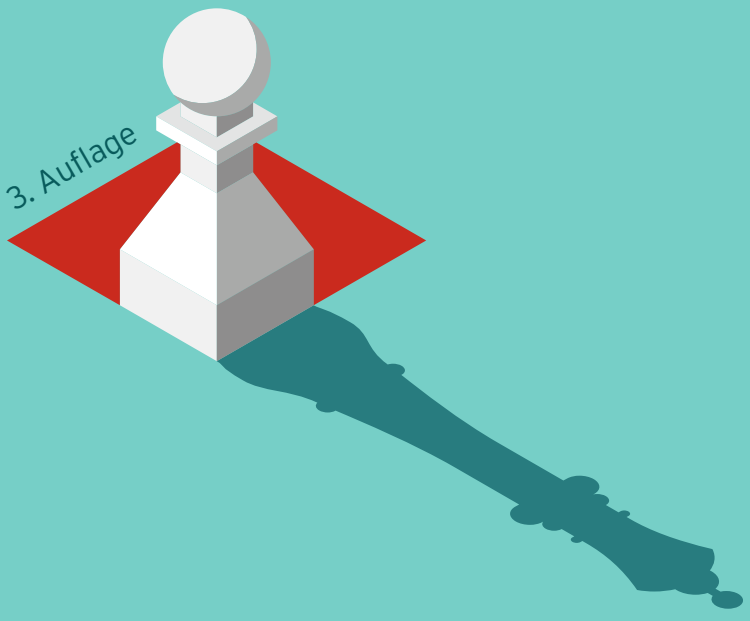
SOFTWARE-ARCHITEKTUREN dokumentieren und kommunizieren

Entwürfe, Entscheidungen und Lösungen nachvollziehbar und wirkungsvoll festhalten



Stefan ZÖRNER

HANSER

Mit einem Geleitwort von Gernot Starke

Zörner

Softwarearchitekturen dokumentieren und kommunizieren

Bleiben Sie auf dem Laufenden!

Unser **Computerbuch-Newsletter** informiert Sie monatlich über neue Bücher und Termine. Profitieren Sie auch von Gewinnspielen und exklusiven Leseproben. Gleich anmelden unter:

www.hanser-fachbuch.de/newsletter

Stefan Zörner

Softwarearchitekturen dokumentieren und kommunizieren

Entwürfe, Entscheidungen und Lösungen nachvollziehbar und wirkungsvoll festhalten

3., überarbeitete und erweiterte Auflage

HANSER

Der Autor:
Stefan Zörner, Buchholz in der Nordheide

Alle in diesem Buch enthaltenen Informationen, Verfahren und Darstellungen wurden nach bestem Wissen zusammengestellt und mit Sorgfalt getestet. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grund sind die im vorliegenden Buch enthaltenen Informationen mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autor und Verlag übernehmen infolgedessen keine juristische Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieser Informationen – oder Teilen davon – entsteht.

Ebenso übernehmen Autor und Verlag keine Gewähr dafür, dass beschriebene Verfahren usw. frei von Schutzrechten Dritter sind. Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Buch berechtigt deshalb auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

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.

Dieses Werk ist urheberrechtlich geschützt.

Alle Rechte, auch die der Übersetzung, des Nachdruckes und der Vervielfältigung des Buches, oder Teilen daraus, vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form (Fotokopie, Mikrofilm oder ein anderes Verfahren) – auch nicht für Zwecke der Unterrichtsgestaltung – reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

© 2022 Carl Hanser Verlag München, *www.hanser-fachbuch.de*
Lektorat: Brigitte Bauer-Schiewek
Copy editing: Petra Kienle, Fürstenfeldbruck
Umschlagdesign: Marc Müller-Bremer, *www.rebranding.de*, München
Umschlagrealisation: Max Kostopoulos,
unter Verwendung von Grafiken von © *shutterstock.com/inimalGraphic*
Foto des Autors: Jan Gentsch, Hamburg
Layout: Manuela Treindl, Fürth
Druck und Bindung: Eberl & Kœsel GmbH, Altusried-Krugzell
Ausstattung patentrechtlich geschützt. Kösel FD 351, Patent-Nr. 0748702
Printed in Germany

Print-ISBN: 978-3-446-46928-0
E-Book-ISBN: 978-3-446-47246-4
epub-ISBN: 978-3-446-47293-8

Inhalt

Geleitwort zur 1. Auflage

Dokumentation – Unwort der IT?

Viele IT-Systeme gelten zu Recht als schlecht erweiterbar, schwer verständlich und ungemein komplex. Teilweise liegt das an ihrer mangelhaften Dokumentation, an fehlenden oder unklaren Erläuterungen. Bei anderen Systemen begegnet mir das Gegenteil: Hunderte von Dokumenten, ungeordnet auf Netzlaufwerken, ohne klaren Einstiegspunkt. Kein Wunder, dass Dokumentation als Unwort gilt.

Die meisten Teams, die ich in meinem IT-Leben begleitet habe, konnten gut bis sehr gut programmieren, viele haben ausgezeichnete technische Konzepte entwickelt und umgesetzt. Aber kaum eines dieser Teams konnte auch nur halbwegs ordentlich dokumentieren. Oft als lästige Nebensache verflucht, mit fadenscheinigen Argumenten auf „später" verschoben oder von Anfang an aufs Abstellgleis verbannt: Dokumentation gilt in Projekten als uncool oder, schlimmer noch, als Strafarbeit: Doku – das sollen andere machen.

Hinter dieser weit verbreiteten, negativen Haltung steckt Unsicherheit: Kaum ein Entwickler, Architekt oder Projektleiter hat jemals gelernt, über Systeme zielorientiert, methodisch und mit moderatem Aufwand zu kommunizieren – und Dokumentation ist schriftliche (d. h. persistente) Kommunikation.

Genau dafür stellt dieses Buch großartige, praxiserprobte und direkt umsetzbare Lösungen bereit: Sie erfahren, wie Sie mit einfachen Mitteln die Anforderungen an langfristige, lesbare und verständliche Dokumentation erfüllen können. Stefan Zörner erklärt Ihnen, wie Sie sowohl den großen Überblick als auch das notwendige kleine Detail für Ihre Leser sachgerecht aufbereiten und darstellen. Besonders freut mich natürlich, dass er etwas Werbung für unser (freies) arc42-Template macht :-)

Ein echtes Novum finden Sie in Kapitel 6 über technische Konzepte: Überall heißt es in der Praxis: „Wir brauchen ein Konzept für <*schwieriges technisches Problem*>" ... aber niemand erklärt uns, wie solche Konzepte denn genau aussehen sollen. Wir überlegen jedes Mal neu, in welcher Struktur wir unsere Lösungsideen darstellen und wie wir argumentieren sollen. Stefan eilt mit diesem Buch zu Hilfe: Er hat (unterstützt durch Uwe Vigenschow) aus den langjährigen Erfahrungen von Lernmethodikern und Hirnforschern genau die Hinweise extrahiert, die wir für verständliche, nachvollziehbare und klare Konzepte benötigen. (Neugierig geworden? Blättern Sie direkt mal zu Abschnitt 6.4.1 und überfliegen das Vier-Quadranten-Modell.)

Aber damit nicht genug: Getreu dem Motto, dass Beispiele die besten Lehrmeister sind, hat Stefan ein wirklich cooles System entworfen, gebaut und für dieses Buch vorbildlich dokumentiert: Seine Schach-Engine DokChess illustriert, wie gute Dokumentation aussehen kann (und spielt außerdem noch ganz passabel Schach).

Ich wünsche Ihnen Freude mit diesem Buch. Als Reviewer durfte ich ja schon vor längerer Zeit frühe Versionen testlesen. Mehr als einmal haben mir Stefans Ratschläge in konkreten Projektsituationen seitdem geholfen.

