1 Einleitung

1.1 Motivation

Agile Methoden legen in Softwareentwicklungsprojekten großen Wert auf direkte Kommunikation von Angesicht zu Angesicht und nehmen gegenüber umfangreicher schriftlicher Dokumentation eine eher kritische Haltung ein. Sie erkennen aber an, dass Dokumentation sinnvoll sein kann und manchmal auch benötigt wird.

Daraus resultiert die Frage, wie die Dokumentation in einem agilen Kontext sinnvoll gestaltet werden kann. Die Möglichkeiten dafür auszuloten ist Gegenstand dieses Buchs. Das Ziel dabei ist, bei der Planung, der Erstellung und der Nutzung von Dokumentation Strategien zu entwickeln, die sich am besten mit dem Wort *bedarfsgerecht* beschreiben lassen – die sich also an den tatsächlichen Bedürfnissen eines Projekts orientieren [Rüping 2011].

Ein bedarfsgerechtes Vorgehen bezieht sich zum einen darauf, *was* in einem Projekt schriftlich dokumentiert werden sollte (und was nicht). Es betrifft zum anderen die Frage, *wie* eine angemessene Dokumentation gestaltet werden kann. Beiden Fragen werden wir in diesem Buch nachgehen.

1.2 Historie der agilen Entwicklung

Um die Sichtweise der agilen Methoden verstehen zu können und speziell auch, was zu der betont kritischen Sicht auf umfangreiche Dokumentation geführt hat, lohnt sich ein Rückblick auf die 1990er-Jahre. In dieser Zeit haben die ersten agilen Verfahren ihren Ursprung. Blicken wir also einmal auf die damals gängigen oder zumindest doch empfohlenen Methoden in der Softwareentwicklung.

Prozesslastige Methoden in den 1990er-Jahren

Die 1990er-Jahre waren geprägt von einer Reihe von Methoden, die großen Wert auf definierte Prozesse und umfangreiche Dokumentation legten. Diese Methoden waren durch das Ziel gekennzeichnet,

den Ablauf eines Projekts im vorhinein detailliert zu planen und sich bei der anschließenden Umsetzung streng am Plan zu orientieren.

Fast alle diese Methoden basieren auf dem klassischen Wasserfallmodell, das den Prozess der Softwareentwicklung in verschiedene Phasen unterteilt (Grobspezifikation, Feinspezifikation, Entwurf, Realisierung, Test, Auslieferung), die typischerweise streng sequenziell durchlaufen werden. Dokumentation spielt beim Wasserfallmodell insofern eine große Rolle, als die Ergebnisse einer Phase in der Regel gründlich dokumentiert werden und die Dokumentation als Input für die nächste Phase bereitgestellt wird.

Typische Vertreter dieser Methoden sind die folgenden:

- Das V-Modell ist ein Vorgehensmodell zur Planung und Durchführung von Softwareentwicklungsprojekten, dessen Ursprünge ins Jahr 1979 zurückgehen. Es erweitert das traditionelle Wasserfallmodell, indem es jeder der ursprünglichen Phasen eine Test- oder Abnahmephase gegenüberstellt. Während das originale V-Modell durch relativ starre Prozesse gekennzeichnet war, lässt die 2005 in Deutschland eingeführte Variante XT bereits eine gewisse Anpassung der Prozesse an spezifische Gegebenheiten zu.
- Das 1986 erstmals vorgestellte Spiralmodell ist ein Vorgehensmodell, das ebenfalls auf dem Wasserfallmodell beruht, es aber bereits um die Idee der iterativen Entwicklung erweitert. Es sieht vor, typische Phasen wie Analyse, Entwurf, Realisierung und Test immer wieder zu durchlaufen und sich so schrittweise dem Projektziel zu nähern.
- Der 1998 erstmals veröffentlichte Rational Unified Process (RUP) [Kruchten 1998] umfasst ein Vorgehensmodell zur Softwareentwicklung, das zur Modellierung die bekannte Unified Modeling Language (UML) [Rumbaugh/Jacobsen/Booch 1998] einsetzt. Auch der Rational Unified Process weicht bereits deutlich vom klassischen Wasserfallmodell ab und empfiehlt stattdessen ein iteratives Vorgehen.

