartet, Stück für Stück heraus in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

Stefan Zörner

1 Einleitung

Lesen Sie dieses Buch nicht. Legen Sie es weg. Jetzt.

Falls Sie dieser Empfehlung nicht gefolgt sind, haben Sie eine wichtige Voraussetzung für die erfolgreiche Lektüre bereits erfüllt: Sie glauben mir nicht alles, sondern denken selbst. Die anderen Voraussetzungen schaffe ich in diesem Kapitel. Ich bringe Ihnen zunächst die Form und das Thema des Buchs näher, erläutere grob den Aufbau und gebe einige (ernst gemeinte) Hinweise zum Umgang mit den Inhalten und Konzepten. Dazwischen erfahren Sie, ob dieses Buch etwas für Sie ist – das „Mission Statement" grenzt die Inhalte zu anderen Büchern ab und definiert die Zielgruppe.



1.1 Kurze Motivation

Ich habe dieses Buch erarbeitet, indem ich meine eigene Vorgehensweise angewandt habe. Dazu gehört, dass unwichtige Teile später bearbeitet werden und, wie es mit einer Timebox nun mal ist, eventuell hinten runterfallen.

So ist es mit der „kurzen Motivation" passiert, die Sie gerade lesen. Sie war ständig niedrig priorisiert und hat es am Schluss nicht geschafft. Sorry! Ich musste mich auf das Wesentliche konzentrieren: das, was Sie mitnehmen können, das, was Ihre Architekturarbeit bereichert, das, was Sie zum besseren Entwickler und Architekten macht. Da ist kein Platz für Motivation. Sie können es positiv sehen: Sie waren motiviert genug, dieses Buch zu kaufen oder zumindest es aufzuschlagen. Das ist doch was!

Das Einzige, was ich Ihnen hier inhaltlich mitgeben kann, ist ein Zitat von Taiichi Ohno[1], das ich schon recht früh in diesen Unterabschnitt geworfen habe, weil es eine zentrale Idee des Buchs gut verkörpert: *„Es gibt so etwas wie Standardarbeit, aber Standards sollten permanent angepasst werden. Wenn Sie vom Standard als das Beste denken, was Sie leisten können, ist alles vorbei."* Er fährt fort, indem er sagt, wenn wir etwas als den *„bestmöglichen Weg"* etablieren, *„wird die Motivation für Kaizen [kontinuierliche iterative Verbesserung] verschwunden sein."* [Pop06][2].

Versuchen Sie in diesem Sinne, die Inhalte dieses Buchs als alleinstehende, aber kombinierbare Verbesserungsideen Ihrer Praxis zu verstehen. Sie kommen aus einem eher klassischen Projektkontext? Lassen Sie sich von leichtgewichtigeren Ideen inspirieren und verzahnen Sie die Architekturdisziplin effektiv mit der Entwicklung. Sie arbeiten in einem agilen Projekt? Experimenten Sie mit den vorgestellten Praktiken, um Architekturaufgaben effektiver im Team zu erledigen oder ein besseres Gefühl für die Architekturdisziplin zu bekommen. Versuchen Sie, Ihren Standardweg zur Architekturentwicklung zu hinterfragen, Schwächen zu erkennen und Stärken auszubauen. Egal, ob Sie nun eher klassisch oder eher agil unterwegs sind: Werden Sie mit Hilfe dieses Buchs ein bisschen besser. Ständig.

1.2 Vorgehensmuster als Mittel der Wahl

Um eine stückchenweise Verbesserung an Ihrer Architekturarbeit gut zu unterstützen, habe ich mich dafür entschieden, Patterns bzw. Muster zu beschreiben. Diese Form der Beschreibung auf Methodik- und Vorgehensebene einzusetzen, ist unüblich[3], ermöglicht es mir aber, gezielt auf Probleme in Softwareprojekten und bei der Produktentwicklung einzugehen. Mit der Zerlegung in zeitgemäße, problemorientierte Architekturpraktiken ist Architekturvorgehen weniger starr und weniger fordernd. Die Praktiken sind leichter erlernbar, einfacher auszuprobieren und generieren weniger Widerstand in der (Projekt-)Organisation. Statt eines aufwendigen „Tailorings" nehmen Sie sich einfach, was Sie brauchen.

Muster sorgen auch dafür, dass Lösungen wiederkehrender Probleme Namen bekommen. Selbst wenn Sie von der beschriebenen Musterlösung abweichen, müssen Sie Ihren Ansatz nicht von Grund auf neu erklären, sondern können den Unterschied zum bekannten Muster erläutern.

Trotz der Stückelung sind die beschriebenen Praktiken nicht voneinander isoliert. Die Muster verweisen aufeinander und können im Verbund eingesetzt werden – sie helfen so auch architektonisch risikoreichen Vorhaben (siehe Abschnitt 2.1.2). Insgesamt entsteht eine Architekturdisziplin, die schnelle Resultate liefert und Stück für Stück einführbar ist.

---

[1] Erfinder des Toyota Production Systems und Urvater von Lean

[2] Originalzitat auf Englisch: Taiichi Ohno: *„there is something called standard work, but standards should be changed constantly. Instead, if you think of the standard as the best you can do, it's all over."* Ohno goes on to say that if we establish something as the *„best possible way, the motivation for kaizen [continuous incremental improvement] will be gone."*

[3] Als ich mit diesem Buchprojekt begonnen habe, war mir kein einziges methodisch orientiertes Pattern-Buch bekannt, zwei ernstzunehmende Vertreter dieses Genres konnte ich jedoch mittlerweile in Erfahrung bringen: [Lef10][Els08]).