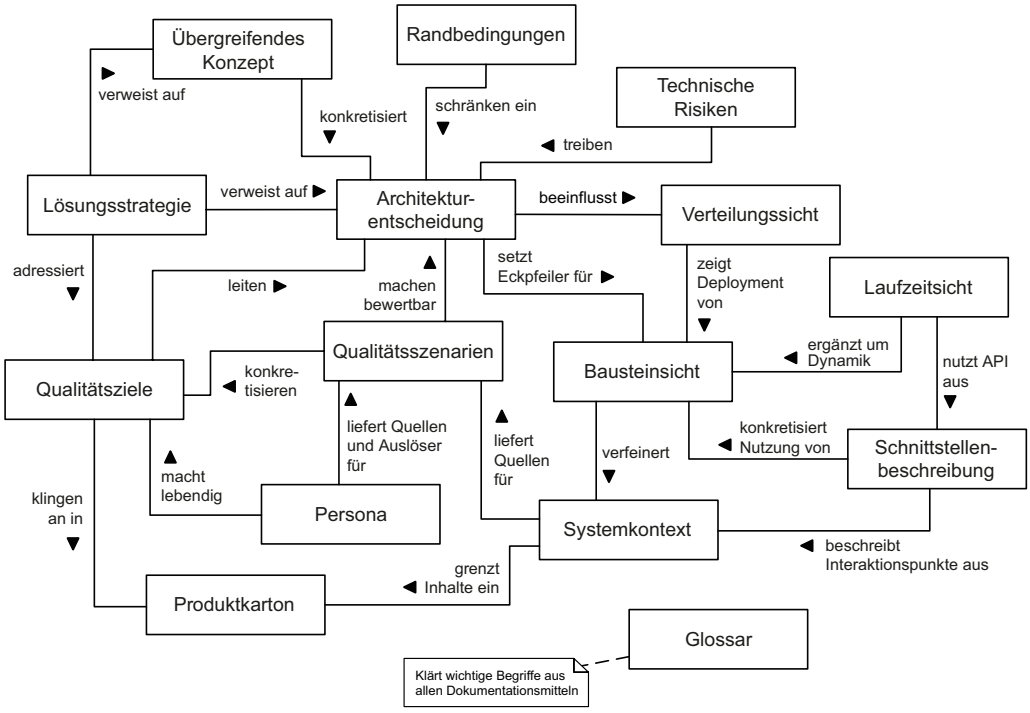
May the force of the proper word and diagram be with you.

Köln, im März 2012

Gernot Starke

Überblick: Dokumentationsmittel im Buch

Die Abbildung zeigt alle im Buch vorgestellten Dokumentationsmittel („Zutaten“) für Softwarearchitektur. Verbindungslinien visualisieren wichtige methodische Zusammenhänge. Die Pfeile an den Linien geben die Leserichtung für die Beschriftung an (Beispiel: Bausteinsicht verfeinert Systemkontext).



Dokumentationsmittel des Buchs mit wichtigen Zusammenhängen

Der Tabelle auf der nächsten Seite können Sie entnehmen, in welchem Abschnitt im Buch Sie den Steckbrief zum betreffenden Dokumentationsmittel finden.

Überblick über die Dokumentationsmittel

Dokumentationsmittel	Nutzen	Steckbrief
Architekturentscheidung	Nachvollziehbare Darstellung einer zentralen, risikoreichen Entscheidung	Abschnitt 3.2.1
Bausteinsicht	Visualisierung der Struktur des Softwaresystems und wie die Teile voneinander abhängen	Abschnitt 5.1.3
Glossar	Etablieren eines einheitlichen Wortschatzes im ganzen Vorhaben	Abschnitt 2.6.7
Laufzeitsicht	Visualisierung von dynamischen Strukturen und Verhalten, vor allem von Abläufen	Abschnitt 5.3.3
Lösungsstrategie	Stark verdichteter Architekturüberblick; Gegenüberstellung der wichtigsten Ziele und Lösungsansätze	Abschnitt 3.4.2
Persona	Archetypische Beschreibung einer Personengruppe und deren Ziele (Stakeholder)	Abschnitt 2.6.2
Produktkarton	Plakative Darstellung der wesentlichen Funktionen, Ziele und Merkmale des Systems	Abschnitt 2.2.2
Qualitätsszenarien	Konkretisierung von Qualitätsanforderungen in kurzen, beispielhaften Sätzen	Abschnitt 2.5.6
Qualitätsziele	Motivation der wichtigsten an das System gestellten Qualitätsanforderungen	Abschnitt 2.5.2
Randbedingungen	Sammlung der technischen bzw. organisatorischen Vorgaben, die beim Entwurf einzuhalten sind (oder waren)	Abschnitt 2.4.2
Schnittstellenbeschreibung	Detaillierte Beschreibung, wie ein Baustein Funktionalität bereitstellt (oder welche er benötigt)	Abschnitt 5.1.8
Systemkontext	Visualisierung der Fremdsysteme und Benutzer, mit denen das System interagiert	Abschnitt 2.3.1
Technische Risiken	Beschreibung der Risiken, die Einfluss auf die Softwarearchitektur haben (oder hatten)	Abschnitt 2.6.5
Übergreifendes Konzept	Darstellung eines übergreifenden Themas, zur Vereinheitlichung im System oder zur Detaillierung eines Ansatzes	Abschnitt 6.4.2
Verteilungssicht	Visualisierung der Zielumgebung, der Inbetriebnahme und des Betriebs des Systems	Abschnitt 5.4.3

1 Warum Softwarearchitekturen dokumentieren?

„Historisch gewachsen."

(dem neuen Teammitglied zum Gruß)

Dieses Kapitel motiviert das Thema und steckt die Aufgabe des Buchs ab. Insbesondere arbeitet es heraus, wo Architekturdokumentation unterstützt, d. h., welche Ziele Sie mit ihr verfolgen können. Am Ende des Kapitels erläutere ich Zielsetzung und Aufbau des Buchs.

1.1 Montagmorgen

Haben Sie schon einmal in einem Softwareentwicklungsprojekt gearbeitet, zu dem im weiteren Verlauf neue Teammitglieder hinzugestoßen sind? Oder waren Sie selbst schon einmal bei einem Vorhaben „der/die Neue"? Es ist also Montagmorgen und ein neuer Mitarbeiter, im konkreten Fall vielleicht eine Entwicklerin, verstärkt das Team.

1.1.1 Fragen über Fragen

Neue im Team haben naturgemäß viele Fragen. Zu Beginn dreht es sich darum, überhaupt arbeitsfähig zu werden. Entsprechend sehen Fragen zum Beispiel so aus:

- Wo soll ich sitzen?
- Welche Tools brauche ich?
- Wie checke ich die Quelltexte aus und wie baue ich die Software?
- Warum sind bei mir die Tests rot?

Die Neuen durchlaufen bezüglich ihrer Fragen einen typischen Zyklus. Nachdem grundlegende Dinge geklärt sind, erkennen Sie an den Fragen, dass es nun in die Praxis geht, unsere Entwicklerin aus dem Beispiel also tatsächlich mitarbeiten will. Später am Tag, vielleicht auch erst am Dienstag oder Mittwoch, könnten ihre Fragen so aussehen:

- Ich finde mich nicht zurecht. Wie finde ich einen Einstieg?
- Diese Teile hier – wie arbeiten die zusammen? Habt ihr das irgendwo aufgemalt?

- Ich soll hier neue Funktionalität hinzufügen, wie stelle ich das an?
- Ich habe hier etwas Ähnliches gefunden, kann ich das wiederverwenden?
- Ich habe hier eine Kleinigkeit geändert. Warum sind jetzt plötzlich bei mir die Tests rot?

Nach ersten Erfolgen wird unser neues Teammitglied mutiger. Fragen drehen sich nicht mehr nur um das „Wie“ oder „Was“, sondern auch um das „Warum“. Jetzt wird auch hinterfragt:

- Diese Software, an der wir hier arbeiten, was macht die überhaupt?
- Warum benutzen wir eigentlich noch Java 5?
- Wieso habt ihr das so gemacht? Ist das nicht viel zu kompliziert?
- Würde man das nicht eigentlich so machen?

Die Beispielfragen sind zwar typisch, aber natürlich fiktiv. Es gibt jedoch eine große Kategorie von Softwarevorhaben, denen sich wunderbar auf die Finger schauen lässt: Open-Source-Projekte. Oft führen sie ihre Kommunikation öffentlich; jeder kann die Mailingliste der Entwicklerinnen und Entwickler abonnieren und ist quasi live dabei. Bild 1.1 zeigt ein echtes Beispiel, den Absender habe ich unkenntlich gemacht.