Ebenso prägend für die 1990er-Jahre waren Bemühungen, durch die Verbesserung der zugrunde liegenden Prozesse die Qualität von Software zu erhöhen. Resultat dieser Bemühungen waren verschiedene Prozessframeworks, die zwar kein spezielles Verfahren vorschreiben, aber doch dazu auffordern, innerhalb eines bestimmten Rahmens Prozesse und Vorgehen zu definieren. Zu den oben genannten Softwareentwicklungsmethoden sind diese Prozessframeworks in dem Sinne orthogonal, dass sie unterschiedliche Schwerpunkte setzen und

sich mit diesen kombinieren lassen. Bekannte Prozessframeworks sind die folgenden:

- Das Capability Maturity Model Integration (CMMI) ist ein Modell zur Beurteilung des Reifegrads sämtlicher Prozesse eines Unternehmens im Zusammenhang mit der Entwicklung und dem Betrieb von Software. Ziel ist die Definition, Planung, Implementierung und Qualitätssicherung dieser Prozesse. Das relativ starre ursprüngliche Modell (CMM) wurde 2003 durch den flexibleren Nachfolger (CMMI) abgelöst.
- Die Normenreihe ISO 9000 ff. legt Mindestanforderungen für die Qualitätssicherung fest. Die Normen beziehen sich auf Produktherstellung und Dienstleistungen generell und werden gelegentlich auch auf die Softwareentwicklung angewendet. Seit Mitte der 1990er-Jahre sind manche Softwareunternehmen bestrebt, sich in Bezug auf ihre Qualitätsstandards nach ISO 9000 zertifizieren zu lassen.

Sowohl die genannten Softwareentwicklungsmethoden als auch die Prozessframeworks zur Qualitätssicherung sind im Laufe der Zeit weiterentwickelt worden. In einigen Fällen drückt sich dies in der Namensgebung aus. So trat zum Beispiel an die Stelle des ursprünglichen V-Modells die Variante XT, und CMM wurde durch CMMI abgelöst.

Die traditionellen Entwicklungsmethoden, die ursprünglich auf dem Wasserfallmodell basierten, haben sich dabei tendenziell auf ein stärker iteratives Vorgehen hin bewegt. Die Prozessframeworks haben im Laufe der Zeit an Flexibilität gewonnen und tragen mittlerweile der Tatsache Rechnung, dass Projekte individuell geprägt sind und dass Prozesse an die individuellen Gegebenheiten angepasst werden müssen. Dementsprechend befinden sich Prozessframeworks wie beispielsweise CMMI heutzutage nicht mehr unbedingt im Widerspruch zu agilen Methoden [Glazer/Dalton/Anderson/Konrad/Shrum 2008].

Ende der 1990er-Jahre stellte sich die Situation allerdings noch anders dar. Die genannten Entwicklungsmethoden waren damals noch bestrebt, allgemeingültige Modelle für die Softwareentwicklung aufzustellen, und begannen sich nur langsam vom Wasserfallmodell zu lösen. Sie wurden zunehmend als starr, schwergewichtig und wenig flexibel wahrgenommen. Mit dem Aufkommen der Prozessframeworks rückten Prozessdefinitionen noch mehr in den Mittelpunkt der Softwareentwicklung, was den Eindruck von Schwergewichtigkeit und mangelnder Flexibilität weiter verstärkte.

Die Kombination aus wasserfallbasierten Entwicklungsmethoden und vergleichsweise starren Prozessdefinitionen hat in der Praxis immer wieder zu massiven Problemen geführt, die in manchen Fällen auch für das Scheitern großer Projekte ursächlich waren:

- Der Overhead, der durch die Prozesslastigkeit und die häufig damit verbundene Bürokratie dieser Methoden entsteht, kann immens sein und kann die Kreativität und Produktivität des Entwicklungsteams regelrecht ersticken. In manchen, nach schwergewichtigen Verfahren durchgeführten Projekten wurde mehr Aufwand darin investiert, den formalen Kriterien des Softwareentwicklungsprozesses zu genügen als tatsächlich Software zu entwickeln.
- Der Wunsch, jede noch so kleine Anforderung, aber auch jeden Entwicklungsschritt im Projekt zu dokumentieren, hat häufig zu Bergen von Dokumentation geführt, die niemand mehr hat lesen können.
- Die exakte Planung eines Projekts für mehrere Monate oder gar Jahre in die Zukunft ist schwierig bis unmöglich. Das Geschehen im Projekt ist schlecht vorhersehbar. Anforderungen ändern sich im Laufe der Zeit, und auch eine gutgemeinte Planung hat sich oft als unzuverlässig herausgestellt.
- Die häufig mit prozesslastigen Verfahren einhergehende sequenzielle Abfolge der einzelnen Projektphasen führt dazu, dass der Kunde die entstandene Software erst gegen Ende des Projekts zu sehen bekommt. Zu einem so späten Zeitpunkt ist es sehr schwer, das Feedback des Kunden auf die entwickelten Systeme noch zu berücksichtigen und möglicherweise noch eine Kurskorrektur vorzunehmen.

Als Konsequenz aus diesen Problemen kam gegen Ende der 1990er-Jahre immer häufiger die Forderung auf, sich von starren, schwergewichtigen Prozessen zu lösen und stattdessen mehr Wert auf Flexibilität zu legen.

Forderung nach mehr Flexibilität

Unter Flexibilität wird dabei die Fähigkeit verstanden, sich an wechselnde Bedingungen und Anforderungen anzupassen. In der Evolutionsbiologie ist diese Fähigkeit eine Grundvoraussetzung für das Überleben von Arten. Analog dazu ist diese Flexibilität auch in Entwicklungsprojekten ein entscheidendes Kriterium, das für den Erfolg oder Misserfolg eines Projekts ausschlaggebend sein kann. Im gleichen Zug wurde gefordert, die eigentliche Erstellung der Software wieder mehr in den Mittelpunkt des Projektgeschehens zu stellen und hingegen Prozesse und Methoden als Mittel zum Zweck zu betrachten und

immer wieder auf ihren Sinn und Zweck hin zu überprüfen. Die Idee eines agilen Vorgehens war geboren.

Agiles Manifest

In der Folge haben einige der frühen »Agilisten« begonnen, Praktiken agilen Vorgehens zu entwickeln und in ihrer alltäglichen Projektpraxis anzuwenden. Im Februar 2001 trafen sich 17 dieser Experten zu einem Workshop mit dem Ziel, ihre Erfahrungen mit dem neuen Vorgehen auszutauschen. Bei den Diskussionen im Rahmen dieses Workshops hat sich dann das herauskristallisiert, was heute als die Prinzipien agiler Entwicklung verstanden wird. Das Ergebnis des Workshops ist später als das Agile Manifest bekannt geworden [Agile Alliance 2001]. Abbildung 1–1 zeigt die Kernaussage des Agilen Manifests in der deutschen Übersetzung.

Manifest für Agile Softwareentwicklung

Wir erschließen bessere Wege, Software zu entwickeln,
indem wir es selbst tun und anderen dabei helfen.
Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:

Individuen und Interaktionen
mehr als Prozesse und Werkzeuge

Funktionierende Software
mehr als umfassende Dokumentation

Zusammenarbeit mit dem Kunden
mehr als Vertragsverhandlung

Reagieren auf Veränderungen
mehr als das Befolgen eines Plans

Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden,
schätzen wir die Werte auf der linken Seite höher ein.

Abb. 1–1
Agiles Manifest

Neben diesen grundlegenden Prinzipien benennt das Agile Manifest auch eine Reihe von Kernpraktiken, die sich für die konkrete Anwendung in der täglichen Projektpraxis eignen. Einige davon sind die folgenden:

- Software wird inkrementell entwickelt. Das Team liefert kontinuierlich die jeweils entwickelten Inkremente an den Kunden aus.
- Die Menge funktionierender Software dient als primäres Fortschrittsmaß.

- Fachexperten und Entwickler kooperieren eng miteinander, idealerweise auf täglicher Basis.
- Die wichtigste Kommunikationsform der Projektbeteiligten ist das Gespräch von Angesicht zu Angesicht.
- Einfachheit ist ein Prinzip. Einfachen Lösungen wird der Vorzug vor komplizierten Lösungen gegeben.
- Das Team reflektiert regelmäßig sein Vorgehen und passt das Vorgehen gegebenenfalls an.