1.3 Gegenstand: Softwarearchitektur

Die Vorgehensmuster dieses Buchs sind nicht nur gut kombinierbar, sie prägen insgesamt auch eine zeitgemäße Vision von Softwarearchitektur aus. Bevor ich in Kapitel 2.1 inhaltlich in diese Vision einsteige, sei ein kurzer Blick auf die Disziplin an sich gestattet: „Was ist Softwarearchitektur?“

Wenn Sie diese Frage zehn Softwareentwicklern stellen, werden Sie neun bis zehn unterschiedliche Antworten erhalten. Und das liegt nicht unbedingt an Unwissenheit: Das Software Engineering Institute der Carnegie Mellon Universität sammelt Definitionen für Softwarearchitektur und hält derzeit bei knapp unter 200 [SEI13]. Eine recht einfache und meist konsensfähige Definition kommt von Martin Fowler:

> *„To me the term architecture conveys a notion of the core elements of the system, the pieces that are difficult to change. A foundation on which the rest must be built.“* [Fow04][4]

Obige Aussage ist deshalb sehr reizvoll, weil sie kein Set von zu erstellenden Artefakten vorgibt und keine Entscheidungsarten definiert, die immer architekturrelevant sind. Stattdessen wird eine klare Botschaft formuliert: Wenn es schwer änderbar ist, ist es Softwarearchitektur. Daraus lassen sich zwei wichtige Feststellungen ableiten:

1. **Softwarearchitektur ist wichtig**:
   „Schwere“ Änderbarkeit definiert sich darüber, dass Änderungen teuer, aufwendig oder qualitätsgefährdend sind. Entsprechende Entscheidungen gefährden zentrale Rahmenbedingungen (Budget, Zeitplan oder Produktqualität). Eine Definition von Eoin Woods stellt diesen Aspekt zentral heraus: *„Software architecture is the set of design decisions which, if made incorrectly, may cause your project to be canceled“* [Roz11][5]
2. **Unwichtige Fragestellungen verdienen keine Architekturaufwände**:
   Es gibt auch Entscheidungen, die leicht zurückzunehmen oder anzupassen sind. Die meisten Fragestellungen in der Praxis fallen in diese Kategorie und sind damit *nicht* architekturrelevant. Architekturaufwände lohnen sich bei diesen Fragestellungen weniger und bremsen Ihre Entwicklung unnötig. George Fairbanks betont: *„You should pay as much attention to software architecture as it contributes risk to the overall project, since if there is little architecture risk, then optimizing it only helps little.“* [Fai10][6]

[4] Deutsch etwa: Für mich drückt Softwarearchitektur die Idee von Kernelementen des Systems aus, jene Teile, die schwer änderbar sind. Ein Fundament, auf dem der Rest aufbauen muss.

[5] Deutsch etwa: Softwarearchitektur ist die Menge der Entwurfsentscheidungen, welche, wenn falsch getroffen, Ihr Projekt scheitern lassen können.

[6] Deutsch etwa: Die Aufmerksamkeit, die Sie Softwarearchitektur entgegenbringen, sollte vom Risiko bestimmt werden, das von Architekturfragen ausgeht. Wenn wenig Architekturrisiko für das Gesamtprojekt besteht, hilft Architekturoptimierung auch wenig.

Softwarearchitektur vs. XY-Architektur

Es gibt viele Architekturdisziplinen und noch mehr, teilweise unternehmensspezifische, Namen dafür. In Anlehnung an [Woo08], möchte ich pragmatisch drei Ebenen definieren:

- **Unternehmensarchitektur** (Geschäftsarchitektur, strategische Architektur, Domänenarchitektur etc.)
- **Softwarearchitektur** (Applikationsarchitektur, Systemarchitektur, Lösungsarchitektur etc.)
- **Betriebsarchitektur** (technische Architektur, Technologiearchitektur, Integrationsarchitektur etc.)

Der Fokus dieses Buchs liegt auf *Softwarearchitektur*. Ich verwende im gesamten Buch die Begriffe „Softwarearchitektur" und „Architektur" synonym. Die beschriebenen Vorgehensmuster sind für die Erarbeitung *einer* Softwarelösung gedacht – sei es im Rahmen der Produktentwicklung oder als einzelnes Projekt.

Nicht systemübergreifende Aspekte der Betriebsarchitektur lassen sich ebenfalls damit bearbeiten (und sind in der Praxis oft mit Softwarearchitekturaufgaben vermischt). Arbeiten Sie übergeordnet auf strategischer Ebene, können Sie von einzelnen Ideen profitieren, müssen die Muster aber an ihren Kontext anpassen.

1.4 Agilität, Scrum und Lean

Ich werde in diesem Buch öfter von agilem Vorgehen, Scrum oder auch Lean schreiben. Ich entlehne diesen Themen einige Ideen und Denkkonzepte und sehe eine der größten Herausforderungen in der Modernisierung von Softwarearchitektur, um in diesen Umfeldern gut zu funktionieren. Der Einsatz agiler Vorgehensweisen ist jedoch keine Voraussetzung für den Einsatz der in diesem Buch enthaltenen Praktiken. Wenn Sie aus einem eher klassischen Kontext kommen, können Ihnen die beschriebenen Vorgehensmuster zu erfolgreicheren Projekten oder effektiverer Produktentwicklung verhelfen – und das relativ „Buzzword-frei".