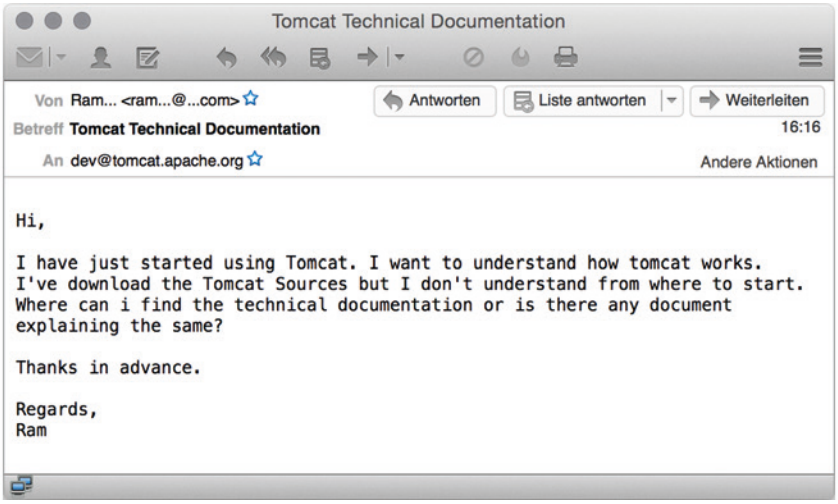


Bild 1.1 Eine Frage auf der Mailingliste von Apache Tomcat

Fragen wie die letztgenannten und konkret auch die von der Tomcat-Mailingliste zielen nicht mehr nur auf Details und die Arbeitsfähigkeit ab, sondern auch auf Strukturen, auf Abhängigkeiten, auf getroffene Entscheidungen, kurz: auf Architektur.

1.1.2 Wer fragt, bekommt Antworten ...

Es gibt einige typische Antworten, die Softwareprojekte bei Fragen neuer Teammitglieder parat haben. Vielleicht haben Sie die ein oder andere schon einmal gehört oder sogar selbst gegeben. Sehen wir uns vier exemplarische Vertreter näher an.

Typische Antwort 1: „Steht alles im Wiki!“

Je nach Füllstand im Wiki ist der Fragesteller nun einige Zeit beschäftigt, sich in den Seiten zurechtzufinden. Nach einigen Stunden kommt er oder sie mit konkreteren Fragen zu Fundstücken wieder und Sie schlagen die Hände über dem Kopf zusammen: „Das machen wir ja schon seit Ewigkeiten nicht mehr so! Wo hast du denn das gefunden?“ – „Steht im Wiki.“

Sie können das Wiki hier natürlich auch durch ein anderes Werkzeug ersetzen, das bei Ihrem Vorhaben dazu dient, Informationen aller Art aufzunehmen und zu bewahren.

Typische Antwort 2: „Das haben wir nicht dokumentiert, wir gehen agil vor.“

Dokumentation und Agilität stehen nicht im Widerspruch. Da es in diesem Zusammenhang aber oft Missverständnisse und Fragen gibt, gehe ich später im Kapitel auf dieses Spannungsfeld noch etwas genauer ein. Der/die Fragensteller/in auf der Tomcat-Mailingliste hat übrigens recht schnell eine sinnverwandte Antwort erhalten, siehe Bild 1.2.

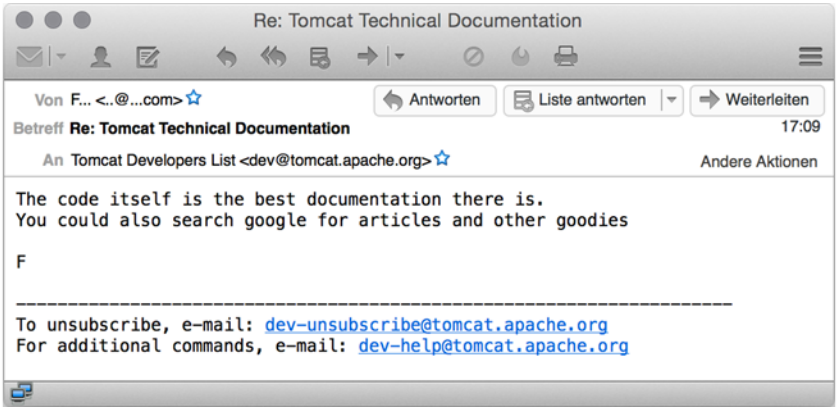


Bild 1.2 Knapp eine Stunde später: Antwort auf der Mailingliste

Anders als bei prominenten Open-Source-Lösungen wie Apache Tomcat fällt Google bei Ihrem Vorhaben als Informationsquelle für projektspezifische Dokumentation vermutlich aus.

Typische Antwort 3: „Das war schon so, als ich neu war.“

Diese Antwort kommt in Vorhaben vor, die schon länger laufen und von einer gewissen Fluktuation im Team geprägt sind. Die entsprechende Frage hatte der Antwortgeber damals auch schon gestellt. Jetzt gibt er (oder sie) die Antwort von damals, falls er oder sie seinerzeit überhaupt eine erhalten hat, an das neue Teammitglied weiter.

Mündliche Überlieferung (aus Wikipedia)

„Mündliche Überlieferung oder Oralität bezeichnet die erzählende Weitergabe von geschichtlichen, gesellschaftlichen und religiösen Informationen – insbesondere in Form von Geschichten, Sagen, Legenden und Traditionen. Sie spielt in allen Kulturkreisen eine große Rolle, insbesondere in jenen, die keine oder erst in Ansätzen eine schriftliche Überlieferung (siehe Schriftlichkeit/Literalität) kennen.“

Und damit sind wir bei der wohl beliebtesten Antwort angelangt:

Typische Antwort 4: „Das ist historisch gewachsen.“

Das hat wohl jeder schon mindestens einmal gehört und/oder gesagt, es gehört zur IT-Folklore wie der Torschrei zum Fußball. An dieser Phrase ist vor allem die Wortwahl bemerkenswert. „Gewachsen“ klingt, als wäre die Software ohne Zutun des Teams entstanden. Und es ist regelmäßig erstaunlich, wie schnell in IT-Vorhaben Dinge als historisch gelten. Anders als in der Geschichte („Die Schlacht von Marathon ist historisch belegt“) vergehen mitunter nur wenige Wochen und Monate und der Begriff taucht zum ersten Mal auf. Im Extremfall, wenn das erste neue Mitglied zum Team stößt.

Versetzen Sie sich in die Rolle eines neuen Teammitglieds! Stellen Sie sich vor, Sie müssten ab nächster Woche Ihr eigenes Projekt unterstützen und hätten keine Ahnung. Welche Fragen würden Sie stellen?

In diesem Buch geht es darum, bessere Antworten parat zu haben als zum Beispiel die vier hier vorgestellten. Wer beim Beantworten auf geeignete Hilfsmittel wie Entwürfe, Diagramme und Konzepte zurückgreifen kann, tut sich leichter. Und spätestens wenn in der Wartung keiner der ursprünglich Beteiligten mehr für Antworten zur Verfügung steht, ist Dokumentation unerlässlich. Aber Moment mal: Ist Dokumentieren nicht voll unagil?

Bild 1.3
Ansichtskarte vom Nord-Ostsee-Kanal, Poststempel 1898

1.2 Voll unagil?

In den Jahren 1887 bis 1895 entstand mit dem Nord-Ostsee-Kanal die heute noch meistbefahrene künstliche Wasserstraße der Welt (siehe Bild 1.3). Am Bau des knapp 100 km langen Kanals waren bis zu 8900 Arbeiter beteiligt, 156 Millionen Goldmark waren ursprünglich für das Vorhaben veranschlagt worden. Das Budget wurde eingehalten, ebenso der Termin. Unglaublich.

1.2.1 Agil vorgehen

Viele IT-Projekte scheitern.[1] Softwareentwicklerinnen und -entwickler kann ich angesichts der obigen Erfolgsgeschichte insofern trösten, als bei weitem nicht alle Bauvorhaben in der klassischen Architektur so erfolgreich verlaufen. Blicken Sie nur auf die Hamburger Elbphilharmonie!

Verglichen mit der Softwareentwicklung ist am Kanalbaubeispiel einiges ungewöhnlich: Die Technologie und die mit ihr verbundenen Risiken galten als beherrscht, Kanalbau – auch in dieser Größenordnung – war kein Hexenwerk. Auch die Anforderungen waren von Beginn an sehr klar umrissen und änderten sich während des Baus nicht.

Moderne Softwareentwicklungsvorhaben arbeiten in der Regel unter andersgearteten Rahmenbedingungen. Es gibt viele Unsicherheiten und Risiken, die anders als beim Bau des Nord-Ostsee-Kanals gelagert sind. Funktionieren die angedachten Lösungen, wie erwartet, aus technologischer genauso wie in fachlicher Hinsicht? Softwaresysteme bilden komplexe Geschäftsprozesse ab, die sich nur schwer fassen und beschreiben lassen. Daher klären sich Anforderungen und Prioritäten erst im Verlauf des Projekts vollständig.