Wenngleich das Agile Manifest diese und eine Reihe weiterer Praktiken empfiehlt, so beschreibt es dennoch keine spezifische Entwicklungsmethode. Durch die Formulierung grundlegender Prinzipien gibt es vielmehr den Rahmen vor, in dem sich die agile Softwareentwicklung bewegt. Für die konkrete Ausgestaltung einzelner Methoden bleibt dabei noch viel Spielraum. Dieser Spielraum ist auch notwendig, weil Projekte sehr unterschiedlich sind, zum Beispiel im Hinblick auf Anzahl der beteiligten Personen, Laufzeit, Umfang der Funktionalität, Komplexität, Technologie und Kritikalität. Nicht alle Projekte können nach derselben Methode durchgeführt werden. Verschiedene agile Methoden interpretieren daher den Rahmen, den das Agile Manifest vorgibt, auf durchaus unterschiedliche Art und Weise. Viele der Methoden weisen durchaus Ähnlichkeiten auf und können auch gut miteinander kombiniert werden, dennoch gibt es auch eine Vielzahl von Unterschieden.

Verschiedene agile Methoden

Im Laufe der Zeit hat sich eine Reihe von agilen Methoden zur Softwareentwicklung etabliert, die in ihrer konkreten Ausgestaltung unterschiedliche Akzente setzen und auch unterschiedliche Verbreitung erfahren haben.

- Eine der frühesten agilen Methoden ist eXtreme Programming, häufig auch einfach als XP bezeichnet [Beck 2000; Wolf/Roock/Lippert 2005]. Seit dem Jahr 2000 hat XP einige Popularität erreicht. XP basiert auf einer Reihe bewährter Praktiken, zu denen beispielsweise inkrementeller Entwurf, testgetriebene Entwicklung sowie kontinuierliche Integration gehören. Bekannt geworden ist XP vor allem für die Einführung von Pair Programming.
- Bei Crystal [Cockburn 2002] handelt es sich um eine Familie von Methoden, die abhängig von der Anzahl der Projektbeteiligten und von der Kritikalität des Projekts verschiedene Strategien zur Kommunikation und zur Qualitätssicherung vorschlagen.
- Scrum ist mittlerweile die bekannteste agile Methode, zumindest im deutschsprachigen Raum [Schwaber/Beedle 2008; Pichler 2007; Cohn 2010; Wolf/van Solingen/Rustenburg 2010; Gloger 2011;

Wirdemann 2011; Röpstorff/Wiechmann 2012]. Kennzeichen von Scrum ist ein iteratives Vorgehen, das sich in regelmäßigen Intervallen, den sogenannten Sprints, ausdrückt. In diesen Sprints werden jeweils Inkremente der Software entwickelt und ausgeliefert. Am Ende jeden Sprints steht unter anderem eine Retrospektive zur Überprüfung des eigenen Vorgehens.

- Feature-Driven Development (FDD) ist ein leichtgewichtiges Verfahren, das ausgehend von einem Gesamtmodell eine Liste einzelner Features entwickelt und die gesamte Entwicklung dann in die Implementierung der einzelnen Features herunterbricht [Palmer/Felsing 2002]. FDD ist vielleicht weniger »revolutionär« als beispielsweise XP oder Scrum, befindet sich aber durchaus im Einklang mit den Prinzipien agiler Entwicklung.
- Eine der neueren agilen Methoden ist Kanban [Anderson 2011], das seinen Ursprung in den Prinzipien des Lean Development hat [Poppendieck/Poppendieck 2003]. Kanban versucht durch die Reduktion paralleler Arbeit den gesamten Fluss aller Projekttätigkeiten zu erhöhen.

Einige dieser Methoden haben mittlerweile einen großen Bekanntheitsgrad erlangt und gehören zum Standardrepertoire heutiger Softwareentwicklung. Insbesondere mit XP, Scrum und Kanban sind bereits viele Projekte erfolgreich durchgeführt worden [Wolf 2011].