Wünschen Sie sich trotzdem etwas Überblick zu agilen Themen, gibt es hervorragende Quellen, die einen guten und knappen Einstieg gewährleisten. Hier eine kleine Empfehlungsliste für Pragmatiker:

- Agile Prinzipien (*http://agilemanifesto.org/iso/de/principles.html*): Auf der Webseite des agilen Manifests findet sich nicht nur der berühmte und vielzitierte Wertvergleich, der vor 14 Jahren klassische Projekte aufrütteln sollte, sondern auch das (meiner Meinung nach interessantere) Verzeichnis der zwölf agilen Prinzipien. Sie machen die grundsätzliche Denkweise von agilem Vorgehen greifbar (siehe Kasten „die agilen Prinzipien").
- Lean Primer (*http://www.leanprimer.com*): eine fantastische Einführung in die Konzepte von Lean. Auf 40 Seiten motivieren Craig Larman und Bas Vodde, warum Lean für die Softwareentwicklung spannend ist, und vermitteln die zentralen Ideen sehr anschaulich.

- Scrum Guide (*http://www.scrum.org/Scrum-Guides*): der Klassiker für Scrum-Einsteiger. Zum Verständnis agiler Denkweisen vielleicht etwas weniger wertvoll als der Lean Primer, werden hier die wichtigsten Elemente von Scrum und deren Zusammenspiel beschrieben. Von den Scrum-Vätern Jeff Sutherland und Ken Schwaber.

Die agilen Prinzipien (zitiert von [agi01])

1. Unsere höchste Priorität ist es, den Kunden durch frühe und kontinuierliche Auslieferung wertvoller Software zufriedenzustellen.
2. Heiße Anforderungsänderungen selbst spät in der Entwicklung willkommen. Agile Prozesse nutzen Veränderungen zum Wettbewerbsvorteil des Kunden.
3. Liefere funktionierende Software regelmäßig innerhalb weniger Wochen oder Monate und bevorzuge dabei die kürzere Zeitspanne.
4. Fachexperten und Entwickler müssen während des Projekts täglich zusammenarbeiten.
5. Errichte Projekte rund um motivierte Individuen. Gib ihnen das Umfeld und die Unterstützung, die sie benötigen, und vertraue darauf, dass sie die Aufgabe erledigen.
6. Die effizienteste und effektivste Methode, Informationen an und innerhalb eines Entwicklungsteams zu übermitteln, ist im Gespräch von Angesicht zu Angesicht.
7. Funktionierende Software ist das wichtigste Fortschrittsmaß.
8. Agile Prozesse fördern nachhaltige Entwicklung. Die Auftraggeber, Entwickler und Benutzer sollten ein gleichmäßiges Tempo auf unbegrenzte Zeit halten können.
9. Ständiges Augenmerk auf technische Exzellenz und gutes Design fördert Agilität.
10. Einfachheit – die Kunst, die Menge nicht getaner Arbeit zu maximieren – ist essenziell.
11. Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbst organisierte Teams.
12. In regelmäßigen Abständen reflektiert das Team, wie es effektiver werden kann, und passt sein Verhalten entsprechend an.

1.5 Mission Statement

Dieses Buch stellt praxiserprobte Praktiken vor, die Ihnen helfen, Herausforderungen der Softwarearchitektur zu meistern. Die Praktiken sind in Musterform beschrieben, um sie möglichst klar darzulegen, einfach verständlich zu machen und vor allem: sie häppchenweise erlern- und anwendbar zu machen. So ermöglicht dieses Buch auch eine skalierbare Methodik. Kleine und einfache Projekte können sich die nötigen Rosinen aus dem Sack von Mustern picken, größere komplexere Entwicklungsvorhaben können mehr Praktiken übernehmen.

Inhaltlich setzt sich dieses Buch folgende Ziele:

- Neue Ideen und gut funktionierende Praktiken aus modernen Vorgehensmodellen, in den Architekturwerkzeugkasten übertragen. Durch den Einsatz der enthaltenen Muster entsteht eine effektivere, zeitgemäße Architekturdisziplin.
- Architektur in Projekte und Produktentwicklungsvorhaben integrierbar machen, die ein leichtgewichtiges Vorgehensmodell haben. Durch den Einsatz der enthaltenen Muster entsteht eine in das Vorgehen integrierte Architekturdisziplin, die so schlank wie möglich und so fundiert wie nötig arbeitet.
- Architekturarbeit dynamischer gestalten. Durch den Einsatz der enthaltenen Muster werden Projekte bis zu mehreren Teams in die Lage versetzt, dezentral und schnell zu tragfähigen Entscheidungen zu kommen. Der Weg zu Entscheidungen bremst so wenig wie möglich bei der produktiven Erstellung des lauffähigen Systems.
- Agile Vorhaben dabei unterstützen, zentrale Architekturfehler zu vermeiden. Durch den Einsatz der enthaltenen Muster entsteht ein stetiger Fluss von ausgelieferten Features ohne große Rückschläge.