Hier setzen agile Techniken und Vorgehensweisen wie z. B. Scrum [Schwaber+2020] oder Extreme Programming [Beck2004] an, indem sie beispielsweise die Tatsache, dass Anforderungen sich ändern, akzeptieren und zu ihrem Vorteil nutzen. Agiles Arbeiten ist mittlerweile im Mainstream angekommen, auch wenn viele Unternehmen und Organisationen noch mit der Umsetzung kämpfen.

Bild 1.4 gibt einen groben Überblick über die agile Welt in Form einer Mindmap. Agilität ist als Begriff nicht klar umrissen, aber das Agile Manifest [Beck+2001] vermittelt einen Eindruck der Weltsicht. Prinzipien, wie sie auch im agilen Manifest formuliert sind, passen zu dieser Anschauung, Praktiken helfen, sie umzusetzen. Vorgehensweisen wie Scrum bilden einen Prozessrahmen zur Integration agiler und klassischer Praktiken in den konkreten Projektkontext und ergänzen sie um Rollen (z. B. ScrumMaster) und ausgewählte Ergebnistypen.

In der Weltsicht des agilen Manifests werden vier Paare von Beiträgen zum Projekterfolg gegenübergestellt (in der Mindmap in Bild 1.4 links oben). Dabei wird bemerkt, dass zwar beide Seiten wichtig sind, aber im Zweifel der jeweils erste Punkt eines Paars der wichtigere ist („X *vor* Y"). Für unseren Kontext ist dabei vor allem das zweite Paar von Interesse: funktionierende Software *vor* umfassender Dokumentation.

[1] Siehe zum Beispiel die Studie „Einfluss klassischer und agiler Techniken auf den Projekterfolg". [Wittwer+2009]

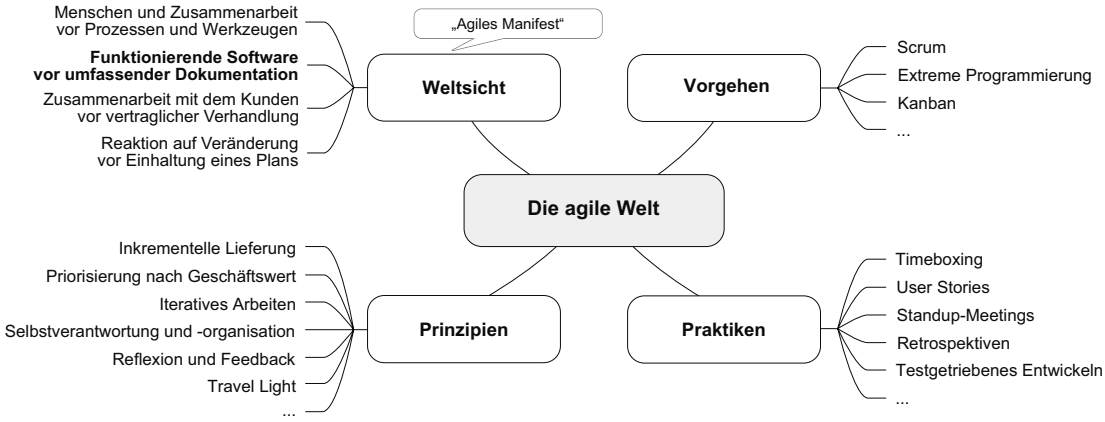


Bild 1.4 Orientierung in der agilen Welt

1.2.2 Funktionierende Software vor umfassender Dokumentation

Diese mitunter als Totschlagargument missbrauchte Aussage will nicht, dass in agilen Projekten überhaupt nicht dokumentiert wird. Sie will, dass das eigentliche Ziel, nämlich funktionierende Software zu bauen, nicht darunter leidet.

Beim Versuch, eine Softwarearchitektur zu dokumentieren, tun sich viele schwer, und die mühsam erstellten Ergebnisse überzeugen oft nicht. Wichtige Fragen, die sich der Leserkreis stellt, werden nicht beantwortet. Oder die Dokumentation ist zwar umfangreich, aber (auch deshalb?) mit der Zeit veraltet. Sie gibt nicht den aktuellen Stand wieder und ist damit nutzlos geworden.

Gesucht ist also „angemessene“ Dokumentation, aber eigentlich scheint nur klar, was das nicht ist: Als nicht angemessen gelten als schwergewichtig bezeichnete Vorgehensweisen, in denen viel Zeit in die Anfertigung umfangreicher Arbeitsergebnisse gesteckt wird, die der funktionierenden Software nicht unmittelbar dienen. Der Extremfall ist das Vorgehen in Wasserfallmanier, d. h. einer umfassenden Analyse der Anforderungen folgen vollständige Architektur- und Designmodelle, die im Anschluss implementiert werden. Die Phasen sind dabei klar abgegrenzt und werden mitunter im deutschsprachigen Raum mit DIN-genormten Dokumenten wie Lastenheft (für die Anforderungen) oder Pflichtenheft (für die Lösung) abgeschlossen.

Das andere Extrem: Bei Reviews erhalte ich auf die Frage nach der Architekturdokumentation die bereits zitierte Antwort: „Solche Arbeitsergebnisse haben wir hier im Projekt gar nicht; wir gehen agil vor. Der Quelltext ist die Dokumentation.“ Was sich bei näherem Hinsehen im schlimmsten Fall als „Wir gehen überhaupt nicht vor“ entpuppt, beinhaltet genau das falsche Verständnis von Agilität.

Dokumentation ist wertvoll, nur hat im Zweifelsfall das Funktionieren der Software eine höhere Priorität. Die spannende Frage ist daher, wie Dokumentation eingesetzt werden kann, um agile Prinzipien wie funktionierende Software oder Zusammenarbeit optimal zu unterstützen.

1.2.3 Dokumentation unterstützt Kommunikation

Ein Blick auf die agilen Werte, Prinzipien und Praktiken (siehe Bild 1.4) offenbart: Kommunikation spielt eine entscheidende Rolle. Projektteams streben einen regen Austausch mit dem Kunden und anderen Beteiligten an, und auch intern im Rahmen von täglichen Besprechungen oder Retrospektiven. Insbesondere bei der Softwarearchitektur ist Kommunikation ausschlaggebend. Architektur ist Mittler zwischen Anforderungen und Realisierung; der Architekt im Elfenbeinturm wird kaum noch akzeptiert. Genau hier findet Dokumentation ihren Platz auch in agilen Vorhaben: Dokumentation unterstützt Kommunikation.

Durch die Wüste – Travel light

Scott Ambler zieht einen Vergleich zwischen einem Softwareprojekt und einer Wanderung durch die Wüste [Ambler2002]. Wer plant, durch die Wüste zu marschieren, nimmt Sonnenschutz, Wasservorräte, und Navigationshilfsmittel mit. „Viel hilft viel“ ist hier fehl am Platz – wer einen Sonnenschirm mit Standfuß aus Granit, einige Kisten Wasser sowie Diercke-Atlas und Globus mit sich schleppen muss, kommt nicht weit. Gleichzeitig käme niemand auf die Idee, ganz ohne entsprechendes Gepäck zu reisen. „Travel light!“ heißt das entsprechende Prinzip, „Reise mit leichtem Gepäck“. Es lässt sich insbesondere auf Dokumentation anwenden.

Denn Aufwand und Nutzen müssen in angemessenem Verhältnis stehen. Der Aufwand bezieht sich auf Erstellung und Pflege der Dokumentation, der Nutzen auch auf die Weiterentwicklung und Wartung der Software. Umfassende Modelle mit dem gleichen Abstraktionsniveau wie der spätere Quelltext können in einem Softwarevorhaben die gleiche Wirkung entfalten wie Pfandflaschen aus Glas in der Wüste. Sie sind unnötiger Ballast.

Dokumentation allein kann nicht alle Fragen beantworten. Im Idealfall unterstützt sie aber wirkungsvoll dabei, Fragen zu beantworten, wie etwa die des neuen Teammitglieds. Was heißt das nun konkret für die Dokumentation von Softwarearchitektur?

1.3 Wirkungsvolle Architekturdokumentation

Dokumentation darf kein Selbstzweck sein. Wirkungsvolle Architekturdokumentation unterstützt Sie bei Ihrer Arbeit im Team und gegenüber Dritten, ohne Sie zu lähmen. Gut gemachte Dokumentation hilft Ihnen, wichtige Ziele innerhalb Ihres Softwarevorhabens zu erreichen. Ich fasse sie für dieses Buch in drei Punkten zusammen. Ihre Reihenfolge sagt nichts über die Wichtigkeit aus.