Neben diesen Methoden haben noch weitere Techniken aufgrund ihres agilen Charakters eine gewisse Bekanntheit erlangt. Zu nennen sind hier insbesondere die folgenden:

- Bei testgetriebener Entwicklung (Test-Driven Development) handelt es sich um eine Strategie, die das Testen von Software in den Vordergrund stellt (nachdem es über lange Zeit hinweg in vielen Projekten nur ein Schattendasein gefristet hatte). Das Prinzip ist dabei, der Entwicklung bestimmter Funktionen immer die Entwicklung entsprechender Tests vorausgehen zu lassen [Westphal 2005]. Testgetriebene Entwicklung ist als Einzeltechnik Bestandteil vieler agiler Verfahren.
- Agile Modeling ist ein Verfahren, das sich auf die Modellierungsaspekte in Projekten konzentriert und dafür leichtgewichtige und durch starke Interaktion geprägte Prozesse empfiehlt [Ambler 2002].
- DevOps ist ein vergleichsweise neuer Ansatz, der das Ziel verfolgt, ein agiles Vorgehen auf den Betrieb von Software auszudehnen [Peschlow 2011]. Schwerpunkt ist dabei die enge Kooperation zwischen Entwicklern und Betriebsabteilung. Durch die regelmäßige

Auslieferung von Software geht deren Inbetriebnahme in einen einfachen, erprobten und reproduzierbaren Prozess über.

Zum Thema Dokumentation gibt es im Agilen Manifest keine Vorschriften, was das konkrete Vorgehen im Projekt angeht. Hingegen vermittelt das Agile Manifest eine grundsätzliche Haltung, die zwar den Wert von Dokumentation anerkennt, diesen Wert aber auch gegenüber anderen Projektzielen relativiert. Gerade vor dem historischen Hintergrund ist dies nur zu verständlich.

Die schwergewichtigen Methoden der 1990er-Jahre hatten allesamt ein gehöriges Maß an Dokumentation mit sich gebracht. Ziel war damals gewesen, jegliche Information, die in einer bestimmten Projektphase benötigt wird, vorher schriftlich zu fixieren. In manchen Projekten wurden über einen Zeitraum von Jahren nur Spezifikationen und Konzeptpapiere erstellt, bevor überhaupt eine einzige Zeile Code programmiert wurde. Dabei sind in der Regel große Mengen an Dokumentation entstanden, die in ihrer Fülle gar nicht gelesen werden konnten und außerdem schneller veraltet sind, als es jedem Projektbeteiligten recht sein konnte. Insider sprechen in einem solchen Fall von *write-only documentation*.

Dokumentation als Mittel zum Zweck

Ein gemeinsames Merkmal aller agiler Methoden ist, dass sie derartige Szenarien vermeiden wollen. Der Grund hierfür liegt in der Erkenntnis, dass Dokumentation nicht automatisch zum Verständnis der Materie führt, wie auch Abbildung 1–2 andeutet. Beim Schreiben wandert nur ein Teil des in den Köpfen vorhandenen Wissens tatsächlich auch ins Dokument, manches hingegen bleibt unausgesprochen. Gleichermaßen erfassen auch gründliche Leser nicht immer alles, was in einem Dokument beschrieben ist.

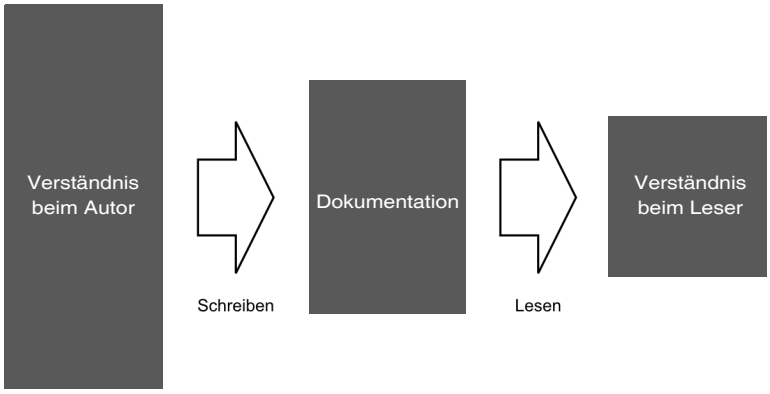

Abb. 1–2
Dokumentation vs. Verständnis

Ein bisschen ähnelt schriftliche Dokumentation damit dem Prinzip der stillen Post, was die Schlussfolgerung zulässt, dass Wissen mithilfe von schriftlicher Dokumentation eben nicht vollständig weitergegeben werden kann und sich beim Leser nicht automatisch ein Verständnis der Materie einstellt. Weil Verständnis der Materie aber das ist, worauf es im Projekt letztlich ankommt, ist eine kritische Haltung gegenüber der Dokumentation nur folgerichtig. In manchen Fällen ist Dokumentation zweifellos sinnvoll, in anderen Fällen gibt es andere Kommunikationskanäle, die für den Transfer von Wissen eben besser geeignet sind.