Dieses Buch wird explizit *nicht* erklären, wie Sie agile Ansätze doch wieder mit klassischer Architekturarbeit ausstatten. Ich werde *keine* Argumente liefern, die Architekturarbeit in jedem Fall einklagbar machen. Gleichzeitig verschreibt sich dieses Buch *keinen* agil puristischen Dogmen. Dieses Buch soll den pragmatischen Umgang mit Softwarearchitektur in der heutigen Projektlandschaft fördern – möglichst unabhängig von modischen Trends.

1.5.1 Abgrenzung zu anderen Büchern

Um den Inhalt dieses Buchs abzugrenzen, möchte ich auf die Fähigkeiten eingehen, die Mitglieder eines Entwicklungsvorhabens in jedem Fall brauchen, um erfolgreich an Softwarearchitekturen zu arbeiten. Bild 1.1 zeigt diese Fähigkeiten in eine Pyramide einsortiert, um den aufbauenden Charakter und das unterschiedlich breite Anwendungsspektrum zu illustrieren.

Basis zur Erarbeitung von Softwarearchitektur ist die Soft-Skills-Ebene, die ich in Bild 1.1 mit „**Kommunikation**“ bezeichnet habe. Hochleistungsteams verfügen über ein Repertoire an Innovations- und Moderationstechniken, gehen produktiv miteinander um und binden Personen außerhalb des Teams gewinnbringend ein. Das Verhalten folgt gemeinsamen Prinzipien und ist auf ein bekanntes Ziel ausgerichtet.

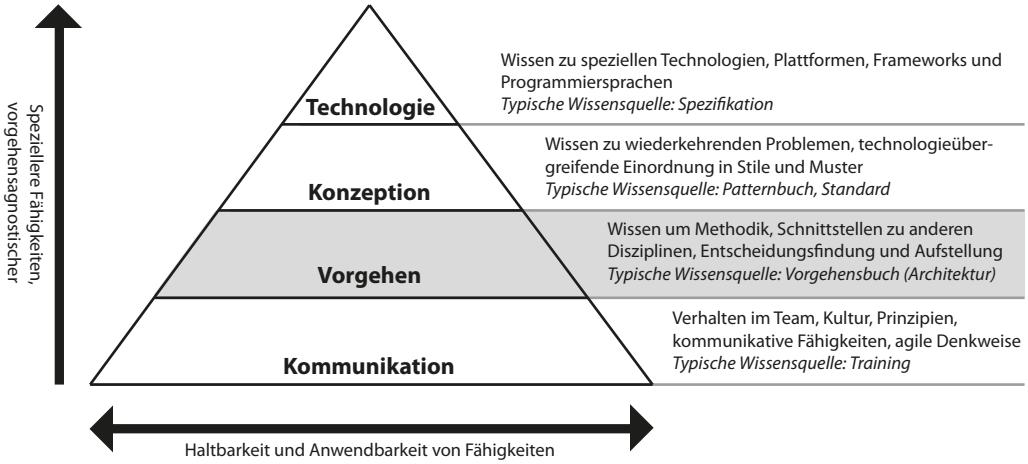


Bild 1.1 Bausteine erfolgreicher Architekturarbeit

Eine Ebene höher liegt die **Vorgehensebene**. Hier sind Praktiken zu finden, die es Teams ermöglichen, effizient zu arbeiten, die richtigen Fragestellungen zum richtigen Zeitpunkt zu behandeln und den Austausch mit anderen Entwicklern, Teams oder Stakeholdern zielorientiert auszurichten. Hier ist das *Wie* der Architekturdisziplin beheimatet.

Die **Konzeptionsebene** beinhaltet Know-how für die Strukturierung von Softwarelösungen und übergreifendes, konzeptionelles Wissen zu Technologien und Frameworks. Hier sind Stile und Muster wie z. B. „Schichten“, „Microservices“, „REST“[7] oder „Adapter“ zu Hause.

Die oberste Ebene beinhaltet **technisches Wissen**. Dazu gehört das Verständnis der eingesetzten Technologien und Frameworks, ihrer Besonderheiten und Möglichkeiten. Hier ist technologische Expertise zu Hause, wie z. B. Wissen zu Spring, ASP.NET, Docker oder Nginx. Dieses Wissen veraltet am schnellsten und ist wichtige Rahmenbedingung für Entscheidungen auf Vorgehensebene.

In der Praxis sollten Sie keine dieser Ebenen vernachlässigen. Ist Ihre Entwicklung beispielsweise auf Kommunikationsebene schwach aufgestellt, werden viele Praktiken von modernen Vorgehensmodellen nicht gut funktionieren. Ist das konzeptionelle Wissen schwach, wird immer mit der einen gut bekannten Technologie gearbeitet – unabhängig vom eigentlichen Problem. Die Architektur und das gesamte Vorhaben leiden unter solchen Lücken. Achten Sie immer auf alle vier Ebenen.

Dieses Buch fokussiert auf die zweite Ebene von unten – das Vorgehen. Es geht um prozessorientierte Entwurfstipps. Wissen zu Architekturkonzepten und Technologien sowie weiche Fähigkeiten rund um Kommunikation sind nicht zentraler Bestandteil (auch wenn Berührungspunkte erläutert werden). Wollen Sie weiter in diese ausgeklammerten Ebenen einsteigen, habe ich folgende Empfehlungen für Sie:

[7] REST steht für „Representational State Transfer“ und ist ein Stil für Webanwendungen, in dem sogenannte Repräsentationen von Ressourcen über HTTP ausgetauscht werden und über URLs eindeutig identifizierbar sind.