- **Ziel 1:** Architekturarbeit unterstützen
- **Ziel 2:** Architektur nachvollziehbar und bewertbar machen
- **Ziel 3:** Umsetzung und Weiterentwicklung leiten

Schauen wir sie uns genauer an!

1.3.1 Ziel 1: Architekturarbeit unterstützen

Mit Architekturarbeit meine ich das Erarbeiten von Architektur. Ein im Deutschen gebräuchliches Tätigkeitswort in diesem Zusammenhang ist „entwerfen“, also Architekturentwurf. Die Arbeit rund um Softwarearchitektur umfasst aber mehr als den Entwurf der Lösung (siehe Kapitel 2).

Ab und zu höre ich in Projektteams, auf die ich im Zusammenhang mit Architekturdokumentation treffe, Entwürfe zu dokumentieren bringe ihnen nichts. Sie programmieren gleich los. Alles vorher genau in UML-Diagrammen aufzumalen, sei sinnlos. Denn wenn sie es später implementierten, sähe die Welt ja schon wieder ganz anders aus, da sich die Anforderungen geändert haben. Und nachher aufmalen sei viel zu mühsam oder die Zeit reiche nicht aus, und wenn man es täte, würde es ohnehin nichts bringen, denn es veralte ja ganz schnell wieder.

Diese Projektteams haben recht. Aber sie vergessen eine dritte Option. Statt zu versuchen, alles vor der Implementierung zu dokumentieren (die Sinnlosigkeit von „Big Design Up Front“ in ihrem Kontext hatten sie ja erkannt) oder nachher, wäre es doch ein interessanter Ansatz, *währenddessen* zu dokumentieren. Und zwar mit Werkzeugen und in einer Granularität, die sie nicht behindern. Im Idealfall halten Sie Entscheidungen zu dem Zeitpunkt fest, wo sie bearbeitet werden, Entwurf und Dokumentation sind eins.

Die richtigen Arbeitsergebnisse helfen Ihnen, Lösungen zu finden und im Team und mit anderen zu kommunizieren. Das betrifft nicht nur die richtige Strukturierung des Gesamtsystems in Teile und wie diese Teile zusammenspielen. Ein weiteres wichtiges Merkmal, das gute Architekturen auszeichnet, ist Konsistenz. Gleiche Probleme sind stets gleich gelöst.

Bei diesem Ziel geht es also darum, bessere Antworten parat zu haben als: „Das haben wir nicht dokumentiert, wir gehen agil vor.“ Es soll gute Lösungen ermöglichen und deren Kommunikation unterstützen.

1.3.2 Ziel 2: Architektur nachvollziehbar und bewertbar machen

Erinnern Sie sich an die neue Mitarbeiterin im Entwicklungsteam, die fragte, warum bestimmte Dinge so sind, wie sie sind?

Nachvollziehbarkeit ist wie Konsistenz untrennbar mit guter Softwarearchitektur verbunden. Die neue Entwicklerin im Team ist nur ein Adressat unter vielen, an den Sie die Lösung, oder zumindest bestimmte Teile davon, kommunizieren müssen. Andere Zielgruppen sind je nach Vorhaben Auftraggeber, Kunden, Produktverantwortliche, Endbenutzer, Betrieb, ... Zum einen geht es bei der Nachvollziehbarkeit darum, die Lösung zu verstehen. Zum anderen aber auch darum, sie bewerten zu können. Ist die Architektur angemessen? Passen die Entscheidungen zur Aufgabe?

Bei diesem Ziel geht es darum, bessere Antworten parat zu haben als „Das war schon so, als ich neu war“ oder „Das ist historisch gewachsen“. Im Idealfall ersparen Sie sich auch die immer gleichen Diskussionen.

1.3.3 Ziel 3: Umsetzung und Weiterentwicklung leiten

Das Ziel des Vorhabens ist funktionierende Software. Daher dient gute Architekturdokumentation insbesondere der Kommunikation der Lösung in Richtung Umsetzung.

Gerade in Fällen, in denen die Software in Teilen oder als Ganzes durch Dritte implementiert wird – im Extremfall räumlich stark verteilt –, sieht sich der Auftraggeber vor große Herausforderungen gestellt. Dass die Verwendung der UML allein nicht sämtliche Kommunikationsprobleme auf einmal löst, hat sich im Markt mittlerweile herumgesprochen.

Softwarearchitekt/in und Entwickler/in sind heute oftmals nicht mehr verschiedene Rollen und Menschen. Die Architektur entsteht im Team und wird gemeinsam umgesetzt. Das macht Architekturdokumentation nicht überflüssig, es verschiebt den Einsatzzweck bestenfalls von Implementierungsvorschriften hin zu gemeinsamen Entscheidungen und Prinzipien. Und es bleiben Fragen wie „Wie finde ich einen Einstieg?“, „Wie finde ich mich zurecht?“.

Wenn die neue Entwicklerin, nachdem sie auf ihre Frage: „Wie mache ich x?“ die Antwort „Steht alles im Wiki“ erhalten hat, dort tatsächlich eine Lösung vorfindet und im Quelltext gleich noch zwei andere, steht sie vor einer interessanten Entscheidung: Wähle ich aus den drei Varianten die mir genehmste aus oder erfinde ich eine vierte?

Bei Ziel 1 ging es darum, Konsistenz zu erreichen. Spätestens bei der Weiterentwicklung und Wartung eines Systems ist es entscheidend, sie zu erhalten. Sonst verwässert die Lösung und wird irgendwann aufgegeben wie ein sinkendes Schiff.

1.3.4 Fremdwort Do|ku|men|ta|tion [...zion] [lat.]

Dokumentieren zählt nicht unbedingt zu den Lieblingsbeschäftigungen vieler Entwickler. Das gilt für Dokumentationen aller Art (Benutzerhandbuch, Betriebshandbuch, ...). Dokumentieren wird als lästige Pflicht angesehen, gern aufgeschoben und dann vergessen. Gute Dokumentation: Fehlanzeige.

Bezeichnenderweise ist das Wort Dokumentation ja tatsächlich ein Fremdwort:

Dokumentation (aus Duden. Das Fremdwörterbuch)

„Do|ku|men|ta|ti|on [...zion] [lat.] die; -, -en: 1. a) Zusammenstellung u. Ordnung von Dokumenten und Materialien jeder Art, durch die das Benutzen und Auswerten ermöglicht oder erleichtert wird ...“

Die Duden-Definition trifft das Vorhaben dieses Buchs recht gut. Im weiteren Verlauf stelle ich verschiedene Techniken und Werkzeuge vor, die Sie bei der Erreichung der drei oben beschriebenen Ziele unterstützen – „Materialien jeder Art“. Zum Arsenal zählen textuelle Arbeitsergebnisse ebenso wie grafische. Diese Zutaten für Architekturdokumentation werden im Buch *Dokumentationsmittel* genannt.

Die richtige „Zusammenstellung und Ordnung“ der vorgestellten Ergebnisse verbessert deren Wirksamkeit bezüglich der Erreichung der Ziele noch einmal drastisch. Hier orientiere ich mich primär an der von Gernot Starke und Peter Hruschka vorgeschlagenen Struktur arc42, diskutiere aber auch Alternativen und Situationen, in denen Anpassungen erforderlich sind.

1.4 Mission Statement für dieses Buch

Im letzten Unterkapitel habe ich die Ziele von Architekturdokumentation herausgearbeitet. Was genau ist nun aber das Ziel dieses Buchs? Die Kurzfassung: Am Ende ist Architekturdokumentation in Ihrem Vorhaben ein integraler Bestandteil und kein Fremdwort mehr. Dieses Buch stellt praxiserprobte Möglichkeiten dar, Softwarearchitektur festzuhalten. Die gezeigten Inhalte lassen sich an den in Abschnitt 1.3 vorgestellten drei Zielen für wirkungsvolle Architekturdokumentation messen.

Das Buch bietet auch Leserinnen und Lesern, die bezüglich Softwarearchitektur noch am Anfang stehen, einen Einstieg ins Thema. Es holt sie ab, gibt ihnen Orientierung und Sicherheit. Das Buch arbeitet heraus, was nach meinem Erfahrungswissen zwingend dazugehört.

Die nötigen Zutaten stellt das Buch lebendig vor, zeigt auf, zu welchem Anlass Sie und Ihr Team sie erstellen und in welcher Tiefe. Dabei skizziert es das Zusammenspiel von Dokumentation mit einem methodischen Softwareentwurf. Außerdem ermöglicht das Buch Ihnen, eigene Ergebnisse einzuschätzen und zu bewerten, etwa durch Abgleich mit Beispielen oder durch die Anwendung der Checklisten.