Diese Erkenntnis hat die 17 Teilnehmer des Workshops, auf dem das Agile Manifest entstanden ist, dazu gebracht, umfassende Dokumentation auf der rechten Seite anzusiedeln (vgl. Abb. 1–3), also dort, wo diejenigen Dinge beschrieben sind, die zwar einen Nutzen für ein Projekt haben, die aber als ein Mittel zum Zweck (und eben nicht als Selbstzweck) betrachtet werden müssen.

Abb. 1–3
Agiles Manifest mit Hervorhebung der Dokumentation

Leider gibt es auch immer wieder Projekte (manchmal auch solche, die sich agil nennen), die die Frage nach der Notwendigkeit möglicher Dokumentation erst gar nicht stellen. In diesen Projekten wird nur rudimentär dokumentiert oder auf die Erstellung von Dokumentation fast vollständig verzichtet – aber eben nicht, weil gute Gründe dafür

sprechen, auf bestimmte Dokumente zu verzichten, sondern weil das Thema Dokumentation einfach ignoriert wird. Die Problematik dieses Ansatzes wird spätestens dann deutlich, wenn Wissensträger das Projekt verlassen haben, die Funktionsweise der Software nicht mehr nachvollziehbar ist und ihre Wartung oder Weiterentwicklung nahezu unmöglich wird, ohne ein umfangreiches Reverse-Engineering zu betreiben, was letztlich eine aufwendige und wenig dankbare Angelegenheit ist. Das ist natürlich kein kluges Vorgehen, und tatsächlich hat das mit Agilität auch nichts mehr zu tun.

Bedarfsgerechte Dokumentation

Beide Extrempositionen helfen uns also nicht weiter: Gar nichts zu dokumentieren ist ebenso unsinnig wie der Versuch, alles dokumentieren zu wollen. Ein vernünftiger Ansatz besteht darin, die Dokumentation auf ihren Nutzen hin kritisch zu prüfen und den Aufwand für die Erstellung transparent zu machen, um letztlich das richtige Maß zu finden.

Hierfür müssen wir die Dokumentation immer auch anderen Formen der Kommunikation gegenüberstellen, insbesondere der direkten Kommunikation von Angesicht zu Angesicht, die aufgrund ihres hohen Interaktionsgrads bei agilen Methoden einen hohen Stellenwert genießt. Allerdings gibt es keine generell *beste* Kommunikationsform – in unterschiedlichen Situationen können verschiedene Kommunikationsformen jeweils ihre unterschiedlichen Vorzüge ausspielen, wie Tabelle 1–1 verdeutlicht.

Tab. 1–1 *Vergleich von direkter Kommunikation und Dokumentation*

Direkte Kommunikation	Dokumentation
Interaktion ■ kurze Frage-Antwort-Zyklen ■ informeller Informationsfluss	Individuelle Geschwindigkeit ■ Wissensaufnahme im persönlichen Tempo
Einschluss nonverbaler Kommunikation ■ Körpersprache ■ Gestik	Schriftliche Ausdrucksmöglichkeit ■ Vorteil für introvertierte Personen ■ möglicher Erkenntnisgewinn beim Schreiben
Unterstützung für Prozesse im Team ■ persönlicher Umgang der beteiligten Personen	Skalierbarkeit ■ Nutzung durch viele Personen ■ Nutzung durch räumlich verteilte Teams
Schnelle Verfügbarkeit ■ Ansprechpartner im Team	Langfristige Verfügbarkeit ■ Verfügbarkeit nach Abschluss eines Projekts

Die Punkte auf der rechten Seite dieser Tabelle beschreiben sehr gut die Motivation dafür, warum und in welchen Situationen wir im Projekt Dinge schriftlich dokumentieren sollten. Wir werden viele dieser

Punkte später im Detail aufgreifen, insbesondere in Kapitel 2, in dem es um die Grundlagen agiler Dokumentation geht.