- **Kommunikation**:
  Kommunikative Fähigkeiten und förderliche Sichtweisen für Projekt- und Unternehmensgestaltung kann man sich nicht einfach anlesen. Trotzdem sind Bücher ein guter Startpunkt, um Ideen für die eigene Praxis zu sammeln. Persönlich sehr anregend finde ich die Soft-Skill-Reihe meines Kollegen Uwe Vigenschow [Vig10, Vig11]. Für eine Erweiterung der eigenen Denkweise Richtung Lean und Agil sind z. B. [Pop03], [Pop06] und [App10] spannend.
- **Konzeption**:
  Zur Konzeption von Architektur und technischen Mustern für Technologien und Plattformen gibt es eine Fülle an guten Büchern. Beispiele wären die Klassiker [Fow02], [Bus96] und [Gam94] oder Bücher mit etwas speziellerem Fokus, wie das Messaging-Werk von Gregor Hohpe [Hoh03]. Empfehlungen für Bücher mit Konzeptionsfokus wären [Roz11], [Sta17] und [Zör15].
- **Technologie**:
  Diese Ebene ist zu speziell für allgemeine Lesetipps. Für jede Technologie gibt es mehr oder weniger brauchbare Spezifikationen, Blogs oder aktive Foren, die helfen, Expertise aufzubauen. Das Wichtigste auf dieser Ebene ist aber die Erfahrung mit hochgekrempelten Ärmeln. Probieren Sie Technologien selbst aus – im Kontext Ihres Systems und nebenbei.

1.5.2 Für wen ich dieses Buch geschrieben habe

Bild 1.2 zeigt *Shu-Ha-Ri* in japanischen Schriftzeichen (kanji) – ein Konzept aus der japanischen Kampfkunst, das drei Phasen von Lernen und Können beschreibt:

- **Shu**:
  In Shu wird *ein* Weg angewandt. Sie sehen nicht links und rechts, sondern befolgen die Ansagen eines Meisters genau (1:1-Umsetzung des Scrum Guide, Befolgung *einer* Methodik).
- **Ha**:
  Ha erweitert die Praxis um Verbesserungen und sinnvolle Alternativen. Sie lernen aus unterschiedlichen Quellen und mixen Ihre eigene Praxis daraus.
- **Ri**:
  Im Ri lernen Sie nicht mehr von anderen, sondern ziehen Erkenntnisse aus der eigenen Praxis. Sie haben die Disziplin durchdrungen und gemeistert.

Bild 1.2
Shu-Ha-Ri in kanji [Wikipedia]

Dieses Buch ist am ehesten für die *Ha*-Ebene gedacht. Sie kennen die agilen Ideen zumindest grob und Ihnen sind die Grundlagen von Architekturarbeit geläufig. Sie wollen sich das Beste aus verschiedenen Quellen suchen, um Ihr Projekt oder Unternehmen vorwärtszubringen. Durch den Einsatz der vorgestellten Muster und das Experimentieren mit den im Buch enthaltenen Ideen rücken Sie ein Stück näher an das *Ri*. Falls Sie bereits anwendender Meister sind, lade ich Sie ein, dieses Buch querzulesen, um vielleicht noch interessante Anregungen zu erhalten.

Typische Rollen typischer Leser:

- **Entwickler**:
  Sie arbeiten als Softwareentwickler an der Umsetzung eines Produkts oder Projekts und interessieren sich für Architekturarbeit. Sie gestalten die Architektur entweder mit oder haben Einfluss auf die Erarbeitung von architektonischen Konzepten und wollen vor allem effektiv mit anderen Entwicklern zusammenarbeiten, Transparenz herstellen und gute Software abliefern.
- **Architekten**:
  Sie sind als Softwarearchitekt auf Produkt- oder Projektebene tätig und wollen dynamischer arbeiten, vielleicht Ihren Platz in einem agilen Entwicklungsprozess finden. Alternativ sind Sie als klassischer (Chef-)Architekt an zeitgemäßen Praktiken für Softwarearchitektur interessiert.
- **Manager**:
  Als Manager auf Projektebene oder darüber interessieren Sie sich für neue Strömungen und Vorgehensideen. Sie wollen sich etwas Kontext verschaffen, um die Disziplin der Softwarearchitektur besser zu verstehen, den Wert für dynamisch oder agil arbeitende Teams abzuschätzen und eventuell Impulse für effektivere Architekturarbeit zu setzen.[8]

1.6 Dieses Buch richtig verwenden

Beim Schreiben dieses Buchs hatte ich als wichtigste Metapher einen Reiseführer im Kopf. Wieso? Weil das die wahrscheinlich am besten getesteten Bücher der Welt sind und weil diese Bücher einfach richtig verwendet werden. Hunderte Reisende verschaffen sich erst mal einen Überblick und schlagen später Details nach. Stimmt etwas nicht, gibt es Rückmeldungen oder Verbesserungsideen. Jeder Leser gestaltet auf Basis der gelieferten Informationen und Einschätzungen seine eigene Reise. Trotzdem bildet sich jeder vor Ort seine eigene Meinung.