Einen Schwerpunkt legt das Buch auf die Werkzeugfrage, die naturgemäß in Büchern über Methodik der Softwarearchitektur sonst wenig Raum einnimmt. Es vermittelt keine einseitigen Produktpräferenzen, sondern gibt konkrete, unvoreingenommene Hilfestellung bei der Auswahl. Damit Sie schnell zu Ergebnissen kommen, wird das Buch durch elektronische Hilfen begleitet: Templates für einzelne Dokumentationsmittel, Rahmen für das Ganze.

Wenden Sie dieses Buch auf Ihre konkreten Systeme an, im Beruf oder bei Open-Source-Projekten, zur Dokumentation Ihrer laufenden Arbeit oder auch zur Nachdokumentation!

Schließlich spricht das Buch auch Unternehmen und Organisationen an, die sich durch das Etablieren von Standards positive Effekte erhoffen. Durch Vorlagen, Tipps und Tricks, aber auch durch die Warnung, dass eine zu regulative Handhabung dieses Themas unerwünschte Effekte mit sich bringt.

Abgrenzung – was dieses Buch nicht ist:

Wer sich auf Architekturdokumentation konzentriert, hat in zweierlei Hinsicht mit Abgrenzungsproblemen zu kämpfen: mit der Methodik (wie entsteht die Softwarearchitektur, die wir beschreiben?) und mit anderen Disziplinen (was gehört zur Architekturdokumentation und was nicht?).

Denn Architekturdokumentation ist – falls es sie gibt – meist nicht die einzige Dokumentation im Projekt. Neben der Softwarearchitektur gibt es weitere Disziplinen, die Dokumente und Materialien aller Art produzieren. Sie erstrecken sich von der Geschäftsprozessmodellierung bis zur Implementierung (zum Beispiel in Form von Kommentaren im Quelltext).

Die Abgrenzung ist nicht immer leicht. Beispielsweise sind technische Risiken sehr interessant für die Nachvollziehbarkeit. Risikomanagement wird aber eher im Projektmanagement angesiedelt als in der Softwarearchitektur. Ein Softwarearchitekt liefert hier Input. Aber gehört dieser nicht auch in die Architekturdokumentation? Ähnliche Beispiele finden Sie auch in anderen Disziplinen, allen voran im Requirements Engineering. Das Buch zeigt auf, warum und wo Abgrenzungen schwerfallen, und macht Vorschläge, wie Sie den Widerspruch zwischen Redundanzfreiheit und Lesbarkeit im Einzelfall auflösen.

Auch Methodik lässt sich nicht völlig ausblenden. Klassische Vorgehensmodelle wie zum Beispiel der Rational Unified Process [Kruchten2003] umfassen Rollenbeschreibungen, Ergebnistypen, Aktivitäten, und dergleichen. Dieses Buch ist kein Vorgehensmodell in diesem Sinne. Da ich konkrete Dokumentationsmittel empfehle, legt es ein gewisses Vorgehen nahe. Bei Architekturdokumentation, so wie sie dieses Buch vermittelt, geht es aber nicht darum, Prozesse und Regularien zu befolgen oder sklavisch Templates auszufüllen.

Ich lasse offen, ob Sie in Ihrem Vorhaben die Rolle Softwarearchitekt/in explizit besetzen oder Ihre Architektur im Team entsteht. Die vorgeschlagenen Werkzeuge und Techniken sind breit anwendbar, ganz egal, ob Sie Scrum oder V-Modell anwenden oder sich in einem regulativen Umfeld bewegen.

Das Buch enthält keinen Einstieg in UML und vermittelt auch kein Wissen über konkrete Technologien oder Architektur im Allgemeinen (z. B. Architekturmuster). Hierzu gibt es bereits reichlich Literatur. Für Softwarearchitektur empfehle ich [Rozanski+2011] und [Starke2020], wobei ich mich bei der Wortwahl am zweiten Buch und am deutschen Lehrplan des International Software Architecture Qualification Board (iSAQB, [iSAQB2021]) orientiere.

1.5 Über dieses Buch

Zum Schluss des Kapitels noch einige „Gebrauchsinformationen“: Für wen wurde das Buch geschrieben, wo kommt es her, und vor allem: wie ist es aufgebaut? Das soll Ihnen Orientierung geben und die Möglichkeit eröffnen, schnell zu starten. Sie brauchen das Buch dazu weder von vorn bis hinten zu lesen, noch vorn zu beginnen. Ein schöner Einstieg ist auch das Fallbeispiel in Kapitel 9.

1.5.1 Für wen ich dieses Buch geschrieben habe

Zielgruppe des Buchs sind Softwareentwickler/innen und -architekt/innen, die die Entwurfsideen ihrer Vorhaben wirkungsvoll festhalten und dabei nicht im Dokumentenstrudel untergehen wollen. Wenn Sie Anregungen, Beispiele und konkrete Hilfestellung dazu suchen, hier sind sie. Das Buch ist programmiersprachen- und technologieneutral und gibt keine Vorgaben bezüglich des Vorgehensmodells. Kenntnisse in UML sind hilfreich, aber zum Verständnis nicht zwingend erforderlich.

Tatsächlich habe ich selbst einmal zur Zielgruppe gehört, und ein Buch wie dieses hätte mir möglicherweise viel unnötige Arbeit und einige Enttäuschungen erspart ...

Wie es zu diesem Buch kam

Mein „Erweckungserlebnis“ bezüglich Architekturdokumentation war übel – es glich dem Anfang dieses Kapitels. Ich stieß als Neuer zum Entwicklungsteam des Auftragnehmers dazu. Der Kunde hatte aufgrund unzulänglicher Ergebnisse und mangelnder Transparenz das Vertrauen verloren. Er machte das unter anderem am völligen Fehlen von Dokumentation fest.

Meine Aufgabe war es, innerhalb kurzer Zeit diese Lücke zu schließen und einen Architekturüberblick anzufertigen. In dem Projekt lernte ich alles, was ich über das Scheitern wissen musste. Insbesondere auch, dass sich die skizzierte Problemstellung durch das Anfertigen von Dokumentation allein nicht wirksam lösen lässt.

Für zukünftige Vorhaben war ich sensibilisiert, was das Festhalten und Kommunizieren von Architektur angeht. Ab 2008 habe ich einzelne Zutaten, die ich dabei als wirksam schätzen gelernt hatte, in einer Kolumne im Java Magazin vorgestellt [Zörner2008]. Die positive Resonanz dazu, Gespräche mit Kundenmitarbeitern, Teilnehmern von Workshops und Zuhörern von Konferenzbeiträgen haben mich weiter ermutigt. Den Ausschlag gab letztlich ein Gespräch mit Gernot Starke, dem ich vom Fallbeispiel DokChess erzählte.

Mit diesem Buch halten Sie nun das Ergebnis meiner Bemühungen in den Händen, Architekturdokumentation in Ihrem Projekt zu einem Nichtübel zu machen. Ich hoffe, Sie begegnen dieser Aufgabe nach dem Lesen mit der Gewissheit, Ihre Architektur schon mit wenigen „Zutaten“ wirkungsvoll festhalten und kommunizieren zu können. Viele Teams, mit denen ich in den letzten Jahren arbeiten durfte, sind zu Recht stolz auf die von ihnen geschaffenen Systeme. Ganz besonders freue ich mich, wenn es Ihnen mit Hilfe der aufgezeigten Werkzeuge gelingt, andere an Ihrer Begeisterung teilhaben zu lassen.

1.5.2 Wie dieses Buch aufgebaut ist

Die insgesamt zehn inhaltlichen Kapitel dieses Buchs lassen sich grob vier Teilen zuordnen, wie in Bild 1.5 gezeigt.

I. Motivation

Im ersten Teil geht es um die Motivation des Themas und die Zielsetzung für dieses Buch. Das zugehörige Kapitel lesen Sie gerade.

II. Basis – Was Sie über das Thema wissen sollten

Im zweiten Teil folgt die nötige Basis. Ich kläre den Begriff „Softwarearchitektur“ für dieses Buch und zeige, welche Faktoren Einfluss auf Architektur haben. Erste aufgabenbezogene Zutaten für eine wirkungsvolle Architekturdokumentation kommen zum Einsatz. Das setzt sich mit Dokumentationsmitteln für den Lösungsentwurf fort. In Kapitel 4 gehe ich auf arc42, einen konkreten Vorschlag zur Gliederung von Architekturdokumentation, und Alternativen dazu ein.