1.3 Muster

Projekte sind individuell geprägt. Was in einem Projekt gut funktioniert, ist in einem anderen Projekt weniger angebracht. Es ist illusorisch zu glauben, man könne konkrete Prozesse definieren, die sich organisations- und projektübergreifend immer wieder erfolgreich einsetzen lassen. Die Rahmenbedingungen sind einfach zu unterschiedlich. Dies gilt insbesondere auch für Dokumentation, weshalb dieses Buch auch gar nicht den Versuch macht, eine einheitliche und allgemeingültige Dokumentationsstrategie zu entwickeln.

Praxiserprobte Muster

Stattdessen möchte ich in diesem Buch eine Reihe einzelner Techniken und Strategien vorstellen, die sich in der Praxis immer wieder bewährt haben. Ich habe diese Techniken und Strategien als Muster (englisch *Patterns*) formuliert, ganz im Sinne der Entwurfsmuster (englisch *Design Patterns*), die in den letzten Jahren eine weite Verbreitung in der praxisorientierten Literatur zur Softwareentwicklung erfahren haben. Entwurfsmuster beschreiben bewährte Lösungen für immer wiederkehrende Probleme, und genau das tun die Muster in diesem Buch auch, nur mit dem Unterschied, dass sie sich nicht auf Programmierpraktiken, sondern auf Dokumentationspraktiken beziehen.

Die Formulierung von Mustern bietet eine Reihe von Vorteilen:

- Muster sind praxiserprobt und geeignet, tatsächliche Erfahrungen zu transportieren.
- Die Struktur als Problem-Lösungs-Paar ist für einen praktischen Einsatz geeignet.
- Muster analysieren die verschiedenen Faktoren, die Einfluss auf mögliche Lösungsszenarien haben, und tragen so zu ausbalancierten Lösungen bei.
- Muster referenzieren sich gegenseitig, was in diesem Buch typografisch durch die Verwendung von Kapitälchen ausgedrückt ist. Muster können so leicht miteinander kombiniert werden.

Sämtliche Muster in diesem Buch (in den Kapiteln 2 bis 7) folgen einem einheitlichen Aufbau:

- Der *Kontext* beschreibt, in welcher Situation ein Muster angewendet werden kann.
- Das *Problem* wirft eine Frage (im Zusammenhang mit Dokumentation) auf, die sich im beschriebenen Kontext immer wieder stellt

und deren Nichtbeantwortung spürbare Nachteile mit sich bringen würde.
- Die *Analyse* beleuchtet mögliche Lösungsansätze, vergleicht sie miteinander und nennt ihre Vor- und Nachteile.
- Die *Lösung* beschreibt eine prinzipielle Strategie oder Technik, mit der sich das Problem vermeiden oder zumindest abmildern lässt.
- Die *Details* beschreiben, wie die Lösung umgesetzt werden kann, und schlagen dafür konkrete Schritte vor.
- Die *Diskussion* nennt weiter gehende Aspekte im Zusammenhang mit der beschriebenen Lösung, insbesondere auch den Zusammenhang zu anderen Mustern.

Zum besseren Verständnis werden die Muster in diesem Buch noch um weiteres Material ergänzt. Da dieses Material über die eigentlichen Muster hinausgeht, ist es in grauen Kästen dargestellt.

- Zunächst stelle ich gelegentlich Beispiele vor, die jeweils ein bestimmtes Muster verdeutlichen. Diese Beispiele stammen aus »echten« Projekten, haben sich also in der Realität so abgespielt (wenngleich ich aus Gründen des Kundenschutzes die Beispiele anonymisiert habe).
- Außerdem ergänze ich die Muster gelegentlich um Hintergrundinformationen, die tiefer gehende oder auch kontroverse Aspekte zu einem bestimmten Thema beleuchten.

Wie bereits angedeutet, beziehen sich die Muster in diesem Buch auf ein agiles Vorgehen im Allgemeinen und sind nicht spezifisch für irgendeine agile Methode. Zur Erläuterung verweise ich allerdings gelegentlich auf Scrum, weil Scrum aktuell das am weitesten verbreitete agile Verfahren ist. Trotzdem sind die in diesem Buch beschriebenen Muster problemlos auch im Kontext anderer Methoden anwendbar. Tatsächlich sind sie auch dann anwendbar, wenn überhaupt keine agile Methode im Einsatz ist, aber das Projektteam den Wunsch hat, agiler zu werden, und diesen Wunsch am Beispiel der Dokumentation in die Tat umsetzen möchte.