Ich möchte nicht, dass Sie dieses Buch nur lesen, ich möchte, dass Sie dieses Buch verwenden. Lassen Sie Ideen auf sich wirken, probieren Sie spannende Dinge aus und gehen Sie kritisch damit um. Sie sind verantwortlich für Ihre Reise! Modifizieren Sie die Muster dieses Buchs, um sie auf Ihren Kontext anzupassen. Falls Sie Schwierigkeiten haben, falls bestimmte Dinge sehr gut funktionieren, falls Sie Erkenntnisse oder Erweiterungen haben: Geben Sie mir Feedback[9]. Lassen Sie dieses Buch leben.

[8] Sie lesen dieses Buch *nicht*, um Praktiken auf Entwicklungs- und Architekturebene vorzugeben ...
[9] E-Mail: Stefan.Toth@embarc.de, Buchwebseite: *www.swamuster.de*

1.6.1 Ein grober Überblick

Bild 1.3 zeigt die grobe Kapitelstruktur und den logischen Aufbau des Buchs.

Bild 1.3
Kapitelstruktur

Kapitel 2 vermittelt die inhaltliche Vision dieses Buchs und schafft wichtige Grundlagen für die weitere Lektüre. Es dient als Einstieg und Wegweiser für die 30 folgenden Vorgehensmuster.

Die Reihenfolge der Patternkapitel ist nicht zufällig gewählt. Ausgehend von Anforderungen und Problemstellungen (Kapitel 3), bespreche ich Vorgehensaspekte beim Treffen von Architekturentscheidungen (Kapitel 4) und unterstützende Praktiken für dynamische Zusammenarbeit (Kapitel 5). Schließlich verbinde ich Architekturideen und -anforderungen mit den tatsächlich beobachtbaren Eigenschaften des Systems (Kapitel 6). Sie können die Kapitel und die enthaltenen Muster gut von vorne nach hinten lesen, selbstverständlich ist aber auch eine nichtlineare Arbeit mit dem Buch möglich. Die Muster verweisen aufeinander und ermöglichen Ihnen Sprünge zu interessanten Ansatzpunkten.

Der zweite große Teil des Buchs zeigt, wie Sie Muster in unterschiedlichen Kontexten anwenden können bzw. welche Anknüpfungspunkte es zu Vorgehensmodellen, etablierten Rollenmodellen und agilen Frameworks gibt. In Kapitel 7 fokussiere ich auf die allgemeine Etablierung von Vorgehensmustern in Projekten oder Unternehmen. Wie experimentieren Sie richtig mit den Ideen dieses Buchs? An welchen Stellen passen die Muster in agile Prozesse wie Scrum? Und welche Auswirkungen hat die Musteranwendung auf die Rolle Architekt? Kapitel 8 ist schließlich der agilen Skalierung gewidmet: Wie spielen die Ideen dieses Buchs mit agilen Skalierungsframeworks zusammen? Wie können große Entwicklergruppen agil an Architektur arbeiten? Und wie fügen sich evolutionäre Architekturansätze in die Welt der beschriebenen Vorgehensmuster?

1.6.2 Patterns lesen

Für die Beschreibung der Muster in diesem Buch habe ich den „Alexandrischen Stil“ gewählt. Diese von Christopher Alexander[10] verwendete Form der Musterbeschreibung zeichnet sich vor allem durch ihre gute Lesbarkeit aus. Die einzelnen Teile des Musters sind nicht durch Überschriften getrennt, sondern lediglich optisch abgesetzt – durch die Verwendung von Bildern, Balken oder Kästen. Um den Lesefluss und die Lebendigkeit der Beschreibung weiter zu erhöhen, habe ich den Musterteil der „Forces“ oder „Einflüsse“ in die Musterbeschreibung integriert und kann so die Lösung direkt im Text motivieren. Bild 1.4 zeigt die Teile eines Musters im Überblick.

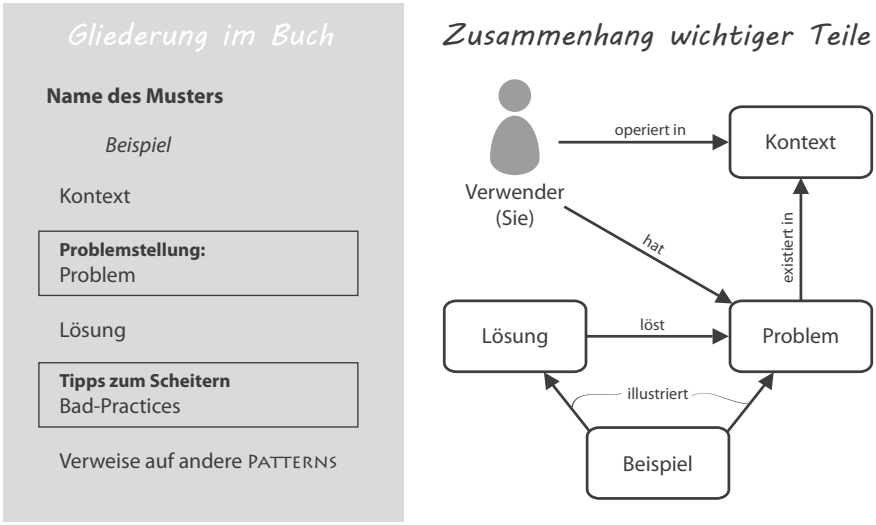


Bild 1.4 Aufbau der Muster in diesem Buch

Generell sollten die beschriebenen Muster gut von oben nach unten lesbar sein. Das Beispiel illustriert einen wichtigen Ausschnitt des Patterns und gibt Ihnen eine grobe Idee. Danach setzt der Kontext die Bühne für das als Frage formulierte Problem. Der Hauptteil des Musters kümmert sich dann um die Lösung dieses Problems. In einem abschließenden Kasten finden Sie schlechte Anwendungsbeispiele des Musters – Anti-Patterns. Die Verweise zu anderen Mustern sind vor allem spannend, wenn Sie das Muster bei sich anwenden wollen. Sie können die Verweise auf verwandte Muster auch nutzen, um das Buch netzwerkartig von interessanten Mustern ausgehend zu lesen, anstatt streng linear vorzugehen.