Damit das Ganze trotz Theorie nicht allzu trocken wird, demonstriere ich die Zutaten bereits innerhalb der Kapitel, in denen sie vorgestellt werden, anhand zweier durchgängiger Beispiele. Dabei handelt es sich um sehr unterschiedliche Architekturen; die fachliche Domäne ist in beiden Fällen Schach. Wenn Sie sich schon ein wenig auf das Thema einstimmen wollen, lade ich Sie ein, den Kasten „Hintergrund: Mythos Computerschach“ zu lesen.

III. Praxis – Wie gehen Sie vor?

Im dritten Teil widmen wir uns zunächst Werkzeugen und Notationen. Mit welchen Tools können Sie die empfohlenen Zutaten für Architekturdokumentation erstellen, pflegen und verbreiten? Welche Formen bieten sich an, und warum?

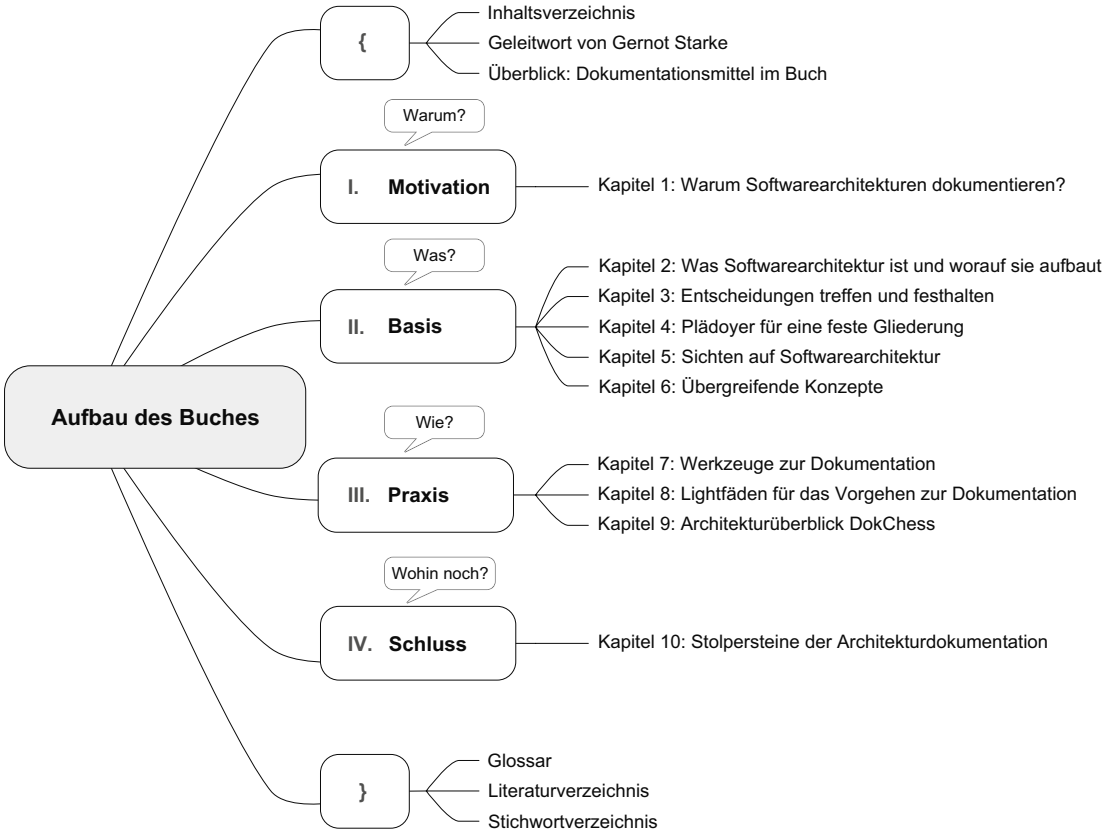


Bild 1.5 Gliederung des Buchs

Dokumentationsanlässe unterscheiden sich drastisch. Ein System, das auf der grünen Wiese entsteht und bei dem Dokumentation zeitgleich zu Entwurf und Realisierung stattfindet, ist in der Theorie ein beliebter Fall. Ihm steht in der Praxis oft das Gegenteil gegenüber: die Dokumentation bestehender und in Betrieb und Weiterentwicklung befindlicher Systeme. In der Theorie gibt es keinen Unterschied zwischen Theorie und Praxis. In der Praxis schon. Dem trage ich im achten Kapitel Rechnung, indem ich unterschiedliche Kochrezepte („Lightfäden“) für typische Szenarien darstelle. Ein Architekturüberblick rundet den Praxisteil ab: eines der Fallbeispiele in geschlossener Form, mit allen vorgeschlagenen Zutaten, gegliedert nach arc42.

IV. Schluss – Wohin geht die Reise?

Sie haben nach dem dritten Teil konkrete Hilfsmittel und Rezepte an der Hand, um für Ihre eigenen Vorhaben eine wirkungsvolle Architekturdokumentation anzufertigen. Auf dem Weg dahin lauern Fallen: typische Handlungsmuster, Missverständnisse und Fehler, die zu unbefriedigenden Ergebnissen und/oder unverhältnismäßig hohem Aufwand führen. Ich zeige Möglichkeiten auf, sie zu umgehen oder zu entschärfen. Damit sollte Ihnen die letzte Angst genommen sein, einfach loszulegen! Ein typischer Stolperstein ist zugleich die größte Chance, um Ihre Ergebnisse zu verbessern: Reviews. Mit ihnen schließe ich die Inhalte ab.

Übungsaufgaben

Falls Sie neben Ihrem eigenen Vorhaben noch ein wenig üben wollen, finden Sie an verschiedenen Stellen kleine Aufgaben. Senden Sie mir Ihre Lösungen per E-Mail zu (Kontakt siehe unten). Sie erhalten eine individuelle Rückmeldung (wenn ich nicht in Einsendungen versinke) und in jedem Fall eine Musterlösung zur betreffenden Aufgabe im Tausch. Ich freue mich auf Ihre Ergebnisse!

Kernaussagen pro Kapitel

Am Ende jedes inhaltlichen Kapitels finden Sie einen Kasten mit der Kernaussage. Dabei geht es weniger um eine Zusammenfassung der besprochenen Inhalte als um meine Haltung dazu. Wir können gerne darüber diskutieren!

Für Softwarearchitektinnen und -architekten

In früheren Auflagen dieses Buchs habe ich bei Personenbezeichnungen nicht immer alle gleich behandelt. In dieser Auflage soll es diesbezüglich zumindest geschlechtergerechter zugehen. In jedem Fall sind alle gemeint, auch wenn ich mich mitunter für den besseren Lesefluss und gegen eine gleichzeitige Nennung mehrerer Sprachformen (z. B. „Entwicklerinnen und Entwickler“) oder eine Neutralisierung (z. B. „Entwickelnde“) entschieden habe.

Feedback

Ich freue mich über Rückmeldungen aller Art, ganz gleich, ob es gefundene Fehler, Lob, Kritik, Anregungen, Fragen oder Lösungen für die Übungsaufgaben sind. Sie erreichen mich per E-Mail unter stefan@swadok.de.

Webseite zum Buch

Auf der Webseite zum Buch *https://www.swadok.de* finden Sie Vorlagen und Werkzeuge zu den im Buch vorgestellten Zutaten und Gliederungen für Architekturdokumentation sowie Links zu weiteren Informationen zum Thema, zu den Fallbeispielen und zu den Übungsaufgaben.

Hintergrund: Mythos Computerschach

Ein Menschheitstraum wie das Fliegen

„Meine Damen und Herren, ich habe eine Maschine gebaut, wie es sie bisher noch nie gegeben hat: Einen automatischen Schachspieler! Er ist in der Lage, jeden Herausforderer zu schlagen …“

Mit etwa diesen Worten leitete Wolfgang von Kempelen (1734-1804) die Vorführung seines berühmten Schachautomaten ein: dem Schachtürken. Das erste Mal geschah dies 1770 am Hofe der Kaiserin von Österreich Maria Theresia in Schönbrunn.

Und es hat noch viele weitere Partien vor berühmten Persönlichkeiten in Europa und Übersee gegeben. Zum illustren Kreis der Gegner sollen Friedrich der Große, Benjamin Franklin und der bekannteste Schachspieler seiner Zeit, Philidor, zählen. In einer überlieferten Partie von 1809 verliert Napoleon Bonaparte gegen den Schachtürken.