Dabei ist es wichtig zu verstehen, dass das Wort »agil« primär eine Haltung, nicht aber eine Technik oder einen Prozess beschreibt. Nicht ohne Grund stehen im Agilen Manifest Prozesse (genau wie die Dokumentation) auf der Seite der Dinge, die eher ein Mittel zum Zweck als ein eigenständiges Ziel darstellen. Die Muster in diesem Buch verfolgen daher nicht den Zweck, einen bestimmten Dokumentationsprozess zu formulieren. Stattdessen können Sie diese Muster als Bausteine nutzen, wenn Sie für Ihre Projekte und die dort bestehenden Rahmen-

bedingungen ein bedarfsgerechtes Vorgehen in puncto Dokumentation entwickeln.

1.4 Nutzung des Buchs

Der Aufbau des Buchs orientiert sich weitgehend am iterativen Vorgehen in agilen Projekten. Die Kapitelstruktur spiegelt daher in etwa den Lebenszyklus von Dokumenten in einem agilen Kontext wider.

- *Einstieg in ein agiles Vorgehen* — *Kapitel 2*
  Bei agilen Verfahren ist die prinzipielle Haltung zur Dokumentation schon ein Stück weit anders als bei traditionellen Methoden der Softwareentwicklung. Das erste Kapitel stellt daher ein paar grundlegende Prinzipien vor, die essenzieller Bestandteil eines agilen Vorgehens sind.
- *Infrastruktur und Werkzeuge* — *Kapitel 3*
  Die Erstellung von Dokumenten erfordert eine geeignete Infrastruktur. Dieses Kapitel beschreibt, wie eine solche Infrastruktur aussehen kann und welche Werkzeuge sich als nützlich herausgestellt haben.
- *Planung der Dokumentation* — *Kapitel 4*
  In diesem Kapitel steigen wir in die konkrete Planung der Dokumentation ein. Das Kapitel gibt konkrete Tipps, wie diese Planung gestaltet werden kann, und beschreibt, welche Art von Dokumentation in einem agilen Kontext typischerweise benötigt wird.
- *Auswahl der richtigen Inhalte* — *Kapitel 5*
  Nachdem wir geplant haben, welche Dokumente wir benötigen, handelt dieses Kapitel davon, welche Inhalte für diese Dokumente sinnvoll sind. Der Schwerpunkt liegt darauf, Inhalte zu identifizieren, die es Wert sind, schriftlich festgehalten zu werden.
- *Gestaltung einzelner Dokumente* — *Kapitel 6*
  Gute Dokumentation ist logisch aufgebaut und vernünftig strukturiert. Die Muster in diesem Kapitel adressieren dieses Thema und gehen auch kurz auf Aspekte des Layouts ein. Tatsächlich sind diese Muster auch außerhalb eines agilen Kontexts sinnvoll, aber sie gelten natürlich auch hier.
- *Umgang mit der Dokumentation* — *Kapitel 7*
  Dieses Kapitel ist der Frage gewidmet, was mit der Dokumentation geschehen soll, wenn sie denn geschrieben ist. Ziel ist es sicherzustellen, dass die Dokumentation nicht »verstaubt«, sondern sinnvoll genutzt werden kann.

Leitfaden zum Lesen des Buchs

Generell gibt es zwei verschiedene Möglichkeiten, dieses Buch zu lesen. Die erste ist die übliche Methode: Sie können das Buch ganz traditionell von vorn bis hinten durchlesen und dabei den Lebenszyklus von Dokumenten im Projekt nachempfinden.

Alternativ können Sie das Buch aber auch einfach durchblättern, sich dabei auf die (kursiv gesetzten) Kernaussagen der einzelnen Muster konzentrieren, und dort tiefer einsteigen, wo Sie ein Thema besonders interessiert. Dadurch, dass die Muster jeweils Verweise auf andere Muster enthalten (in Kapitälchen gesetzt), können Sie auf diese Art und Weise durch das Buch »navigieren«.