[10] Christopher Alexander wird von vielen als der Urvater der Beschreibung von Mustern gesehen. Sein einflussreichstes Werk ist „A Pattern Language: Towns, Buildings, Construction“ [Ale78].

1.6.3 Patterns anwenden

Die stumpfe Übernahme von in Mustern beschriebenen Praktiken wird Ihnen und Ihrem System nur bedingt helfen. Finden Sie interessante Ansätze und Ideen, können Sie gerne so schnell wie möglich loslegen. Um die Musteranwendung jedoch erfolgreich und zielführend zu gestalten, lesen Sie die Hinweise, die ich in Kapitel 7 („Vorgehensmuster anwenden“) gesammelt habe, insbesondere Abschnitt 7.1 („Muster richtig einsetzen“).

1.7 Webseite

Auf der Webseite

www.swamuster.de

finden Sie einen schlanken Musterüberblick, einige Inhalte, die aus Platzgründen nicht mehr in das Buch gepasst haben, sowie Links und weiterführende Informationen zum Thema und zu einzelnen Vorgehensmustern. Ich gebe dort auch aktuelle Vortrags- und Veranstaltungshinweise.

1.8 Danksagung

Neben der Academy möchte ich mich bei allen Helfern, Unterstützern und Kunden bedanken, die dieses Buchprojekt möglich gemacht haben. Ganz besonderer Dank gebührt Stefan Zörner, der nicht nur große Teile des Buchs gereviewed hat und viele wertvolle Anregungen beisteuerte: Er ist auch verantwortlich dafür, dass ich mich überhaupt monatelang von allen sozialen Bindungen gelöst habe, um mich dem geschriebenen Wort zu widmen. Danke für den Impuls!

Danke an Markus Wittwer für den wichtigen Input im Bereich der gemeinsamen Entscheidungsfindung und das beigesteuerte Vorgehensmuster aus Abschnitt 5.2.

Danke an meine Diskussionspartner, Reviewer und Unterstützer, die mir mit Hinweisen, Kommentaren und Ideen geholfen haben, dieses Buch zu realisieren: Jan Gentsch, Claudia Schröder, Uwe Vigenschow, Tadeusz Malek, Mischa Soujon, Kai Münch, Jan Dittberner, Wolfgang Werner, Gernot Starke, Axel Müller, Robert Uhl, Stephan Roth, Roland Mast, Niko Köbler, Harm Gnoyke, Simon Brown, Matthias Bohlen, Peter Götz, René Weiß.

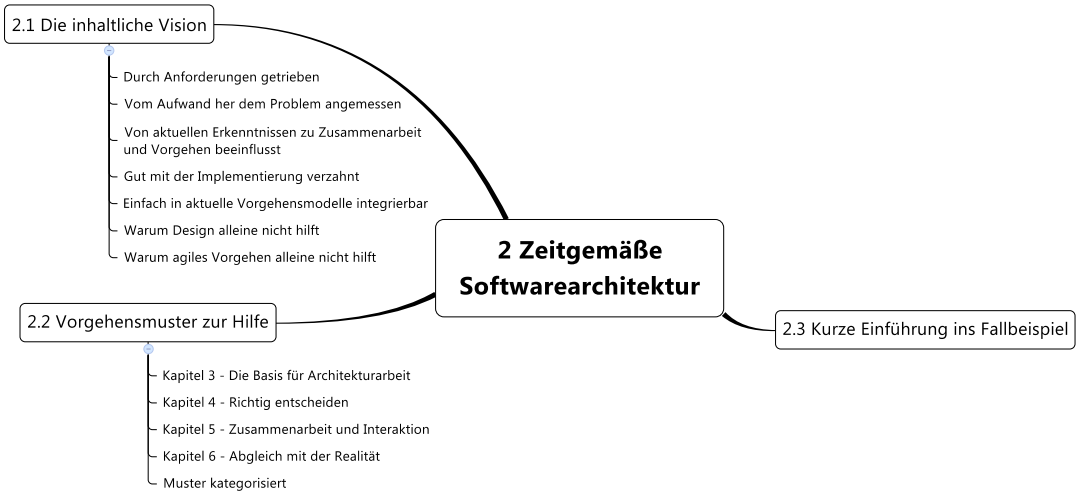
Weil ich mir das Beste immer für den Schluss aufhebe, kommt nun der Dank an jene Frau, die mich während der Arbeit an diesem Buch unterstützt, angefeuert und bei Laune gehalten hat: Marion. Danke für die gemeinsame Zeit, die mich zu dem gemacht hat, was ich heute bin. Ich versuche, deine positive und liebevolle Art weiterleben zu lassen. Irgendwann werden wir uns wiedersehen und dann habe ich viel zu erzählen.