Teil des Automaten war eine türkisch gekleidete Puppe, die die Figuren auf dem Brett führte. Auf YouTube sind Filme von Nachbauten zu bewundern, das Ganze mutet ziemlich unheimlich an. Es ist klar, dass die Menschen im ausgehenden 18. Jahrhundert fasziniert waren, unter ihnen später auch Edgar Alan Poe. Die Begriffe „getürkt“ und „einen Türken bauen“ sollen angeblich auf den Schachtürken zurückgehen. Denn tatsächlich handelte es sich um einen Trick; verborgen im Innern der Kiste steuerte ein menschlicher Spieler das Gerät.

Doch das kam erst sehr viel später ans Licht. Zuvor regte dieses Wunderwerk der Mechanik die Diskussion an, ob eine Maschine dem Menschen geistig ebenbürtig sein kann. Oder ihm sogar überlegen ... Schach ist für diese Fragestellung geradezu prädestiniert.

Schach und Computerschach

Das Schachspiel ist sehr alt. Die Ursprünge liegen in Indien und Persien; auf verschiedenen Wegen gelangte es nach Europa und ist hier spätestens seit dem 13. Jahrhundert präsent. Schach begleitet uns durch die Geschichte, in jeder Epoche finden sich Darstellungen von Menschen, die Schach spielen. Immer noch gilt es bei uns als das bedeutendste Brettspiel. Kein Wunder also, dass nach von Kempelen viele weitere Erfinder und Wissenschaftler sich der Aufgabe annahmen, Maschinen zu konstruieren oder Algorithmen zu erdenken, die Schach spielen. Viele gute Bekannte aus der Informatik finden sich darunter, von Charles Babbage über Alan Turing bis zu Konrad Zuse.

Noch heute lesenswert ist der schöne Artikel „Programming a Computer for Playing Chess“ von Claude Shannon [Shannon49]. Gleich zu Beginn führt er aus, dass diese Aufgabe vielleicht von keinerlei praktischem Nutzen ist. Gleichzeitig äußert er aber die Erwartung, dass, wenn man einem Computer beibringen kann, Schach zu spielen, auch andere, ähnlich gelagerte Aufgaben lösbar sein müssten: das Handeln mit Aktien etwa, das Lösen logistischer Probleme oder das Führen militärischer Operationen.

Und so wurde Schach zum Standardbeispiel für künstliche Intelligenz. Während zunächst noch bestritten wurde, dass ein Computer jemals einen menschlichen Spieler schlagen könnte, wurde die Messlatte nach und nach angehoben – entsprechend den Fortschritten in Hard- und Software. Die 90er-Jahre brachten dann medienwirksame Gladiatorenkämpfe aufs Brett: der aktuelle Schachweltmeister stellvertretend für die ganze Menschheit gegen eine Maschine (z. B. Kasparow gegen Deep Blue 1997). Moderne Schachprogramme sind von Normalsterblichen nicht mehr zu schlagen. 2018 erzielte Google im Zusammenhang mit Computerschach noch einmal große Aufmerksamkeit. Die Machine-Learning-Lösung AlphaZero hatte sich das Spielen durch Partien gegen sich selbst quasi selbst beigebracht („Verstärkendes Lernen“). Es kannte zuvor lediglich die Schachregeln und Siegbedingungen. Nach dem Training konnte es mit klassischen Schachprogrammen mehr als nur mithalten und sie mitunter sogar schlagen – ein weiterer Meilenstein für die künstliche Intelligenz.

Eine eigene Schach-Engine

Für Entwicklerinnen und Entwickler stellt das Schreiben eines eigenen Schachprogramms – einer sogenannten Engine – auch heute noch eine interessante Herausforderung dar. Kann ich etwas entwerfen und implementieren, das mich selbst schlägt? Wie schneidet es gegen die Lösungen anderer Entwickler ab? Wie gegen den aktuellen Computerschachweltmeister?

Ich verwende Schach-Engines schon länger als Beispiel in Entwurfsworkshops. Sie eignen sich exzellent, um Designprinzipien und -muster zu demonstrieren, und machen den Teilnehmenden regelmäßig Riesenspaß. Vielleicht kann ich Sie ja auch anstecken und Sie probieren sich selbst einmal daran, eine Engine zu programmieren. Das Fallbeispiel DokChess vermittelt Ihnen ganz nebenbei das nötige Rüstzeug (eigentlich ist es ja ein Beispiel für Dokumentation).

1.5.3 Wem ich danke schön sagen möchte

In einem Buchprojekt wie diesem gibt es viele helfende Hände. Ganz besonders danken möchte ich aber meinem lieben Kollegen Stefan Toth für den tollen Austausch und die vielen guten Ideen! Auch von Gernot Starke habe ich viele wertvolle Anregungen erhalten. Vor allem hatte Gernot mich ermutigt, das Vorhaben überhaupt anzugehen. Ihm und Peter Hruschka gebührt zudem der Dank für die Steilvorlage arc42.

Viele Kolleginnen, Kollegen und Bekannte hatten das Buch (oder Teile davon) vor dem Erscheinen Probe gelesen und Rückmeldungen geliefert. Der ersten und auch der zweiten Auflage folgte reichlich Feedback, in Blogs und Rezensionen, per E-Mail und auch im direkten Kontakt mit mir in Workshops und Kundenprojekten. Auch dafür möchte ich mich bedanken. All dies fließt neben frischen Projekterfahrungen nun in die dritte Auflage ein.

Meiner Familie (Anna, Konstantin und Katharina) möchte ich auch ganz dick Danke sagen!

Darüber hinaus haben mich folgende Personen bei der Erstellung der ersten, zweiten und/oder dritten Auflage besonders unterstützt und ich danke ihnen sehr herzlich dafür: Melanie Andrisek, Brigitte Bauer-Schiewek, André Friedrich, Andreas Flügge, Jan Gentsch, Kay Glahn, Tabea Hentschel, Peter Hruschka, Petra Kienle, Margarete Metzger, Ralf D. Müller, Kay Münch, Karen Paul, Stephan Roth, Kristin Rothe, Axel Scheithauer, Philip Schönholzer, Sandra Schweighart, Falk Sippach, Christel Sohnemann, Markus Stäuble, Uwe Vigenschow, Irene Weilhart und Tim Weilkiens.

Stefan Zörner, im August 2021

Kernaussage dieses Kapitels

Ein wirkungsvolles Festhalten der Architektur hilft Ihnen beim Softwareentwurf, leitet die Umsetzung und führt zu einer nachvollziehbaren und bewertbaren Lösung. Angemessene Dokumentation unterstützt die Kommunikation innerhalb des Teams und gegenüber Dritten. Sie steht nicht im Widerspruch zum agilen Weltbild.

2 Was Softwarearchitektur ist und worauf sie aufbaut

„Architecture is about the important stuff. Whatever that is."[1]

Martin Fowler

Dieses Kapitel klärt zunächst kurz und knapp, was Softwarearchitektur eigentlich ist, wie sie entsteht und wer sie macht. Dabei spielen verschiedene Einflussfaktoren eine Rolle: die Ziele des Auftraggebers etwa, Zeit und Budget, die Skills der Teammitglieder, Performanceanforderungen und, und, und ...

Wer verstehen will, warum eine Lösung so ist, wie sie ist, muss diese Einflussfaktoren kennen. Wenn wir Architekturen dokumentieren wollen, müssen wir daher nicht nur die Lösung selbst, sondern zu einem gewissen Grad auch diese Aspekte festhalten. In diesem Kapitel schlage ich praktikable Werkzeuge und Arbeitsergebnisse dazu vor. Ich demonstriere sie anhand zweier Fallbeispiele, die Sie konsequent durch das Buch begleiten werden.

2.1 Softwarearchitektur-Freischwimmer

Bevor wir uns ins tiefe Wasser begeben, lege ich fest, was im weiteren Verlauf dieses Buchs unter Softwarearchitektur verstanden wird. Anschließend gehe ich kurz auf ein mögliches Vorgehen und die Rolle Softwarearchitekt/in ein und zeige auf, welche Konsequenzen das für Architekturdokumentation hat.

2.1.1 Was ist Softwarearchitektur?

Für den Begriff „Softwarearchitektur" gibt es keine allgemein anerkannte Definition. Stattdessen gibt es sehr viele unterschiedliche Definitionen, die sich in einschlägigen Fachbüchern, Artikeln, Blogs etc. finden. Eine umfangreiche Sammlung solcher Definitionen hat das Software Engineering Institute der Carnegie Mellon University zusammengetragen und stellt sie online bereit [SEI]. Mein persönlicher Favorit:

[1] Deutsch etwa: Bei Architektur geht es um die wichtigen Sachen, welche das auch immer sind. [Fowler2003]