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.

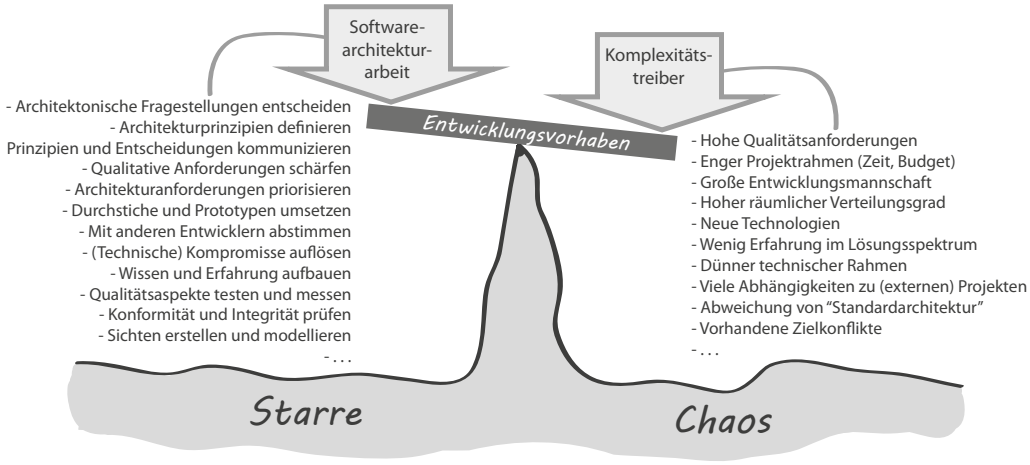


Bild 2.1 Das richtige Maß für Softwarearchitekturarbeit

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

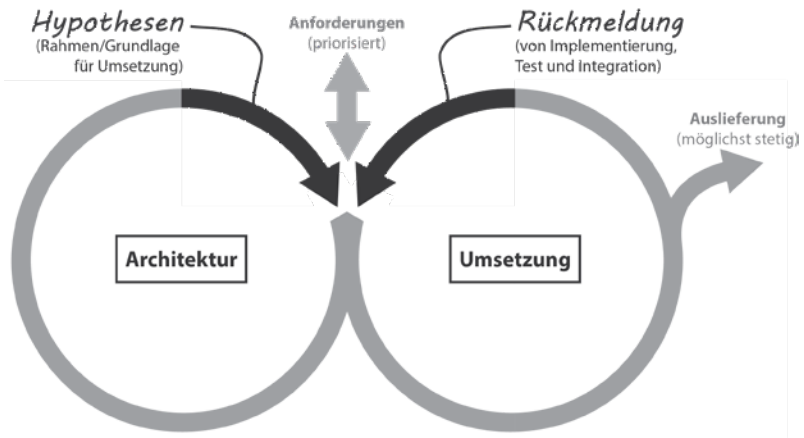


Bild 2.2 Iterative Architekturarbeit mit Umsetzung verzahnt

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.

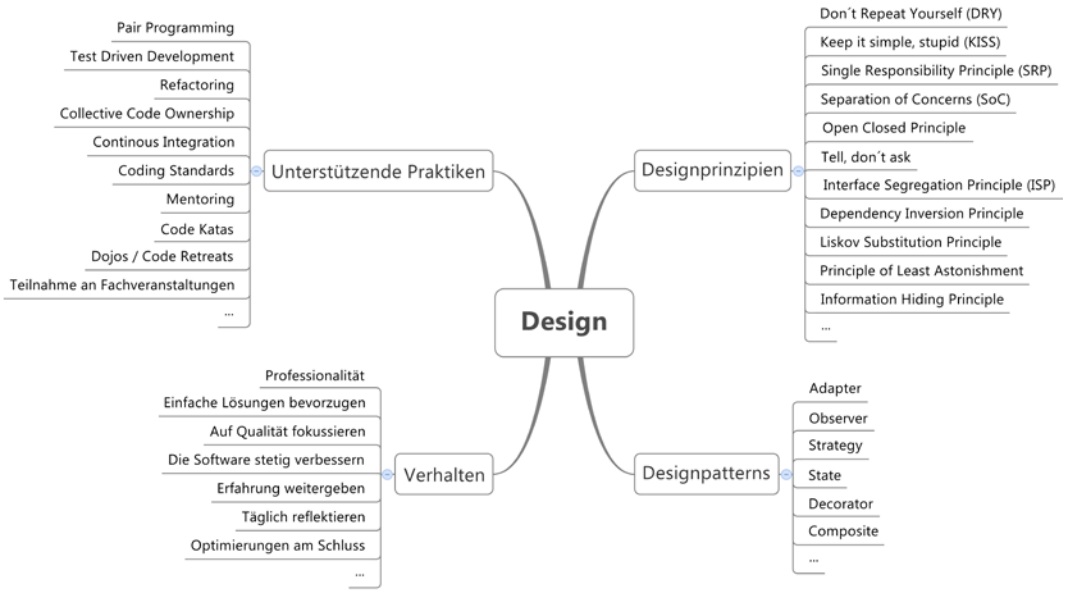


Bild 2.3 Praktiken, Prinzipien und Haltung für das Design von Software